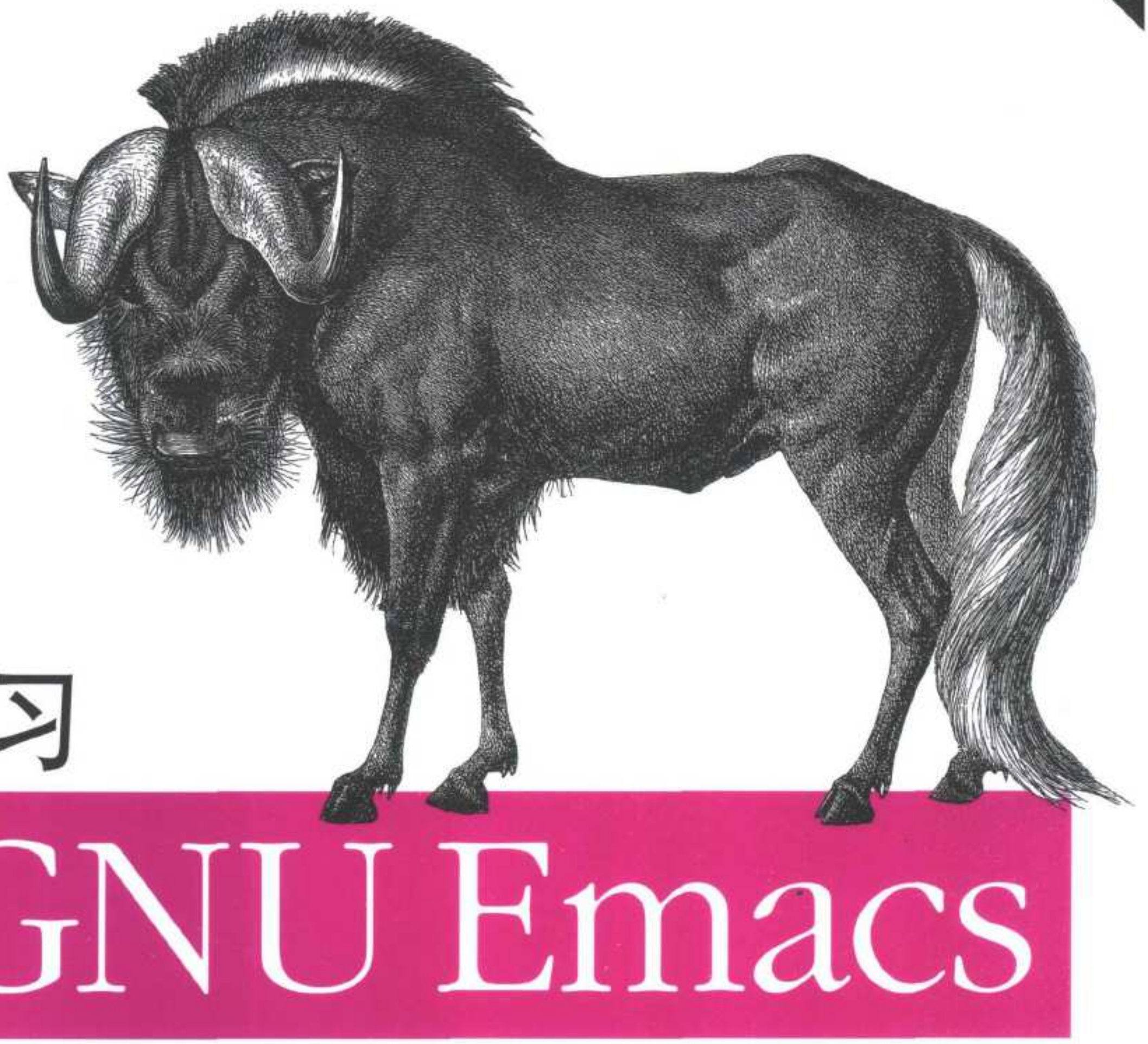
Learning GNU Emacs

第二版



学习 GNU Emacs

O'REILLY®
机械工业出版社
China Machine Press
HZ BOOKS

Debra Cameron, Bill Rosenblatt
& Eric Raymond 著
杨涛 杨晓云 王建桥 等译

UNIX

O'REILLY®

学习GNU Emacs

GNU Emacs是Emacs编辑器家族中最受欢迎、传播范围最广、也是最强大和最灵活的UNIX文本编辑器。与其他文本编辑器的重要区别在于它是一个完备的工作环境，使用Emacs可以完成各种日常工作。本书循序渐进地讲述Emacs的入门知识，随着本书的深入，读者的Emacs使用水平将从初级（只会进行简单的文字编辑）提高到足以完成相当复杂的定制和程序设计任务的阶段。

本书对Emacs 19.30中的新增功能做了全面的介绍，内容涉及字体和颜色、下拉菜单、卷屏条、增强的X窗口系统支持，以及对大多数标准按键进行正确的绑定。此外，书中还对Emacs自带的新闻阅读器程序Gnus和文件传输协议的透明接口ange-ftp模式进行了介绍。

本书内容包括:

- 把Emacs当做一个因特网工具箱来使用
- Emacs丰富而又全面的在线帮助功能
- 如何使用Emacs来编辑文件
- 把Emacs当做一种“shell环境”来使用
- 如何利用Emacs内建的排版功能
- 如何使用多个编辑缓冲区、多个Emacs窗口、多个X窗口
- 对Emacs进行定制
- Emacs到X窗口系统的接口
- 用宏来完成重复性工作的理由和方法
- 作为程序设计环境的Emacs
- Emacs LISP程序设计入门
- 如何获得Emacs

书后所附的速查卡列出了书中介绍的全部命令。

ISBN 7-111-10348-3

9 787111 103486 >

O'Reilly & Associates, Inc. 授权机械工业出版社出版

ISBN 7-111-10348-3

定价：68.00元

学习 GNU Emacs

第二版

Debra Cameron, Bill Rosenblatt
& Eric Raymond 著

杨涛 杨晓云 王建桥 等译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly & Associates, Inc. 授权机械工业出版社出版

机械工业出版社

图书在版编目（CIP）数据

学习 GNU Emacs（第二版）/（美）卡马伦（Cameron, D.）等著；杨涛等译．
－北京：机械工业出版社，2002.2

书名原文：Learning GNU Emacs, Second Edition
ISBN 7-111-10348-3

I. 学… II. ①卡… ②杨… III. 文字处理系统，Emacs IV. TP391.12

中国版本图书馆 CIP 数据核字（2002）第 039495 号

北京市版权局著作权合同登记
图字：01-2002-1850 号

Copyright©1996 by O'Reilly & Associates, Inc.

Simplified Chinese Edition, jointly published by O'Reilly & Associates, Inc. and China Machine Press, 2002. Authorized translation of the English edition, 1996 O'Reilly & Associates, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly & Associates, Inc. 出版 1996。

简体中文版由机械工业出版社出版 2002。英文原版的翻译得到 O'Reilly & Associates, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 —— O'Reilly & Associates, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

书　　名 / 学习 GNU Emacs（第二版）
书　　号 / ISBN 7-111-10348-3/TP.2457
责任编辑 / 史宗海，陆颖
封面设计 / Edie Freedman，张健
出版发行 / 机械工业出版社
地　　址 / 北京市西城区百万庄大街 22 号（邮政编码 100037）
经　　销 / 新华书店北京发行所发行
印　　刷 / 北京牛山世兴印刷厂
开　　本 / 787 毫米 × 1092 毫米　16 开本　37.5 印张　554 千字
版　　次 / 2003 年 6 月第一版　2003 年 6 月第一次印刷
印　　数 / 0001-4000 册
定　　价 / 68.00 元（册）

（凡购本书，如有倒页、脱页、缺页，由本社发行部调换）

O'Reilly & Associates 公司介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly & Associates公司授权机械工业出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly & Associates 公司是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到 WebSite（第一个桌面PC的Web服务器软件），O'Reilly & Associates 一直处于Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly & Associates公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly & Associates 还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly & Associates 依靠他们及时地推出图书。因为O'Reilly & Associates紧密地与计算机业界联系着，所以O'Reilly & Associates 知道市场上真正需要什么图书。

O'Reilly 新书推荐

《学习 bash》（第二版）

作者：*Cameron Newham & Bill Rosenblatt*

bash 是自由软件基金会发布的“Bourne Again Shell”的缩写，是流行的 UNIX Bourne shell 的开源软件替代产品，供全球 Linux 用户选用。《学习 bash》（第二版）正是学习 bash 的权威指南。

本书教授了如何使用bash的高级命令行特性，如历史命令、命令行编辑和自动完成命令，还介绍了 shell 编程（这是没有一个UNIX 或Linux 用户能够不使用的技巧），阐述了如何使用 bash 的编程特性完成各种功能。读者还将学习到流控制、信号处理、命令行处理和 I/O，及如何调试 bash 程序。

《WEB 设计技术手册》（第二版）

作者：*Jennifer Niederst*

本书涵盖了在设计 Web 页面的时候需要知道的所有东西。它的编排和组织让读者可以很快地找到问题的答案。这个经过全面修正和扩展的第二版充分展现了各种前沿技术信息，以及Web设计人员必须了解的技术信息。

本书是有关 HTML 4.01 标签的优秀参考（包括表格、框架、表单、颜色和级联样式表），同时还包括对浏览器的支持、平台特性和标准的相关信息。读者还可以查阅到很多关于使用图形、多媒体、音频和视频的最新信息，以及诸如动态 HTML、JavaScript和XML等的高级技术，本书还增添了关于XHTML、WML和SMIL的新章节。本书是各类 Web 设计人员不可缺少的工具。

《lex 与 yacc》（第二版）

作者：*John R. Levine, Tony Mason & Doug Brown*

《lex 与 yacc》是唯一一本专门介绍这两个重要的UNIX 编程工具的书。这本新版是完全的修订版，并以很多新的扩充示例代替了旧的示例。几个介绍性章节已经完全重写，还有一章专门介绍实现SQL语法，给出了有经验的程序员希望看到的各种细节。

本书对lex和yacc的重要主题提供了详尽的参考手册。对所有主要的lex和yacc的MS-DOS和UNIX版本，本书都进行了介绍，包括AT&T lex和yacc、Berkeley yacc、Berkeley/Gnu flex、Gnu bison、MKS lex 和 Yacc、Abraxas PCYACC，等等。

《ActionScript 权威指南》

作者：*Colin Moock*

本书是关于 ActionScript 的完整而深入说明，其目标读者既包括初来乍到的Flash开发者，又包括那些要将技术转移到 ActionScript 上的JavaScript程序员（两种语言都是以 ECMAScript 标准为基础的）。

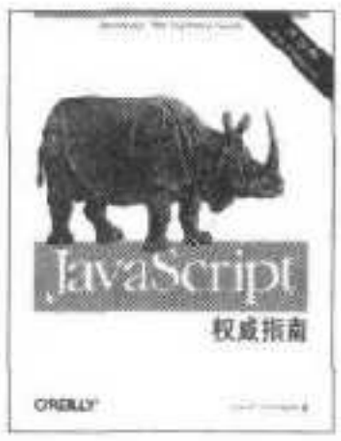
本书可以让新的ActionScript程序员获得迅速的进步，它用传统的形式来说明 ActionScript，给读者打下坚实的理论基础。富有经验的程序员可以在学习Flash的复杂部分时利用他们的JavaScript知识。在理论之上，本书还包含了很多实际技巧和现实的例子，包括了滚动文本域、菜单按钮、多项选择测试、XML 驱动的站点、物理视频游戏和实际的多用户环境等等。本书还谨慎地讨论了一些未证明或者正在证明的主题。

O'REILLY®
奥莱理软件（北京）有限公司
北京海淀区知春路 49 号希格玛公寓 B-809　邮政编码：100080　http://www.oreilly.com.cn　info@mail.oreilly.com.cn

O'Reilly 新书推荐

《JavaScript 权威指南》（第四版）

作者：*David Flanagan*



本书全面介绍了JavaScript语言的核心，以及Web浏览器中实现的遗留和标准的DOM。它运用了一些复杂的例子，说明如何处理验证表单数据、使用cookie、创建可移植的DHTML动画等常见任务。本书还包括详细的参考手册，涵盖了JavaScript的核心API、遗留的客户端API和W3C标准DOM API，记述了这些API中的每一个JavaScript对象、方法、性质、构造函数、常量和事件处理程序。

《JavaScript权威指南》是JavaScript程序设计者的完整指南和参考手册。对于使用最新的、遵守标准的Web浏览器（如Internet Explorer 6、Netscape 6和Mozilla）的开发者，它尤其有用。HTML作者可以从中学习如何用JavaScript创建动态网页。经验丰富的程序设计者可以从中快速地找到编写复杂JavaScript程序需要的信息。

《Unix 备份与恢复》

作者：W. Curtis Preston

本书完整地覆盖了Unix系统备份和恢复领域的方方面面，并且为各种规模、各种预算的环境提供了实用的、经济的备份和恢复解决方案。本书从介绍Unix系统管理员可用的本地备份工具开始，最后给出了商业备份工具选择的实用建议。对于管理员来说，本书可以说是无价之宝。

《学习 vi 编辑器》（第六版）

作者：*Linda Lamb & Arnold Robbins*

vi是一种配备在大部分UNIX系统上的全屏幕文本编辑器。对许多用户来说，在UNIX环境下工作就意味着使用vi。

这本畅销书的最新修订版是使用vi进行文本编辑的完全手册。新增加的内容涵盖了四种vi版本：nvi、elvis、vim和vile，以及它们对vi的扩展功能，如多窗口编辑、GUI接口、扩展的正则表达式和为程序员提供的增强特性等等。还增加了一个描述vi在UNIX世界和因特网文化中的重要地位的附录。

《UNIX 操作系统》（第五版）

作者：*Jerry Peek, Grace Todino & John Strang*

本书主要包含以下内容：

- 登录到系统和从系统中注销
- 使用窗口系统
- 管理UNIX文件和目录
- 发送和接收邮件
- 阅读新闻组和向新闻组发布消息
- 浏览Web
- 与朋友和同事交谈
- 理解管道和过滤器
- 后台处理
- 基本网络命令

本书适合UNIX初学者使用。

O'REILLY®
奥莱理软件（北京）有限公司
北京海淀区知春路49号希格玛公寓B-809 邮政编码：100080 http://www.oreilly.com.cn info@mail.oreilly.com.cn

作者简介

Debra Cameron 是 Cameron 咨询公司的总裁。10 多年来，Deb 写了很多与计算机工业有关的书。她写了大量关于局域网、电子商务和因特网安全方面的文章。她的作品曾经获得过技术通信协会（Society for Technical Communication）的表彰。除写作和咨询工作外，她还发表演说和讲授关于如何有效地使用因特网方面的课程。

Deb 毕业于佛罗里达大学。她与她的丈夫 Jim 及两个孩子 Megan 和 David 住在宾夕法尼亚州的 Bellefonte。她喜欢散步、眺望山景和给她的孩子讲故事。她最喜欢的作家包括查尔斯·狄更斯、维克多·雨果、乔治·麦克唐纳、C.S. 刘易斯和多萝西·塞亚。

Bill Rosenblatt 居住在费城。他现就职于纽约市 Sun 公司，任企业 IT 体系和数字媒介策略分析师。

他在普林斯顿大学获得 B.S.E 学位，又在位于 Amherst 市的麻萨诸塞大学获得了 M.S. 和 A.B.D 学位，这两个学位都是与计算机科学有关的。他在计算领域的兴趣包括数字图书馆、数字知识产权和因特网/局域网的软件开发工具。在计算以外的领域，他的兴趣集中在法式烹调、古典音乐、爵士乐和福尔摩斯推理探案小说。Bill 与他的妻子 Jessica 居住在曼哈顿的上西区，在他家周围有很多一流的餐馆和书店。

Eric Raymond 是一个自由软件高手，偶尔也写写书。他在各种 Emacs 版本上的程序设计经验差不多有 15 年了。他为 Emacs 19 设计了几个新的功能，包括 VC 模式和 GUD 模式、帮助系统的程序包查找器和程序包上传代码。他还负责几个 FAQ（常见问题答疑）和 Jargon File（行话文件）的维护工作。Eric 住在离费城以西 20 英里远的地方，并在当地帮助运行一个免费因特网的站点，这个站点的名字叫 Chester County Interlink（CCIL）。大家可以在主页 *http://www.ccil.org/~esr/home.html* 上找到更多关于他的自由软件项目和 CCIL 的资料。在他忙里偷闲的自由时间里，人们会看到他在读科幻小说、吹笛子、弹吉他、练空手道、开车兜风，或者与他女朋友的猫一起练心灵感应。

封面介绍

出现在本书封面上的动物是一只非洲角马。这种动物原产非洲，在Serengeti平原繁衍生息。雄性非洲角马高度不超过52英寸，重量不超过500磅，但在同类物种中却有着最具威力的角。成年雄性角马的领土性很强，往往独来独往；而雌性和幼年角马则选择群居生活，在迁徙途中它们往往会形成一个数以千计的大军。非洲角马是狮子最喜欢的一种食物。

目录

前言 1

第一章 Emacs 的基本概念 17

Emacs 简介 17
理解文件与编辑缓冲区 19
编辑模式 20
启动 Emacs 22
Emacs 的编辑画面 23
Emacs 命令 26
打开一个文件 29
保存文件 33
退出 Emacs 34
获取帮助 34
小结 37

第二章 文件编辑 40

光标的移动 42
文本的删除 48

文本块及其编辑操作 53
段落重排 59
编辑技巧和快捷键 61
命令的中止和修改的撤销 63
对 Emacs 进行定制 67

第三章 查找和替换操作 73
查找操作 73
查找和替换操作 80
拼写检查 89
单词简写模式 99

第四章 使用编辑缓冲区和窗口 107
文件、编辑缓冲区和窗口 107
同时使用多个编辑缓冲区进行工作 109
使用窗口进行编辑 115
在文档中使用书签 130
临时性地挂起 Emacs 135
使用多个 X 窗口进行编辑 137

第五章 Emacs 工作环境 143
在 shell 编辑缓冲区里执行 UNIX 命令 143
文件和目录操作 155
Emacs 中的打印操作 171
用 Emacs 查阅 UNIX 的在线文档 173
时间管理工具的使用 173
用好 Emacs 工作环境 183

第六章 电子邮件和 Usenet 新闻 184

Emacs 的电子邮件功能 184
用 Emacs 发送邮件 185
用 Emacs 读取邮件 195
用 Gnus 读取 Usenet 新闻 216

第七章 Emacs 的因特网工具箱 240

Emacs 的 Telnet 模式 241
Emacs 的 Ange-ftp 模式 245
用 W3 模式浏览 Web 主页 249

第八章 简单的文字排版和特效编辑 261

文本的缩进 262
文本的居中 272
插入分页符 274
矩形编辑 275
绘制简单的图形 281
Emacs 的大纲模式 291

第九章 用 Emacs 设置排版标记 300

设置 troff 和 nroff 排版标记 302
设置 TEX 和 LATEX 排版标记 309
编写 HTML 文档 316
Emacs 的 Html-helper 模式 317

第十章 Emacs 中的宏 334

什么是宏 334
定义宏 335
向现有的宏里增加编辑命令 342
命名并保存宏 343

执行一个已命名的宏 344
建立复杂的宏 345
LISP 函数 —— 宏的补充 350

第十一章 对 Emacs 进行定制 352
键盘的定制 353
终端支持 360
Emacs 变量 364
Emacs 的 LISP 程序包 366
自动模式的定制 366

第十二章 程序员的 Emacs 369
语言编辑模式 370
C 和 C++ 模式 377
LISP 模式 391
FORTRAN 模式 399
对程序进行编译 405

第十三章 用 LISP 语言对 Emacs 做进一步开发 408
LISP 语言简介 409
LISP 语言的基础函数 420
Emacs 的内部函数 425
主编辑模式程序设计实例 441
对现有编辑模式进行定制 451
建立自己的 LISP 开发库 458

第十四章 Emacs 编辑器和 X 窗口系统 461
Emacs 的 X 界面 461
让 Emacs 使用 X 字体和颜色 465
定制 Emacs 在 X 环境中的显示情况 467

通过 .Xdefaults 文件进行定制 467
属性、窗格、菜单和鼠标事件 469
与 X 服务器进行通信 479
良好的 X 程序设计风格 480

第十五章 Emacs 下的版本控制 482
版本控制的用途 482
版本控制的有关概念 483
VC 对基本操作的辅助作用 485
修改注释的编辑 486
VC 命令汇总 487
VC 模式的标志 488
使用哪一种版本控制系统 488
VC 命令细说 489
对 VC 进行定制 495
对 VC 进行扩展 497
VC 的不足之处 498
有效地使用 VC 499

第十六章 在线帮助 500
Emacs 的自动补足功能 500
帮助命令 504
针对复杂 Emacs 命令的帮助功能 512

附录一 如何获得 Emacs 软件 513

附录二 解除他人对 Emacs 的定制设置 521

附录三 Emacs 变量 523

附录四 Emacs LISP 程序包 534

附录五 软件漏洞及其修补 541

附录六 Emacs 的版权文件 543

附录七 请支持自由软件基金会 555

附录八 Emacs 编辑命令速查表 556

词汇表 575

前言

Emacs是迄今为止功能最为强大的文本编辑器。它与其他大多数编辑器（特别是UNIX操作系统的标准编辑器vi）的不同之处在于Emacs是一个完备的工作环境。不管你做什么，都可以在清晨启动Emacs，然后一整天都用它来工作：可以用它对文件进行编辑、重命名、删除和复制等操作；可以对程序进行编译；可以与UNIX操作系统的shell进行交互式操作；可以阅读和组织电子邮件；可以访问因特网等等。在X等窗口系统流行之前，人们通常把Emacs单独当做一个完备的窗口化系统来使用。只要有一台终端，就可以在Emacs环境里永不停息地工作。Emacs还具备无穷的灵活性：你可以编写自己的命令，能够更改与Emacs命令关联的按键；如果愿意花时间，可以用它做任何你想做的事情。

学习本书的目的

因为Emacs能够完成如此多的工作，所以人们普遍认为它是一种极其复杂的编辑软件。到目前为止，已经出版的Emacs书籍大都是一些综合性的参考手册，不太适合作为Emacs初学者的入门教材。这正是我们编写本书的原因：教大家如何从最基本的东西开始去熟悉和掌握Emacs。本书首先介绍Emacs的基础，然后过渡到一些更高级的功能。

我们努力使本书最大限度地满足不同读者的学习需要：也许你是一位只想用Emacs来写电子邮件消息和办公室留言的企业管理人员；也许是一位需要用Emacs来撰写

充满版式代码的复杂文档的职业作家；还可能是一位希望用Emacs来对源代码进行排版的高级程序员。不管你想用Emacs做些什么，你都会发现它其实是很容易学习和掌握的。只要有一两次实际使用经验，你就能够了解基本的文件编辑方法。

在掌握了Emacs的基本用法之后，大家可以继续学到更高级的技巧。只有掌握了它们，你才能真正体会到Emacs的妙处。这些高级技巧包括：

- 使用多个窗口和编辑缓冲区，以同时操作多个文件。
- 对键盘命令进行定制。
- 使用变量设计Emacs，使它适应用户个人的工作习惯。
- 使Emacs成为能够完成各种日常任务的工作环境，比如阅读电子邮件、编译程序和执行shell命令等。
- 创建宏编辑命令以快速完成重复性的操作任务。
- 用Emacs支持多种程序设计语言（其中包括C、C++、LISP和FORTRAN）的编程工作。
- 利用各种标记语言（markup language）对文件进行排版。
- 利用Emacs的单词缩略功能，避免拼写过长的短语或者纠正常见的拼写错误。
- 利用Emacs访问因特网资源。

当然，这么多的论题不见得都适用于你的具体情况。有些专题可以在掌握了Emacs的基本用法之后再深入学习，第一次阅读本书时可以先跳过去。在这篇前言的结尾，我们给出了一些学习本书的不同方法，请大家根据自己的兴趣和经验作出具体的选择。

阅读本书需要满足一些最基本的要求。我们假定大家对UNIX操作系统都有一个基本的了解（或者，如果你不了解UNIX，那么至少应该对正在使用的操作系统有一定的了解）。具体说来，你必须知道文件和目录指的是什么，怎样给它们起名字，对它们都能进行哪些基本操作（如复制、删除和重命名等操作；在UNIX操作系统里，这些操作要分别用**cp**、**rm**和**mv**命令来完成）。如果你对UNIX完全不了解，我们建议你先去学习一下Grace Todino、John Strang和Jerry Peek等人合著的《Learning the UNIX Operating System》（《UNIX操作系统》）一书，这本书也是由O'Reilly & Associates出版公司出版的。

Emacs 的版本

Emacs 有许多不同的版本，提供了丰富多彩的功能。单就 UNIX 操作系统来说，Emacs最重要的版本包括GNU Emacs（也就是我们将在本书里讨论的）、Unipress、JOVE、MicroEmacs、Freemacs、MG、Epsilon和CCA。某些版本，包括GNU Emacs在内，可以在MS-DOS下的个人电脑上运行；其他一些版本只提供了有限的功能集，但优点在于占用的内存和CPU周期要相对节省。“主流”的Emacs版本要求计算机必须有比较好的配置，多个用户共享一台中央计算机的时候更是如此。有些类Emacs的编辑器（比如Epoch和xemacs）与X窗口系统结合得非常紧密。有些版本是“free”（有自由、免费等几种含义）软件，另外一些则可能是相当昂贵的商业化产品。还有一些软件程序（比如FrameMaker）虽然不在Emacs的家族中，但却有与之相似的键盘命令集。

作为一个编辑器家族，不同的Emacs编辑器之间会有一定的差别，我想这一点大家能够有所了解。但这些编辑器之间还是有很多相似之处的，比如说：它们都可以自由地把文本和编辑命令混在一起使用；大多数编辑器都支持多窗口操作；都有一些供编辑特殊类型文件（比如C语言程序、TEX文件等等）用的支持功能；它们中的大多数都允许使用者进行大量的定制，而这些定制通常都是通过某些LISP程序设计实现的。

既然有这么多让人迷惑的差异，我们到底介绍哪一种版本好呢？这本书介绍的是GNU Emacs编辑器。自从GNU Emacs编辑器问世以来，它已经成为UNIX界最流行的Emacs实现版本，而这一状况目前还没有改变的迹象。Emacs家族本身已经是异常强大和灵活的了，而GNU Emacs又可以说是其中之最。只要你掌握了GNU Emacs，就可以毫不费力地掌握其他Emacs实现版本；反之，却不见得有这么简单。基于以上几点，我们惟一合乎逻辑的选择就是把这本书的重点放在GNU Emacs上。

不过，这本书的读者群却并非仅限于GNU Emacs用户。因为不同的Emacs实现版本之间有很多相似之处，所以本书完全可以帮你学习任何一种Emacs编辑器。一种Emacs编辑器之间所使用的键盘命令与另一种编辑器所使用的几乎没什么差别——你会看到“**C-n**”（即**CTRL-n**组合键）几乎总是代表“移动到下一行”。各种Emacs编辑器之间的差别主要体现在它们的高级命令和特色功能方面；不过，如果你准备利用某些高级功能而使用的却不是GNU Emacs，那最好还是换成GNU Emacs。

我们将在这本书里讨论的GNU Emacs编辑器是它的第19版。准确地说，是它的19.30版。本书的第一版是以GNU Emacs的第18版为重点的。在第19版里，GNU Emacs的操作界面有了很大变化，除增加了一些字体和颜色之外，在以下几方面也有了升级：菜单、卷屏条、增强的X窗口系统支持和大多数标准按键（比如**PAGE UP**、**PAGE DOWN**、**HOME**和**END**等按键）的功能绑定等。有些过去需要由用户自行安装的LISP包，现在已经包括在标准的发行版本中，其中两个值得注意的是Gnus工具和ange-ftp模式，前者是一个Usenet新闻阅读器，后者是FTP（file transfer protocol）的一个透明接口。如果没有特殊说明，我们在这本书里介绍的命令、菜单和功能都将以GNU Emacs 19.30版为准（注1）。

GNU Emacs和自由软件基金会

使用Emacs编辑器并不意味着你必须了解它的发展史，但它的起源确实是近代计算机史中不能不提的故事。负责维护和发行GNU Emacs的自由软件基金会（Free Software Foundation，简称FSF）已经成为计算机文化中一个非常重要的角色。

很久以前（1975年），Richard Stallman在麻省理工学院编写出了第一个Emacs编辑器。据说，最初的Emacs编辑器是TECO的一组宏编辑命令，这是一种很难学习的行编辑器，目前已经退出了历史舞台。而“Emacs”这个名字的意思就是“Editing Macros（编辑用宏命令）”。另外一种说法是“Emacs”起源于Richard Stallman特别喜欢的某个冰激凌店的名字。1975年之后发生了许多事情，TECO逐步退出了人们的视野，而Emacs则被重新编写为一个独立的软件程序。Emacs曾经有过几个商业化版本，其中最重要的是Unipress Emacs和CCA Emacs。曾有一段时间，在学术界以外你最有可能遇见的Emacs编辑器就是这些商业化的实现版本。

注1：大家必须了解，Emacs 19从第19.12版到第19.30版已经有了很大的改进，特别是第19.12版和第19.22版之间在功能方面有巨大的跃进。第19.30版为文本界面提供了菜单选项。此外，第19.30版对Gnus包做了升级，使它成为一个与早期版本中的Gnus包有很多不同之处的新版本。同时，菜单选项的安排也更加合理化了。如果你习惯于使用Emacs的菜单，不管是通过X界面还是通过文本界面，可能会发现以前熟悉的菜单选项变到另外一个不同的菜单里去了。这些小变化通常不会造成什么麻烦，但如果你按照我们的指导去进行操作却发现结果稍有差别，现在就该知道是什么原因了。欢迎大家把自己的疑问（或者意见、建议）寄到电子邮件地址 *deb@ora.com*。

Stallman的Emacs在自由软件基金会（FSF）和GNU项目出现之后逐步占据了主导地位。GNU是“GNU's Not UNIX（GNU不是UNIX）”的首字母缩写，它指的是一个由Stallman及其伙伴们共同努力构建的完备的类UNIX操作系统。Stallman创建自由软件基金会的目的就是为了保证某些软件能够永保自由（free）。这里所说的“自由”并不意味着廉价。你可能必须支付一笔软件发行成本方面的费用，它的确切含义是指把人们从软件使用许可方面的限制条款中解放出来。

为了理解“自由”这个词的含义，我们有必要先来看看软件通常是如何发行的。大多数商业软件在发行时都带有一个限制性很强的许可证。你必须先支付费用才能使用某个程序；你可能必须为运行此程序的每台计算机分别支付一定的费用；而为了能够继续使用此程序，你可能还得每年再交一些费用，在某些情况下，你甚至可能不得不按分钟计算来支付使用费。这些许可证几乎百分之百地不允许你把有关程序赠送给朋友，而且你几乎永远也见不到程序的源代码（除非你非常有钱）。如果某个商业化的程序出了问题或者它不具备你需要的功能，就只能等卖给你程序的那家公司的怜悯了，而那些公司极有可能根本不理睬你。

而GNU Emacs则没有这样的限制条款。只要你能找到一个愿意把它送给你的人（通常总能找到这样的人），就可以完全免费地得到它。它在许多开放性的软件仓库站点（包括一些匿名FTP站点）上都可以找到（我们将在附录一中讨论这个问题）。由于GNU Emacs在发行时永远带着源代码，因此，如果你是一位程序员，就能给它添加自己的功能，或者自行修补你发现的程序漏洞（注2）。你可以把副本赠送给你的朋友，永远不会有人要求你支付使用费用。你惟一不能做的事情就是不得自行增加GNU Emacs在使用许可方面的限制条款。也就是说，如果你向他人赠送了副本或者对它进行了改进，你不能因此而开始收取所谓的许可费。GNU Emacs是自由的，而这种自由将永远保持下去。作为一名使用者，你的权利和责任都已经在通用公共许可证（General Public License，GPL）里描述得非常全面和清楚了，该许可证的具体内容请参考附录六。

创建自由软件基金会来发行软件程序的目的，是为了鼓励大家去共享而不是占有软件。制定通用公共许可证的目的，是为了防止出现一种自私而又常见的行为，即某

注2：幸运的是，GNU Emacs中的程序漏洞是非常少的。事实上，与大多数谋求暴利的公司相比，FSF开发的代码要好得多，而且他们还非常重视程序漏洞方面的报告。不过，如果你真的发现了一个程序漏洞却又等不及让他们去修补，你完全可以自己动手。

个公司以公开软件代码为基础，在做了若干改进和纠错之后就宣称拥有改进版本的版权而谋取利益。一旦有公司这样做了，从本质上讲，那个程序就将成为一种私有财产而不再属于公用范畴。正是出于对这种行为的厌恶，Stallman才创建了自由软件基金会这一组织。正如他在GNU宣言里所说的："我不会在理智的情况下签署一项保密合约，或者一份软件许可证合约……因此，为了让自己能够不在不名誉的情况下继续使用计算机，我必须做出'把足够多的自由软件集合在一起'的决定。这样，我就能够不依赖任何不自由的软件而继续生活和工作下去。"在这份宣言中，Stallman还把软件的共享称为"程序员之间最基本的友谊行为"。程序员编写出来的软件是自由的，因为它们可以共享并将永远是共享的，而且这种共享不应该有任何附加的限制（注3）。FSF软件不受限制性版权法律的约束，那是Stallman从根本上反对的做法。事实上，他专门造了一个词"*copyleft*"来描述FSF的可共享软件在版权方面的基本观点。

自从GNU Emacs问世之后，GNU操作环境许多其他的组成部分也逐步到位，其中包括：C和C++编译器（*gcc*和*g++*），功能强大的调试器（*gdb*），词法分析器**lex**和语法分析器**yacc**的替代品（分别叫做*flex*和*bison*），一个UNIX操作系统的shell（*bash*，"Bourne-Again Shell"的缩写），以及许多其他程序和库。许多已经存在的软件工具，比如源代码控制系统RCS等，也纳入了FSF的copyleft版权规定中。FSF基金会还发行一个Linux版本（即Debian Linux）。有了Linux和GNU软件工具，拥有一个能够完全体现FSF价值观的完备的操作环境就成为可能。

学习 Emacs 的方法

就像Emacs有许多版本一样，它的用户群也各不相同。这本书的写作目的是帮助大家（不论是有经验的计算机用户还是新手）尽快进入Emacs的大门。开篇的头两章内容向大家介绍应该了解的基本概念，其余各章都是建立在这些基本概念之上的。在学习完前两章之后，大家不必一定按顺序学习其余各章的内容，你可以直接跳到自己感兴趣的主题上去。此外，这本书尽量做到深入浅出，你既可以仔细研读每一个细节，也可以快速查阅有关的命令清单和应用示例。

注3：Emacs等FSF程序经常会随商业化系统一起发行。但即使在这样的情况下，通用公共许可证也能保证你自由传播FSF程序的权利不受任何制约。当然，这个许可证对与GNU软件工具一起发行的其他专有软件是没有效力的。

你可以参考下面列出的阅读顺序进行学习:

如果	请阅读
你是一名系统管理员用户	前言、第一~三章、第十六章
你是一名非专业用户	前言、第一~三章、第十六章
你是一位程序员	前言、第一~五章、第十~十二章
你是一位作家或专业人士	前言、第一~四章、第八~九章、第十六章
你想对 Emacs 进行定制	第十一章，也许还要再学习第十三章
你想在 Emacs 里使用电子邮件	第六章
你想在 Emacs 里使用 UNIX 命令	第五章
你想从 Emacs 里访问因特网	第六~七章和第九章

上表给出的阅读顺序仅供大家参考。Emacs 是一个庞大而又功能丰富的编辑器。我们已经对它进行了分解以方便大家的学习和掌握，所以，你完全不必因它庞大和丰富的功能而丧失信心。学习Emacs的最佳办法是蚂蚁啃骨头：先学会最基本的编辑功能，其他功能可以等到你对它们有了兴趣、或者等到你想做什么事却又不知该如何在Emacs里做的时候再去进一步学习。Emacs很可能已经具备你需要的功能，即使没有，你也可以通过编写 LISP 函数把它添加到 Emacs 里去（具体细节请参考第十三章）。GNU Emacs的在线帮助系统是一个快速学习掌握新功能的好地方。我们在第一章里对在线帮助的具体使用方法做了介绍，在第十六章里又对它做了更深入的讨论。

下面列出了一些大家可能打算在闲暇时学习的功能:

- 单词简写模式（第三章）
- 如何使用宏编辑命令（第十章）
- 如何把键盘上的功能键映射为 Emacs 命令（第十一章）
- 如何发出（和编辑）shell 命令（第五章）
- 如何使用多个窗口（第四章）
- 如何在图形模式（picture mode）里绘制简单的图形（第八章）
- 如何访问因特网（第七章）

PDG

- 如何发送电子邮件和阅读 Usenet 新闻（第六章）

最后，如果你坚持要从头到尾通读本书，那么请参阅下面对各章内容的简单介绍:

第一章，Emacs 的基本概念：介绍怎样启动 Emacs 和怎样对文件进行操作的方法。这一章还对在线帮助系统做了一个简单的介绍。

第二章，文件编辑：介绍Emacs的编辑操作，包括光标移动命令、文本的复制和粘贴命令、撤销修改命令等。这一章还介绍了几种比较初级的定制方法，这将使Emacs按照你设定的方式去完成工作。

第三章，查找和替换操作：介绍了更多的编辑功能，包括查找和替换、单词的简写模式、拼写检查等功能。

第四章，使用编辑缓冲区和窗口：介绍多个编辑缓冲区、Emacs 窗口和 X 窗口系统的使用方法。这一章还介绍了在文件里插入书签以便日后检索定位的有关操作。

第五章，Emacs 工作环境：介绍能够在 Emacs 中的 shell 提示符下进行的各种操作。比如，发出shell命令，对文件和目录进行操作，或者使用一些基本的时间管理工具等。

第六章，电子邮件和 Usenet 新闻：介绍用 Emacs 发送、阅读和管理电子邮件的方法。Gnus 新闻阅读器使你能够在 Emacs 环境中完成对 Usenet 新闻组的访问。

第七章，Emacs的因特网工具箱：介绍利用Emacs编辑器远程访问其他计算机、用其 FTP 功能检索文件及浏览万维网的方法。

第八章，简单的文本排版和特效编辑：介绍Emacs中基本的文本排版操作（例如段落缩进和居中等）和某些使用较少的专业化编辑功能（如图形模式和大纲模式等）。

第九章，用Emacs设置排版标记：介绍Emacs对**troff**（及其相关软件）、TEX、LATEX和 HTML 等标记语言的支持功能。

第十章，Emacs 中的宏：介绍利用宏编辑命令简化重复性工作的方法。

第十一章，对Emacs进行定制：介绍根据个人喜好对Emacs进行定制的方法：定制屏幕画面、定制键盘命令和编辑环境、加载 Emacs 扩展包以实现特殊功能等。

第十二章，程序员的Emacs：介绍Emacs在程序设计环境方面的有关功能，包括对C、LISP、FORTRAN和其他一些程序设计语言的编辑支持，还介绍了对编译器和UNIX操作系统的**make**工具的接口。

第十三章，Emacs LISP程序设计：介绍Emacs LISP的基本概念，这是一种能够对Emacs做进一步定制的程序设计语言。

第十四章，Emacs编辑器和X窗口系统：介绍Emacs与X窗口系统的接口。如果你使用的是一个图形工作站，这个接口将使你能够通过鼠标和弹出菜单来进行操作。

第十五章，Emacs下的版本控制：介绍对文件版本进行控制的VC模式。如果你维护的某些程序或文档需要附带一个修订方面的历史记录，Emacs的版本控制功能将大大简化这类操作。

第十六章，在线帮助：介绍Emacs丰富而又易于使用的在线帮助功能。

附录一，如何获得Emacs软件：介绍获得GNU Emacs和其他几种Emacs版本的方法。

附录二，解除他人对Emacs的定制设置：告诉大家如何解除他人对自己的Emacs进行的定制设置，使它能够按这本书里所描述的那样运行和工作。

附录三，Emacs变量：列出了许多重要的Emacs变量，包括本书涉及到的全部变量。

附录四，Emacs LISP扩展包：列出了Emacs自带的几个最有用的LISP开发包。

附录五，软件漏洞及其修补：介绍怎样（以及何时）提交在Emacs中发现的程序漏洞。

附录六，Emacs的版权文件：给出通用公共许可证的完整内容，GNU Emacs就是在这些规则下发行的。

附录七，请支持自由软件基金会：为了更多地推出高质量的软件，自由软件基金会在不懈地奋斗着，而你也可以为此尽一份力量。请支持他们的工作。

附录八，Emacs编辑命令速查表：对本书介绍的各种Emacs重要命令的汇总。

词汇表：对 Emacs 术语的解释。

GNU Emacs 速查卡：这张可以撕下来的卡片对附录八中列出的 Emacs 命令做了进一步的筛选，列在这张卡片里的都是最基本的 Emacs 编辑命令。

本书没有讨论的 Emacs 功能

GNU Emacs 是一个功能丰富而且强大的编辑器。限于篇幅，我们在这本书里对它的介绍和讨论并不完整。作为一本针对初学者的教材，Emacs的许多功能都没有包括进来，而新的功能还会不断地添加进来。下面是一些没有包括在这本书里的功能：

- 各种兼容模式（compatibility modes）：GNU Emacs 为 UNIX 操作系统下的 vi 编辑器和 VAX/VMS 操作系统下的 EDT 编辑器分别提供相应的兼容模式。本书没有讨论这些模式。就我们的经验而言，这些兼容模式只是对真实情况的拙劣仿真。如果读者确实想使用 vi 或 EDT，尽管去尝试好了。我们认为既然大家学习的是 Emacs，那最好还是用 Emacs，而不是把它假装成其他的东西。
- 多种程序设计语言模式：我们在这本书里讨论了 C++、LISP 和 FORTRAN 语言的编辑模式。其实Emacs里还有许多针对其他程序设计语言的模式；有些工作做得还相当不错，而且新的模式还会定期地添加到其中去。可惜我们没有办法做到面面俱到。
- 高级 LISP 程序设计：GNU Emacs 内含一个完备的 LISP 解释器。这本书只简要地介绍了Emacs LISP中那些最基本的东西；第十三章虽然足够让你入门，但也只能算是涉及到皮毛。自由软件基金会出版了一本完整的《Emacs LISP 参考手册》，O'Reilly & Associates 出版公司很快也将出版一本这方面的书，其作者是本书的一位技术顾问，Bob Glickstein。
- 移植、调试和安装：这本书没有涉及把 GNU Emacs 移植到其他系统上时可能会遇到的问题。我们相信人们需要一本对此问题进行讨论的书籍，可惜的是它超出了本书的讨论范围。如果你认为这样一本书会对自己有所帮助，请告诉我们，我们将考虑编写这方面的书。
- 游戏和娱乐：GNU Emacs 附带很多游戏和娱乐项目。比如，它能够随机抽取《Zippy the Pinhead》（《死脑筋 Zippy》）一书中的精彩语句，作为著名搞笑心理医生“Eliza”的诊断语。（你可以在闲暇时输入“**ESC x psychoanalyze-**

pinhead RETURN”命令，看看会发生什么有趣的事情。）不过，这些“非核心”的东西只能放到其他书里介绍了。

本书的编写体例

下面介绍这本书所使用的编写体例。

Emacs 命令

Emacs 命令是由一个修饰符，比如CTRL（CONTROL）或ESC（ESCAPE），后面再跟一或两个字符组成的。这本书里的命令把CTRL 简写为大写字母“C”，比如:

C-g 按住 **CTRL** 键并按下 **g** 键。

为了输入一条命令，你可能还需要按下回车键，如下所示:

RETURN 按下回车键（这个键在你的键盘上可能为 **ENTER**）。

除 **CTRL** 键外，大多数 Emacs 手册还会提到一个 **META** 键。因为大部分键盘上都没有 **META** 键，所以本书将用 **ESC** 键来代替 **META** 键，如下所示:

ESC x 按下 **ESC** 键，放开它（注 4），再按下 **x** 键。

有的读者的键盘上可能真的有 **META** 键。比如，在 Sun 工作站上，紧靠空格键左右两端的就是 **META** 键。在许多其他型号的键盘上，**ALT** 键的功能也相当于 **META** 键（注 5）。如果你的键盘上有 **META** 键，那它的用法就像刚才介绍的 **CTRL** 键一样，即按住 **META** 键再按下所需要的按键，比如 **g**。因为可以在按住 **META** 键的同时多次按下其他的按键，所以 **META** 键有一定的优越性；而 **ESC** 键在需要多次

注 4: 我们在此强调放开 **ESC** 键的原因是，在 Emacs 19 的某些版本里，两次按下它（**ESC ESC**）或者因按住它超过大约一秒钟，而使它产生自动重复会导致一条错误信息。如果你遇到了这个问题，请参考第二章“重新绑定键盘按键”小节里给出的解决方案。

注 5: 如果你是以远程方式来使用 Emacs 的，按下 **ALT** 键可能会切断连接。虽然这种情况很少见，但我们认为还是应该让大家知道有这种可能性。

按下其他按键的时候就不那么方便了。一般来说，如果你的键盘上有一个**META**键，那你肯定会更喜欢用它而不是 **ESC** 键。

有几个鼠标命令（仅用在X窗口系统里，参见第十四章中的讨论）需要**SHIFT**键作为修饰符，通常还要与 **CTRL** 键联合使用。我们把这种情况简写为:

S-right 按住 **SHIFT** 键的同时按下鼠标右键。

C-S-right 按住 **SHIFT** 键和 **CTRL** 键的同时按下鼠标右键。

全部 Emacs 命令都有一个完整的名字，即使是最简单的也不例外。比如，使用 **forward-word** 命令相当于按下“**ESC f**”组合键；而使用 **forward-char** 命令相当于按下“**C-f**”组合键。有些命令只有全名而没有对应的按键。

在介绍一条命令的时候，我们将给出它的全名和组合键（如果有的话），按下那个组合键就等于发出了对应的命令。我们假设你使用的是默认的按键绑定（即按键动作与 Emacs 命令的默认映射关系）。

表格

如果你想快速查找一组命令，请在有关章节里查看相应的命令汇总表。这些表格的格式如下所示:

键盘操作	命令名称	动作
C-n	**next-line**	移动到下一行
C-x C-f *Files→Open File*	**find-file**	打开一个指定的文件
（无）	**yow**	把 minibuffer 缓冲区里保存的游戏人物 Pinhead 的笑话打印出来

表中第一列是命令的默认按键绑定，第二列是命令的全名，第三列是命令的功能。比如，按下“**C-n**”（即执行 **next-line** 命令）将把光标移动到文件的下一行。有些命令，比如“**C-x C-f**”，还可以通过菜单来执行。如果有一个与命令对应的菜单选项，我们就用斜体字把它列在该命令的组合键的下面。比如，想执行 **find-file** 命令，则既可以在键盘上按下“**C-x C-f**”，也可以从“**Files**”菜单里选择“**Open File**”选

项。有时候，键盘操作栏里给出的是“(无)”，这并不代表你无法使用这个命令，它的意思是这个命令没有对应的键盘操作。要想使用没有对应的键盘操作命令，必须先按下“**ESC x**”，然后输入命令的全名，再按下**RETURN**键(有时你可以输入“**ESC x yow RETURN**”试试)。

示例

在本书中，经常会看到下面这样的情况：先是需要进行键盘操作，然后是一个给出操作结果的Emacs屏幕画面。如下所示：

输入：**emacs** *myfile*

```
Buffers Files Tools Edit Search Help

-----Emacs:myfile          (Fundamental)--L1--All----------------------
(New file)
```

在UNIX提示符处输入**emacs**和一个文件名，开始一次Emacs编辑任务。

用粗体字给出的“**emacs**”表示必须精确输入的内容；而用斜体字给出的“*myfile*”表示可以把它替换为任意一个文件名，用不着精确地按照书里给出的样式。Emacs屏幕画面中的光标(即你当前正在进行编辑的字符位置)是以反显字体给出的，就像画面底部的Emacs模式条(mode line)那样。

在本书的后半部分，当我们讲到各种程序设计模式、定制和LISP程序设计的时候，反复给出雷同的Emacs屏幕画面没有什么实际的意义。所以到那时，我们将不再显示上面这样的画面，我们只给出一两行文本，如果有必要，我们会以反显字体给出光标的位置，就像下面这样：

```
/* This is a C comment */
```

字体体例

本书使用了以下几种字体体例：

- UNIX命令、Emacs键盘操作、命令的名字、菜单选项和变量都用粗体字给出。如“**emacs**”。
- 文件名用斜体字给出，如“*myfile*”。
- 编辑缓冲区的名字、LISP代码、C语言代码、Emacs消息以及各种程序节选段落都用等宽字体给出，如“`constant width`”。
- 将被实际值替换掉的示范性参数用斜体字给出（如果它们出现在一段程序里，将用等宽斜体字给出，如“*`constant width italic`*”）。

建议与评论

本书的内容都经过测试，尽管我们尽了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们:

美国:

O'Reilly & Associates,Inc.
101 Morris Street
Sebastopol, CA 95472

中国:

100080北京市海淀区知春路49号希格玛公寓B座809室
奥莱理软件（北京）有限公司

如果你在学习Emacs的过程中遇到了问题，或者有针对本书的建议，请按电子邮件地址 *deb@ora.com* 发送给我们。如果你能够访问Usenet新闻，那还可以到 *gnu.emacs.help* 新闻组去寻求帮助。

你也可以发电子邮件。要想参加O'Reilly的邮件列表或者要一些目录，请给 *info@oreilly.com* 发邮件。要询问技术问题或者对本书做评论，请发邮件给 *bookquestions@oreilly.com*。

我们有一个关于本书的网站，上面有信息、勘误表和关于将来版本的一些计划。你可以访问:

http://www.oreilly.com/catalog/gnu2

最后，你可以在 WWW 上找到我们:

http://www.oreilly.com
http://www.oreilly.com.cn

致谢

Debra Cameron: 本书的第二版是在许多人的帮助下完成的。有许多热心的读者给我们发来了良好的建议。Emacs 本身也发生了变化，从第 18 版升级为第 19 版。但到目前为止，还没有一本书能够把Emacs讲得面面俱到。Emacs远远超出了一个编辑器的概念，它本身就是一个不断被开发的世界。它就像是你刚搬进去的一套新居，你先按照自己的想法把它变成自己的家；再不断地添添改改，让它更能折射出你工作和生活的方式。

我要感谢Bill Rosenblatt，他曾经帮我安装了Emacs的两个版本并让它们运行起来，一步一步地带我走过整个安装过程。在我安装Emacs 19的第三个版本的时候，Lar Kaufman和Matt Welsh又帮我做了大量的录入工作。Linux和自由软件基金会在软件的安装方面做了大量的工作，使我们只需花费几个小时去FTP和编译Emacs就能顺利地把它安装起来。我还要感谢Lenny Muellner，他耐心地帮我完成了本书的校对工作。

我还要特别感谢我丈夫 Jim 对我的支持和鼓励；还有我的两个孩子 —— Meg 和 David —— 容忍我把自己的时间和精力用在这本“Emacs 书”的写作上。

Bill Rosenblatt: 我要向以下诸位表示感谢：普林斯顿大学古典艺术系的Richard Martin教授，是他在我心中播下的种子使我能够把枯燥的写作当做一种艺术的享受；Intermetrics公司，这家公司只给我分配了很少的工作，使我能够有空在工作时间里钻研GNU Emacs；Hal Stern，是他带我进入Emacs的大门；Sandy Wise，他帮我完成了X窗口系统的有关章节；Jessica Lustig，感谢她的爱和支持；最后还要感谢研究生学院里那些睡在我同宿舍的兄弟，我整日整夜地占用着电话线，而他们对此却毫无怨言。

Eric Raymond: 首先，我要感谢“无法无天”的黑客社团，他们缔造了丰富的Emacs LISP程序设计传统，而正是这些传统把Emacs的定制工作从一种纯粹理论意义上的可能性，转变为现实生活中的软件工具。我有很多知识都是通过阅读别人的程序代码而掌握的，这些“前辈”包括：Olin Shivers、Jamie Zawinski、Kyle Jones、Barry Warsaw、Roland McGrath、Richard Stallman本人（当然得有他）和许多其他程序员。其次，我要向Catherine C. Olanich献上我的感谢和热爱，当我埋头写作由我负责的有关章节时，她在许多方面给了我支持。最后，我要感谢O'Reilly出版公司里那些热心、专业而又负责的人们。他们不仅知道如何把一本好书带给读者，还知道如何善待每位作者。摆在大家面前的这本书就是明证。

我们这三位作者必须向本书的技术审校人员表示感谢，他们在紧迫的时间压力下体现出了工作效率和聪明才智。这本书因Ellen Siever娴熟的审校而更紧凑和实用，我们在此向他表示感谢。Bob Glickstein以一位LISP专家的眼光对本书进行了认真的审核，并督促我们在遣词造句方面精益求精。

我们还要向我们的编辑Mike Loukides表示感谢，他使本书第二版的写作和出版工作能够按预定计划顺利完成。他的指导意见和采编功力对我们帮助良多。我们还要感谢他介绍Eric Raymond成为本书的第三作者，Eric Raymond对本书的部分章节做了认真的修订，并使本书新增加了一章关于版本控制问题的重要内容。Eric的才华进一步充实了这次改版。我们还要特别感谢Gigi Estabrook，她在紧迫的时间压力下接手了本书出版的工程，并协调远在各地的作者、编辑和技术审校人员圆满地完成了任务。

最后，我们要向O'Reilly & Associates出版公司负责装帧和印刷的员工们表示感谢。Mary Anne Weeks Mayo是本书的生产经理。Robine Andrau负责本书稿的编辑工作。Clairemarie Fisher O'Leary、Kismet McDonough-Chan和Sheryl Avruch对本书的印刷质量进行了把关。Seth Maislin编写了本书的索引并在Cynthia Grabke的帮助下完成了有关的出版工作。这本书在编辑方面融进了Erik Ray、Ellen Siever和Lenny Muellner等人多年的经验，在印刷方面又得到了Michael Deutsch和John Files等人的大力协助。本书的插图由Chris Reilley负责，内部设计由Nancy Priest和Mary Jane Walsh负责，Hanna Dyer设计了本书的封底，Edie Freedman设计了本书的封面。

本章内容：

- Emacs 简介
- 理解文件与编辑缓冲区
- 编辑模式
- 启动 Emacs
- Emacs 的编辑画面
- Emacs 命令
- 打开一个文件
- 保存文件
- 退出 Emacs
- 获取帮助
- 小结

第一章

Emacs 的基本概念

有些读者可能已经迫不及待地把手放在键盘上开始打字了。我们不打算制止大家这样做，翻到“启动 Emacs”那一小节开始行动吧。不过，等你稍后准备停下来的时候，最好还是从头开始学习一下本章的内容。如果你对 Emacs 所涉及到的各种基本概念有了比较清晰的理解，学习起来就更容易了。我们将在以下各个小节里对 Emacs 的基本概念进行讨论。

Emacs 简介

GNU Emacs 是目前 UNIX 世界里最为常用的文本编辑器。与 **vi**（UNIX 操作系统的标准编辑器）或者其他内置在各种现代窗口系统里的编辑器相比，许多用户更喜欢使用 GNU Emacs。那么，它为什么这么流行呢？ Emacs 并不是市面上最新的编辑工具，它也肯定不是最好的。但它却将是你可以找到的最有用的编辑工具。我们希望这一章能够让大家对 Emacs 的基本概念有清楚的认识，从而帮助大家有效地利用 Emacs 来完成自己的工作。本书是针对 Emacs 用户而编写的一本指南。在编写它的时候，我们就已经把“满足尽可能多的读者群的学习需要”当做我们的写作目的之一，从需要撰写各种备忘录和报告的系统管理员及企业管理人员，到需要使用多种程序设计语言编写源代码的程序员都在我们考虑的读者范围内。

但也正是因为如此，我们势必无法把Emacs的一切功能都面面俱到地介绍给大家。Emacs有许多功能和命令没有在这本书里出现。可我们并不认为这是一个问题，因为Emacs本身带有一个详尽的在线帮助功能，完全能够帮助大家做进一步的学习。而我们则把注意力集中在介绍Emacs是如何完成各种实用性工作方面。本书的前三章主要是介绍Emacs的基本编辑功能，在学习完这三章之后，我们将向大家介绍怎样才能把Emacs当做一个完备的工作环境来使用，怎样才能发送和接收电子邮件，怎样才能在不离开这个编辑器的前提下执行UNIX命令，怎样使用各种特殊的文本编辑模式，怎样利用Emacs来编辑各种特殊类型的文件（比如**troff**、TEX和各种程序设计语言的源代码文件等）等高级论题。我们将把最重要的命令和最重要的文本编辑模式介绍给大家。但大家千万不要忘记这条原则：Emacs的确可以把许多事情都做得很好，不过人们并不是因为这一点才说它重要的。Emacs的重要性体现在“它能把你想做的许多事情都集成到一起来”这一点上。

那么，所谓“集成”的意思又是什么呢？下面这个简单的例子就很能说明问题。假设有人给你发来一封电子邮件，向你介绍了一条访问新打印机的特殊命令。那么，你可以先用Emacs来阅读这封电子邮件，然后试试这条新命令：从Emacs里启动一个UNIX shell把新命令复制过去，再直接执行它。如果新命令很好用，你可以编辑自己的*.cshrc*文件，给这条新命令创建一个假名（或者缩写）。在做这些事情的时候，你根本用不着离开Emacs编辑器，也根本用不着重复输入那条命令。这就是人们说Emacs功能强大的原因。它远不仅是一个文本编辑器，它是一个能够改变你工作方式的完备的操作环境。

再提前给大家一个忠告：许多人认为Emacs是一种非常难以掌握的编辑器，但我们不明白这有什么道理。Emacs的功能确实是非常之多，但这并不意味着你都必须学会和使用这么多的功能。就任何一种文本编辑器而言，不管它是多么简单或者多么复杂，其基本功能都应该是相同的。只要你能学会一种，就完全能够学会它们当中的任何一种。虽说我们为帮助大家记住Emacs的各种命令而给出了一些教条化的说明（比如“**C-p**代表把光标上移一行”），可我们并不认为这是必不可少的。当然，这些助记说明确实能帮助大家越过初学阶段的拦路虎，但从长远看并不会造成什么区别。学习使用编辑器其实就是学习手指的习惯性动作：学会在想把光标移动到上一行去的时候应该把手指放到什么地方。只要大家肯在Emacs里做练习，用不了多长时间你就能很快适应手指的这些习惯性动作了。而一旦你掌握了这些习惯性动作，就永远也不会忘记它们，就像你永远不会忘记如何骑自行车一样。在使用Emacs一

两天之后，我们就再也用不着去想什么“**C-p**代表把光标上移一行”之类的东西了，手指自己就知道该放到什么位置上去。到了这一阶段，你就算入门了。也正是从这一阶段起，你将能更有创造性地使用Emacs来进行工作。接下来就该考虑怎样才能让Emacs的高级功能为自己服务的事情了。由于Emacs里有很多扩展菜单，所以新的鼠标点击式操作界面将使Emacs的使用更简便。不过，即使你有鼠标，我们也建议你学习一些最常用命令的键盘操作。良好的手指习惯动作无疑会使你成为一个打字快手；而把手指从键盘移到鼠标上去肯定会降低你的打字速度，特别是在写作的时候。

学习手指习惯动作这种办法还暗示着本书一种不同的阅读方式。毫无疑问，读一遍书肯定会学到很多东西，但你每天能够形成的新习惯却不可能太多，除非它们都是些坏习惯。第二章将把大多数常用的基本编辑技巧介绍给大家。大家可能需要多读几遍，每次学习都略有侧重。比如，Emacs有许多办法能够让你把光标向前移动：你可以一次移动一个字符、一个单词、一行文本、一句话、一个段落和一张打印页等等。这些技巧都将在第二章里进行介绍。先从前、后移动开始学起，再逐步增加更复杂的命令。类似地，Emacs还提供了很多在文件里进行文本搜索的办法，这比我们在其他编辑器里见过的要多很多。这些搜索操作都将在第三章里讨论。你用不着一次把它们都学会，先学会几个，多做些练习，再去学习后面的章节。即使你花了好几遍的功夫才能熟练掌握本书前三章所介绍的内容，也没有人会笑话你。在培养好习惯方面多花点时间是值得的。

理解文件与编辑缓冲区

编辑器并不是对某个文件本身进行编辑。事实上，它们会先把文件的内容放到一个临时性的缓冲区里，然后再对缓冲区里的东西进行编辑。在通知编辑器保存缓冲区的内容之前，存放在磁盘上的原始文件是不会发生任何改变的。记住：虽然缓冲区看起来与文件非常相像，但它只是一个临时性的工作区域，里面可能包含的是文件的一份副本。

Emacs的编辑缓冲区和文件一样也有名字。缓冲区的名字通常就是正在编辑的文件的名字，但也有例外的情况。有些缓冲区没有与它们关联的文件，比如说，`*scratch*`就只是一个临时性的辅助性缓冲区，它的作用有点像草稿簿；而帮助功能会把帮助信息显示在一个名为`*Help*`的缓冲区里，它也是一个与任何文件都没

有关联的缓冲区。不过我们此时还用不着为此操心。就眼前来说，只要记住Emacs会在开始编辑一个文件的时候，把该文件复制到一个缓冲区里去就行了。编辑文件的时候，修改的是缓冲区而不是文件本身；可以等到把文本编辑得比较满意时再去保存它们，而文件本身只有在你明确地选择了存盘操作时才会发生变化。如果对自己的文本编辑工作不满意，可以在退出Emacs时选择不保存文件，这样就不会影响到原始的文件了。

编辑模式

Emacs有各种各样功能略有差异的编辑模式，而它灵活多能的声誉也部分来源于此。“模式”一词听起来技术味很浓，好像还挺复杂，其实它真正的含义不过是Emacs对当前的文本编辑工作更“敏感”而已。当你在输入长篇大论的时候，通常需要字换行（word wrap）等功能，这样你就不必在每一行的末尾按回车键了。而当你进行程序设计的时候，就必须遵守程序设计语言在语句格式方面的规定。对写作来说，Emacs有文本模式（text mode）；对程序设计来说，Emacs与各种程序设计语言对应的编辑模式，比如C语言模式（C mode）。也就是说，编辑模式将使Emacs成为能满足你不同工作任务要求的“专用”编辑器。

文本模式和C语言模式都是主模式（major mode）。一个编辑缓冲区每次只能处于一种主模式中，退出一种主模式的办法是进入另一个主模式。表1-1列出了部分Emacs的主模式、它们的作用，以及所在的有关章节。

表1-1：Emacs编辑器的主模式

模式	功能
基本模式（fundamental mode）	默认模式，无特殊行为
文本模式（text mode）	书写文字材料（第二章）
邮件模式（mail mode）	书写电子邮件消息（第六章）
RMAIL模式（RMAIL mode）	阅读和组织电子邮件（第六章）
只读模式（view mode）	查看文件，但不进行编辑（第五章）
shell模式（shell mode）	在Emacs里运行一个UNIX shell（第五章）
FTP模式（ange-ftp mode）	下载或者查看远程系统上的文件（第七章）
Telnet模式（telnet mode）	登录到远程系统（第七章）

表 1-1: Emacs 编辑器的主模式（续）

模式	功能
大纲模式（outline mode）	书写大纲（第八章）
缩进文本模式（indented text mode）	自动缩进文本（第八章）
图形模式（picture mode）	绘制简单的线条图形（第八章）
nroff 模式（nroff mode）	按 **nroff** 的要求对文件进行排版（第九章）
TEX 模式（TEX mode）	按 TEX 的要求对文件进行排版（第九章）
LATEX 模式（LATEX mode）	按 LATEX 的要求对文件进行排版（第九章）
C 模式（C mode）	书写 C 语言程序（第十二章）
C++ 模式（C++ mode）	书写 C++ 程序（第十二章）
FORTRAN 模式（FORTRAN mode）	书写 FORTRAN 程序（第十二章）
Emacs LISP 模式（Emacs LISP mode）	书写 Emacs LISP 函数（第十二章）
LISP 模式（LISP mode）	书写 LISP 程序（第十二章）
LISP 互动模式（LISP interaction mode）	书写和求值 LISP 表达式（第十二章）

当编辑一个文件的时候，Emacs会根据正在进行的编辑工作尝试进入正确的主模式。如果编辑一个以“*.c*”结尾的文件，它会转入C语言模式；如果编辑一个以“*.el*”结尾的文件，它会转入LISP语言模式。有时候，Emacs会根据文件的内容而不仅仅是文件名来做出判断。如果编辑一个按TEX格式排版的文件，它会转入TEX模式。如果它判断不出应该放到哪个编辑模式里，就会转入基本编辑模式（fundamental mode），也就是最普通的编辑模式。

在主模式之外还有一些副模式（minor mode）。副模式定义的是Emacs某些特定的行为，可以在某个主模式里打开或者关闭。比如，自动换行模式（auto-fill mode）表示Emacs将对文本自动换行；当你输入一个长句子的时候，它会在适当的位置自动插入一个换行符。表1-2列出了一些副模式和它们的作用，以及所在的有关章节。

表 1-2: Emacs 编辑器的副模式

模式	功能
自动换行模式（auto-fill mode）	开启字换行（word wrap）功能（第二章）
改写模式（overwrite mode）	打字时替换而不是插入字符（第二章）

PDG

表 1-2: Emacs 编辑器的副模式（续）

模式	功能
自动保存模式（auto-save mode）	把文件按一定周期自动保存到一个特殊的临时文件里（第二章）
行号模式（line number mode）	在状态行上显示当前文本行的编号（第二章）
临时标记模式（transient mark mode）	对被选取的文本区做高亮反显（第二章）
缩略语词模式（abbrev mode）	允许使用单词的简写形式（第三章）
大纲模式（outline mode）	书写大纲（第八章）
VC 模式（VC mode）	在 Emacs 下使用各种版本控制系统（第十五章）

大家可能已经注意到，大纲模式既是一个主模式，又是一个副模式。这表示它既可以作为一种主模式单独使用，也可以作为一种副模式用在其他主模式里。

Emacs还有其他一些没有列出的编辑模式，比如一些不常见但很有意思的程序设计语言（如 **Scheme** 等）所对应的编辑模式等。还有一些模式是 Emacs 自己使用的，比如对应于目录编辑功能的 Dired 模式（这个模式在第五章里介绍）。

最后，如果你擅长LISP程序设计，还可以自行增加新的编辑模式。Emacs的可扩展性似乎是无穷无尽的。

启动 Emacs

启动 Emacs 的办法很简单，输入“**emacs**”再加上要编辑（注 1）的文件名就行了。如果给出的文件名不存在，Emacs 将创建一个新的文件。你将看到类似于下面这样的画面:

注 1: 这个命令的名字在不同的计算机上可能会有所不同。GNU Emacs 有时候被叫做 *gnumacs* 或 *gmacs*，在安装有多种编辑器的站点上经常会出现这样的情况。也许你还得修改自己 UNIX 操作系统的搜索路径。如果你在输入“*emacs*”、“*gnumacs*”或者“*gmacs*”的时候出现一条“Command not Found（命令未找到）”错误信息，请向你的系统管理员求助。

PDG

% **emacs** *myfile*

```
Buffers Files Tools Edit Search Help

-----Emacs:myfile          (Fundamental)--L1--All----------------------
(New file)
```

在 UNIX 操作系统的提示符处输入 **emacs** 和一个文件名来开始一次 Emacs 会话。

如果给出的文件已经存在，Emacs 将读入文件并把它显示在屏幕上，如下所示：

% **emacs** *dickens*

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it
was the age of wisdom,it was the age of foolishness,it was the epoch
of belief,it was the epoch of incredulity,it was the season of
Light,it was the season of Darkness,it was the spring of hope,it
was the winter of despair,we had everything before us,we had nothing
before us,we were all going direct to Heaven,we were all going
direct the other way -in short,the period was so far like the present
period,that some of its noisiest authorities insisted on its being
received,for good or evil,in the superlative degree of comparison only.
-----Emacs:dickens         (Fundamental)--L1--All----------------------
```

也可以省略文件名。如果只输入了“**emacs**”，屏幕上将出现你所运行的Emacs的版本信息、如何启动在线帮助系统和其他一些提示性信息。这些信息会在开始输入第一个字符的时候消失；而Emacs将把输入内容放到一个名为*scratch*的空缓冲区里去，这是个试做各种练习的好地方。

Emacs 的编辑画面

进入Emacs的时候，你将在屏幕上看到一个很大的工作区（一般是20多行），你的编辑工作就将在这里进行（如图1-1所示）。一个光标将标识出你在文件里的前后位置。这个光标也叫做“插入点”。Emacs 老手们或者 Emacs 的在线帮助系统比较习惯于使用这个词，因此记住这个术语将会很有用。

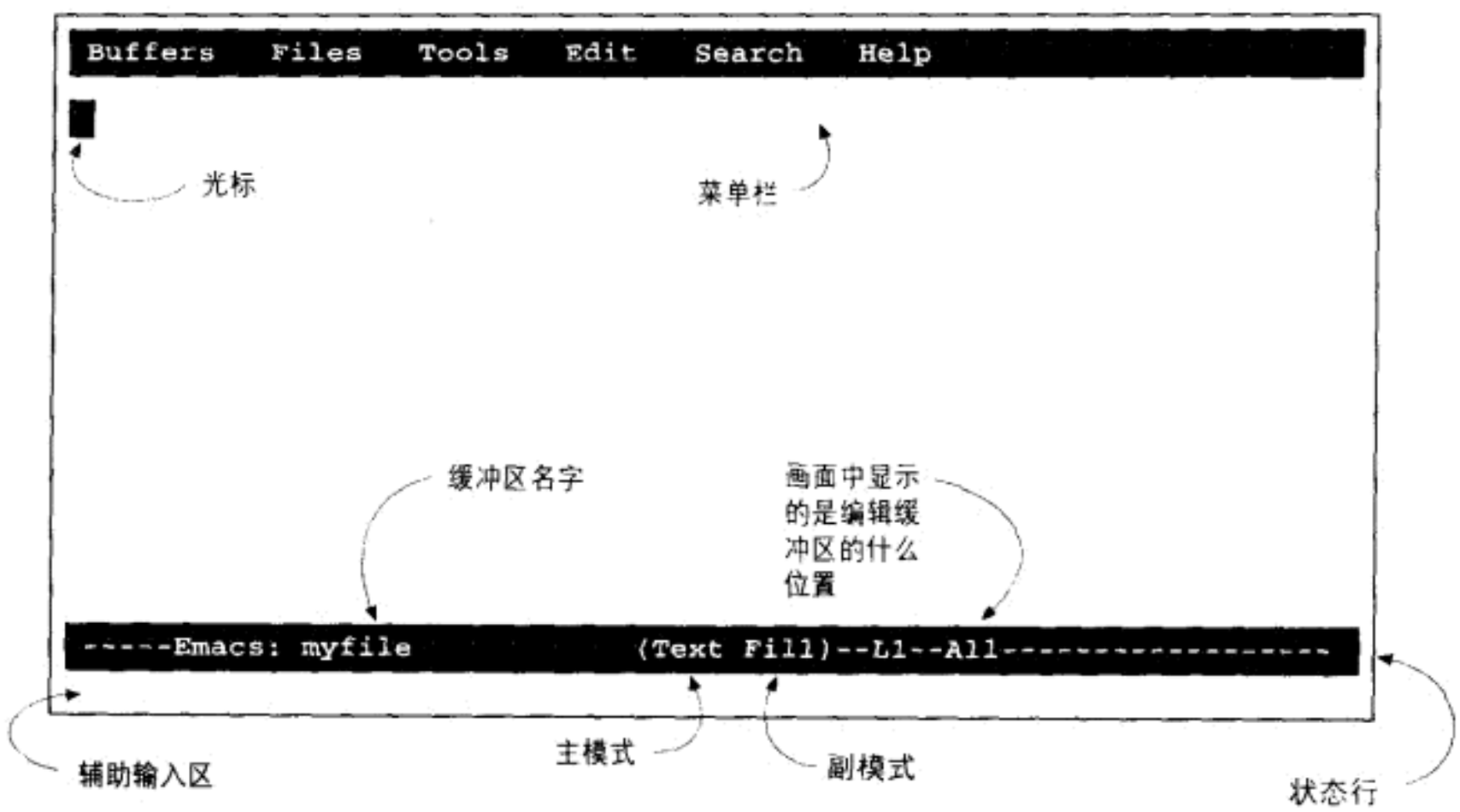


图 1-1：Emacs 的编辑画面

开始工作不需要做什么特殊的操作，在键盘上直接打字就行了。只要输入的是字母/数字字符和标点符号，Emacs 就会把它们插入到文本里去。光标指示 Emacs 将插入新字符的位置，它会随着打字动作而移动。Emacs 并没有为插入文本和输入命令分别准备编辑模式，这是它与许多编辑器（特别是 **vi**）的一个不同之处。现在就来输入些东西试试，你将发现 Emacs 是很容易使用的。（万一遇到了麻烦，请按 **C-g** 组合键。）

屏幕的顶部有一个菜单栏，上面排列着 **Buffers**（缓冲区）、**Files**（文件）、**Tools**（工具）、**Edit**（编辑）、**Search**（搜索）和 **Help**（帮助）等选项；这些选项将导向其他菜单。菜单栏的用法我们稍后再做介绍。

在屏幕的底部（倒数第 2 行），Emacs 会给出一大堆关于当前工作情况的信息。这一行叫做“状态行”。状态行靠左边的地方可能会有两个星号（**）。这两个星号的作用是表明正在编辑的文件在上次存盘之后又被修改过。如果没有做过任何修改，那里就不会出现两个星号。接下来，Emacs 显示“Emacs:”和正在编辑的缓冲区的名字（myfile）。接下来的圆括号里给出当前所处的编辑模式（编辑模式在本章前面刚刚讲过）。在它的后面，Emacs 给出在文件的前后位置：哪一行（图中的“L1”表示是第 1 行）和相对文件其余部分所处的位置。如果是文件的开头，Emacs 给出单

词“Top”；如果是文件的末尾，Emacs 给出“Bot”；如果是文件中间，它会给出一个百分数（比如，“50%”表示现在看到的是文件中部的内容）；如果整个文件都显示在屏幕上，Emacs 将给出单词“ALL”。

一个熟练的Emacs用户经常会同时打开多个缓冲区进行工作。如果是这种情况，则每个缓冲区都将有自己的一个状态行。我们现在还不用操心这些事情，只要知道每个缓冲区都有一个描述其工作情况的状态行就行了。

屏幕画面最底部、状态行的下面是辅助输入区（*minibuffer*）。Emacs 这个辅助性的输入区有许多用途，比如，Emacs 会把发出命令的执行结果回显在这里，在此输入文件名让Emacs 去查找，搜索和替换所使用的值也要输入在这里等等。Emacs 的出错信息也将显示在这个辅助输入区里。如果你困在辅助输入区里出不来了，请再次按下 **C-g** 组合键。

X 技巧：X 窗口系统下的 Emacs 画面

X窗口系统下的Emacs屏显画面与它在字符终端上的显示效果看起来稍微有些不同。每个Emacs窗口都有一个标题，它通常是“emacs@systemname”。但这并不是一成不变的：如果在一次Emacs会话里使用了多个X窗口，窗口的标题就会出现一些变化，具体情况我们将在第四章里介绍。在画面的右边是一个卷屏条，它能直观地反映出在文件中的位置，其作用与状态条上的百分比差不多。还可以利用卷屏条在文件中前后移动。图 1-2 给出了 Emacs 一个示范性的 X 窗口。

```
Buffers Files Tools Edit Search Help
        It was the best of times, it was the worst of times, it was
        the age of wisdom, it was the age of foolishness, it was the
        epoch of belief, it was the epoch of incredulity, it was the
        season of Light, it was the season of Darkness, it was the
        spring of hope, it was the winter of despair, we had
        everything before us, we had nothing before us, we were all
        going direct to Heaven, we were all going direct the other
        way--in short, the period was so far like the present period,
        that some of its noisiest authorities insisted on its being
        received, for good or evil, in the superlative degree of
        comparison only.

-----Emacs: dickens          9:45am 0.16    (Text Fill)--All-------------------
```

图 1-2：X 窗口系统下的 Emacs 编辑画面

Emacs 命令

我们马上就要开始学习一些Emacs命令了，所以，我们现在先来对它们做一个基本介绍。怎样才能发出命令呢？每个命令都有一个正式的名字，它们（如果你喜欢刨根问底儿的话）实际上是Emacs内部LISP例程的名字。这些名字一般都比较长，大多数人都不太喜欢输入完整的名称。所以，我们需要一些能够简化命令输入的办法。

Emacs把一个命令名与一个以**CTRL**或**ESC**打头的组合键关联起来。命令与组合键之间的这种联系被称为“绑定”。在X窗口系统环境里，Emacs会把一些命令与鼠标动作（单独按鼠标键或者先按下**SHIFT**或**CTRL**键再按鼠标键）和菜单里的某些选项绑定在一起。

Emacs的创作者们已经尽量把最常用的命令，与手指最容易触到的组合键绑定在了一起。大家将会遇到的各种组合键如下所示:

- 最常用的命令（比如那些光标移动命令）都被绑定为“**C-*n***”（*n*可以是任意字符）的形式。“**C-*n***”组合键的输入方法是：按住**CTRL**键，再按下“*n*”键，然后释放这两个键。
- 次常用的命令被绑定为“**ESC *n***”的形式，而*n*可以是任意字符。“**ESC *n***”的输入方法是：按下**ESC**键，释放它，再按下“**n**”键（注2）。
- 其他常用命令被绑定为“**C-x** *something*”（即C-x后面再加上一些东西——可能是一个或者多个字符，也可能是另外一个控制组合）的形式。在大家将要学习到的各种命令中，文件操作类命令通常被绑定为“**C-x** *something*”的形式。
- 某些特殊命令被绑定为“**C-c** *something*”的形式。这类命令通常都与某些特殊的编辑模式有关——比如图形模式或邮件模式等。对这类命令的介绍将出现在本书比较靠后的部分。

注2： Emacs文档和在线帮助功能里说的都是**META**键，它的简化形式是大写的字母“**M**”。就各种实际应用目的而言，**META**键与**ESC**键是完全等价的。这个按键在大多数键盘上都不存在（或者被隐藏起来了），所以本书用**ESC**键来代替它。如果你的键盘上确实有一个**META**键，那么它与**ESC**键还是有一点很重要的区别的。如果你准备发出一连串的**ESC**命令，就必须在每一个命令的前面按下**ESC**键；而如果你有个**META**键，那么你就可以按住**META**键来输入那一连串的命令。从这方面看，**META**键与**Ctrl**键有一定的相似之处。在Sun工作站上，空格键左右两端的按键就是**META**键。在某些键盘上，**Alt**键与**META**键作用相同。

- 上面这些规则依然没有顾及到所有的可能性，有些命令无法绑定为上述几种形式。这类命令的输入方法是“**ESC x** *long-command-nam* **RETURN**”。这种方法其实适用于全部的命令，但组合键通常比较容易学习。

Emacs还允许用户自己定义组合键。如果总在使用一些长格式的命令，那么这个功能就将非常方便。我们将在第十一章介绍更多关于自定义组合键的内容。

X 技巧：使用下拉菜单

X 窗口系统的用户可以通过下拉菜单来访问常用的命令。如果想看看 **Files**（文件）菜单里都有哪些东西，请先把“I”形鼠标指针移到屏幕顶部的单词“**Files**”上，再按下鼠标左键(除非我们另有说明，否则都是指“一直按住它”)。屏幕上将出现**Files**菜单，而鼠标光标的形状则变为一个箭头。这一章将介绍这个菜单里的四个选项，它们是“**Open File**（打开文件）”、“**Save Buffer**（保存缓冲区）”、“**Save Buffer As**（将缓冲区另存为）”和“**Exit Emacs**（退出Emacs）”。如果某个菜单项不可用，就呈灰色，就像此时 **Files** 菜单里的“**Delete Frame**（删除窗格）”选项那样。

按住鼠标左键的同时在菜单里上下移动光标。在菜单里移动光标的时候，光标下的菜单项将呈高亮反显状态。如果想选中某个选项，在该选项上释放鼠标左键即可。(知道为什么要一直按住鼠标按键了吧。如果随随便便地松开鼠标按键，天知道会选中哪一个选项!) 现在，既然我们只是随便试试而不是真的要进行什么操作，那么请按住鼠标按键的同时把光标移到Emacs画面的主窗口里，离开菜单，再松开鼠标按键。

菜单里的每个选项都可以通过键盘命令来访问。在菜单里，与各个菜单项对应的组合键就列在该菜单项的旁边。同时学习鼠标操作和键盘操作是个很不错的主意。有时，键盘操作的效率要高一些（比如在写入文稿、不想把手移到鼠标上去的时候）。学会键盘命令还有一个好处，那就是即使需要在没有 X 窗口系统的情况下使用Emacs，你也能够应付自如。

使用基于文本的菜单

Emacs 在第 19.30 版里增加了基于文本操作界面的菜单。至少有两大理由支持增加这些菜单:

- 不必再去记忆组合键，现在可以简单地通过选择一个菜单选项来执行有关的命令。
- “鼠标手”的患者不用再移动手指去够**CTRL**键，这个动作已经被研究证实会加重病症。

举个例子：假设你想使用“**find-file**（查找文件）”选项（具体含义马上就要介绍）。按下**F10**键或“**ESC** `”（单反引号键，通常位于键盘的左上角，紧靠在数字“1”旁边）将打开主菜单（注3）。屏幕画面显示如下：

按下：**F10**

```
Buffers Files Tools Edit Search Minibuf Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way-in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
-----Emacs:dickens          (Text Fill)--L1--All----------------------
Press PageUp Key to reach this buffer from the minibuffer.
Alternatively,you can use Up/Down keys (or your History keys)to change
the item in the minibuffer,and press RET when you are done,or press the
marked letters to pick up your choice.Type C-g or ESC ESC ESC to cancel.
In this buffer,type RET to select the completion near point.

Possible completions are:
B==>Buffers                         F==>Files
T==>Tools                           E==>Edit
S==>Search                          H==>Help
--**-Emacs:*Completions*.     (Completion List)--L1--All----------------
(up/down to change,PgUp to menu):B==>Buffers
```

Emacs打开一个列出菜单选项的编辑缓冲区。

选择菜单选项有3种方法：

- 可以使用**PgUp**键切换到*Completions*缓冲区里去，然后使用方向键移动到想执行的选项上，再按下回车键。
- 可以直接按下回车键选中出现在辅助输入区里的默认选项。如果想要的是另一

注3：虽然功能键在大多数情况下都能正常使用，但如果是通过Telnet或者终端仿真软件来使用Emacs，它们就不会起作用。因此，在这种情况下，必须使用“**ESC** `”组合键。

个选项，按动上、下方向键直到想要的选项出现在辅助输入区里，然后再按下回车键选中它。

- 可以输入 *Completions* 缓冲区里列出的各选项前面的字母。比如说，按下“**F**”键等于选中了“**Files**”选项。

在选择了一个菜单项之后，该菜单的各种选项就会出现在屏显画面里。重复上述过程直到你找到你想要的选项为止。

打开一个文件

在启动Emacs的时候，可以给出一个文件名以打开一个文件（就像我们刚才做的那样），也可以按下“**C-x C-f**”组合键（与此对应的长格式命令名是“**find-file**”）。“**C-x C-f**”的作用是创建一个新的编辑缓冲区，它的名字与文件的名字相同。

输入：**C-x C-f**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.

-----Emacs:dickens.      (Fundamental)--L1--All----------------------
Find file:~/
```

Emacs 提示输入一个文件名。请输入一个文件名，然后按下回车键。

输入：*newfile* **RETURN**

```
Buffers Files Tools Edit Search Help

-----Emacs:newfile          (Fundamental)--L1--All----------------------
(New file)
```

Emacs 创建一个新的缓冲区，新文件的内容将显示在这个缓冲区里。

使用“**C-x C-f**”组合键的方法是：先按住**CTRL**键，再依次按下“**x**”键和“**f**”键，然后松开**CTRL**键。这个操作过程听起来挺复杂，不过尝试几次之后你就会很熟练了。

按下“**C-x C-f**”组合键之后，Emacs会通过辅助输入区提示输入文件名。注意：只要是Emacs需要进一步提供信息，它就会把光标自动放到辅助输入区里。完成在辅助输入区里的输入之后，需要按下回车键来确认输入了一条命令。在普通编辑命令（比如，使用**CTRL**和**ESC**键的命令）后面是用不着按下回车键的。

如果对同一个文件尝试进行两次读入操作会发生什么样的事情呢？Emacs不会创建一个新的缓冲区，它会进入该文件所在的编辑缓冲区里去。

如果你还没有在Emacs中与输入过任何东西，现在正是试着输入一些东西的好机会。你很快就会发现自己需要多学些光标移动和编辑方面的命令，这再正常不过了。你完全可以跳过本章后面的内容去开始学习第二章的内容，但我们建议你还是先读完本章的“保存文件”和“退出Emacs”小节比较好。为了方便大家今后的参考，我们在这一章的末尾给出了一个命令速查表。如果你愿意多学一些文件操作方面的知识和有帮助作用的快捷操作方法，请随我们一起去完成这一章的学习。

如果读入了错误的文件

如果错误地读入了另外一个文件（比如，因为在另外一个目录里或者出现了输入错误），找到正确文件最简便的办法是按下“**C-x C-v**”组合键（对应于“**find-alternate-file**”命令）。这个命令的意思是“读取另一个文件来代替刚才读入的那一个”。按下“**C-x C-v**”组合键之后，Emacs会把当前文件的名字放到辅助输入区里去；对输入错误或不正确的文件路径（它们是读错文件最常见的两个原因）进行修改。输入正确的文件名，再按下回车键。Emacs会用新打开的文件替换编辑缓冲区里的内容。

Emacs的名称自动补足功能

Emacs有一个很有用的功能叫做“自动完成（completion）”。如果想打开一个已经存在的文件，只需输入该文件名的头几个字母，以构成一个惟一识别的文件名即可；然后按下**TAB**键，Emacs会自动补足文件名的剩余部分。请看下面的例子，假设要打开一个已经存在的文件，它的文件名是*dickens*。

输入：**C-x C-f di**

```
Buffers Files Tools Edit Search Minibuf Help

-----Emacs:newfile          (Fundamental)--L1--All--------------------
Find file:~/di
```

按下“**C-x C-f**”之后，Emacs 提示输入文件名。输入前几个字母。

按下：**TAB**

```
Buffers Files Tools Edit Search Minibuf Help

-----Emacs:newfile          (Fundamental)--L1--All--------------------
Find file:~/dickens
```

按下 **TAB** 键时，Emacs 将自动补足文件名的剩余部分。

按下：**RETURN**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in,short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.

-----Emacs:dickens          (Fundamental)--L1--All--------------------
```

Emacs 读入 *dickens* 文件后，可以开始对其进行编辑。

如果有不止一个文件的名字以“*di*”开头，则 Emacs 将打开一个窗口，把以这个字符串开头的文件都列出来。再多输入几个字符（足够惟一地确定文件即可）并再次按下**TAB**键，就可以从中挑选出想要编辑的文件了。也可以在自动补足窗口里移动到想要的条目上，然后按下回车键，这样也能从文件清单里挑选出想要的文件。（X 窗口系统的用户可以利用鼠标中键来实现自动补足功能。）

自动补足功能也能用在需要输入长格式命令名的场合。Emacs这个了不起的功能能够节省不少的时间。它不仅启发了Korn shell和**tcsh**的开发者，还被内置在自由软件基金会的shell **bash**里。关于自动补足功能的进一步讨论请参考第十六章。

注意：Emacs不允许在“**C-X C-f**”操作里使用UNIX通配符（*、?等）。（但有一个简单的办法可以绕过这条限制：如果你需要对一组文件进行编辑，可以在shell提示符下使用通配符来启动Emacs。比如，如果想对以“*projectx*”开头的全体文件进行编辑，就可以在shell提示符处输入“**emacs projectx***”。）可以使用波浪符（~）作为主目录的简写形式。

插入和追加文件

如果想把一个文件插入另外一个文件，只需移动到文件的适当位置，再按下“**C-x i**”即可。（是的，我们还没有向大家介绍如何在文件里前后移动。这些操作将在下一章里讨论。）如果想追加一个文件，移动到文件的结尾（**ESC >**）（注4），然后按下“**C-x i**”。与“**C-x C-f**”的情况相类似，Emacs会提示用户在辅助输入区里输入文件名。

Emacs如何确定默认目录

使用任何一个需要进一步给出文件名的命令时（比如“**C-x C-f**”），Emacs会在辅助输入区里给出一个默认的目录，然后由用户来输入文件名的其余部分。那么，Emacs如何确定默认目录呢？默认目录是根据光标当时所在的编辑缓冲区确定的。按下“**C-x C-f**”组合键的时候，如果正在编辑一个位于主目录里的文件，Emacs会假定用户要编辑所登录目录里的另外一个文件；如果正在编辑的文件的文件名是*/sources/macros/troff.txt*，Emacs会把默认子目录设置为*/source/macros*。如果想查找的是另外一个目录里的某个文件，就需要对Emacs提供的默认目录进行修改，或者是删除它再重新输入一个新目录。

出个难题考考大家：如果你正在编辑的那个缓冲区与文件没有联系，那么会出现什么样的情况？比如，如果你在启动Emacs时没有给出文件名，你就将在`*scratch*`

注4：“**ESC >**”的使用方法是：按下**ESC**键，释放后，再按下“>”键。

缓冲区里进行编辑，此时的默认目录又该如何设置呢？这当然是有规定的，但你其实不必为此费心——你只需按下“**C-x C-f**”组合键，看看Emacs在辅助输入区里给出的目录是什么，如果不合心意，把它改过来就是了。在猜测用户想设置的默认目录方面，Emacs还是相当有门道的。

保存文件

如果想保存正在编辑的文件，请按下“**C-x C-s**”组合键或者在“**Files**”菜单里选择“**Save Buffer**”（保存缓冲区）选项。Emacs会把文件存盘。为了让用户了解文件已被正确地保存，它会在辅助输入区minibuffer里显示一条“*Wrote filename*”（文件已存盘）信息。如果没有对文件进行任何修改，Emacs会在辅助输入区里显示一条“(No changes need to be saved)（没有需要保存的修改）”信息。

如果想用“**C-x C-s**”组合键把在*scratch*缓冲区中输入的内容保存起来，Emacs会提示你为它指定一个文件名。给出文件名之后，Emacs会修改状态行以反应你设置的文件名——把*scratch*替换为给出的新文件名。

如果按下“**C-x C-s**”组合键时发生了死机现象（再输入什么都没有反应），那么这时就需要将“**C-s**”和“**C-q**”当做流控制字符来使用了。这里，**C-s**的意思将是“停止接收输入”；而**C-q**则相当于重新启动这次会话。一个名为“**enable-flow-control**”（激活流控制）的命令（最早出现在Emacs 19里）可以快速解决这一问题。先输入“**ESC x enable-flow-control RETURN**”。然后，可以用“**C-**”代替“**C-s**”，用“**C-^**”代替“**C-q**”。不过这条命令只能解决本次会话中的这个问题。永久性的解决办法请参考第二章末尾“对Emacs进行定制”一节里的内容。对流控制问题的进一步讨论请参考第十一章。

如果在使用“**C-s**”时遇到了麻烦（甚至即使没有遇到麻烦），你可能更愿意用**write-file**（写文件）命令（对应于“**C-x C-w**”组合键）来保存自己的文件。**write-file**命令与**save-buffer**命令在做法上稍微有些差异。**save-buffer**命令假设你不想改变文件的名字；而**write-file**命令却认为你想修改文件名，它会让你在辅助输入区minibuffer里输入一个新的文件名。不过，如果你直接按下回车键而不是输入一个新的文件名，**write-file**命令会按原来名字对文件进行存盘——就像“**C-x C-s**”做的那样。

write-file命令可以用来编辑没有修改权限的文件。先用**find-file**命令把想查看或者编辑的文件读入一个缓冲区，再通过**write-file**命令用另外一个名字（也许还要用另外一个路径）把它存为私用的版本。这个办法能够把本来无权修改的文件复制为自己的一个文件；然后就可以对它进行编辑了。当然，原始文件是不会受到影响的。

退出 Emacs

如果想结束一次Emacs会话，可以按下“**C-x C-c**”组合键或者在**Files**菜单里选择“**Exit Emacs**（退出Emacs）”选项。如果对文件进行了编辑却没有保存，Emacs会问你是否想保存那些修改。如果回答是“**y**”，Emacs将在对文件存盘后退出；如果回答是“**n**”，Emacs会再次提问是否真的想放弃所做的修改并退出，这一次必须输入完整的“**yes**”或“**no**”作为回答。如果回答是“**no**”，则这次Emacs会话将持续下去，就像你根本没有按下过“**C-x C-c**”组合键一样。如果回答是“**yes**”，就将退出Emacs，在这次Emacs会话过程中所做的修改也就都不会保留下来。如果不打算把所做的修改保留下来，那么不存盘地退出就是最合理的做法。

顺便说一句，Emacs对所回答的是“**y**”还是“**yes**”是很挑剔的。它有时要求这样回答，有时又要求那样回答。如果它预期的回答是“**y**”，那么输入“**yes**”往往也能奏效；但反过来就未必能行。如果听见报警声并看到“`Please answer yes or no`”（请回答**yes**或**no**），就说明你没有完整地输入这两个单词之一而它却要求这样做。在19.29之前的版本里，它甚至对字母的大小写都很挑剔；如果使用的是早期的版本，请根据它的要求输入小写的“**y**”或“**n**”，或者小写的“**yes**”或“**no**”作为回答。

获取帮助

GNU Emacs的在线帮助功能非常丰富，具体细节我们将在第十六章里做详细介绍。进入帮助功能的方法是敲入“**C-h**”组合键或者从**Help**（帮助）（注5）菜单里选择一个选项。按下“**C-h**”组合键，会出现一个选项清单。按下“**C-h t**”组合键将启动Emacs教程，这是一个非常好的Emacs入门介绍。

注5： 在某些键盘上，用来进入帮助功能的按键是**F1**。

如果想了解按键组合的含义，请按下对应于“**describe-key**（按键释义）”命令的“**C-h k**”组合键。比如，如果按下“**C-h k C-x i**”，Emacs 将给出一个关于 **insert-file**（插入文件）命令的用法说明——这条命令被绑定在“**C-x i**”组合键上。而按下“**C-h f**”（命令名是 **describe-function**）则是让 Emacs 对某个函数（实际就是某个命令的完整名字，比如“**find-file**”）进行解释。从理论上讲，“**C-h k**”和“**C-h f**”给出的信息是完全一样的。它们之间的区别是：“**C-h k**”后面要跟着一个组合键，并让 Emacs 对这个组合键的功用做出解释；而“**C-h f**”后面要跟着一个命令名，并要求 Emacs 对这个函数的功用做出解释。

如果想了解“**C-x i**”组合键的作用。那么首先要输入“**C-h k**”（如图 1-3 所示）：

```
Buffers   Files   Tools   Edit   Search   Help

It was the best of times, it was the worst of times, it was the age of
wisdom, it was the age of foolishness, it was the epoch of belief, it
was the epoch of incredulity, it was the season of Light, it was the
season of Darkness, it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we
were all going direct to Heaven, we were all going direct the other
way--in short, the period was so far like the present period, that
some of its noisiest authorities insisted on its being received, for
good or evil, in the superlative degree of comparison only.

-----Emacs: dickens        (Fundamental)--L1--All----------------------
Describe key:
```

你现在准备查询关于某个键盘命令的帮助信息

图 1-3：查询关于某个键盘命令的帮助信息

然后按下“**C-x i**”组合键查出关于 **insert-file** 命令的帮助信息（如图 1-4 所示）。

注意几个问题：屏显画面现在被分为两个部分，这是因为用户正在查看两个不同的编辑缓冲区。每个缓冲区都有它自己的状态条。下半部分的缓冲区是 `*Help*` 缓冲区。它包含着关于 **insert-file** 命令的帮助信息，也就是想查找的东西。Emacs 把编辑光标留在 `dickens` 缓冲区里，因为它知道一般情况下不会有人想修改 `*Help*` 缓冲区里的内容。如果想让 `*Help*` 缓冲区从屏幕上消失，需要输入“C-x 1”命令（可以用“返回1号缓冲区”来帮助记忆这条命令）（注6），或者从**Files**菜单里选中“**One Window**”选项。其他帮助命令的行为都与此差不多。

注 6：这实际上是等于跳到了第四章，但不介绍这条命令而让 `*Help*` 缓冲区总占据着你屏显画面的一半好像不太公平。

```
Buffers   Files   Tools   Edit   Search   Help

It was the best of times, it was the worst of times, it was the age of
wisdom, it was the age of foolishness it was the epoch of belief, it
was the epoch of incredulity, it was the season of Light, it was the
season of Darkness, it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we
were all going direct to Heaven, we were all going direct the other
way--in short, the period was so far like the present period, that
some of its noisiest authorities insisted on its being received, for
good or evil, in the superlative degree of comparison only.

-----Emacs: dickens          (Fundamental)--L1--All-----------------------

C-x-i runs the command insert-file:
Insert contents of file FILENAME into buffer after point.
Set mark after the inserted text.

-----Emacs: *Help*           (Help View)--L1--All-----------------------

C-x 1 to remove help window. M-C-v to scroll help.
```

Emacs给出关于插入文件命令的帮助信息

图 1-4: Emacs 的帮助画面

大家可能还会注意到：在对命令进行解释的时候，Emacs把光标称为“*point*（插入点）”。GNU Emacs的各种文档里——包括在线文档和GNU Emacs的使用手册都使用了这种称呼。

Help 菜单

也可以通过**Help**（帮助）菜单快速访问帮助命令。在这个菜单上，大家可以看到刚才介绍过的几个选项：“**Tutorial**”（教程，键盘命令组合是“**C-h t**”）、“**Describe Key**”（按键释义，键盘命令组合是“**C-h k**”）和“**Describe Function**”（函数释义，键盘命令组合是“**C-h f**”）等。此外，还有其他一些选项可供使用，比如获取按键绑定对应表的简单办法、访问UNIX命令的使用手册页（manpages）和Emacs常见问题答疑（frequently asked questions，简称FAQ）文件的办法和对当前编辑模式的一个介绍等。还有一个可以用**info**命令来查阅Emacs在线文档的接口。选中“**Browse Manuals**（浏览使用手册）”就能启动**Info**命令。

本小节对进入Emacs帮助功能的几种途径进行了介绍。帮助功能还有很多种，第十六章对它们做详细的讨论。现在介绍的这些帮助功能应该足以满足大家开始使用Emacs的需要了。如果你还有兴趣多学点东西，可以直接跳到第十六章。

小结

这一章对启动、退出Emacs以及简单的文件操作进行了介绍。第二章将以此为基础继续介绍一些在Emacs里进行编辑工作所需要的基本命令。表1-3对本章涉及到的各种命令进行了汇总，**Files**和**Help**菜单里的部分选项也包括在其中。

表1-3：与文件操作有关的命令

键盘操作	命令名称	动作
C-x C-f *Files→Open File*	**find-file**	查找文件并在一个新缓冲区里打开它
C-x C-v	**find-alternate-file**	读入另外一个文件替换掉用“**C-x C-f**”读入的文件
C-x i *Files→Insert File*	**insert-file**	把文件插入到光标的当前位置
C-x C-s	**save-buffer**	保存文件
C-x C-w *Files→Save Buffer As*	**write-file**	把缓冲区内容写入一个文件
C-x C-c *Files→Exit Emacs*	**save-buffers-kill-emacs**	退出 Emacs
C-h	**help-command**	进入 Emacs 的在线帮助系统
C-h f *Help→Describe Function*	**describe-function**	给出某个给定命令名的在线帮助信息
C-h k *Help→Describe Key*	**describe-key**	给出某个给定击键序列的在帮助信息
C-h t *Help→Emacs Tutorial*	**help-with-tutorial**	启动 Emacs 教程
C-h i *Help→Browse Manuals*	**info-goto-emacs-command-node**	启动 **Info** 文档阅读器

疑难解答

- **在试图进入在线帮助系统时意外退出。**按下了“**C-h**”组合键（或者映射到“**C-h**”的其他键），而这个组合键被绑定为退格操作的ASCII控制序列。在Emacs

里，“**C-h**”是求助键，按下“**C-g**”组合键将退出帮助系统。如何把求助键从“**C-h**”组合键改为其他的绑定，比如“**C-x ?**”？具体设置办法请参考第二章中“对Emacs进行定制”一节。（如果只是想在文件中后退一个字符，可以通过按下“**C-b**”组合键来完成，这是下一章介绍的一个光标移动命令。）

- **无法进入在线帮助系统**。“**C-h**”组合键可能被映射为键盘上的**DEL**键了。试试**F1**键管不管用。如果**F1**键不管用，请参考第二章“对Emacs进行定制”一节。

- **死机**。你可能无意中按下了“**C-s**”组合键，而它将产生一个暂停数据流的流控制字符。有的工作站在一般情况下允许使用“**C-s**”组合键，但如果是通过**rlogin**、Telnet或者调制解调器来使用Emacs的，也往往会出现死机现象。要想重新恢复工作，请按下“**C-q**”组合键。如果问题还没有解决，请通过键盘输入“**ESC x enable-flow-control RETURN**”。可以用“**C-**”代替“**C-s**”，用“**C-^**”代替“**C-q**”。

- **Emacs的工作情况与本书所介绍的不一样**。请输入“**ESC x version RETURN**”以确认自己运行的确实是GNU Emacs 19（而不是其他版本的Emacs或者GNU Emacs的早期版本）。如果确定版本是GNU Emacs 19，请输入“**emacs -q**”退出Emacs并再次进入。第二章的“对Emacs进行定制”一节给出了一个比较彻底的解决方案。

- **启动Emacs或者查找文件时屏幕上出现乱码**。问题的原因可能是正在编辑的文件是一个二进制文件或者其他某种特殊的文件。这种情况经常出现在拼错了某个文件名的时候。请输入“**C-x C-v** *filename* **RETURN**”重新查找出正确的文件。如果什么事情都不对头——根本分辨不出是否有一个缓冲区、一个状态条或者一个辅助输入区minibuffer，那么可能是终端设置不适合Emacs，这时需要系统管理员或者专家的帮助。按下“**C-l**”（“L”的小写字母）也可能会有所帮助。

- **看到出错信息**“This terminal is not powerful enough to run Emacs（本终端功能不足，无法运行Emacs）”。终端设置有问题，或者（只是可能是）使用的终端太旧，无法支持Emacs的运行。不过，这样的终端现在已经非常少见了。这条消息通常意味着环境变量**TERM**的设置不正确。如果不知道如何对环境变量**TERM**进行设置，请向系统管理员求助。

- **在X窗口系统下，状态条或者辅助输入区没有出现在屏幕上**。Emacs窗口超出了显示器屏幕。请参考第十四章，对名为**geometry**的X变量进行调整，以使Emacs在启动时有一个合理的窗口尺寸。作为一种临时措施，重新调整窗口尺寸也可以解决这一问题。

- **画面中一个菜单也没有**。是否能够看到菜单取决于正在运行的Emacs版本和是否运行X窗口系统。在19.30版本之前，菜单只有在X窗口系统下才能工作。如果有X窗口系统却看不到任何菜单，可以输入“**ESC x menu-bar-mode**”把它们显示出来。就19.30版本而言，任何用户都应该能够看到菜单。请输入“**ESC x version**”命令来检查运行的Emacs版本。

- Emacs**报告**“Please enter y or n（请输入y或n）”，**而你已经这样做了**。问题的原因可能是输入的是大写字母，而Emacs要求的是小写字母。请检查是否按过键盘上的“**CAPS LOCK**”键。

第二章

文件编辑

本章内容：

- 光标的移动
- 文本的删除
- 文本块及其编辑操作
- 段落重排
- 编辑技巧和快捷键
- 命令的中止和修改的撤销
- 对 Emacs 进行定制

现在，大家已经了解了如何进入、退出Emacs以及如何对文件进行基本的操作，我们接下来将学习如何在文件里前后移动并进行编辑。Emacs为我们准备了许多在文件里前后移动的方法。但正是因为完成同一件事情的方法太多，所以刚开始的时候大家可能很容易把它们弄混。不要着急，随着学习的深入，这种混乱会逐步地减少，而大家也将逐渐学会如何去欣赏Emacs那丰富的命令。学会的方法越多，到达文件中准备开始编辑工作的位置所需要的击键次数也就越少。

如果你打算在学习编辑命令的同时对它们加以练习 —— 这是帮助你尽快掌握它们的好办法，可以从你手边随便什么东西开始输入一两页内容，报纸就很不错。这等于是替你自己准备了一段练习用的文本，可以用它们来练习自己在本章学到的各种编辑技巧。不要害怕出错，一口气输入下去。可以在掌握了本章所介绍的基本编辑技巧之后再回过头来改正那些错误。学习使用一种编辑器实际上是要形成某种习惯性的手指动作，而不是死板地记忆书上是怎样说的。只有亲自动手开始打字，才能学会正确的手指习惯动作。

在打字的时候，如果到达了屏幕显示画面的右端，会有两个选择。可以按回车键转到下一行，也可以继续打下去。如果在输入一个长句子的时候没有按回车键，Emacs会在到达显示画面右端的时候，自动在这一行的末尾加上一个反斜线（\）并转到下一行去。请看下面这个长句子：

```
Buffers Files Tools Edit Search Help
This is one very very very very very very very very very very very very v \
ery very very very very very very very very very very very very very very \
 very very very very very very very very very very very very very very ve \
ry very very very long line.
--**-Emacs: myfile          (Fundamental)--L1--All----------------------
```

当一行文本长于显示画面的宽度时，Emacs 会自动在行尾加上一个反斜线。

这个反斜线字符不是文本的一部分；它们只是一种标记，提醒显示画面里的下一行其实是属于上一行的（注 1）。

在每一行末尾都按一次回车键比较麻烦，而反斜线看起来又比较乱，Emacs 提供了一种比这两种做法都要好的方法 —— 自动换行模式（auto-fill mode），这种副模式把在什么地方断行的工作交给Emacs去决定。Emacs会在句子接近行尾的时候等待你输入一个空格，然后它会把下一个单词（有时候是好几个单词）转到下一行去。这种行为有时候也被叫做“字换行（word wrap）”功能，它在需要录入大段文字的时候是很有用的。出于这个理由，自动换行模式和文本模式（text mode）通常是形影不离。文本模式通常用来录入英文；但在编写程序的时候，人工添加换行符往往是人们更喜欢用的方法。

自动换行模式不会被默认地设置为on。请查看状态行。如果它上面有单词“`Fill`”，就说明已经在自动换行模式里了；有人（很可能是系统管理员）替用户把它设置为on。如果在状态行上没有看到这个单词，可以输入“**ESC x auto-fill-mode RETURN**”（注 2），把这个编辑缓冲区的自动换行模式设置为on，但这只对这一个缓冲区有用。如果不想要自动换行模式，请再次输入“**ESC x auto-fill-mode RETURN**”。这个命令就像是一个信号灯开关，只不过它切换的是自动换行模式的开、关状态。

注 1：　在某些情况下（我们稍后会讲到），Emacs 会在显示画面的右边界放上一个美元符号（$），这一行的其余部分就不显示出来了。

注 2：　当我们说“输入‘**ESC x this-outrageously-long-string RETURN**’”的时候，别忘记Emacs的自动完成功能。在输入几个字符后按下**TAB**键，Emacs通常就会自动填上单词的其余部分或者命令的其余部分。应该尽量把自己的工作转移一些给Emacs。

用户可能（就像我们一样）希望在每次进行文件编辑的时候都自动进入自动换行模式。它的具体做法将在本章快要结束的时候进行介绍。

光标的移动

移动光标最简单的方法是点击鼠标左键（如果有鼠标可用的话）或者按动方向键。把光标向右移动一个字符位置的Emacs命令是“**C-f**”（“**f**”表示“forward”，向前）；而“**C-b**”将把光标向左移动一个字符位置。把光标向上移动一行，按下“**C-p**”（对应的命令名是**previous-line**）；向下移动一行，按下“**C-n**”（对应的命令名是**next-line**）。知道了这些字母所代表的单词，就可以很容易地记住这些命令了。图2-1给出了如何使用Emacs命令让光标做上、下、左、右方向的移动。

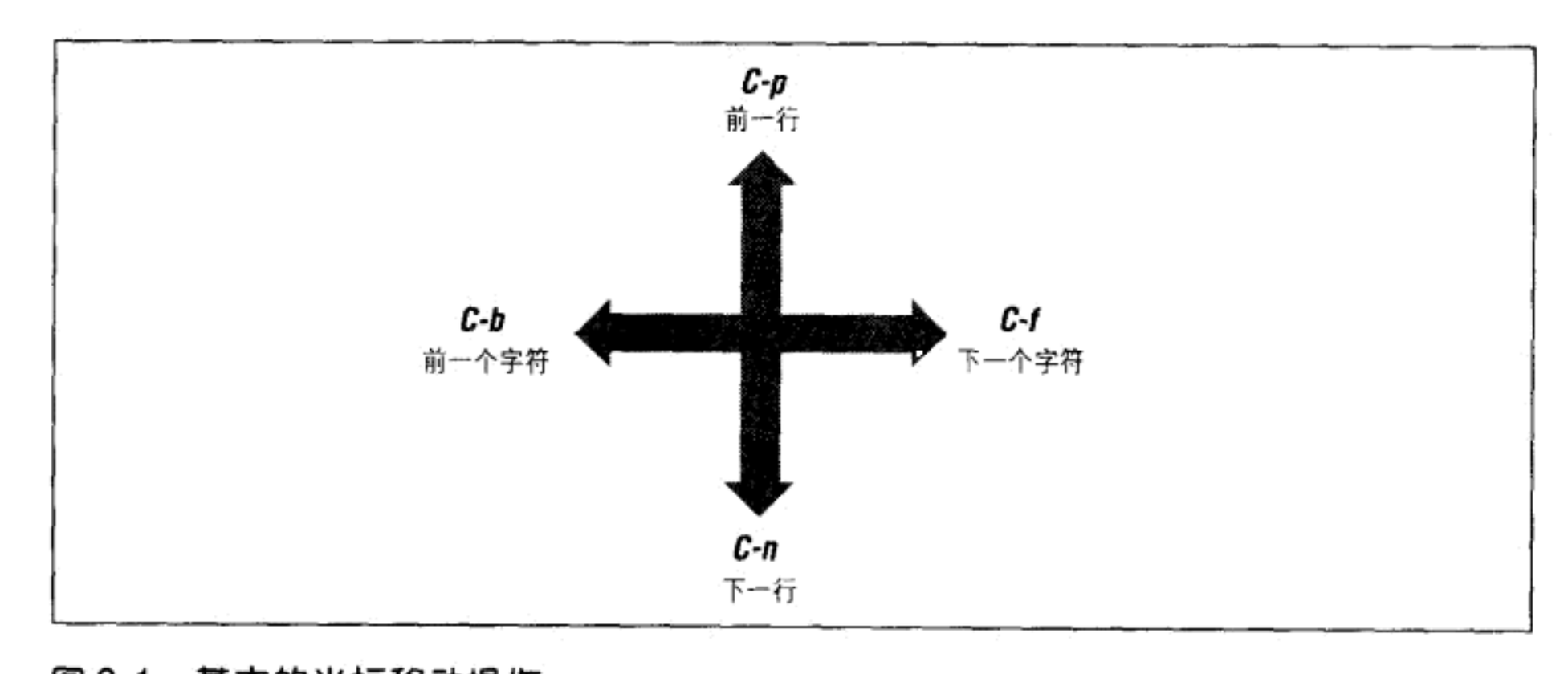


图2-1：基本的光标移动操作

如果是在一行的末尾，“**C-f**”将把光标移到下一行的第一个字符；类似地，如果是在一行的开始，“**C-b**”将把光标移到上一行的最后一个字符。如果光标不能被移动，Emacs会鸣叫并显示出错信息“Beginning of buffer（缓冲区头）”或“End of buffer（缓冲区尾）”。这条规则有一个重要的例外：如果在缓冲区的最后一行按下“**C-n**”组合键，将把光标移动到下一行，即等于给缓冲区增加一个新行。此时（也只有在此时）的“**C-n**”不仅把光标移动到下一行，还创建了一个新行。别问我们为什么会这样。

移动光标的其他方法

现在我们来学习更高级的光标移动方法。一种常见的情况是把光标向前或者向后移过一个单词："**ESC f**"把光标右移一个单词；"**ESC b**"把光标左移一个单词。还可以直接把光标移动到一行的开始或者结束。"**C-a**"把光标移到一行的开始（就像"*a*"是字母表的第一个字母一样），"**C-e**"把光标移到一行的结束。如果想把光标左移一个句子，按下"**ESC a**"；如果想把光标右移一个句子，按下"**ESC e**"。如果想把光标下移一个段落，按下"**ESC }**"；如果想把光标向上移一个段落，按下"**ESC {**"。如果光标处在一个句子或者一个段落的半中间，那么把光标向回移过一个句子或者段落将把光标移到这个句子或者段落的开始。

图 2-2 用维克多·雨果的小说《悲惨世界》里的几段话做例子，演示各种把光标一次移动一个以上字符的方法。

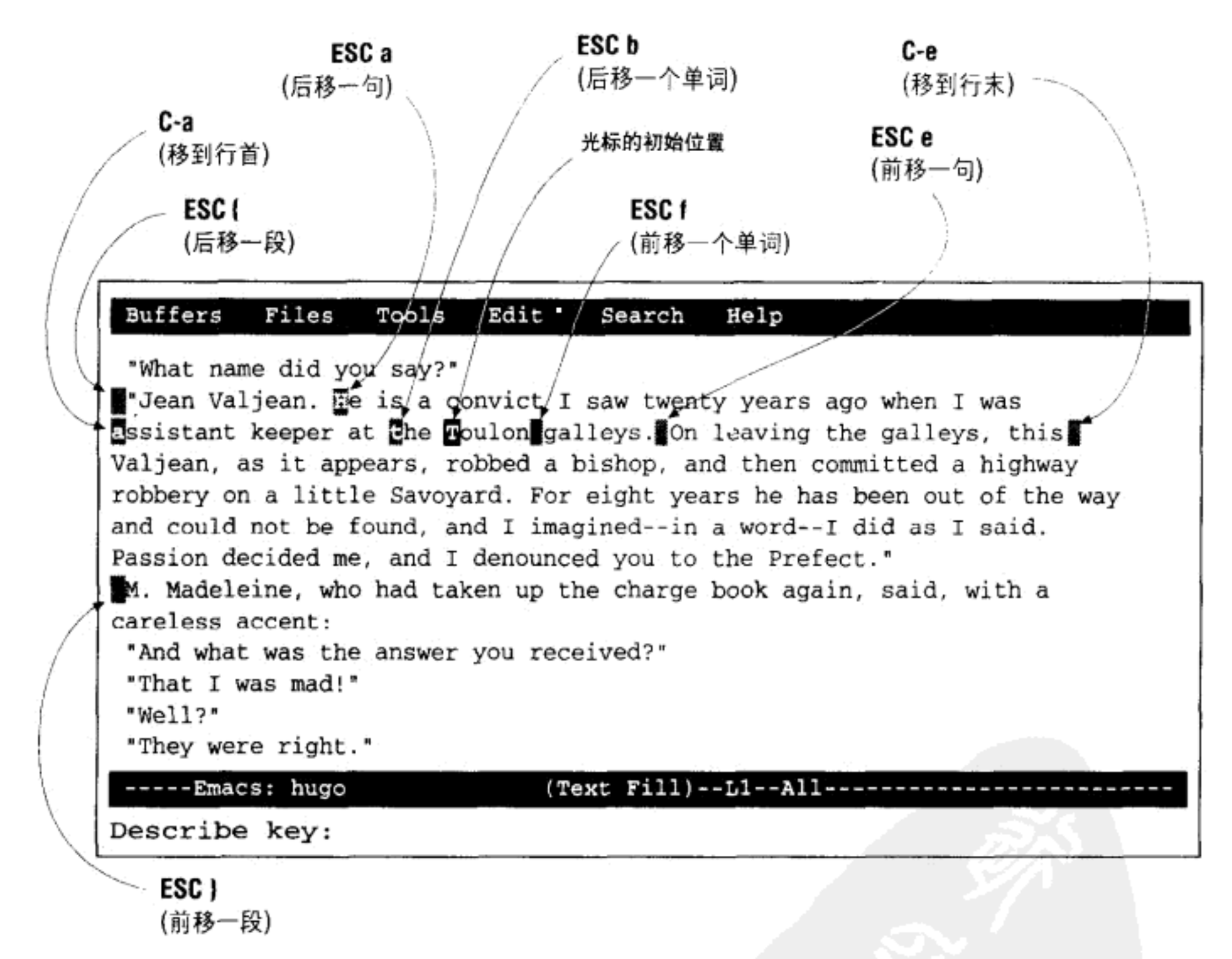


图 2-2：光标的快速大范围移动

下面是记忆这些基本编辑命令的重要提示。请注意以“**CTRL**”开头的命令与以“**ESC**”开头的命令之间的区别。以“**CTRL**”开头的命令的光标移动距离通常都要比对应的以“**ESC**”开头的命令移动距离短。比如，“**C-b**”把光标向左移动一个字符，而“**ESC b**”把光标向左移动一个单词。类似地，“**C-a**”把光标移动到文本行的开始，而“**ESC a**”把光标移动到一个句子的开始。

在以句子或者段落为单位来移动光标方面有一个限制性的前提：Emacs对句子的定义是很严格的。如果句子的结束位置不是一个文本行的结尾，那么它最后一个标点符号的后面必须有两个空格。如果只有一个空格，Emacs就无法分辨。类似地，把光标向前或者向后移过一个段落也涉及到Emacs对段落的定义。对Emacs（以及我们当中的大多数人）来说，段落通常要缩进一个制表位 —— 至少要缩进一个空格，或者两个段落之间增加一个空白行(现代书信体)。用户可以对这些设定情况进行修改，但必须先掌握正则表达式的使用方法 —— 我们将在第三章对正则表达式做简单的介绍，在第十三章做进一步的讨论。第十一章将涉及有关修改Emacs变量的内容。

如果文件里有分页符，可以通过敲入“**C-x]**”（命令名为**forward-page**）或“**C-x [**”（命令名为**backward-page**）组合键，来把光标移动到下一页或上一页。类似于段落和句子的光标移动方式，按页移动光标又将涉及到Emacs对页面的定义。Emacs所使用的分页符是由一个名为**page-delimiter**的变量定义的。如果文件里没有Emacs能够识别出来的分页符，Emacs就会把编辑缓冲区看做是一个非常长的页面。在这种情况下，**forward-page**命令会把光标移到缓冲区的末尾，而**backward-page**命令则会把光标移到缓冲区的开头。

文本模式里的分页符是一个进纸换页字符，它的作用是告诉打印机先前进到下一个打印页（在连续纸上，这等于是让打印机走纸到下一个页面，这就是“*formfeed*”进纸这个术语的来历）再继续打印。如果想在文本模式下在文件里插入分页符，可以按下“**C-q C-l**”（小写的“L”字母）组合键。“**C-q**”叫做“引用”命令，它的作用是告诉Emacs要把“**C-l**”字符插入到文件里去，而不是把它解释为重新绘制屏显画面命令。“**C-l**”字符在文件里的样子就像是两个字符（^L），但实际上是一个。（大家用**DEL**键试着删除一个这样的字符就明白了。）

把光标一次移过一个（或者多个）屏显画面

如果想一次把屏幕上的文件上卷一页，可以使用键盘上的**PgDn**键或者按下“**C-v**”（命令名是**scroll-up**）组合键。Emacs会把文件的下一屏内容显示到屏幕上。它会把上一屏幕内容的最后几行留在屏幕顶部，好让用户掌握文件的上下文关系。类似地，按下“**ESC v**”（或者**PgUp**键）将把文件的前一屏内容显示在屏幕上。“**C-v**”和“**ESC v**”提供了快速翻卷文件的简便方法。

如果输入的光标移动命令的目标位置在当前屏显文本内容以外，Emacs会自动进行卷屏。比如说，如果在屏幕最后一行按下“**C-n**”组合键，Emacs就会上卷一行。类似地，如果在屏幕最顶部按下“**C-p**”组合键，Emacs就会下卷一行。

有时候，需要直接到达某个文件的开头或者结尾。在Emacs里，按下“**ESC >**”或者按下**END**键将移动到编辑缓冲区的末尾；而按下“**ESC <**”或者按下HOME键将移动到文件的开始。请记住：“>”指向缓冲区尾，而“<”指向缓冲区头。

另外还有两种很方便的光标移动方法。“**ESC x goto-line *n* RETURN**”把光标移动到文件的第*n*行。当然，Emacs是从文件头开始计算行数的。类似地，“**ESC x goto-char *n* RETURN**”把光标移动到文件的第*n*个字符，从文件头开始计数。两个命令中的“*n*”都代表着一个数字。

这两个命令对程序员来说是非常有用的，这是因为许多编译器给出的出错信息都是“Syntax error on line 356（第356行语法错误）”这样的格式。这两个命令可以让用户迅速找到发生错误的位置。还有一些更复杂的方法可以把Emacs与编译器、或其他程序产生的出错报告联系起来；另外还有几个光标移动命令只能用在编辑程序代码的场合，这些内容都将在第十二章里讨论。

X技巧：使用卷屏条

用鼠标来移动光标就更简单了。只要在屏显画面上的某个位置点击一下鼠标左键，光标就会跳到那个地方。使用卷屏条时的直观性要稍差一些。点击屏显画面右边的卷屏条可以在文件中移动。当把鼠标指针移动到卷屏条上的时候，它的形状会变成一个指向上下方向的双向箭头。此时，点击左键将使文件内容向上移动，点击右键将使文件内容向下移动。按住鼠标中键后可以上下拖动卷屏条中的滑块。把滑块拖

到卷屏条的顶部可看到缓冲区开始部分的内容；而把滑块一直向下拖动可看到缓冲区结尾部分的内容。也可以把滑块放到卷屏条的任何中间位置上；把滑块放到卷屏条的中部可到缓冲区中间部分的内容。

命令的重复执行

现在来学习几个提高操作效率的技巧。Emacs的任何命令都允许重复执行多次。第一种方法是在准备重复执行的命令前面加上“**ESC** *n*”，其中的“*n*”是准备重复执行的次数。这个命令被称为**digit-argument**命令。

如果准备重复执行一个命令多次，可以用“**ESC** *n*”指定一个很大的数字。举个例子：假设正在编辑一个大约有1000行的大文件。如果输入“**ESC 500 C-n**”，光标就将向下移动500行而到达文件的中部。在大文件里用这个命令来移动光标通常是最快的。如果在“**ESC** *n*”中给出的数字超出了命令所能执行的范围，它将执行尽可能多的次数之后停下来。

另外一个可以重复执行编辑的命令是“**C-u**”（命令名是**universal-argument**）。可以像使用“**ESC** *n*”那样也在“**C-u**”后设定一个参数，“**ESC 5**”和“**C-u 5**”都会把随后的命令重复执行5次。但“**C-u**”与“**ESC** *n*”的不同之处在于前者即使没有参数也可以重复执行命令。如果不带参数，“**C-u**”将把随后的命令重复执行4次。如果按下的是“**C-u C-u**”，它将重复执行命令16次。也就是说，重复使用“**C-u**”可以按4的幂次来重复执行随后的命令：16次、64次、256次，依此类推（注3）。

重新绘制屏显画面

很多情况下需要用“C-l”（小写“L”字母）组合键来重新绘制屏显画面。比如，如果是在一个字符终端上而不是X窗口系统下来使用Emacs的，那么，随着时间的推移，屏显画面就可能会因为其他进程发送的各种消息而变得混乱不堪。比如，可能

注3：在大多数情况下，可以按我们这里的介绍来使用“**C-u**”。可它并不总是一个命令重复因子；有时候，“**C-u**”的作用会是改变某个命令的功能。大家将在本章的后面看到一个这样的例子。不过，只要正在做的事情与命令的重复执行有关，“**C-u**”就几乎总是能起到命令重复因子的作用。

会收到通知有电子邮件到达、有人想用**talk**聊天软件请你吃午饭，或者磁盘空间不足等各种各样的消息。如果有的是用调制解调器，屏幕上还会显示出许多“垃圾”字符来。Emacs是不会被这些来来往往的字符弄糊涂的，不过它们却可能把用户给弄糊涂，因为屏幕上多出了一些不属于自己文件的文本，而光标也没有处于它应在位置上等等。如果遇到这种情况，按下“**C-l**”就可以解决问题；Emacs会正确地重新绘制屏显画面。作为一般原则，只要屏幕有点乱，就可以试试“**C-l**”组合键。

“**C-l**”还有另外一个好处：它会把正在编辑的那一行放到屏显画面的中心位置。（准确地说，是放到“窗口的中心位置”，但我们还没有讲到窗口。我们将在第四章讨论它们。）当打字打到画面的底部或者顶部时，这个命令的好处就体现出来了。按下“**C-l**”会把用户最关心的文件内容放到屏显画面的中心位置，方便用户看到它的上下文。

表2-1列出了我们前面介绍的各种光标移动命令。

表2-1：光标移动命令速查表

键盘操作	命令名称	动作
C-f	**forward-char**	光标前移一个字符（右）
C-b	**backward-char**	光标后移一个字符（左）
C-p	**previous-line**	光标前移一行（上）
C-n	**next-line**	光标后移一行（下）
ESC f	**forward-word**	光标前移一个单词
ESC b	**backward-word**	光标后移一个单词
C-a	**beginning-of-line**	光标移到行首
C-e	**end-of-line**	光标移到行尾
ESC e	**forward-sentence**	光标前移一个句子
ESC a	**backward-sentence**	光标后移一个句子
ESC }	**forward-paragraph**	光标前移一个段落
ESC {	**backward-paragraph**	光标后移一个段落
C-v	**scroll-up**	屏幕上卷一屏
ESC v	**scroll-down**	屏幕下卷一屏
C-x]	**forward-page**	光标前移一页
C-x [	**backward-page**	光标后移一页

表 2-1：光标移动命令速查表（续）

键盘操作	命令名称	动作
ESC <	**beginning-of-buffer**	光标前移到文件头
ESC >	**end-of-buffer**	光标后移到文件尾
（无）[a]	**goto-line**	光标前进到文件的第 *n* 行
（无）	**goto-char**	光标前进到文件的第 *n* 个字符
C-l	**recenter**	重新绘制屏显画面，当前行放在画面中心处
ESC *n*	**digit-argument**	重复执行 *n* 次后续命令
C-u *n*	**universal-argument**	重复执行 *n* 次后续命令（省略 *n* 时重复 4 次）

a. 表中第一栏里的“（无）”表示如果想执行这个命令，就必须先按下“**ESC x**”，再输入该命令的全名，最后按下回车键。它们没有对应的默认组合键。

Emacs 命令与你的键盘

许多 Emacs 命令可以通过键盘上的标准按键来快速输入，比如 **PgDn**（前进到下一页）和 **Home**（跳到缓冲区的开头）等。图 2-3 给出了一个键盘布局样板和各有关按键的功能，各个用户按键可能与图中给出的位置稍有差别。不过，如果键盘上有同名或者名称相似的按键，它们就应该能用。“应该能用”的意思是这些按键在少数情况下会不能用，比如你在一台远程机器上使用 Emacs 的时候。建议大家记住这些标准的 Emacs 命令，因为它们在任何一种键盘上都能用；而且，一旦掌握了它们的用法，用手指够起来往往也比较容易。

文本的删除

在用文本删除命令做练习之前，我们想先把撤销操作（undo）命令介绍给大家，本章后面将对这个命令做详细的讨论。按下“**C-x u**”组合键将撤销最近一次的编辑操作；再次按下这个组合键会撤销再上一次的编辑操作，以此类推。

Emacs 为我们准备了很多种删除文本的方法。删除文本最简单的方法是按下键盘上的 **DEL** 键，它的作用是删除紧靠光标左侧的那个字符。**DEL** 键在键盘上的位置请参考图 2-3。**DEL** 键的功能是最容易定义的：删除光标前面的那个字符。在打字的

时候，如果想删除刚刚输入的那个字符，应该按下键盘上的哪个按键呢？在Emacs里，应该按下**DEL**键。

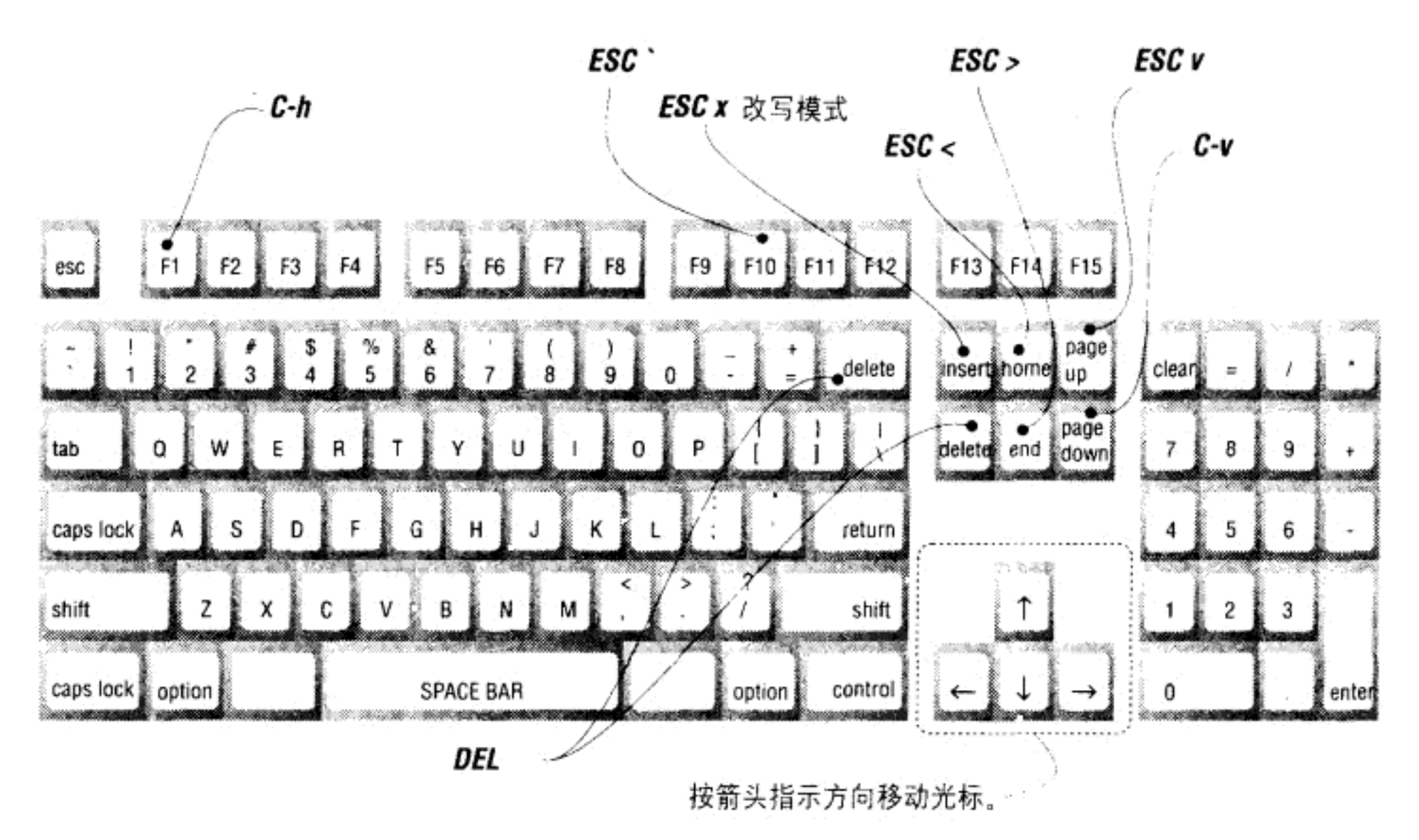


图 2-3: Emacs 命令与键盘

Emacs提供了许多其他的删除命令，比如用来删除光标所在位置处字符的“**C-d**”命令（命令名是**delete-character**）等——虽然命令很多，不过它们都是有存在价值的。删除下一个单词的命令是“**ESC d**”（命令名是**kill-word**）。再次提醒大家注意**ESC**是如何扩大命令的作用范围的：“**C-d**”对字符进行操作，而“**ESC d**”对单词进行操作。

Emacs提供了向前、向后删除单词、句子和段落的编辑命令。可以根据这些命令的名字猜出它们在光标处于单词、句子或者段落之间时会有什么样的操作结果。可是，如果光标位于一个单词、句子或者段落的中间，它们的操作结果就有些出人意料了：它们会删除当前那个单词、句子或者段落的一部分，具体是向前删还是向后删要看命令本身是向前操作还是向后操作。请看下面这些操作实例，“**ESC d**”会根据光标的不同位置采取不同的行动：

如果光标位置在:	"ESC d" 的操作结果:
It was the worst of times	It was the w of times
It was the worst of times	It was the of times
It was the worst of times	It was the wors of times

类似地，如果在某个单词的中间让 Emacs 去删除前一个单词（**ESC DEL**，命令名是**backward-kill-word**），它将从光标位置开始删除一直到那个单词的第一个字母。

如果想全部或者部分地删除一行文本，可以使用"**C-k**"（命令名是**kill-line**）命令。这个命令将把从光标位置到行尾之间的文本全删除。在空白行上按下"**C-k**"组合键将删除这一行本身。因此，一般要用两个"**C-k**"才能删除一行：一个用来删除文本，另一个用来删除剩下来的空白行。如果想删除的是从行首到光标位置之间的东西，可以试试复杂点的组合命令"**ESC – C-k**"（即先按**ESC**键，后面跟一个短划线，然后是"**C-k**"）。

恢复已删除的文本

大家可能已经注意到 Emacs 中某些删除命令的名称里有个单词"*kill*"，如**kill-region**、**kill-word**等。在 Emacs 里，删除并不意味着永远消失。事实上，在许多情况下，已经被删掉的文本都可以再恢复。被删掉的文本并没有消失，它们被隐藏在一个被称为"删除环（kill ring）"的地方。删除环与大家熟悉的剪贴板有着异曲同工之处。要想恢复已经删除的东西，可以按下"**C-y**"（命令名是**yank**）组合键（注4），或者按下**SHIFT-INSERT**组合键，或者从**Edit**（编辑）菜单里选择"**Paste Most Recent**"选项。更方便的是，如果连续删除了多行文本，Emacs 会把它们收集起来做为一个整体放到删除环里去——这时用一个"**C-y**"命令就可以把它们都恢复回来。在下面的例子里，我们先用4个"**C-k**"删除小说《双城记》选段的前两行（记住：第一个"**C-k**"用来删除文本，第二个"**C-k**"用来删除剩下来的空白行），然后用一个"**C-y**"就把它们都恢复回来了。

注4： 正在学习 Emacs 的 vi 用户请注意：vi 里也有一个术语叫做"*yank*"，但这两者的含义几乎正好相反。千万不要弄混了。

初始状态:

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
-----Emacs:dickens          (Text Fill)--L1--All----------------------
```

光标位于左上角。

按下: **C-k C-k C-k C-k**

```
Buffers Files Tools Edit Search Help
as the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
--**-Emacs:dickens          (Text Fill)--L1--All----------------------
```

用“**C-k**”删除了前两行。

按下: **C-y**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
as the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
--**-Emacs:dickens          (Text Fill)--L3--All----------------------
```

只用一个操作就把所有东西都恢复回来了。

到底有哪些东西会被放到删除环里去呢？进入删除环的内容包括用“**C-k**”命令删除的所有东西、用“**C-w**”命令删除的所有东西和用“**ESC w**”命令复制的所有东西（后两个命令马上就会学到）。用“**ESC d**”和“**ESC DEL**”命令及其变体删除

的单词、句子和段落也将进入删除环。此外，用“**C-u**”加**DEL**或“**C-d**”删除的文本也将进入删除环。不会被Emacs保存到删除环里去的内容大概只有用**DEL**或“**C-d**”删除的单个字符了。（如有必要，可以用**undo**撤销操作命令“**C-x u**”来恢复这类删除操作。）

Emacs对删除环中文本的处理也很聪明：它总是能够把一组删除操作所删除下来的东西，正确地拼凑成一个大段的文本。比如，可以先按下几个“**ESC d**”，再来几个“**ESC DEL**”、中间还可以再夹杂几个“**C-k**”。可当按下“**C-y**”组合键的时候，Emacs会把刚才删除的文本按正确的顺序都恢复回来。

但必须注意这样一个问题：如果给出了一个不属于“kill”类命令的命令，Emacs就将停止拼凑被删除文本的工作。比如，假设先按了一次“**C-k**”组合键，然后用“**C-d**”删除了一个字符，最后又按了一次“**C-k**”组合键。这就不能算是一组连续的删除操作了；实际进行的是两次互不相干的删除操作。稍后再向大家介绍如何把前面的删除操作恢复回来。

表2-2对文本的删除和恢复命令进行了汇总，**Edit**菜单里的有关选项也包括在其中。

表2-2：文本删除命令速查表

键盘操作	命令名称	动作
C-d	**delete-char**	删除光标位置上的字符
DEL	**delete-backward-char**	删除光标前面的字符
ESC d	**kill-word**	删除光标后面的单词
ESC DEL	**backward-kill-word**	删除光标前面的单词
C-k	**kill-line**	从光标位置删除到行尾
ESC k	**kill-sentence**	删除光标后面的句子
C-x DEL	**backward-kill-sentence**	删除光标前面的句子
C-y 或 **SHIFT-INSERT** *Edit→Paste Most Recent*	**yank**	恢复被删除的文本
C-w 或 **SHIFT-DELETE** *Edit→Cut*	**kill-region**	删除文本块
（无）	**kill-paragraph**	删除光标后面的段落
（无）	**backward-kill-paragraph**	删除光标前面的段落

文本块及其编辑操作

如果想删除一个短语该怎么办？段落的一部分呢？几个段落呢？可以组合使用各种删除命令来删除想去掉的东西，不过Emacs已经准备了一个更简单的方法：用标记文本的方法把打算删除的东西定义为一个区域。这个被标记出来的区域就叫做“文本块（region）”。

为了对文本块进行定义，要用到一个称为“文本标记（mark）”的辅助指针。某些Emacs版本的文本标记在屏幕上是可见的，可惜GNU Emacs的文本标记是不可见的。

把光标移到文本块的一端，按“**C-@**”或“**C-SPACE**”组合键设置一个文本标记；然后移动光标，文本标记和光标当前位置之间的文本就构成了一个文本块。（在提到光标的时候，人们也经常使用“插入点（point）”这个词。不过，光标和插入点却有一个细小但很重要的区别。光标显示在一个字符上；而Emacs中的插入点实际是在光标位置处字符与其前一个字符之间的夹缝里。两者的区别确实很小，但在标记文本的时候就值得细加注意了。）图2-4给出了插入点、文本标记和文本块之间的关系。

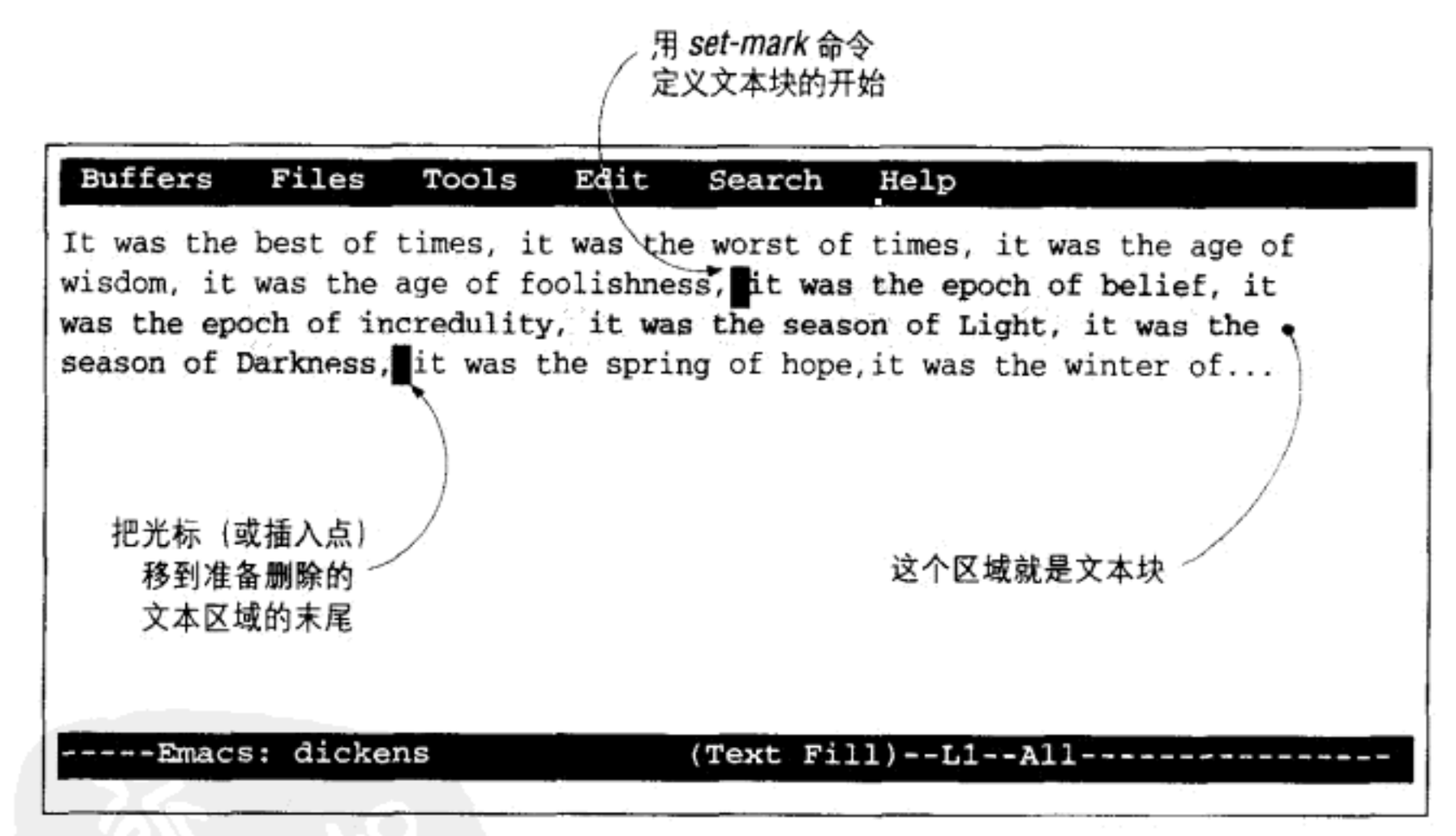


图2-4：插入点、文本标记和文本块

我们来做一个标记文本块的示范。在下面的例子里，打算删除的是“it was the worst of times.”这句话。首先，把光标移到这段话的开始，设置文本标记；然后，把光标移动到这段话的结尾；最后执行删除操作。给被选取的文本块加上阴影以示醒目（但书里的阴影实际上是不会反映在屏幕上的，除非使用了X）。

把光标移到“it”的首字母，按下“**C-@**”组合键。

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.

-----Emacs:dickens          (Text Fill)--L1--All----------------------
Mark set
```

设置文本标记；辅助输入区里出现“Mark set”字样。

把光标移到“it was the age of wisdom.”中“it”的字母“i”处。因为插入点的精确位置在字母“i”的前面，所以它正是我们需要的位置。

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it
was the age of wisdom,it was the age of foolishness,it was the epoch
of belief,it was the epoch of incredulity,it was the season of
Light,it was the season of Darkness,it was the spring of hope,it
was the winter of despair,we had everything before us,we had nothing
before us,we were all going direct to Heaven,we were all going
direct the other way—in short,the period was so far like the present
period,that some of its noisiest authorities insisted on its being
received,for good or evil,in the superlative degree of comparison
only.

-----Emacs:dickens          (Text Fill)--L1--All----------------------
```

把光标移到准备标记的文本块的末尾。

现在，文本块就标记好了。在执行删除命令之前最好先检查一下，看文本块标记得是否正确。按下“**C-x C-x**”（命令名是**exchange-point-and-mark**）组合键，它的作用是互换插入点和文本标记的位置。如果光标移到了应该有文本标记的地方，就说明文本块已经被正确地标记出来了。因为GNU Emacs中的文本标记是不可见的，

所以最好养成一个好习惯：在对文本块做删除操作之前先用“**C-x C-x**”检查一下它的位置。即使是一些已经使用Emacs很多年的人也会忘记检查文本标记，随手一删，不知道会删除哪些东西。（这时候就用得着撤销操作命令“**C-x u**”了。）

要想删除文本块，可以按下“**C-w**”（命令名是**kill-region**）组合键，或者按下**SHIFT-DELETE**组合键，或者从**Edit**菜单里选择执行**Cut**（剪切）操作。

按下：**C-w**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.

--**-Emacs:dickens          (Text Fill)--L1--All----------------------
```

C-w删除了这个文本块。

如果对删除的东西不放心，可以按下“**C-x u**”组合键来撤销这次操作。撤销回来的文本仍带有文本标记；如果愿意，可以再次用“**C-w**”命令删除它。如果想移动文本，可以先给它加上文本标记，按下“**C-w**”删除该文本块；再把光标移到打算插入这段文本的位置，按下“**C-y**”组合键。如果弄错了地方，马上用“**C-x u**”撤销这次操作，再移到正确的位置并按下“**C-y**”组合键即可。

在定义文本块的时候，比较常见的做法是先在该区域的一头设置文本标记，然后再把光标移动到该文本块的另一头。下面介绍几个对大多数情况都很有用的快捷键。假设要标记一个段落。当然可以采用“把文本标记放在段落的一头，再移动到另一头”的方法。可这里有个更简便的方法：按下“**ESC h**”（命令名是**mark-paragraph**）。它会自动地把文本标记放到段落的结束位置，把光标放到段落的开始位置。类似地，“**C-x h**”（命令名是**mark-whole-buffer**）会把整个缓冲区标记为一个文本块，光标在缓冲区的开始位置，文本标记在缓冲区的结束位置。最后，如果处在文本模式里，并已经把“**C-l**”字符定义为页面的分页符，就可以用“**C-x C-p**”组合键快速标记出当前页面。把段落、页面或者缓冲区标记为文本块通常都是后续操作，比如删除操作“**C-w**”等的准备工作。

X 技巧：用鼠标标记文本块

用鼠标标记准备删除、移动或者复制文本块是很直观的。按住鼠标左键并拖动鼠标，就可以选取文本块，被选取的文本将被高亮显示。可一旦松开了鼠标左键，高亮显示区域就会消失；这虽然不会影响对文本块进行的标记工作，但人们都愿意在松开鼠标键之后还能看到文本块被标记出来。如果想在松开鼠标之后还能看到高亮显示的文本块标记，请启用 Emacs 的“临时标记模式（transient mark mode）”。之所以把它称为“临时”标记模式是因为只要对缓冲区做了修改，它就会取消文本标记，好在它带来的不便并不是很大。在 X 窗口系统下，很多人愿意加上这个模式，好让文本块显得醒目一些。输入“**ESC x transient-mark-mode RETURN**”将启用这个模式（注 5）。现在，在文本块的开始处按下鼠标左键，然后按着鼠标左键移动到文本块的结尾并释放，两次动作之间的文本就被标记为一个文本块了。在标记文本的时候，你可以朝屏显画面边界外的方向移动；Emacs 会自动卷屏以便能够对多屏信息进行标记。高亮显示的文本都是已经被标记的。

用鼠标标记文本的另一个方法是：把光标移动到文本块的开始，按鼠标左键并释放，然后把光标移动到文本块的结尾，按鼠标右键一次。如果是在临时标记模式里，刚才选取的文本就都将呈高亮显示状态。如果不在临时标记模式里，现在该干什么了？对，用“**C-x C-x**”来检查文本块是否被正确标记。

如果只想标记一个单词或者一行文本，这里有个简单点的做法：在单词或文本行上多按几次鼠标左键。双击选取一个单词；三击选取一行文本。

文本的复制

文本的复制操作是这样的：先选取一个文本块，按下“**ESC w**”（命令名是**kill-ring-save**）组合键或者在从 **Edit** 菜单里选择 **Copy**（复制）操作；然后把光标移动到准备插入文本的位置，按下组合键“**C-y**”。文本的复制与文本的删除基本相同，只不过复制操作中的 Emacs 不会删除选取的文本。被复制的文本将被放到删除环里去，所以可以用“**C-y**”命令对它做任意次操作。

注 5：　如果打开了临时标记模式，那么不管文本标记是用鼠标还是用键盘命令加上去的，所有的文本块都将呈高亮反显状态。如果想在自己每次进入Emacs的时候都让临时标记模式自动打开，请阅读本章后面“对 Emacs 进行定制”一节里的有关内容。

“**ESC w**”的一个优势是它可以对只读文件或只读缓冲区进行操作。比如，如果想自己创建个关于Emacs使用技巧方面的文件，就可以用“**ESC w**”从在线帮助里复制些文本到某个缓冲区里去。

下面是文本块各种常见操作的基本步骤。

标记一个文本块:

1. 把光标移动到准备标记的文本区域的开始位置。
2. 按下“**C-@**”或“**C-SPACE**”组合键。Emacs会显示“`Mark set`”。
3. 把光标移动到准备标记的文本块的结束位置。
4. 如有必要，按下“**C-x C-x**”互换插入点和文本标记的位置，检查文本块是否已标记正确。

删除一个文本块:

1. 标记好准备删除的文本块。
2. 按下“**C-w**”删除该文本块。Emacs将删除该文本块。

移动文本:

1. 按“标记一个文本块”和“删除一个文本块”中的操作步骤，删除准备移动的文本。
2. 把光标移动到准备插入文本的位置。
3. 按下“**C-y**”组合键。Emacs将把刚删除的文本插入到该位置。

复制文本:

1. 给准备复制的文本做好标记。
2. 按“**ESC w**”复制文本。
3. 把光标移动到准备插入复制文本的位置，按下“**C-y**”组合键。Emacs将把刚复制的文本插入到这里。

恢复早先的删除操作

我们前面对删除环做过简单的介绍，Emacs会把删除的东西保存在这个临时性的存储区域里。我们此前一直假定试图恢复的都是最新被删除的东西。其实，删除环的用处远不止此。事实上，删除环可以把最近30次删除的文本都保存起来。我们已经看到“**C-y**”可以把最新删除的文本恢复回来；继续按下“**ESC y**”组合键，Emacs会用删除环里保存的倒数第二次删除的文本把刚恢复的文本替换掉。

请看下面的例子。假设刚删除的文本是“most recent”。“**C-y**”将把这几个单词从删除环里恢复回来。按下“**ESC y**”时，Emacs会用删除环里的下一个条目（例如“second-last”）替换“most recent”。如下所示：

键盘操作	动作
C-y	This was the most recent deletion.
ESC y	This was the second-last deletion.
ESC y	This was the third-last deletion.
ESC y	This was the fourth-last deletion.

可以继续按下“**ESC y**”组合键把更早先的删除内容依次恢复回来，直到到达删除环的末端（此时将循环回最新删除的文本；这也是它被称为“环”的原因）。

如果觉得保存最近30次删除操作还不够，可以自行加大删除环的尺寸。但很少有人需要扩大删除环，除非是有很特殊的理由或者是个非常有耐心的人。不过，如果真的想试试，可以使用下面这个命令：“**ESC x set-variable RETURN kill-ring-max RETURN** *new-value* **RETURN**”（*new-value* 是一个数字）。

文本块的选取和粘贴

可以通过菜单以更直观的方式访问删除环里的文本：从 **Edit** 菜单里选择执行“**Select and Paste**（选取与粘贴）”操作。选择执行“**Select and Paste**”操作的时候，屏幕上会出现一个列有删除文本的窗口，最近删除的排在最顶部。为了能够尽可能多地显示以前删除的文本，各次删除操作在这个窗口里都只能用一行文本来表示。这就是说，即使删除了一个大文本块，比如500行，在这个窗口里也只能看见它的第一行，后面是省略号。（用户可能需要多熟悉几次才能适应这种安排。）虽然

窗口显示仅为一行，但选取这一行就等于是选取了被删除的整个文本块。选好准备粘贴的文本块之后，弹出窗口消失，选取的文本块就会被粘贴到编辑缓冲区里的光标位置处。

表 2-3 对与文本块有关的编辑命令进行了汇总。

表 2-3: 文本块操作命令速查表

键盘操作	命令名称	动作
C-@ 或 C-SPACE	**set-mark-command**	标记文本块的开始（或结束）位置
C-x C-x	**exchange-point-and-mark**	互换插入点和文本标记的位置
C-w 或 SHIFT-DELETE *Edit→Cut*	**kill-region**	删除文本块
C-y 或 SHIFT-INSERT *Edit→Paste Most Recent*	**yank**	粘贴最近删除或复制的文本
ESC w 或 C-INSERT *Edit→Copy*	**kill-ring-save**	复制文本块（以便用“**C-y**”命令来粘贴它）
ESC h	**mark-paragraph**	标记段落
C-x C-p	**mark-page**	标记页面
C-x h	**mark-whole-buffer**	标记整个缓冲区
ESC y *Edit→Select and Paste*	**yank-pop**	在用过“**C-y**”命令以后粘贴更早删除的文本

段落重排

如果是在自动换行模式里，Emacs 会自动在每行末尾添加一个换行符；这样，每一行的长度都差不多，文件看起来也就比较整齐美观。当然，这种效果只会出现在第一遍草稿中。随着编辑工作的进行，肯定会有一些语句被改得长长短短，整个文件的格式不可能再像当初那样整齐了。经过编辑后，有些文本行会变短，有些文本行会长得超出屏显画面的边界。

Emacs不会主动重排文本。如果想让自己的文件重新恢复当初的整齐效果，必须给出一个勾空（**fill**）命令。重排文本最简单的方法是按下“**ESC q**”组合键以执行**fill-paragraph**命令。Emacs将重排那个段落，然后把光标放到该段落的末尾。

但这里有一个必须引起大家注意的问题。在文本模式里，段落或者必须缩进，或者其前后必须有一个空行。如果没有在文件里给段落前后加上空行，Emacs就会把它整个看做是只有一个段落。如果不小心按下了“**ESC q**”组合键，Emacs就会把文件中的全部文字当做一个段落进行重排，忽略所有的换行符，结果是把它重排为一个很长的段落。如果使用的是**nroff**、**troff**或者TEX等排版系统，这种结果当然不会是你所期望的。幸好还可以用“**C-x u**”（命令名是**undo**）组合键来恢复文件。如果使用**troff**或TEX排版系统，或者（出于种种原因）需要创建段落前后不带空白行的文件，请按下面的建议操作：

- 对**nroff**或**troff**排版系统，请使用Emacs编辑器的nroff模式，该模式将在第九章介绍。这样，就可以用“**ESC q**”组合键来重排段落，而排版命令将不受影响。对TEX用户来说，请使用Emacs编辑器的TEX模式，该模式也在第九章里讨论。这些特殊的模式重新对段落进行了定义，使**fill-paragraph**命令能够正确地执行。除此之外，这两种模式与Emacs编辑器的文本模式是很相似的。
- 不对段落进行重排，而对文本块进行重排。先定义好准备重排的文本块，再按下“**ESC x fill-region RETURN**”或者从**Edit**菜单里选择执行**Fill**操作。这个命令对文本块进行处理，能够把文本块里各个段落的格式都重新排好。

再提醒大家一句：如果在“ESC q”的前面加上了“**C-u**”（或者“**ESC －**”），Emacs会对勾空操作稍微做些变化。普通的勾空命令会使右页边参差不齐。加上这两个前缀中的任何一个之后，勾空命令将会用在文本行里插入额外的空格的方法来调整右页边。不过，平心而论，我们认为这样的居右调整会降低文档的可读性。

表2-4对段落重排方面的编辑命令进行了汇总。

表2-4：段落重排命令速查表

键盘操作	命令名称	动作
ESC q	**fill-paragraph**	重排段落
（无）	**fill-region**	对某个文本块中的段落进行重排
Edit→Fill		

PDG

编辑技巧和快捷键

学会基本的编辑操作——把光标移动到正确的位置、删除/复制/移动文本后，我们再介绍一些简化编辑工作的技巧。

交换位置

最常见的输入错误之一是把前后两个字符的位置打错了，通常会在敲过键盘之后立刻发现这类错误。按下“C-t”组合键可以交换两个字符的位置，把它们修改为正确的顺序，如下所示：

按“C-t”之前	按“C-t”之后
the best of timsc, it	the best of times, it

要想交换两个字符的位置，先要把光标放到准备交换的第二个字符上。按下“**C-t**”组合键。(如果经常弄错两个字符的前后顺序，请考虑用第三章介绍的单词简写模式(word abbreviation mode)来自动改正这类打字错误。)

还可以交换两个单词、文本行、段落或者句子的位置。交换两个单词的位置的方法是：把光标放在两个单词的中间，然后按下“**ESC t**”组合键。Emacs在完成操作后会把光标放到这两个单词的后面。如下所示：

按“ESC t”之前	按“ESC t”之后
one three two	one two three

交换两个文本行的方法是：把光标放在第二个文本行的任意位置上，按下“**C-x C-t**”组合键。Emacs会把第二个文本行放到第一个文本行的前面去。如下所示：

按“C-x C-t”之前	按“C-x C-t”之后
second line first line third line	first line second line third line

表2-5对交换位置类的编辑命令进行了汇总。

表 2-5: 位置交换命令速查表

键盘操作	命令名称	动作
C-t	**transpose-chars**	交换两个字符的位置
ESC t	**transpose-words**	交换两个单词的位置
C-x C-t	**transpose-lines**	交换两个文本行的位置
(无)	**transpose-sentences**	交换两个句子的位置
(无)	**transpose-paragraphs**	交换两个段落的位置

改变字母的大小写

在字母的大小写方面也经常会弄出一些常见而又恼人的输入错误。Emacs 专门为改正字母大小写错误而准备了一些命令。如果想把某个单词的第一个字母改为大写，请把光标放到第一个字母上再按下“**ESC c**”组合键；如果想把一个单词全部改为小写字母，请按下“**ESC l**”组合键；如果想把一个单词全部改为大写字母，请按下“**ESC u**”组合键。这些键盘命令可以这样来记忆：**ESC** 加“**c**”表示单词的首字母要改为大写；加“**l**”表示要把单词的所有字母都改为小写；加“**u**”表示要把单词的所有字母都改为大写。但要注意：如果把光标放在单词的半中间，Emacs 只对从光标位置开始到单词尾之间的字母进行处理；也就是说，可以用“**ESC l**”组合键方便地把单词的前半部分改为小写。

如果发现刚输入的单词有错误，可以在以上几个命令的前面加上一个“**ESC -**”（按 **ESC** 键，再按连字符键）来修改。这种方法不会改变光标的位置。如果光标放在某个单词的半中间，在命令前加上一个“**ESC -**”会只对单词的前半部分（光标位置之前的单词字母）进行改正，光标位置之后的单词字母不受影响。

请看下面的例子，我们从“abcd▌fghij”开始进行操作：

如果你按下	你将得到
ESC u	abcdEFGHIJ▌
ESC - ESC u	ABCD▌fghij
ESC c	abcdEfghij▌
ESC - ESC c	Abcd▌fghij

表 2-6 对与字母大小写有关的编辑命令进行了汇总。

表 2-6: 字母大小写编辑命令速查表

键盘操作	命令名称	动作
ESC c	**capitalize-word**	把单词的首字母改为大写
ESC u	**upcase-word**	把单词的字母全部改为大写
ESC l	**downcase-word**	把单词的字母全部改为小写
ESC – ESC c	**negtive-argument; capitalize-word**	把前一个单词的首字母改为大写
ESC – ESC u	**negtive-argument; upcase-word**	把前一个单词的字母全部改为大写
ESC – ESC l	**negtive-argument; downcase-word**	把前一个单词的字母全部改为小写

文本的改写模式

大家可能习惯了用新输入的字符取代以前的内容而不是删除它们。用这种方法来删除不想要的旧文本确实是很方便的。在 Emacs 里你也可以这样做 —— 只要进入名为改写模式（overwrite mode）的副模式即可。在进入改写模式之后，输入的新文本将取代光标下面的旧字符。如果不在改写模式里（即在普通的Emacs环境里），则输入的新文本将插入到光标位置，而光标后面的旧文本将被推到右边去。（其他软件把这种模式叫做“插入模式”；因为这是 GNU Emacs 的正常工作行为，所以 GNU Emacs 没有给它专门起个名字。）

进入改写模式的办法是按下键盘上的**INSERT**键。状态行上将出现“`Ovwrt`”字样。如果不起作用（或者键盘上没有 INSERT 键），请输入“**ESC x overwrite-mode RETURN**”。再次输入“**ESC x overwrite-mode RETURN**”将退出改写模式。利用Emacs命令的自动补足功能（completion），只要输入“**ESC x ov**”再按下**RETURN**（回车键）即可。这已经足以让 Emacs 知道要切换为改写模式了。自动补足功能是 Emacs 最好的快捷操作方式之一，我们将在第十六章对它进行介绍。

命令的中止和修改的撤销

有时候，用户可能会不小心执行了一个错误的命令，或者当命令执行到中途的时候又不想让它完成了。别担心：在 Emacs 里，既可以在命令执行到中途的时候取消，也可以在它完成后撤销它做的编辑修改。

命令的中止

如果想在命令执行到中途的时候停止它，请按“**C-g**”组合键。命令区里将出现“Quit”字样。

修改的撤销

如果在编辑过程中出现错误怎么办？可以按下“**C-x u**”、“**C-_**”或者“**C-/**”（命令名是**undo**）等组合键，或者从**Edit**菜单里选择执行**Undo**（撤销）操作来撤销刚才的编辑修改。连续按下“**C-x u**”组合将逐步退回到出错之前的位置。虽然**undo**命令的功能很强，但定期存盘永远是最保险的方法。我们通常会在打字过程中停顿的时候把文件存一次盘——即使只停顿几秒钟。请训练手指在准备稍事休息的时候按下“**C-x C-s**”组合键，这是一个值得培养的好习惯。

如果在执行了**undo**命令之后又想再次执行一个命令该怎么办呢？Emacs没有提供“再次执行某个命令”的命令，但可以利用**undo**命令的特性来进行以下操作：随便移动一下光标，然后再次按下“**C-x u**”组合键。Emacs会把过去撤销的命令再执行一遍。可以用这种办法恢复执行前面撤销的命令。

虽然**undo**是一个重要的命令，但如果想撤销一次大规模的修改，那么它的执行速度也会变的很慢。表2-7对3种用来撤销编辑修改的方法，以及它们各自的适用场合进行了总结。

表2-7：撤销编辑修改的3种方法

如果你	请使用以下命令
不喜欢刚做的修改，想一个一个地撤销它们	**C-x u**（命令名是**undo**）
想撤销自上次对文件存盘之后的所有修改	**ESC x revert-buffer RETURN**
想回到该文件以前的版本（即这个文件在开始这次编辑工作之前的样子）	**C-x C-f** *filename*~**RETURN C-x C-w** *filename* **RETURN**

我们已经讨论过使用**undo**命令撤销编辑修改的情况；接下来我们将介绍如何用某个文件，来取代当前编辑缓冲区里的内容，和如何返回到文件的某个早期版本。

用文件内容取代编辑缓冲区中的内容

如果 **undo** 命令不够用，还有其他方法可以把文件恢复到其早期的状态。如果想让文件回到存盘时的样子，请输入“**ESC x revert-buffer RETURN**”或者从 **Files** 菜单里选择执行“**Revert Buffer**（回复缓冲区）”操作。Emacs 会做如下所示的提问：

```
Revert buffer from file filename? (yes or no)
```

其中 filename 是原文件名。如果想重新调入那个文件，回答“**yes**”；如果改变了主意，回答“**no**”。Emacs 会从保存在磁盘上的文件里读出其内容，放到当前编辑缓冲区里去，自你上次存盘之后的一切改动都将不复存在。虽然这个命令的名字是 **revert-buffer**，但请注意它只能对与文件有关联的缓冲区进行恢复。

返回文件某个以前的版本：备份文件

Emacs 会在第一次保存某个文件的时候创建一个备份文件。如果发生了灾难性的意外，而其他用来撤销编辑改动的技巧又帮不上忙的时候，就可以求助于这个备份文件。备份文件的名字与你正在编辑的文件差不多，只是在最后加上了一个波浪符（~）。比如说，如果你正在编辑的文件是 *text*，则其备份文件就是 *text*~。

Emacs 没有提供从备份文件恢复编辑缓冲区的专用命令。最简单的方法是直接编辑备份文件，然后再把它保存为一个正式的文件。假设正工作在一个名为 *text* 的文件上，需要如下操作：先用“**C-x C-c**”组合键退出 Emacs，然后用“**emacs** *text*~”重新启动 Emacs；在备份文件出现在屏幕上以后，用“**C-x C-w** *text* **RETURN**”把它保存为一个正式的文件。Emacs 会在覆盖原始文件之前提问：

```
File text exist; overwrite ? (y or n)
```

输入“**y**”，Emacs 将用备份文件覆盖原始文件。

GNU Emacs 还有一种编号备份功能。如果启用了编号备份功能，Emacs 会在每次对文件进行存盘的时候都创建一个备份文件（后缀是“~*n*~”）。“*n*”是一个数字，每存一次盘，这个数字就加“1”。如果文件的旧版本很重要，就应该考虑使用这项功能，只要愿意，文件的旧版本可以永远保存下去。不过，编号备份功能往往会浪费大量的磁盘空间，一个比较好的折中办法是让 Emacs 只保存最近的 *n* 个版本，这里

的 *n* 是要保存的版本个数。编号备份的控制变量将在附录C里介绍。如果对完备的版本控制有兴趣，可以参阅将在第十五章讨论的版本控制模式（VC mode）。表2-8对与“命令的中止执行和撤销修改”有关的命令进行了汇总。

表2-8: 命令的中止和撤销

键盘操作	命令名称	动作
C-g	**keyboard-quit**	放弃当前命令
C-x u	**advertised-undo**[a]	撤销上一次编辑（可以重复使用）
C-_ 或 **C-/** *Edit→Undo*	**undo**	撤销上一次编辑
（无）	**revert-buffer**	把缓冲区恢复到上次对文件进行存盘(或者自动存盘）时的状态

a. **advertised-undo** 和 **undo** 命令之间并没有真正的不同。它们的工作情况都是一样的。

恢复丢失的编辑修改

我们已经介绍了如何撤销不想要的编辑修改；可找回丢失的修改又是另外一回事了。丢失编辑修改的情况可能会发生在以下几种场合：电源突然断电、计算机突然死机、不小心关掉了计算机的电源等。在非正常退出Emacs的时候，也可能会丢失所做的编辑修改。幸好Emacs想得非常周全，它可以定期把文件保存在“自动保存（auto-save)”文件里。如果用户足够细心，就会发现“`Auto saving`”消息会不时地闪现在辅助输入区里。有了自动保存文件，就可以找回大部分编辑工作——即使不是全部的话。自动保存文件的文件名和正在编辑的文件一样，只是在其前后分别加上了一个井字符（#)。比如，如果正在编辑的文件是 *text*，它的自动保存文件就将是 *#text#*。

从自动保存文件恢复文本的方法是输入“**ESC x recover-file RETURN**”。Emacs会打开一个窗口，窗口里列出了文件和与它对应的自动保存文件以便对存盘时间、文件长度等进行比较；Emacs会提出下面这个问题：

```
Recover auto-save file #text# ?(yes or no)
```

如果想把自动保存文件的内容复制到当前文件里去，回答“**yes**”；如果又改了主意，回答“**no**”。（如果不那么肯定，可以在使用 **recover-file** 命令之前先用“**C-x C-f**”

把自动保存文件*#text#*读到一个缓冲区里并对它做仔细的检查。如果想具体比较这两个版本之间的不同之处。请参考第四章“对比两个窗口中的文件”一节中。)

Emacs会在什么时候创建自动保存文件呢？Emacs会在每敲了几百次键盘之后或者每当电源被切断或Emacs非正常结束时创建一个自动保存文件（注6）。可以用修改**auti-save-interval**变量的方法，来改变Emacs对文件进行自动保存的频率。在默认的情况下，Emacs会每隔300次击键就自动存盘一次。修改变量值的具体操作请参考第十一章。

Emacs的自动保存文件有一个重要的注意事项：如果在文件里进行了一次大规模的删除操作，Emacs将停止自动保存这个文件并显示一条消息通知用户。要想让Emacs再次开始自动保存这个文件，请用“**C-x C-s**”组合键给这个文件存一次盘，或者输入“**ESC 1 ESC x auto-save RETURN**”（命令中的“1”是数字1）。

Emacs的文本编辑命令到这里就介绍得差不多了，它们已经足以完成大部分的编辑工作。我们下面来学习如何对Emacs的某些功能——比如启用自动换行模式（auto-fill mode）——进行设置，这样就不必在每次进入Emacs的时候去启用它们了。下一小节对Emacs的定制情况做了一个简单的介绍；这个问题在第十一章有更详细的讨论。

对 Emacs 进行定制

如果各位是捧着这本书一直读下来的，那么可能已经列出了一个每次进入Emacs时都要去做的工作清单。这些工作可能包括：

- 启用自动换行模式，让 Emacs 去处理字换行问题。
- 如果有 X 窗口系统，启用临时标记模式使 Emacs 高亮显示文本块。
- 如果在流控制方面遇到了问题（比如“**C-s**”会造成死机），一劳永逸地解决它们。

注 6：准确地说，这句话应该是Emacs试图在这些场合这样做，但也有Emacs力所不及的时候。这可没有办法打保票。不过我们可以为Emacs的这个功能投一张信任票。在使用Emacs进行写作的时候，我们从来没有丢失过作品，对其他软件我们可不敢这么说。

我们将介绍怎样才能给Emacs列出一个初始化工作表，让Emacs在用户每次进入的时候自动完成有关的设置工作。这些选项都是在一个名为“*.emacs*”的初始化文件里定义的。初始化文件是一些能够自动运行的文件。有些初始化文件，比如“*.login*”，会在每开始一次UNIX会话时自动运行；而其他的初始化文件，比如“*.emacs*”，会在启动一个与之关联的软件程序时自动运行。总之，“*.emacs*”文件会在启动Emacs的时候自动运行，把该文件里定义的各种选项都设置好。Emacs没有这个文件也能运行；这个初始化文件的惟一作用是让Emacs能够如人所愿地去工作。

我们将在这本书里把你可能想添加到自己的“*.emacs*”文件里去的语句文本都给出来。若想把一行文本添加到你的“*.emacs*”文件里，请按以下步骤操作：

1. 进入Emacs（如果还没有进入）。
2. 输入“**C-x C-f ~/.emacs**”，再按下回车键。
3. 按下“**ESC >**”组合键移动到文件的末尾。
4. 把本书给出的语句准确地输入到文件里去（注7）。
5. 按下“**C-x C-s**”，保存“*.emacs*”文件。
6. 按下“**C-x C-c**”，退出Emacs。
7. 输入“**emacs**”，重新启动Emacs，让新添加的语句起作用。

如果在输入方面出了点小错误（比如漏掉了一个问号或者一个括号），就会在重新启动Emacs的时候看到一条“`Error in initial file`”（初始化文件有错）的出错信息。请重新编辑“*.emacs*”文件，把添加的语句与它们的出处（不管是本书还是其他用户的“*.emacs*”文件）进行对比。只要足够仔细，通常都能把错误找出来；如果自己找不出错误，请向有“*.emacs*”文件（最好还懂LISP语言）的其他人求助。改正错误，保存文件，然后重新启动Emacs。

注7：*.emacs*文件包含LISP函数调用表达式，因此它对语法非常挑剔。（如果“LISP函数调用表达式”使读者感到困惑，这并不意外。只要原样键入，它就会工作。现在无需深究。读完第十三章的详细说明，就会明白。）

自动启用文本模式和自动换行模式

如果想把文本模式设置为默认的主模式，并在进入Emacs的时候自动启用自动换行模式，请把下面这些语句添加到“*.emacs*”文件里：

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

第一条语句告诉Emacs把文本模式设置为默认的主模式；等于在说“如果没有另行说明，请启用文本模式”。第二条语句的作用是只要在文本模式里，就启用自动换行模式。

自动启用临时标记模式

如果想自动启用临时标记模式，请把下面这个语句添加到“*.emacs*”文件里。它的作用是为X用户启用高亮显示功能，使文本块的标记工作更容易进行。

```
(setq-default transient-mark-mode t)
```

这条语句的作用是启用临时标记模式。语句结尾处的“t”表示“true”；“t”的相反取值是“nil”。

解决流控制问题

如果“**C-s**”组合键在系统上会引起死机，就说明它被用做一个流控制字符了。如果把下面这条语句添加到“*.emacs*”文件里，就可以用“**C-**”来代替“**C-s**”，用“**C-^**”来代替“**C-q**”：

```
(enable-flow-control)
```

现在，再输入要求使用“**C-s**”组合键（却引起死机）的命令时，就可以用“**C-**”来代替它。类似地，“**C-q**”（撤销“**C-s**”命令的操作结果）被重新映射到“**C-^**”上。

重新绑定键盘按键

“*.emacs*”文件的另外一种主要用途是对Emacs中干扰工作的东西重新进行定义。这

里讨论的大多数键盘绑定方面的问题在Emacs 19.29都已经得到了解决。如果使用的是一个早期的版本，就可能需要改变这些绑定。即使使用的版本比较新，也可以根据这些例子体现的原则重新绑定其他按键。

一个比较常见而又烦人的错误是当按下**ESC**键的时候，系统会认为按了两次而不是一次。在许多键盘上，如果按键超过一秒钟，它就会自动重复。如果发生了这样的事情，就会撞进**eval-expression**命令，而这个命令通常只有LISP程序员才会用到。在默认的情况下，这个命令是被禁用的，Emacs会显示一个窗口询问是否真的打算使用它。如果用户经常不小心输入“**ESC ESC**”，就会无所适从。另外一个经常出现意外的按键是“**C-x C-u**”，它也有与“**ESC ESC**”同样的麻烦。（我们经常会在重复输入**undo**命令“**C-x u**”时错误地弄成“**C-x C-u**”。）要想让“**ESC ESC**”和“**C-x C-u**”不再显示那些烦人的提示信息，请把下面这些语句添加到“*.emacs*”文件里去：

```
(global-unset-key "\e\e")
(global-unset-key "\C-x\C-u")
```

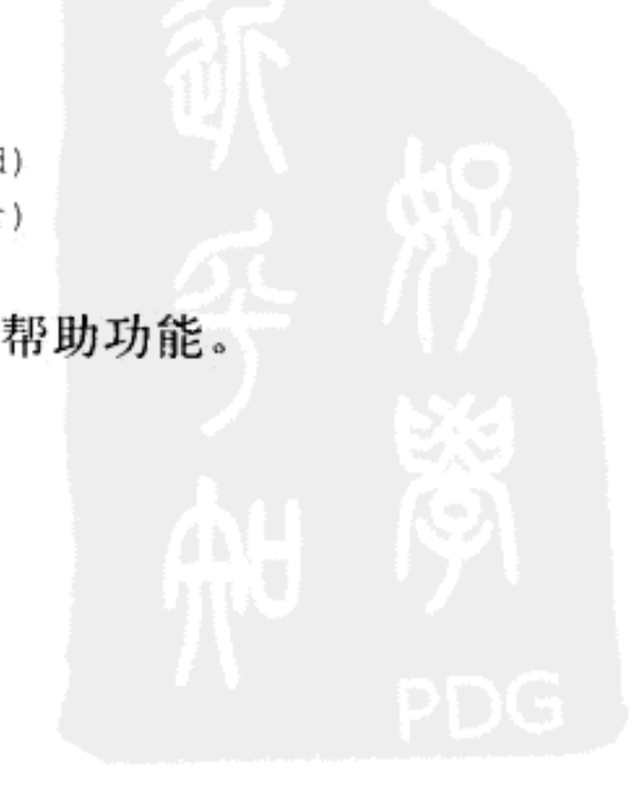
请注意击键动作在“*.emacs*”文件里的表示方法。**ESC**是“**\e**”，所以“**ESC ESC**”就是“**\e\e**”，中间不能有空格。类似地，“**C-x**”和“**C-u**”的前面要加上反斜线（\），“**\C-x**”和“**\C-u**”之间也不能有空格。把这两条语句添加到“*.emacs*”文件里后，如果又错误地按下了这些组合键，就只会听到一声蜂鸣，但工作却不会被打断了。

在线帮助功能方面的问题

另外一个常见的错误是意外地进入了在线帮助功能。进入帮助系统的方法是按下“**C-h**”组合键。但在某些键盘上，“**C-h**”组合键被映射为一个退格字符或者左箭头键，想退格的时候Emacs却认为是在求助。有时候，“**C-h**”又被映射为**DEL**键，致使很难进入帮助功能。把下面这些语句添加到“*.emacs*”文件里可以解决这两个问题：

```
(define-key global-map "\C-x?" 'help-command)
(define-key global-map "\C-h" 'backward-char)
```

做完这些改动之后，你可以用“**C-x ?**”组合键进入帮助功能。



关闭按键提示功能

在Emacs 19.30里，如果某个长命令名有一个对应的组合键，Emacs就会在输入这个命令的时候在辅助输入区里给出一条提示符。设计这个功能的目的是帮助那些初学者，但时间一长，有的用户可能就想关掉这项功能了。请看下面这个例子。在输入“**ESC x find-file**”时请注意会发生什么事情。

输入：**ESC x find-file**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
-----Emacs:dickens          (Fundamental)--L1--All----------------------
You can run the command 'find-file' by typing C-x C-f
```

Emacs会提示这个命令的按键绑定。它会在一两秒钟之后消失，然后开始执行**find-file**命令。

如果不想让Emacs继续提示按键绑定，请把下面这些语句添加到“*.emacs*”文件里：

```
(setq suggest-key-bindings nil)
```

Emacs的定制功能是很强的，用户可以让Emacs完全按照自己的意愿工作。更深入的定制设置将在第十一章讨论。此处这些简短的介绍一是为了引起大家的兴趣，二是为了让大家知道在必要的时候怎样往“*.emacs*”文件里添加简单的语句。

下一章的主题是Emacs提供的各种搜索功能，包括查询－替换、拼写检查、单词简写模式（word abbreviation mode，主要用在自动改正打字错误方面）等。如果你想学习这些功能，请继续阅读下一章。从这里开始，各位可以根据自己的情况有选择地来学习这本书，看哪些东西是自己最想掌握的，不必按部就班地阅读本书的其他章节。

疑难解答

- **出现错误信息**“auto-save error（自动保存错误）”**或**“cannot create backup file（无法创建备份文件）”。这些错误信息往往让人摸不着头脑。它们的真实意思是用户正在一个没有写权限的子目录里进行编辑工作，因此Emacs无法创建一个自动保存文件或备份文件。如果这个子目录不存在，也会显示这条错误信息。Emacs允许创建一个名字像“*foo/bar*”这样的编辑缓冲区，即使这个子目录实际并不存在，也可以开始工作。在Emacs试图自动保存文件之前，一切都会显得很正常。可以把正在编辑的用“**C-x C-w**”文件备份到另外一个子目录里去，也可以用UNIX命令改变这个子目录上的权限。

- **执行移动光标、插入或者删除等操作时，屏幕画面看起来不正常**。比如画面上有反显的小区块、文本行尾部消失等等。终端或显示器的设置有问题。作为一种临时性的解决措施，可以试试重新绘制屏幕命令“**C-l**”（小写的“L”字母）以清除屏幕。然后让系统管理员把配置弄正确。

- **在修改过“*.emacs*”文件后重新启动Emacs时出现了一条**“Error in initial file（初始化文件有错）”**的信息**。重新编辑“*.emacs*”文件，把添加的语句与它们的出处进行对比，看看有没有细微的输入错误。漏掉连字符或者括号这类小错都会引起这个问题。改正错误，保存文件，退出Emacs，再重新进入。

- **在X下，用鼠标左键选取文本的时候，它们没有呈高亮显示状态**。首先，输入“**ESC x transient-mark-mode RETURN**”进入临时标记模式。如果文本仍然不能在选取的时候呈高亮显示状态，那么原因可能是高亮显示颜色和窗口的背景颜色设置相同（Emacs高亮显示文本块，可却无法分辨）。可以按第十四章介绍的方法对“*.Xdefaults*”文件中的显示颜色进行修改。

本章内容:

- 查找操作
- 查找和替换操作
- 拼写检查
- 单词简写模式

第三章

查找和替换操作

前两章里介绍的编辑命令带大家入门已经足够了，但离专业编辑水平的要求还有不小的差距。如果想用Emacs来编辑“宏篇巨作”，就应该掌握本章介绍的高级文档编辑技巧。我们将在这一章介绍在Emacs里查找和替换文本的各种方法。Emacs不仅提供了能在任何一种编辑器里看到的传统的查找、查找/替换功能，还提供了几种重要的变体，其中包括递增查找、正则表达式查找和查询－替换等。Emacs的拼写检查功能也会在这一章加以介绍，因为它也是一种替换操作（查找拼错的单词并用正确的替换之）。最后，我们将把单词简写模式（word abbreviation mode）介绍给大家，它也可以看做是一种自动替换功能，用好了可以节省大量的时间。

查找操作

在做文本编辑工作的时候，经常需要查找一些以前输入的东西。不用采取把文件从头到尾读一遍的办法来查找要找的东西，因为任何一种编辑器都提供有某种形式的查找功能，这类功能能够迅速地从文件里把特定的文本字符串找出来。Emacs也不例外，它有自己的查找命令——事实上，它提供的查找命令极为丰富。下面是对Emacs的各种查找操作的一个简单介绍:

简单查找

给出一个文本字符串（叫做“查找字符串”），Emacs会把该字符串在文件里的下次出现位置找出来。几乎所有的编辑器都有这项功能。

递增查找

Emacs的递增查找操作是这样工作的：只要输入了查找字符串的第一个字符，Emacs就开始进行查找。它会随着敲入的更多字符而继续进行查找。

单词查找

Emacs的单词查找操作与简单查找操作很相似，但前者只找出完整的单词或短语。比如，如果想查找的是单词“*hat*”，就不必担心找到的是单词“*that*”。想查找的短语跨越两个文本行时，单词查找也很有用。

正则表达式查找

正则表达式查找功能可以根据匹配模板，来查找稍有不同但又有一些共性的一组字符串。比如，如果想把文件里的“B1”和“B2”字样都找出来，就可以利用正则表达式“B[12]”来查找。有时候，正则表达式本身可能会相当复杂。我们这一章只对此做一个比较概括的介绍，对它的详细论述请参考第十三章内容。

递增正则表达式查找

把递增查找和正则表达式查找两种操作结合起来，就得到了递增正则表达式查找。

可以向前或者向后进行查找。查找既可以区分字母的大小写，也可以不区分字母的大小写。前者表示Emacs将认为大写字母和小写字母不同（比如，把单词“*This*”和“*this*”看做是不同的字符串）；后者表示Emacs不把大写单词和小写单词区别对待（即把“*This*”和“*this*”看做是一样的字符串）。在默认的情况下，Emacs的查找操作是不区分大小写的，也就是说，它认为大写字母和小写字母没有区别。一个例外：如果输入的单词里有一个以上的大写字母，Emacs就会按区分大小写的情况来对待整个查找字符串。因为Emacs会认为既然你多花了工夫去输入一个以上的大写字母，就应该是想查找与之精确匹配的东西。

替换操作与查找操作是紧密相关的。类似于查找操作的情况，Emacs也已经准备了好几种替换操作，其中包括：

简单查找与替换

在这个操作里，Emacs将把文件里的一个字符串用另外一个字符串全部替换。但这往往不会是我们想要的结果，可以用查询－替换操作来有选择地进行替换。

查询－替换

在查询－替换操作中，Emacs 会有条件地把文件中的每个字符串替换掉。Emacs把查找字符串在文件里的出现位置都找出来，每找到一个，就会询问是否要进行替换。如果只打算部分地替换文件里的某个单词或短语而不是其全部，那么这种替换操作最有用。

正则表达式替换

正则表达式替换利用UNIX强大的“正则表达式”功能（请参考本章后面的内容）来找出替换字符串。

大家可以根据自己的需要有选择地学习这一章。千万不要被Emacs众多的查找操作吓倒。在实际工作中，可能用一个查找命令和一个替换命令就能应付百分之九十九的工作。比如，在编写本书的时候，大部分工作都是用递增查找和查询替换完成的；如果是作家，则单词查找应该最能满足需要；而如果是一位程序员，正则表达式查找就将是最得心应手的工具。如果刚开始学习Emacs，那么最好先掌握递增查找，再继续学习本章的其他内容。总之，只有在知道都有哪些查找命令可用的情况下，才会在必要时想起它们来。

递增查找

递增查找从输入查找字符串的第一个字符起就开始行动了。对初学者而言，它并不是最容易使用的查找操作，可它有许多优点。很多用户都比较喜欢递增查找的效率；但如果在打字时经常出错的话，递增查找反而会影响工作和心情。如果真是这样，那么也许简单查找更适合一些。可话又说回来，Emacs的递增查找确实有很多优点，从纯粹的使用角度讲也最简便，所以我们将从它开始来展开本章的讨论。

递增查找的操作方法是这样的：先敲入“**C-s**”组合键，再输入想查找的文本。Emacs将临时性地进入Isearch模式（知不知道这一点其实无关紧要，但我们认为还是告诉大家比较好）。请看递增查找操作的工作情况：Emacs会根据输入的每一个字符在文件里进行查找。假设想查找的单词是“*meter*”。在递增查找操作中，敲入字母“*m*”的时候，Emacs会把光标位置以后的第一个“*m*”找出来；再输入字母“*e*”的时候又会把光标位置以后的第一个“*me*”找出来；当你再输入字母“*t*”的时候又会把光标位置以后的第一个“*met*”找出来；以此类推。最终，或者找到了想要找的东西，或者什么也没找到。如果找到了想要的东西，按下回车键退出这次查找，光标将停

留在文件的当前位置上。如果Emacs没有找到能够与查找字符串相匹配的东西，它就会在屏幕画面的底部显示一条“Search failed”（查找失败）信息，并蜂鸣报警。

下面是查找单词“*meter*”的情况；图中的数字指明了每在查找字符串里输入一个新字母后光标将会到达的位置。

输入：**C-s meter RETURN**

```
 Buffers Files Tools Edit Search Help
made
 1
means
  2
method
    3
meteor
     4
meter
      5
 -----Emacs:mwords          (Text Fill Isearch)--L9--All---------------
I-search:meter
```

在这次递增查找操作中，Emacs会随着输入查找字符串“*meter*”而把光标依次移动到位置1、2、3等等。另外，请注意状态条上出现的“I-search”字样。

如果找到了正在查找的字符串，可它的位置却不是你所希望的，该怎么办呢？比方，想查找的单词是“*eschatology*”并已经找到了一个，如果它不是想找的那个，再次按下“**C-s**”组合键，使Emacs继续查找下一个出现“*eschatology*”的地方。Emacs会用当前查找字符串去进行查找，用不着重新输入。

找到想找的文本后，千万要记得按下回车键。忘记结束查找操作（即按下回车键或者输入其他光标移动命令）是一个很常见的错误——输入一些东西后，Emacs一下子跑到文件中的其他地方去了。出现这种情况的原因是Emacs认为还在进行查找，它会把刚才输入的字符看做查找字符串的一部分。

如果在输入查找字符串时打错了字，递增查找往往会让人心烦意乱。如果真的在查找字符串里错误地输入了另外一个字符，可以用**DEL**键加以改正；Emacs会根据改正后的字符串返回到刚才的位置。如果继续按动**DEL**键来删除查找字符串里的字符，Emacs会依次返回到以前的匹配位置。

取消一次查找操作（即放弃这次查找操作）的办法是按下“**C-g**”组合键。这个命令会回到开始查找操作之前的地方去。

注意：当Emacs正在进行查找操作的时候，“**C-g**”组合键的含义会有点小变化。这一点很重要，特别是当需要在Emacs找到它正在查找的东西之前，取消这次查找操作的时候更要注意。按一次“**C-g**”组合键只会回到Emacs刚才找到的地方，光标将在最后一个字符上；而这次查找操作仍未取消。按两次“**C-g**”组合键才能彻底取消这次查找操作，回到最初开始查找操作的地方去。

如果想向后（即向文件头方向）进行查找，请使用“**C-r**”组合键。它除了查找方向相反之外，与“**C-s**”完全一样，它会把光标放在找到的文本的开始。就像重复按下“**C-s**”组合键时的情况一样，也可以再次按下“**C-r**”组合键继续查找下一个出现查找字符串的地方而不用再次输入查找字符串。

如果不想输入查找字符串，那么可以把编辑缓冲区里的文本复制到查找字符串里去。用“**C-s C-w**”组合键（可以把“**C-s C-w**”看做“复制单词”命令）可以把从光标位置到下一个标点符号或空格符之间的文本都复制到查找字符串里去。如果想把从光标位置到行尾之间的文本都复制到查找字符串里，就要使用“**C-s C-y**”组合键。用这种办法得到的查找字符串会把原文中的大写字母都转换为小写，这就保证了查找操作不区分字母的大小写情况。用“**C-s ESC y**”组合键可以把删除环里的文本复制到查找字符串去；而在用过这条命令之后，还可以用“**ESC p**”组合键来查看删除环里的上一个条目。如果用“**ESC n**”跑过了头，可以再用“**ESC p**”倒退回来。

在递增查找操作开始之后，有些按键（比如回车键**RETURN**和退格键**DEL**）的功能与平时有所相同。这听起来虽然很费解，可习惯起来却很快。表3-1对递增查找操作中特殊按键的功能进行了总结。

表3-1：递增查找命令速查表

键盘操作	命令名称	动作
C-s	**isearch-forward**	向前（朝文件尾方向）开始递增查找操作；后面是查找字符串。另外，（向前）查找下一个出现查找字符串的地方

表 3-1：递增查找命令速查表（续）

键盘操作	命令名称	动作
C-r	**isearch-backward**	向后（朝文件头方向）开始递增查找操作；后面是查找字符串。另外，（向后）查找下一个出现查找字符串的地方
RETURN	（无）	退出查找操作
C-g	**keyboard-quit**	取消递增查找操作（你可能需要连按它两次）
DEL	（无）	删除查找字符串中的字符
C-s C-w	（无）	开始递增查找操作；把光标位置处的单词用做查找字符串
C-s C-y	（无）	开始递增查找操作；把光标位置到行尾之间的文本用做查找字符串
C-s ESC y	（无）	开始递增查找操作；把删除环中的文本用做查找字符串
C-s C-s	（无）	重复刚才的查找操作

简单查找

如果在输入查找字符串的时候没有出现打字错误，那递增查找的效率还是很高的。可要是打错了字，就不得不改正打错的字母后才能继续操作。如果总是出这样的错，你大概就不会喜欢用递增查找了。Emacs 中的简单查找（或者叫非递增查找）可能更适合你的口味。向前（朝文件尾方向）执行的简单查找的操作方法是：先按下“**C-s RETURN**”或者从“**Search**（查找）”菜单里选择执行“**Search**”操作，然后输入查找字符串，再按下回车键；Emacs就开始查找了。再次按下“**C-s**”将让Emacs去继续查找下一个。向后（朝文件头方向）执行的简单查找的操作方法是：先按下“**C-r RETURN**”或者从“**Search**”菜单里选择执行“**Search Backwards**（向后查找）”操作，然后输入查找字符串，再按下回车键。表 3-2 对简单查找的有关命令进行了汇总。

表 3-2: 简单查找命令速查表

键盘操作	动作
C-s RETURN *searchstring* **RETURN** *Search→Search*	向前（朝文件尾方向）开始一次非递增查找操作
C-s	向前查找下一个
C-r RETURN *searchstring* **RETURN** *Search→Search Backwards*	向后（朝文件头方向）开始一次非递增查找操作
C-r	向后查找下一个

单词查找

如果在查找一个短语的时候，明明文件里有这样的短语，可用递增查找却找不出来，那请试试单词查找。（用递增查找操作找不到这个短语的原因，极可能是这个短语跨越了两个文本行；也就是说，它中间多了一个换行符。）单词查找是一种非递增性的查找，它不会受换行符、空格和标点符号的影响，但要求查找字符串必须与文件里的单词完整地匹配。

单词查找的操作方法是：按下“**C-s RETURN C-w**”（命令名是 **word-search-forward**）组合键，辅助输入区里会出现“Word search”（单词查找）字样（不要被它前面的提示信息所迷惑：按下“**C-s**”后会先出现一个“I-search”，按下回车键后又会出现一个“search”提示符，别管它们。）；然后输入查找字符串，再按下回车键。Emacs 开始查找给定的字符串。如果想向后（朝文件头方向）进行单词查找，按下“**C-r RETURN C-w**”。举个例子，假设对下面这段文本进行编辑，并且光标放在文本块的第一个字符上：

```
He said, "All good elephants are wise, aren't they?"
She answered, "Some are smarter than others, but we
think this is socially conditioned."
```

则使用“**C-s RETURN C-w they she RETURN**”命令，将把光标放到单词“*She*”的后面。虽然这条命令看起来有点繁琐，可它的意思还是很清楚的：这是一条用来查找（“**C-s RETURN C-w**”）单词“*they*”的单词查找命令，“*they*”的后面必须跟有单词“*she*”，而“*they*”和“*she*”之间的标点符号（?"）和换行符不会影响查找结果。

假设想查找的单词是“*the*”，并且不想撞到*thence*、*there*、*theater*、*thesis*和*blithe*或者碰巧包含字母“*the*”的其他单词。这种情况用递增查找和简单查找都不太好对付——最佳办法是采用单词查找。如果撰写一篇论文，那么单词查找往往会是最需要的方法。就这三种基本的查找操作而言，单词查找是惟一能够在将要查找的短语跨越两个文本行的情况下、找到该短语的查找操作。

在学完这三种最常用的查找操作之后，可以多做些练习，看哪一种最能满足自己的工作需要。

查找和替换操作

在工作中，往往需要把一个查找操作和一个替换操作结合起来使用。比如，如果拼写能力比较弱，就可能出现需要把文件中所有的“*recieve*”都替换为“*receive*”的情况。最笨的办法是先查找“*recieve*”，每找到一个就手动改正。Emacs准备了一个查找和替换操作，用一条命令就可以把所有的“*recieve*”都改正过来。

基本查找与替换操作

假设正好遇上我们刚讲的情况——想把文件里的某个字符串都替换为另一个字符串。我们不敢肯定是不是把所有的“*receive*”都错拼为“*recieve*”了，但肯定得把拼错的地方都改过来。当想把文件里的某个字符串都替换掉的时候，只需用一条简单的命令就能让Emacs干好这件事——先输入“**ESC x replace-string RETURN**”，再输入查找字符串，按下回车键；然后输入替换字符串，再次按下回车键。Emacs将从光标位置开始，把文件里所有出现查找字符串的地方都替换掉。如果想对整个文件进行查找和替换，可以在使用这个命令之前先用“**ESC <**”组合键或**HOME**键移动到文件的开头。请看下面**replace-string**命令的用法示例。

初始状态：

```
Buffers Files Tools Edit Search Help
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks,provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo          (Text Fill)--L4--All----------------------
```

在这个画面里，“*receive*”被错拼为“*recieve*”三次，但光标位置在第一个被拼错的单词后面。

现在来改正它们。

输入：**ESC x replace-string RETURN recieve RETURN receive RETURN**

```
Buffers Files Tools Edit Search Help
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve received it, we'll begin work on filling the order.
You should receive the merchandise in about six weeks,provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L5--All----------------------
Replaced 2 occurrences
```

只有光标位置后面的拼写错误被改正过来了，第一个句子里的单词“*recieve*”仍然是不正确的。注意**replace-string**命令是如何把“*recieve*”改正为“*receive*”的。

如果对字母的大小写情况没有把握怎么办？单词“*recieve*”可能会出现在句子的开始，也就是说，它的第一个字母可能是大写的。是不是要分别进行两次查找呢？没有必要。Emacs在进行替换操作时能够把字母大小写方面的问题处理好。

查询－替换操作

有时候，可能不需要把出现查找字符串的每一个地方都替换过来；如果文件的长度超过五行，对整个文件进行全局性的替换就有些不够慎重。如果想一个一个地决定是否需要进行替换，可以使用查询－替换操作，以便能够有选择地对文件里的某个字符串进行改正。Emacs找到一个出现查找字符串的地方后，会询问是否进行改正，可以根据具体情况做出回答。

查询－替换的操作方法是：按下“**ESC %**”组合键或者从“**Search**”菜单里选择执行“**Query Replace**（查询－替换）”操作。辅助输入区里会提示“Query replace:”。输入查找字符串并按下回车键。接下来将出现提示信息

```
Query replace searchstring with:
```

输入替换字符串并按下回车键。到目前为止，情况与简单查找和替换操作差不多；只是提示符不同。

现在，Emacs开始搜寻查找字符串的第一个出现位置。找到之后，将出现一个新的提示符，如下所示：

```
Query replacing searchstring with newstring
```

Emacs会停下来等待指示，然后根据回答来决定是否需要进行替换。表3-3列出了可能的回答及其结果。

表3-3：查询－替换操作中的响应

键盘操作	动作
ESC % *Search→Query Replace*	开始查询－替换操作
SPACE 或 **y**	用新字符串替换查询字符串searchstring，然后前进到下一个位置
DEL 或 **n**	不替换；前进到下一个位置
.	在当前位置做替换后退出查询－替换操作
,	替换并显示替换情况（再按空格键或“**y**”后才移动到下一个位置）
!	对后面的文件内容全部进行替换，不再提问是否要进行替换
^	返回上一次进行了替换的位置
RETURN 或 **q**	退出查询－替换操作
C-r	进入递归编辑状态（马上就要讲到）
C-w	删除此处内容并进入递归编辑状态（好做其他修改）
ESC C-c	退出递归编辑状态，继续完成查询－替换操作
C-]	退出递归编辑状态和查询－替换操作

看起来好像有很多键盘命令需要记忆，其实只要知道了其中的两三个就已经够用了。在大多数情况下，可以按空格键告诉Emacs进行替换并进到下一个位置，或者按“**n**”键跳过这个位置进到下一个地方。如果对替换后的情况没有把握，按一次逗号（,）

键；Emacs完成此处的替换但不再继续前进，再按一次空格键它才会往下进行。在做了几次之后，可能发现没有必要再挨个查看每一处修改；此时可以按下惊叹号(!)键让Emacs把以后的替换都做完，用不着再询问是否需要进行替换了。只要记住了这几个按键，一切就尽在掌握之中。

为了让大家有一个真实的感受，我们再来看看刚才的例子。还是假设“*receive*”被拼错了。

输入：**ESC % recieve RETURN receive RETURN**

```
Buffers Files Tools Edit Search Help
Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.

-----Emacs:ordermemo        (Text Fill)--L4--All----------------------
Query replacing recieve with receive:(? for help)
```

做完替换第一个单词的准备工作，按空格键继续。

按下：**SPACE**

```
Buffers Files Tools Edit Search Help
Now that weve received it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L5--All----------------------
Query replacing recieve with receive:(? for help)
```

按下空格键之后，Emacs替换了第一个单词，查询－替换操作进到第二个单词。

重复上述过程直到到达文件的末尾。我们已经说过，按下惊叹号(!)键将把文件其余部分的有关位置都改正过来。

根据表3-3，大家可能已经发现有几个键(比如空格键)在查询－替换操作过程中有着特殊的含义。在实际工作中，把这些按键用做其他用途不会造成混乱，但在书里写起来却有点怪怪的。为了掌握好查询－替换操作各种响应的使用方法，可能需要找个文件实际操练一下。

查询－替换（以及其他复杂命令）的重复执行

在学完查询－替换操作的基本用法之后，我们再介绍一种快捷操作方式。这种快捷操作方式不仅能够用在查询－替换操作里，还可以用在Emacs几乎所有的地方：复杂命令的重复执行（前后两次操作可能会稍有不同）。有时候，会不小心退出了查询－替换操作，或者又想对替换字符串再稍加修改。就拿刚才的例子来说，对文件进行的查询－替换操作并没有从头开始。用不用再把整个命令重新输入一遍呢？不用。只要把光标移到文件的开始再按下“**C-x ESC ESC**”组合键就行了。最后一次输入的复杂命令将被调出来。如果它不是想要的，可以再用“**ESC p**”组合键调出它前面的那个命令（随便按多少次都行；如果过了头，可以用“**ESC n**”后退一条命令）。举个例子，把光标移到文件的开头，然后重复执行一次刚才进行的查询－替换操作。

按下：**ESC <**，然后再按下 **C-x ESC ESC**

```
Buffers Files Tools Edit Search Minibuf Help
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve received it, we'll begin work on filling the order.
You should receive the merchandise in about six weeks,provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L1--All----------------------
Redo:(query-replace "recieve" "receive" nil)
```

先用“**ESC <**”组合键移动到文件的开头；然后按下“**C-x ESC ESC**”组合键，最后一条复杂命令被调了出来。

因为它就是想要的命令，所以用不着再按“**ESC p**”组合键查看前一条命令了。如果愿意，还可以在按下回车键之前对查询－替换字符串做些改动。

按下：**RETURN**

```
Buffers Files Tools Edit Search Help
We were very anxious to receive your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L1--All----------------------
Query replacing recieve with receive:
```

按下回车键将再次执行这条命令。

刚才已经讲过，“**C-x ESC ESC**”这种快捷操作方式对在辅助输入区里有补充性输入内容的一切命令都有效，绝不仅限于查询－替换命令。只不过在查询－替换操作里用得最多而已（它还很适合用来重复执行键盘宏命令，我们将在第十章讨论这方面的内容。）。

递归编辑

在进行查询－替换操作的时候，几乎不可避免地会看到还有其他一些地方需要修改。不信就试试——肯定没错！第一反应是记下那些错误，等完成查询－替换操作之后再来修改它们；第二反应是着急上火——等完成替换操作后才发现自己已经想不起来刚才看到哪儿还出了错。

幸好Emacs已经想到这一点了。它允许在一次查询－替换操作的过程中再开始一次递归编辑。“开始一次递归编辑”的意思是，暂时放下手里的查询－替换操作去任意做一些其他的修改。当退出递归编辑状态的时候，查询－替换操作将从刚才暂停的位置开始继续进行。

要想在查询－替换操作的过程中开始一次递归编辑，需要按下“**C-r**”组合键。（注意：类似于许多其他的组合键，“**C-r**”在查询－替换操作中的含义与它在正常Emacs编辑活动中的含义是不同的。）进入递归编辑状态的时候，状态条上将会多出一对方括号（[ ]）。再以订单备忘录为例对此做个说明。已经用查询－替换操作找到了第一个“*recieve*”并正准备按下空格键来改正它。就在这时，发现没有把“*we've*”里面的上撇号打上去。让递归编辑来帮忙吧。

输入：**C-r**

```
Buffers Files Tools Edit Search Help
Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo        [(Text Fill)]--L4--All----------------------
```

在进入递归编辑状态的时候，请注意“(Text Fill)”两端的方括号。

做你想做的编辑工作吧——现在进入一个类似于Emacs标准编辑状态的编辑模式。把光标向回移动几个字符并改正“*we've*”。可以重新开始查询－替换操作时，按下

"**ESC C-c**"组合键（这个组合键的输入方法是：按下再释放**ESC**键，然后按"**C-c**"）。这个命令让Emacs退出递归编辑，并重新开始继续执行查询－替换操作。Emacs将回到刚才离开的位置；可以像什么事情也没发生过那样继续进行替换操作。

给"weve"中间加一个上撇号使之变成"*we've*"，然后按"**ESC C-c**"组合键

```
Buffers Files Tools Edit Search Help
Now that we've recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L4--All----------------------
Query replacing recieve with receive:(?for help)
```

递归编辑的工作完成后，返回查询－替换操作。按空格键修改"*recieve*"。

如果想在退出递归编辑的同时也取消查询－替换操作，可以按下"**C-]**"（命令名是**abort-recursive-edit**）或"**ESC x top-level RETURN**"而不是"**ESC C-c**"。

Emacs高级用户请注意：可以随时（不仅仅是在查询－替换操作的过程中）开始一次递归编辑。"**ESC x recursive-edit RETURN**"命令会进入递归编辑状态；而"**ESC C-c**"会退出递归编辑状态，返回到此前的工作中去。甚至可以在递归编辑里再来一次递归编辑，不过每多增加一个递归层次，把自己弄糊涂的可能性也将增大一层。

找操作中的字母大小写问题

在默认的情况下，Emacs的查找操作是不区分大小写字母的。在对单词"*random*"进行查找的时候，"*random*"、"*Random*"、"*RANDOM*"以及古怪的"*RanDoM*"、"*rANdOM*"等单词都会被找出来。在进行替换的时候，Emacs会根据被替换单词中的字母大小写情况做相应的替换。如果用"*tandem*"来替换"*random*"，那"*Random*"将被替换为"*Tandem*"，而"*RANDOM*"将被替换为"*TANDEM*"。

也就是说，默认的查找和替换操作差不多完全能够满足要求：它们在不考虑字母大小写情况的前提下找到一个查找字符串，然后根据其上下文来调整替换字符串中字母的大小写情况。但有时可能需要更精确的控制。变量**case-fold-search**决定查找操作是否要区分字母的大小写情况。它对各种查找操作——包括递增查找、单词查找、查找-替换操作中的查找等等都有影响。在默认的情况下，**case-fold-search**

的值被设置为“**t**”，意思是“如果用户输入的字符串不是大小写混杂或全都是大写字母，就不区分大小写字母”。这种默认设置通常也是需要的。但如果真的需要在查找操作中区分大小写字母，就要用“**ESC x set-variable RETURN**”命令把**case-fold-search**的值设置为“**nil**”。Emacs会提示输入一个变量名，输入“**case-fold-search RETURN**”。Emacs再提示输入一个新值，输入“**nil RETURN**”。

类似地，如果不想对替换字符串中的字母大小写情况进行调整，可以修改变量**case-replace**的值。这个变量的缺省值也是“**t**”（代表“true”），意思是“根据原始文本调整替换字符串中的字母大小写情况”——如果原单词的首字母是大写的，替换字符串的首字母也要大写等等。把这个变量的值设置为“**nil**”意味着“不要对替换字符串中的字母大小写情况进行调整，就按输入进行替换”。如果想修改变量**case-replace**的值，请按刚才介绍的步骤来调用**set-variable**命令。我们认为把调整替换字符串字母大小写情况设置为“on”是有用的，特别在使用的查找是不区分大小写字母的时候更有必要。但大家应该在这个问题上花点时间练练手，这样才能真正体会其中的奥妙。

set-variable命令只能临时性地改变一下Emacs的行为。只要启动了一次新的Emacs会话，甚至只要在同一次Emacs会话里开始编辑另一个文件，就将回到默认的行为。这可能正是需要的，因为分别对首字母大写的单词和小写单词进行查找是不方便的。如果把下面这些语句添加到“*.emacs*”文件里，就可以永久性地改变这些变量的设置值：

```
(setq-default case-fold-search nil)   ; 要求精确匹配
(setq-default case-replace nil)       ; 替换时不改变大小写情况
```

要想让这些语句起作用，需保存“*.emacs*”文件，再重新启动Emacs。

查找与替换操作中的正则表达式

前面介绍的查找、查找和替换操作虽然不算很复杂，但用来完成日常工作基本上够用了。但有些场合单靠这些比较简单的操作是无法应付的。正则表达式是UNIX操作系统一项基础性功能，并且已经在Emacs里得到了实现。正则表达式是程序员或UNIX黑客们手里的法宝，它能够利用含有各种通配符的字符串进行复杂的查找。对正则表达式细致全面的讨论请参考第十三章；本小节的目的是把正则表达式在查找操作中的基本用法介绍给大家，让各位对利用正则表达式进行文本查找的基础性知

识有一个了解。如果想对正则表达式做深入的研究，请参考O'Reilly出版公司出版的《sed & awk》，这本书的作者是Dale Dougherty和Arnold Robbins。

表3-4列出了一些可以用在正则表达式里的字符。

表3-4：用来建立正则表达式的字符

字符	匹配情况
^	匹配行首
$	匹配行尾
.	匹配任意单个字符（类似于文件名中的问号？）
.*	匹配任意（零或以上）个字符（这是一个真正的通配符，类似于文件名中的星号*）
\<	匹配单词的开头
\>	匹配单词的结尾
[]	匹配方括号中的任何一个字符；比如“[a-z]”将匹配任意一个字母表字符

如果用“**^word$**”做一次正则表达式查找，那么找到的将是独占一个文本行的单词“*word*”。字符“**^**”的含义是字母“**w**”必须是文本行的第一个字符，而字符“**$**”的含义是字母“**d**”必须是文本行的最后一个字符。

如果想把前有“*beg*”、后有字母“*s*”的单词都找出来，就应该选用“beg[a-z]*s”做为正则表达式。它将把单词“*begins*”、“*begets*”、“*begonias*”以及“*shibegrees*”、“*altbegaslia*”等不常用的单词都找出来。如果不想要那些不常用的单词——即如果只想找出以“*beg*”开头、以字母“*s*”结尾的单词，就要用“\<beg[a-z]*s\>”做为正则表达式。特殊字符组合“\<”的作用是匹配一个单词的开头；而字符组合“\>”的作用是匹配单词的结尾。如果只想找出“*beg*”、“*big*”、和“*bag*”等以一个字母“*b*”开头、中间有任意一个字母、再以一个字母“*g*”结束的单词，但不想找到“*begonias*”之类的单词，或者其他含有“*beg*”等字符串的古怪单词，就要用“\<b[a-z]g\>”做为正则表达式。

如果想对“^”、“$”、“.”、“*”、“[”、“]”或其他特殊字符进行查找，显然不能直接使用这些字符本身。必须在它们的前面加上一个反斜线（\）。也就是说，如果想查找一个英文句号，就必须查找“\.”。举个例子，如果想查找下面这个电子邮件地址：

```
howie@mcds.com
```

正则表达式就将是:

```
howie@mcds\.com
```

可以把正则表达式用在递增查找和查询－替换操作中。表3-5列出了正则表达式查找操作所使用的各种命令。除初始命令稍有不同以外，这类查找与本章前面介绍的查找操作都是一样的。

表3-5: 正则表达式查找命令速查表

键盘操作	命令名称	动作
ESC C-s RETURN *Search→Regexp Search*	**re-search-forward**	向前（朝文件尾方向）查找一个正则表达式
ESC C-r RETURN *Search→* *Regexp Search Backwards*	**re-search-backward**	向后（朝文件头方向）查找一个正则表达式
ESC C-s	**isearch-forward-regexp**	向前（朝文件尾方向）递增查找一个递增正则表达式
ESC C-r	**isearch-backward-regexp**	向后（朝文件头方向）递增查找一个递增正则表达式
（无） *Search→* *Query Replace Regexp*	**query-replace-regexp**	查询－替换一个正则表达式
（无）	**replace-regexp**	无条件地对一个正则表达式做全局性替换（谨慎使用）

拼写检查

Emacs中的拼写检查可以采用两种方法：使用Ispell或者使用UNIX操作系统的拼写检查器**spell**。就我们个人的观点而言，Ispell是一种更好的选择，所以我们的讨论将从它开始。

在开始之前，我们先讲点关于拼写检查器的故事。拼写检查器都有一个词汇表，里面是它们能够认出来的单词，人们把这类词汇表叫做“字典”。如果单词没有出现在

字典里，拼写检查器也就无法识别它 —— 它会认为这个单词拼错了。有些单词虽然拼写检查器不认识，可单词本身却是正确的（比如人名或者专业技术术语等）；将要介绍的两种拼写检查器都允许跳过这类单词，并且还允许把这些“额外的”单词插到个人字典里，这样，当拼写检查器下次遇见它们的时候就能认出它们来了。当然，拼写检查器绝对无法取代人工校对；比如，它们根本发现不了把“*two*”或“*too*”错写为“*to*”这样的错误。尽管拼写检查器存在着各种不足，我们仍然认为它是现代社会最伟大的发明之一。请跟我们一起来学习它在 Emacs 中的使用方法吧。

Ispell 的使用方法

Ispell提供了对一个编辑缓冲区、一个文本块、一条电子邮件消息或者一个单词进行拼写检查的操作选项（注 1）。但在输入命令告诉 Ispell 去检查哪个区域之后，这些操作选项的工作情况都将是一样的。以 **ispell-buffer** 命令为例进行讨论。如果编辑缓冲区里的单词都拼写对了，Ispell将给出一条“`Spell-checking done.`”的消息。如果 Ispell 找到了一个拼错了的单词，就会看到类似与下面这样的画面：

输入：**ESC x ispell-buffer RETURN**

```
Buffers Files Tools Edit Search Help
(0)relieve (1)receive
- *Choices* - word: recieve
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs: ordermemo        (Text Fill)--L1--All----------------------
C-h or ?for more options; SPC to leave unchanged, Character to replace
word
```

Ispell 移动到它不认识的第一个单词。

注 1： 如果 Ispell 能够正常工作，它的这些操作选项还是很容易使用的。先去试试看，如果真的有问题，再来看这个脚注不迟。Emacs 要求使用 Ispell Version 3.1.x。如果安装的是 Ispell Version 4，它就不能工作。在 Emacs 19 某些早期的版本里，我们就遇到过 Emacs-Ispell 接口方面的问题。所以，即使真的遇到了问题，你也不是孤独的。在本章最后一个小节里，我们给出了一些解决这类问题的建议。

辅助输入区里出现“C-h or ? for more options”的字样。因为这是第一次使用Ispell，所以按下“**?**”来看看它还有哪些操作选项：

输入：**?**

```
Buffers Files Tools Edit Search Minibuf Help
(0)relieve (1)receive
- *Choices* - word: recieve
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo        (Text Fill)--L1--All----------------------
[r/R]eplace word; [a/A]ccept for this session; [i]nsert into private
dictionary [l]ook a word up in alternate dictionary; e[x/X]it;
[q]uit session [u]ncapitalized insert into dictionary.Type
```

Ispell给出一个操作选项清单。

如果Ispell列在屏幕顶部的某个单词就是想要的，按下对应的数字键，Emacs将进行替换。在这个例子里，请按下“**1**”。如果想自己修改那个单词，请按下“**r**”，Emacs会在输入了“正确的”单词之后再检查一遍。如果按下的是“**R**”，Ispell将开始一次查询－替换操作，以便把编辑缓冲区里所有同样的拼写错误都改正过来。

除了对“错误的”单词进行替换之外，可能还想让Ispell跳过它。按下空格键使Ispell跳过此处拼错了的单词。如果想让Ispell在检查编辑缓冲区余下的内容时，把某个拼错了的单词都跳过去，请按下“**a**”（表示接受）。按下大写的“**A**”有另外一层含义：它告诉Ispell在这次编辑工作里接受那个单词，但只有在这个编辑缓冲区里才有效。

如果再次遇到了这个单词，而它确实拼写正确，那么可以按下“**i**”让Ispell把它插到个人字典里；这个字典保存在你主目录中一个名为“*ispell.words*”的文件里。如果这个文件不存在，Ispell会自动创建它。如果想让Ispell把单词以小写形式插入到个人字典里，按下“**u**”，Ispell会把单词转换为小写形式后再放到个人字典里。

如果发现了比较复杂的错误并打算亲自改正，可以按下“**C-r**”开始一次递归编辑。改正错误之后，按“**ESC C-c**”退出递归编辑，Ispell将重新开始继续检查。（可以再去看看本章前面对递归编辑的介绍。）

因为备忘录里多次出现拼错“*receive*”的情况，找一个改一个未免有些麻烦。一个更好的方法是按下“**R**”——在编辑缓冲区里对这个单词进行查询－替换。

按下：**R**

```
Buffers Files Tools Edit Search Minibuf Help
(0)relieve (1)receive
- *Choices* - word: recieve
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo        (Text Fill)--L1--All----------------------
Query-replacement for:recieve
```

Ispell提示输入对“*recieve*”的改正。

把“recieve”改正为“receive”，然后按下回车键

```
Buffers Files Tools Edit Search Help
(0)relieve (1)receive
- *Choices* - word: recieve
We were very anxious to receive your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L5--All----------------------
Query replacing recieve with receive:(? for help)
```

Ispell开始一次查询－替换操作。

因为想把这个拼写错误一次性地都改正过来，所以按下“!”，表示“全部替换，用不着再查询了”。

按下：!

```
Buffers Files Tools Edit Search Help
(0)w-eve (1)w eve (2)wove (3)wive (4)were (5)wee (6)weave
(7)we 've (8)wave (9)neve (:)eve
- *Choices* - word: weve
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should receive the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L4--Bot----------------------
C-h or ? for more options; SPC to leave unchanged, Character to replace
word
```

Ispell 替换掉这个单词，然后进到下一个拼写有误的地方："*weve*"。

按下：7，选择"*we've*"

```
Buffers Files Tools Edit Search Help
(0)relieved (1)received
- *Choices* - word:.recieved
We were very anxious to receive your order and were pleased when it
came in today.

Now that we've recieved it, we'll begin work on filling the order.
You should receive the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L4--All----------------------
C-h or ? for more options; SPC to leave unchanged, Character to replace
word
```

Ispell 改正"*weve*"，然后进到下一个拼写出错的地方。

虽然前面已经改正了"*recieve*"，但 Ispell 会认为"*recieved*"是另外一个单词，所以这个拼写检查器会把它当做另外一个拼写错误找出来（注 2）。

注 2：请注意：Ispell 中的查询－替换操作与我们前面介绍的情况有一个细微的差别。普通的查询－替换操作会把"receive"的各种形式都找出来，不管它是不是内含在另外一个单词里（比如"recieved"等）。但在 Ispell 和我们马上将要讲到的 UNIX 拼写检查器里，单词的不同形式会被认为是不同的。

按下：**l**

```
Buffers Files Tools Edit Search Help
We were very anxious to receive your order and were pleased when it
came in today.

Now that we've received it, we'll begin work on filling the order.
You should receive the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
--**-Emacs:ordermemo        (Text Fill)--L1--All----------------------
Spell-checking done
```

Ispell 结束拼写检查工作。

查单词

在打字的时候，有时会说："哎，这个单词怎么拼呀？"或者"我好像拼得不对"。如果想对光标位置上的单词进行检查，请按下"**ESC $**"（命令名是**ispell-word**）组合键，或者从"**Spell**"（拼写）菜单里选择执行"**Check Word**"（检查单词）操作，Ispell 将对那个单词进行检查。如果单词拼写无误，Ispell 将显示"*word*: ok"；如果单词拼写错了，Ispell 会显示一个窗口，窗口里是刚介绍过的各种操作选项。

如果实在想不起某个单词应该怎样拼，可以先输入那个单词的头几个字母，然后按"**ESC TAB**"（命令名是**ispell-complete-word**）组合键。Ispell 会把单词的各种补足形式列出来，可以输入字符从中挑选一个。注意：Ispell的自动补足功能只能用在文本模式里。（其实它也可以用在另外几个用来对文本进行置标排版的编辑模式里，比如 nroff 模式。另外，"**ESC TAB**"组合键在那些程序设计类编辑模式里另有其他含义。）比如，不知道"*disambiguate*"该怎样拼写。我们将像下面这样进行操作：

输入：**dis ESC TAB**

```
Buffers Files Tools Edit Search Help
(0)dis (1)disaccharide (2)disambiguate (3)disastrous (4)disburse
(5)disc (6)discern (7)discernible (8)disciple (9)disciplinarian
(:)disciplinary (;)discipline (<)disco (=)discoid (>)discomfit
(@)discordant (B)discovery (C)discreet (D)discrepant (E)discrete
(F)discretion (G)discretionary (H)discriminable (I)discriminant
(J)discriminate (K)discriminatory (L)discus (M)discuss (N)discussant
- *Choices* - word: dis
Many words have multiple meanings.If a single meaning is possible for
a word, it takes a linguist to dis
--**-Emacs:semantics        (Text Fill)--L2--All----------------------
C-h or ? for more options; SPC to leave unchanged, Character to replace
word
```

按"2"键选择单词"*disambiguate*"的正确拼写。

Ispell 进程

只要启用过 Ispell，它就将一直运行在后台等待再次使用。如果想杀掉 Ispell 进程（比如，因为它而使系统速度变慢了），请输入“**ESC x ispell-kill-ispell**”或者从“**Spell**”菜单里选择执行“**Kill Process**（终止进程）”操作。

表 3-6 对 Ispell 命令进行了汇总，包括 **Spell** 菜单中的有关操作选项。

表 3-6：Ispell 命令速查表

键盘操作	命令名称	动作
ESC $ *Edit→Spell→Check Word*	**ispell-word**	检查光标位置上的单词或者光标后面的单词
（无） *Edit→Spell→Check Region*	**ispell-region**	检查文本块里的单词
（无） *Edit→Spell→Check Buffer*	**ispell-buffer**	检查缓冲区里的单词
（无） *Edit→Spell→Check Message*	**ispell-message**	检查电子邮件正文里的单词
C-u ESC $ *Edit→Spell→Continue Check*	**ispell-continue**	让 Ispell 重新开始继续执行；这个命令只有在运行过 Ispell 并用“**C-g**”组合键暂停过它的执行时才有效
（无） *Edit→Spell→Kill Process*	**ispell-kill-ispell**	杀死（即结束）Ispell 进程
ESC TAB *Edit→Spell→Complete Word*	**ispell-complete-word**	在文本模式下，自动补足当前单词

使用 UNIX 拼写检查器

Emacs 还有一个面向 UNIX 拼写检查器的接口。可以用它来检查一个单词、一个编辑缓冲区或者一个文本块。Emacs 将使用 UNIX 操作系统提供的字典，来检查单词的拼写情况。

spell-word命令对光标位置上的单词进行拼写检查（如果光标处于两个单词的中间，就将对前面的那个进行检查）。它会先在辅助输入区里显示一条下面这样的消息：

```
Checking spelling of word
```

这条消息的含义是“正在进行拼写检查，别再做任何编辑了”。如果单词拼写无误，Emacs会告知正确；如果Emacs认为那个单词拼写有误，它会显示一条下面这样的消息：

```
word is not recognized; edit a replacement: word
```

如果报错原因只是字典里查不到它，可单词实际上是正确的话（比如人名或者专业术语），可以直接按下回车键。如果确是单词拼写错误，请用标准的Emacs编辑命令来改正。Emacs将会开始一次查询－替换操作，把那个单词的上下文显示出来；再通过查询－替换操作的键盘响应命令，告诉Emacs是否需要替换那个单词即：按空格键改正拼写错误，按“**C-g**”退出且不做任何修改，按“**n**”不修改此处并前进到下一个出现单词拼写错误的位置等等。完整的键盘响应命令请参考本章前面“查询－替换操作”小节中的速查表。请看下面这个例子，我们将用UNIX拼写检查器来改正我们的备忘录文件。

输入：**ESC x spell-word RETURN**

```
Buffers Files Tools Edit Search Minibuf Help
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo        (Text Fill)--L1--All----------------------
'recieve' not recognized; edit a replacement:recieve
```

在辅助输入区里改正“*recieve*”的拼写。完成后按回车键开始一次查询－替换操作。

改正拼写错误后按下回车键。

```
Buffers Files Tools Edit Search Help
We were very anxious to recieve your order and were pleased when it
came in today.

Now that weve recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.

-----Emacs:ordermemo        (Text Fill)--L1--All----------------------
Query replacing \recieve\ with receive: (?for help)
```

按空格键对单词进行替换并前进到下一个拼错“receive”的地方。

按下：**SPACE**

```
Buffers Files Tools Edit Search Help
We were very anxious to receive your order and were pleased when it
came in today.

Now that we've recieved it, we'll begin work on filling the order.
You should recieve the merchandise in about six weeks, provided that
we can get materials from the factory quickly to do the necessary
customization.
-----Emacs:ordermemo        (Text Fill)--L4--All----------------------
Query replacing \recieve\ with receive: (?for help)
```

继续执行，直到完成整个编辑缓冲区的拼写检查。

在使用UNIX拼写检查器时有两件事情需要大家特别注意：

- **spell-word**命令根据所给单词，在整个编辑缓冲区的范围内查找有无拼错的情况。它会从缓冲区的开始一直查找到缓冲区的末尾。因此，当它开始对拼错的单词（本例中是“*recieve*”）进行查询－替换的时候，它会把光标移动到编辑缓冲区里第一次出现该单词的地方——不一定是光标原先的位置。
- **spell-word**命令只检查给定单词的拼写情况，不包括单词的各种变形（比如时态和单复数形式等）。在我们的例子里，它所查找和改正的单词是“*recieve*”而不是“*recieved*”。不过，当你看出“*recieved*”也拼错了的时候，完全能够用递归编辑的办法来改正它。

如果想对某个编辑缓冲区进行全面的拼写检查，请输入“**ESC x spell-buffer RETURN**”。Emacs会从编辑缓冲区的第一个单词开始查找不正确的单词。当它找到一个拼错了的单词时，会显示一条如下所示的消息：

```
word is not recognized; edit a replacement: word
```

如果该单词的拼写确实是正确的，按回车键（就像使用**spell-word**命令时那样），Emacs会前进到下一个拼写出错的单词处。如果单词是拼错了，用标准的编辑命令改正它；Emacs将开始一次查询－替换操作，以便把缓冲区里所有拼错这个单词的地方都改正过来。这次查询－替换操作完成之后，Emacs将继续顺序检查其他的拼写错误。

还可以对一个选定的文本块进行拼写检查。把文本标记放到文本块的一头，再把光标移动到另外一头，然后发出“**ESC x spell-region RETURN**”命令。**spell-region** 命令的行为与 **spell-buffer** 命令是完全一样的。

如果确认某个单词的拼写没有错误（按了回车键），那么在这次编辑活动期间，在此编辑缓冲区里，Emacs 将不会再过问这个单词。不过，只有 **spell-buffer** 和 **spell-region** 命令才有这种“记性”，它不适用于 **spell-word** 命令。

往 UNIX 拼写检查器的字典里添加新单词

可以把任何特殊的拼写情况（比如人名等）保存到一个名为“*.spell*”的文件里。要想利用UNIX拼写检查器的这个特长，需要修改变量 **spell-command** 的值，这个变量给出的是Emacs用来进行拼写检查的UNIX命令。在默认的情况下，这个变量的值是“**spell**”。在“*.emacs*”文件里增加一条语句就能把变量 **spell-command** 的值改为“**spell +.spell**”，如下所示：

```
(setq spell-command "spell +.spell")
```

注意：在“`spell`”和加号之间必须有一个空格，在加号和“`.spell`”之间却不能有空格。保存“*.emacs*”文件，再重新进入Emacs使这条语句生效。接着，把个人词汇放到一个名为“*.spell*”的文件里面。这个文件必须按字母表顺序排序，并且每行只能有一个单词。这个命令要求每一个子目录都必须有它们各自的“*.spell*”文件；对大多数应用情况来说，这种做法是很适用的，因为每一个工作项目有它们各自的专用词汇表（人名、技术术语等等）。

表 3-7 对 **spell** 拼写检查器的使用命令进行了汇总。

表 3-7：UNIX 拼写检查命令速查表

键盘操作	命令名称	动作
（无）	**spell-word**	检查光标位置上的单词或者光标后面的单词
（无）	**spell-buffer**	检查当前编辑缓冲区的拼写
（无）	**spell-region**	检查当前文本块的拼写

单词简写模式

单词简写模式（word abbreviation mode）可为特殊的单词和短语定义一些简写形式，而那些简写词则有很多种用法。就其传统用法而言，简写模式使得不必逐字逐句地输入很长的单词或者很长的短语。假设正在起草一份合同，合同里会反复用到“National Institute of Standards and Technology”这个短语却不允许使用首字母缩写。重复输入这么长的短语确实很费事，为了能省点力气，可特意定义一个简写词“*nist*”。有了这个简写词定义之后，只要输入“*nist*”加一个空格或一个标点符号，Emacs就会把那个短语完整地插入到文本里。Emacs会密切关注内容，一旦输入了一个简写词，并按下了空格键或者某个标点符号键，它就会立刻自动地将此简写词展开为其对应的短语。

先来看一个例子，然后再介绍进入单词简写模式和自行定义简写词汇表的方法。单词简写模式有一个非传统用法是我们非常喜欢的——用它在打字的过程当中来改正拼写错误。几乎每个人都有一些会习惯性打错的单词，这也许跟人的记忆神经有关系。可以采取一种很简单的办法来解决这一问题，即告诉Emacs这些拼错的单词是其正确形式的“简写词”，而Emacs会在输入这些单词的时候把它们自动改正过来；在Emacs把打错的单词改正过来之前，用户可能根本不知道自己打错了字。举个例子，假设已经进入单词简写模式，并在此前把“*recieve*”定义为“*receive*”的一个简写形式；现在，打字时又不小心打错了这个单词。如下所示：

输入：**You will recieve**

```
Buffers Files Tools Edit Search Help
You will recieve

--**-Emacs:letterb          (Text Fill)--L1--All----------------------
```

输入了错误的单词但还没有按下空格键。一旦按下空格键，Emacs将把它自动改正过来。

输入：**SPACE the materials you requested shortly.**

```
Buffers Files Tools Edit Search Help
You will receive the materials you requested shortly.

--**-Emacs:letterb          (Text Abbrev Fill)--L1--All----------------
```

按下空格键时，Emacs将自动改正这个单词。用户不需要停止录入工作，甚至可能根本就不知道自己打错了一个单词并已经被纠正过来了。

用户不仅能为自己经常用到的短语定义一些简写词以方便工作，还可以为自己经常打错的单词定义一个简短的“简写词汇表”以节约花在文件校对上的时间并减少常见打字错误的次数。

在为短语定义简写词时，千万不要使用现有的英文单词；否则，即使不想让Emacs对简写词进行扩展，它也会那样做（Emacs在对简写词进行扩展的时候不会事先提问）。举个例子，如果经常用到“World Association for Replicant Technology”这个短语，就不要把它的简写词定义为“*wart*”（英文的意思是“瘤状物”）；否则，一输入这个词就会被扩展为短语，再想用“*wart*”来描述癞蛤蟆之类的东西可就不容易了。（如果极少用到单词“*wart*”，认为使用“*wart*”作为简写词非常方便，就可以在需要输入单词“*wart*”的时候用“**ESC x unexpand-abbrev RETURN**”命令告诉Emacs取消这个简写词定义。）

Emacs只会严格按简写词的定义情况来对它进行扩展。如果已经把“*recieve*”定义为“*receive*”的一个简写词，就还得把“*recieves*”、“*recieving*”、“*recieved*”等也定义为简写词，以包括可能打错的这个单词的各种时态。

在开始定义自己的简写词汇表之前，还有一件更重要的事要记住：Emacs会根据简写词被定义时的编辑模式对它们进行分类。全局性简写词可以用在任何一种编辑模式里；而局部性简写词则只能用在定义时所在的编辑模式里。比如，如果想让某些简写词只能用在文本模式里，而不能用在C语言模式里，就必须在文本模式里把它们定义为局部简写词（local）。如果想让简写词用在任何一种编辑模式里，就必须把它们定义为全局简写词（global）。记住：简写词的作用范围是编辑模式，而非文件或编辑缓冲区。

Emacs还提供一种简写词的逆向定义方法。说它是“逆向定义方法”的原因是因为先输入的是简写词，后输入的才是其定义。有些命令（书中没有对这些命令进行介绍）允许先输入其定义，后输入简写词；但必须使用一些特殊的按键组合，才能让Emacs知道简写词是由光标前哪些字符组成的。相比之下，逆向定义方法不仅更方便，而且无论与简写词对应的短语定义是一个单词还是十个单词，它都能应付自如。

为一次编辑工作定义临时性的简写词汇

一般来说，既然已经辛苦地定义了单词的简写形式，那么肯定会在多次的Emacs编辑工作中用到。可如果想先试试单词简写模式，然后再决定是否需要让它成为编辑器启动时初始化工作的一部分，那么请按以下步骤进行操作。

为编辑工作临时定义一些简写词汇:

1. 用输入“**ESC x abbrev-mode RETURN**”的办法进入单词简写模式。“Abbrev”字样将出现在状态行上。
2. 如果想定义的是全局性简写词，请在输入简写词之后按下“**C-x a i g**”组合键（命令名是**add-inverse-global**）。如果想定义的是局部性简写词，请在输入简写词之后按下“**C-x a i l**”组合键（命令名是**add-inverse-local**）。Emacs将提示输入其扩展短语。
3. 输入该简写词的扩展短语后按下回车键。Emacs对简写词进行扩展并会在此后每次输入这个简写词再跟上一个空格或标点符号的时候照章行事。定义的简写词只在此次Emacs编辑工作期间有效。

如果喜欢使用Emacs的单词简写模式，可以让它成为编辑器启动时初始化工作的一部分；具体操作步骤列在下一小节里。

为多次编辑工作定义永久性的简写词汇

当编辑工作到了离不开单词简写模式的地步时，把它加到“*.emacs*”文件里是最简单的解决方案(再也用不着每次都要回忆上一小节里列出的操作步骤了)。下面给出的操作流程将创建出一个用来保存简写词汇的永久性文件，Emacs会在每次启动时自动加载这个文件。还可以随时从这个文件里删除简写词汇。删除简写词汇的操作步骤将在下一小节里介绍。

定义简写词汇并使它们成为编辑器启动时初始化工作的一部分:

1. 把下面这些语句添加到“*.emacs*”文件里:

```
(setq-default abbrev-mode t)
(read-abbrev-file "~/.abbrev_defs")
(setq save-abbrevs t)
```

2. 保存“*.emacs*”文件并重新启动Emacs。“Abbrev”字样将出现在状态行上。还会出现一条错误信息，即Emacs无法加载简写词汇文件（这条信息很容易理解，因为你还没有创建这个文件）。不要理睬这条错误信息，它以后就不会再出现了。

3. 输入一个简写词，然后按下“***C-x a -***”或者“**C-x a i g**”组合键。“**C-x a i g**”创建的是全局性简写词；如果你创建局部性简写词，请按下“**C-x a i l**”。Emacs于是提示输入其扩展短语。

4. 输入该简写词的扩展短语后按下回车键。Emacs对简写词进行扩展并会在此后每次输入这个简写词再跟上一个空格或标点符号的时候照章行事。根据需要重复第3步和第4步以定义任意多个简写词。

5. 输入“**ESC x write-abbrev-file RETURN**”保存简写词汇文件。Emacs提示输入一个文件名。

6. 输入“**~/.abbrev_defs RETURN**”。Emacs保存该文件。

上述流程的第5步和第6步只有在第一次定义简写词汇的时候才需要执行。这个文件存在以后，Emacs会自动加载这个简写词汇文件。

在此后的编辑工作中再定义简写词汇的时候，Emacs会询问是否想保存简写词汇文件（注3）。回答“**y**”以保存刚才新定义的简写词汇，它们会自动生效。

删除简写词汇条目

如果经常使用简写词汇，那么可能会不断地做些改动。如果想对简写词汇表进行编辑，请输入“**ESC x edit-abbrevs RETURN**”命令。如果你想查看（而不是编辑）简写词汇表，请输入“**ESC x list-abbrevs RETURN**”命令。

在简写词汇表显示出来以后，按下“**C-k**”组合键（或者其他任意编辑命令）来删除不想要的简写词条目。这个词汇表的格式是由Emacs自己安排的，所以不要去编

注3： Emacs 19的某些早期版本不会询问是否想保存简写词汇文件。如果情况真是这样，就必须在每次定义了一些简写词汇之后输入“**ESC x write-abbrev-file RETURN**”以明确地保存自己的简写词汇。因为不可能每天都要定义一些简写词汇，所以这个要求不会有多不方便。不管怎么说，我们认为在简写词汇上多花些功夫是值得的。

辑这个文件的文本行，或者给它添加新的文本行；删除操作大概是惟一安全的选择。下面是编辑简写词汇表文件时将会出现的简写词排列情况。这个文件会根据简写词是全局性的还是某个模式局部性的而划分为不同的区。如下所示：

```
"iwthout"        1     "without"
"prhase"         1     "phrase"
"teh"            1     "the"
"fo"             1     "of"
"eamcs"          2     "Emacs"
"wrok"           1     "work"
"aslo"           1     "also"
"sotred"         1     "stored"
"inforamtion"    1     "information"
"esc"            6     "ESC"
"taht"           1     "that"
"chatper"        1     "chapter"
"adn"            1     "and"
"iwth"           1     "with"
"chpater"        1     "chapter"
"tex"            36    "TeX"
"loaction"       1     "location"
"wart"           1     "World Association for Replicant Technology"
```

列在第一栏里的是简写词；上例中的简写词基本上都是一些容易拼错的单词。第二栏供内部记录使用，不用管它。第三栏给出了简写词的定义情况，Emacs 在看到简写词时会用与之对应的单词或短语进行替换。

删除简写词汇条目的方法是：先把光标移动到定义该条目的文本行上，用“**C-k**”组合键把整个文本行删掉，最后输入“**ESC x write-abbrev-file RETURN**”保存这个文件。按下“**C-x b**”将返回此前正在编辑的编辑缓冲区（这是一个多编辑缓冲区操作命令，我们将在第四章里介绍它。）

禁用简写词汇

以下两种办法中的任何一种都可以用来完全禁用简写词汇。第一种办法是输入“**ESC x kill-all-abbrevs RETURN**”命令，这个命令的作用是在本次编辑工作中禁用简写词汇。

第二种办法是删除保存简写词汇表的文件。如果简写词汇是编辑器启动时初始化工作的一部分，就要从“*.emacs*”文件里删除“**read-abbrev-file**”那一行。

简写词汇与字母的大小写

Emacs对简写词所做的字母大小写设置，通常都会让人称心如意。但如果对简写词的字母大小写有特殊的要求，那么多知道些内幕往往会有很多好处。请看下面这些规则:

- 只要简写词的短语定义里有大写字母，Emacs在插入这个短语定义的时候就不会对它做任何改动。比如说，如果把“*ora*”定义为“*O'Reilly & Associates*”的一个简写词，那么“*O'Reilly & Associates*”的字母大小写情况就将保持原样。
- 如果简写词的短语定义全都是小写字母，Emacs将根据以下规则对它进行字母大小写设置:
  - 如果把简写词的全部字母都输入为小写，Emacs就将以小写字母插入其短语定义。
  - 只要输入的简写词里有大写字母，Emacs就将把其短语定义的第一个单词的第一个字母设置为大写。
  - 如果把简写词的全部字母都输入为大写，并且变量“**abbrev-all-caps**”设置为“**nil**”，Emacs就将把其短语定义的每一个单词的第一个字母设置为大写；如果变量“**abbrev-all-caps**”设置为“**t**”，Emacs就将把其短语定义的每一个单词的每一个字母设置为大写。

表3-8给出了一些例子。

表3-8: 单词简写情况示例

简写词	短语定义	输入	扩展为	说明
lc	lamb chop	**lc**	lamb chop	“*lc*”是小写，所以“*lamb chop*”是小写
lc	lamb chop	**Lc**	Lamb chop	“*Lc*”里有一个大写字母，所以“*Lamb*”的首字母大写
lc	lamb chop	**lC**	Lamb chop	“*lC*”里有一个大写字母，所以“*Lamb*”的首字母大写

PDG

表 3-8: 单词简写情况示例（续）

简写词	短语定义	输入	扩展为	说明
lc	lamb chop	**LC**	Lamb Chop	"*LC*" 全都是大写，所以两个单词的首字母都大写
lc	Lamb Chop	**lc**	Lamb Chop	短语定义里的字母大小写情况永远不变
lc	Lamb Chop	**LC**	Lamb Chop	短语定义里的字母大小写情况永远不变

要想记住这些规则还真得下点功夫，但用不着死记硬背。Emacs 通常能够很好地完成字母大小写方面的处理工作；只要做点练习，就能掌握 Emacs 的行为规律。

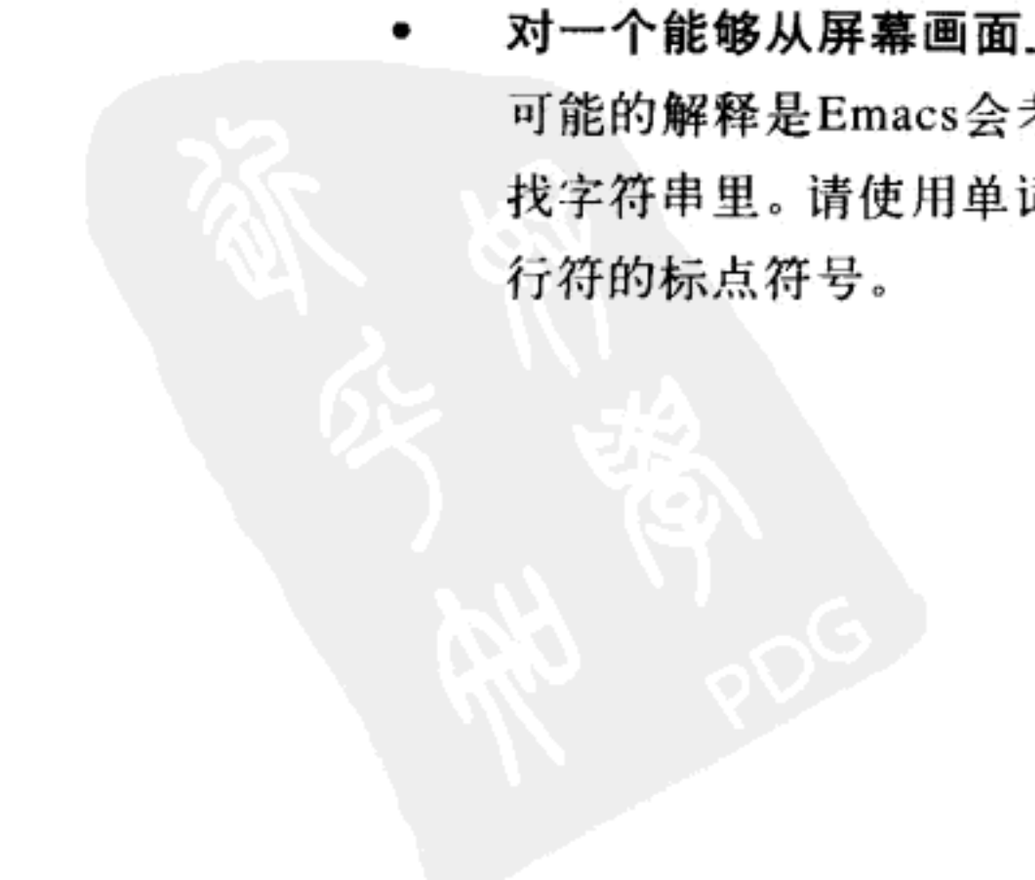
表 3-9 对与简写词有关的操作命令进行了汇总。

表 3-9: 简写词编辑命令速查表

键盘操作	命令名称	动作
（无）	**abbrev-mode**	进入（或退出）单词简写模式
C-x a - 或 **C-x a i g**	**inverse-add-global-abbrev**	输入全局性简写词之后，输入其短语定义
C-x a i l	**inverse-add-mode-abbrev**	输入局部性简写词之后，输入其短语定义
（无）	**unexpand-abbrev**	撤销最近一个简写词定义条目
（无）	**write-abbrev-file**	保存简写词汇表文件
（无）	**edit-abbrevs**	编辑简写词汇表
（无）	**list-abbrevs**	查看简写词汇表
（无）	**kill-all-abbrevs**	本次编辑工作禁用简写词功能

疑难解答

- **对一个能够从屏幕画面上看到的字符串进行查找，可 Emacs 就是找不到。**最可能的解释是Emacs会考虑换行符和标点符号，而你可能没有把它们包括在查找字符串里。请使用单词查找操作，它在对字符串进行查找的时候，会忽略换行符的标点符号。



- **Ispell工作不正常**。造成这种情况的原因有很多。有时候，**ispell-region**命令可以工作而其他命令却不行。请用“**C-x h**”把整个编辑缓冲区标记为一个文本块，然后输入“**ESC x ispell-region**”命令。如果这条路走不通，请继续往下看。

- **看到一条错误信息“Problem starting Ispell; use old-style spell instead?（启动Ispell时故障：是否使用老式的spell?）”**。很可能是Ispell没有安装。如果回答“**y**”，Emacs将启动UNIX拼写检查器**spell**程序。与**spell**程序有关的讨论请参考“使用UNIX拼写检查器”一节。

- **看到一条错误信息“ispell version 3.1.* is required: try renaming ispell4.el ispell.el（要求使用ispell 3.1.*，请试试把ispell4.el更名为ispell.el）”**。在Emacs 19里使用了Ispell Version 4。Emacs 19需要的是Ispell Version 3。请让系统管理员把Ispell降级为Ispell Version 3（也可以首先建议系统管理员把文件*ispell4.el*更名为*ispell.el*。）。

- **在使用spell拼写检查器时，看到一条错误信息“Searching for program: no such file or directory, spell（查找程序：没有文件或子目录，spell）”**。有些UNIX系统（特别是Linux发行版本）会用Ispell替代UNIX的**spell**拼写检查器程序。如果情况如此，则只能使用Ispell或者找个有**spell**软件的人把它安装在自己的系统上。

- **在使用Ispell-buffer的时候，看到一条错误信息“spell can't read *.spell*（spell无法读出“*.spell*”文件）”**。需要在文件所在的子目录里创建一个“*.spell*”文件作为拼写检查文件。

- **定义了一些简写词，可Emacs无法保存它们**。如果没有在自己的“*.emacs*”文件里进行设置，Emacs在默认情况下是不会保存简写词汇的。请参考本章中“为多次编辑工作定义永久性的简写词汇”一节里的讨论。如果已经在自己的“*.emacs*”文件里进行了设置却还是没有办法保存简写词汇，那么使用的可能是Emacs 19的早期版本，它们的单词简写模式有一个小bug。只能明确地在自己每次定义了一些简写词汇之后输入“**ESC x write-abbrev-file**”以保存它们。

本章内容：

- 文件、编辑缓冲区和窗口
- 同时使用多个编辑缓冲区进行工作
- 使用窗口进行编辑
- 在文档中使用书签
- 临时性地挂起Emacs
- 使用多个X窗口进行编辑

第四章

使用编辑缓冲区和窗口

Emacs最有用的功能之一是它能够同时对多个编辑缓冲区进行编辑，并利用窗口同时显示一个以上的编辑缓冲区。有关的操作命令并不复杂，只需掌握有限的几个命令就可以大大提高工作效率。使用多编辑缓冲区和多窗口的次数越多，就越能发现它更多的用途。

在这一章里，我们将介绍编辑缓冲区和窗口的区别，与多编辑缓冲区有关的操作，以及如何利用窗口来显示多个编辑缓冲区。接着，我们将继续讨论如何对编辑缓冲区列表清单进行操作(这是管理多编辑缓冲区的一个简便方法)，以及如何利用书签保持正在处理的各种文件里的位置。有时候，可能需要临时性地把Emacs挂起来。确实可以这样做，即使正使用多个窗口和编辑缓冲区时也完全没有问题，我们会介绍如何操作。最后，我们在本章的末尾专门为X用户准备了一节，介绍如何在Emacs编辑工作中使用多个X窗口的内容。

文件、编辑缓冲区和窗口

Emacs里的一切编辑操作都发生在编辑缓冲区(buffer)里。虽然编辑缓冲区往往会有许多其他方面的用途，但它通常就是某个文件的工作副本。如果你正在编辑的编辑缓冲区里包含着某个文件的一个副本，那么当你保存自己所做的改动时，Emacs将把编辑缓冲区的内容复制到文件里。在实际工作中，在对编辑缓冲区进行完编辑

之后，你完全可以不保存自己所做的修改。从另一方面讲，在一次编辑工作中，想把编辑缓冲区里的改动保存到文件里多少次也都可以。如果所做的编辑修改确实非常重要，频繁存盘还是一个很好的习惯呢。

除了工作在文件副本上的编辑缓冲区以外，你还会接触到一些只能用做临时工作区域的编辑缓冲区，这些编辑缓冲区与文件一般没有什么关联。这些临时性的工作区域很适合用来做练习或打草稿。可以随时使用**write-file**命令（“**C-x C-w**”）或**save-buffer** 命令（“**C-x C-s**”）来保存这类临时性编辑缓冲区里的内容。如果试图保存的编辑缓冲区与任何文件都没有关联，“**C-x C-s**”命令会要求提供一个文件名——它会在辅助输入区里显示提示信息“File to save in:(把文件保存为:)”，后面跟着一个默认的子目录名。

可以不受限制地打开任意多个编辑缓冲区。在大部分时间里，同时显示在屏幕上的编辑缓冲区也就是一两个；但尽管看不到它们，在一次Emacs编辑工作中创建的编辑缓冲区也仍然是活动的。可以把它们想像为一摞纸，而显示在屏幕上的是最顶上的那张。可以随时翻看另外一张纸（即转到另一个编辑缓冲区），也可以随时给这摞纸再添上一张（即创建一个新的编辑缓冲区）。

特殊用途编辑缓冲区的创建工作由Emacs负责。这些内部编辑缓冲区的名字通常采用“*buffer name*”的格式。“*Help*”、“*scratch*”、“*mail*”和“*Buffer List*”等几个编辑缓冲区就是由Emacs自己创建的。

每个编辑缓冲区都有一个相关的主编辑模式，这个主模式左右着Emacs在这个编辑缓冲区里的大部分行为。比如，用来书写英文文本的文本模式，与用来编写LISP程序代码的LISP模式这两者的行为是不一样的。

那窗口又是些什么东西？窗口是显示器屏幕上用来显示编辑缓冲区内容的区域（注1）。换句话说，需通过一个窗口来查看一个编辑缓冲区。可能没有想过自己的显示器屏幕就是一个大窗口，但实际情况却是如此。每个窗口都有一个相关的状态

注1：这是Emacs对“窗口”的定义，它与X窗口系统和微软公司的Windows中的图形化用户界面意义上的“窗口”是不同的概念。在这些用户界面里，窗口是屏幕上一些有程序运行在其中的区域。为了与Emacs窗口有所区别，我们将把X窗口系统下的多窗口称为“窗格（frame）”，前者指的是Emacs屏幕画面上的区域，而后者则是图形化用户界面意义上的窗口。

行，并且通常也只有一个。从理论上讲，可以在屏幕上不受限制地拥有任意多个不同的窗口。从一个窗口转移到另一个窗口，就能做到同时对多个文件进行编辑、对比两个窗口里的文件、从一个文件复制到另一个文件等等。

虽然窗口和编辑缓冲区是两个很容易混淆的概念，可是两者之间的差异并不复杂。我们已经讲过，编辑缓冲区是文件的一个工作副本；而窗口则是显示器屏幕上的一个区域。可以在屏幕上让好几个窗口显示同一个编辑缓冲区——比如想查看同一文件的两个不同位置、或者想把一个地方的内容复制到另一个地方去的时候。比较常见的情况是在屏幕上用不同的窗口来代表不同的文件；或者是用一个窗口来显示某个文件的编辑缓冲区，用另一个窗口来显示Emacs某个内部的编辑缓冲区——内部编辑缓冲区将在本章后半部分探讨。随着大家对本章学习的深入，文件、编辑缓冲区和窗口之间的区别将越来越直观。

同时使用多个编辑缓冲区进行工作

输入“**emacs** *filename*”命令就能进入Emacs是大家都已经知道的事情了。如果还想再创建一个包含着另一文件内容的编辑缓冲区，只需按下“**C-x C-f**”组合键来查找文件。Emacs将自动创建出第二个编辑缓冲区并转移至此。如果已经在某个编辑缓冲区里打开过这个文件的一个副本，“**C-x C-f**”命令将进到那个现有的编辑缓冲区里去——这种情况可能正是想要的；如果“**C-x C-f**”每次都从磁盘上读出文件，那么文件就将形成许多个版本，而各版本之间又几乎没有什么区别。如果给“**C-x C-f**”提供的文件名不存在，Emacs就会认为要用那个名字创建一个新文件，它会进到一个空白的编辑缓冲区里。（如果在输入文件名时打错了字，按下“**C-x C-v**”组合键将把你带回原来的文件。）

“**C-x C-f**”后面永远要跟有一个文件名；而在编辑缓冲区之间进行切换的“**C-x b**”命令的后面要跟着一个编辑缓冲区名。是否注意到状态行上只显示有编辑缓冲区的名字而没有文件名？有些版本的Emacs会把这两个名字都显示出来，但GNU Emacs只显示编辑缓冲区名。如果没有对它们进行过修改（请参考本章后面“编辑缓冲区的重新命名”小节），那么编辑缓冲区名和文件名（如果有的话）就将是一样的。

可以用“**C-x b**”来完成以下几件事：

如果在“C-x b”的后面输入了	Emacs 将
一个新的编辑缓冲区名	新创建一个与任何文件都没有联系的编辑缓冲区并转移到那里
一个已有的编辑缓冲区名	进入那个编辑缓冲区（不论该编辑缓冲区是否与某个文件相关联）

在编辑缓冲区之间进行切换要按下“**C-x b**”组合键。Emacs 会显示一个默认的编辑缓冲区名。如果它就是想去的编辑缓冲区，直接按下回车键；否则请输入正确的编辑缓冲区名的前几个字母再按下**TAB**键，Emacs将自动补足这个名字的剩余部分。现在，按下回车键就能转移到那个编辑缓冲区。

如果想创建第二个（或者第三个、第四个等等）空白的编辑缓冲区，按下“**C-x b**”组合键，Emacs将要求提供一个编辑缓冲区名。可以随意起个名字，比如parctice，然后按下回车键；Emacs 将创建这个编辑缓冲区并进入其中。举个例子，假设在dickens编辑缓冲区里工作时忽然想起了点东西，决定打开一个新的编辑缓冲区来编辑 James Joyce（英国诗人乔叟）的作品。如下所示：

初始状态：正在 dickens 编辑缓冲区里编辑狄更斯的作品。
按下：**C-x b**

```
Buffers Files Tools Edit Search Minibuf Help
It was the best of times, it was the worst of times, it was the age of
wisdom,it was the age of foolishness it was the epoch of belief, it
was the epoch of incredulity, it was the season of Light, it was the
season of Darkness,it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we
were all going direct to Heaven, we were all going direct the other
way--in short,the period was so far like the present period, that
some of its noisiest authorities insisted on its being received, for
good or evil, in the superlative degree of comparison only.

-----Emacs:dickens          (Text Fill)--L1--All----------------------
Switch to buffer:(default *scratch*)
```

准备输入一个新的编辑缓冲区名。

输入：**joyce RETURN**

```
Buffers Files Tools Edit Search Help

-----Emacs:joyce          (Text Fill)--L1--All----------------------
```

现在，可以往这个名为 joyce 的新编辑缓冲区里输入东西了。

上述操作与使用“**C-x C-f**”（命令名是 **find-file**）组合键的情况并没有什么不同之处，两者惟一的区别是新编辑缓冲区 joyce 现在还没有与某个文件关联起来。因此，如果现在就退出 Emacs，编辑器不会询问你是否想保存它。

“**C-x b**”特别适用于没有记住正在编辑的文件的名字的场合。假设正在编辑一个名字非常难记的文件——比如 */tmp/80b16.12344* 文件（注 2）；不小心按了某个键使这个编辑缓冲区从屏幕上消失了（这类命令我们马上就会在本章后面讲到）。怎样才能让 */tmp/80b16.12344* 文件重新回到屏幕上来呢？记住这个文件的全名或者哪怕是文件名的一部分了吗？没记住？别着急，在没有进行其他操作之前，立刻按下“**C-x b**”组合键。默认的编辑缓冲区恰好是刚才消失的那个；按下回车键，它又会重新出现。

还有一种办法是选择“**Buffer**（编辑缓冲区）”菜单，所有的编辑缓冲区都列在其中，从中挑选一个就行了。如果看到一些不是自己亲自创建的编辑缓冲区，可别大惊小怪——Emacs 也会创建编辑缓冲区，这类编辑缓冲区的名字多是像“*Help*”和“*scratch*”的形式。

多个编辑缓冲区的保存

大家已经知道要想一个一个地保存编辑缓冲区要按下“**C-x C-s**”组合键。如果正在使用多个编辑缓冲区，就该知道按下“**C-x s**”（命令名是 **save-some-buffers**）组合键可把它们都保存起来。Emacs 会依次询问是否想保存每一个编辑缓冲区；回答

注 2：这个文件名还不算最古怪的。许多程序创建的临时文件都有类似于这样的名字。你可以想像自己正在调试一个这样的程序。

“**y**”保存，回答“**n**”不保存。按下“!”键将使Emacs无条件地保存所有的编辑缓冲区。如果只想保存某个编辑缓冲区，其他的都不要，请输入一个英文句号(.)。如果想取消这个命令并且不保存当前的编辑缓冲区，按下“**q**”键（在按下“**q**”之前保存的编辑缓冲区都已经保存起来了；**ESC**对那些文件没有影响）。在决定是否需要保存某个编辑缓冲区之前，如果想先看看它里面的内容，请按下“**C-r**”组合键；Emacs将进入查看模式（view mode）——这个模式里的编辑缓冲区只能看，不能改。按下“**C-c**”组合键退出查看模式并回到保存编辑缓冲区的操作上来。

编辑缓冲区的删除

编辑缓冲区创建起来很容易，想删除它们也不难。如果觉得Emacs因为有太多的编辑缓冲区而变得磕磕绊绊，就会要删除一些编辑缓冲区。开始工作时也许使用了5个编辑缓冲区，可现在想换成另外5个继续工作；把前5个编辑缓冲区都删掉肯定会使工作更顺手一些。删除编辑缓冲区也是一种有用的应急措施——比如某些替换操作引起灾难性后果的时候，可以先删除那个编辑缓冲区并选择不保存所做的修改，然后再重新读取文件。

删除一个编辑缓冲区不会删除它所关联着的文件，也不等同于隐藏一个缓冲区的操作。被隐藏起来的编辑缓冲区依然是活动的，而被删除的编辑缓冲区则将不再是本次Emacs工作的一部分。还用一摞纸为例，删除一个编辑缓冲区等于从这摞纸里把一张纸抽出来扔掉。

删除编辑缓冲区也不会让用户冒丢失编辑修改的风险。如果对编辑缓冲区进行过修改（并且这个编辑缓冲区代表着某个文件），Emacs会询问是否需要在删除这个编辑缓冲区之前保存所做的修改。虽然对与文件没有关联的编辑缓冲区所做的修改将会丢失，但可能这些编辑缓冲区的命运无关紧要。

删除一个编辑缓冲区的方法是按下“**C-x k**”（命令名是**kill-buffer**）组合键或者从“**File**（文件）”菜单里选择执行“**Kill Current Buffer**（删除当前编辑缓冲区）”操作。Emacs会给出当前显示的编辑缓冲区的名字，按下回车键就可以删除它。如果Emacs给出的编辑缓冲区名不是想删除的，那么可以输入另外一个编辑缓冲区的名字再按下回车键。如果准备删除的编辑缓冲区里还有一些没来得及保存的修改，Emacs会显示一条如下所示的消息：

```
Buffer buffer name modified. Kill anyway? (yes or no)
```

如果不想保存所做的修改，回答“**yes**”，Emacs将删除这个编辑缓冲区。如果想停止编辑缓冲区的删除操作，回答“**no**”。此后，可以先用“**C-x C-s**”组合键保存这个编辑缓冲区，再用“**C-x k**”组合键删除它。

还可以让Emacs就准备删除的每一个编辑缓冲区进行提问，从而有机会一个一个地决定是否要删除它们。如果想用这种办法来删除已经不再需要的编辑缓冲区，请输入“**ESC x kill-some-buffers**”。Emacs将依次列出每个编辑缓冲区的名字，并指出它里面有没有还未来得及存盘的修改，然后询问是否真的想删除它。

编辑缓冲区的重新命名

编辑一个文件的时候，编辑缓冲区的名字就采用文件的名字。如果觉得长文件名在使用中不方便，可以给编辑缓冲区重新起个名字（这种重命名只改变编辑缓冲区的名字，对文件名不会有影响）。这个功能最适用于某些自动补足功能不太完备的Emacs版本。GNU Emacs的自动补足功能是很不错的：当需要输入编辑缓冲区的名字时，只要输入其前几个字母再按下**TAB**键，Emacs就会自动补足那个名字。但肯定会有你想改变编辑缓冲区名字的时候。

重新命名编辑缓冲区的方法是：输入“**ESC x rename-buffer**”，Emacs提示输入新名字；输入它，再按下回车键。新名字将出现在状态行上。

前面讲过，GNU Emacs的状态行上只显示编辑缓冲区的名字，而不是一个编辑缓冲区名和一个文件名。即使给包含着文件的编辑缓冲区重新命名，Emacs也不会忘记编辑缓冲区和文件之间的对应关系；保存文件（“**C-x C-s**”）或查看编辑缓冲区清单（参见本章后面的内容）试试，就会明白了。

可是若有两个同名的编辑缓冲区该怎么办呢？比如，正编辑主目录里一个名为*memo*的文件和自己某个子目录里一个名字也为*memo*的文件的情况。虽然两个编辑缓冲区都叫做memo，但Emacs会给第二个编辑缓冲区的名字后面加上一个“<2>”以区别它们。用户可以用查看编辑缓冲区清单的办法来区分它们，查看编辑缓冲区清单的操作在本章后面就要讲到。下面是编辑缓冲区重新命名操作一种很有用的用法：如果想把哪个文件来自哪个子目录也表示出来，可以在修改编辑缓冲区的名字

时把它们的子目录名也包括在其名字里，比如像“home:memo”和“staff:memo”的样式。

注意：如果有好几个编辑缓冲区的名字是“memo”、“memo<2>”、“memo<3>”的样式，就很可能是在给出文件名时弄错了它们的目录名。如果读取文件的时候弄错了目录，Emacs 会认为想开始编辑一个新文件。举个例子，假设想编辑的文件是“*~/work/memo*”，但错误地输成了“*~/novel/memo*”。因为“*~/novel/memo*”原来并不存在，所以 Emacs 会新创建一个空白的“memo”编辑缓冲区。如果改正了这个错误（“**C-x C-f ~/work/memo**”），Emacs 也将相应地重新命名编辑缓冲区：那个空白的编辑缓冲区“memo”与“*~/novel/memo*”文件关联，而真正想要的编辑缓冲区的名字将是“memo<2>”。

弄错文件的名字是一个很常见的错误，用下面这个办法可以解决它。如果发现“**C-x C-f**”命令找到的文件不对，可以使用“**C-x C-v**”命令，用自己想要的文件来替换它。“**C-x C-v**”会把查找到的文件放到当前编辑缓冲区里，而不是去创建一个新的编辑缓冲区。它的含义是“把真正想要的文件找出来并要它替换现在这个”。这个命令可以解决不必要地出现了大量带编号的编辑缓冲区（比如“memo”、“memo<2>”等）的问题。

只读编辑缓冲区

在工作中，可能需要查阅一些并不想对其进行修改的文件 —— 只想浏览它们的内容。但因误碰键盘而造成意外修改的情况很容易发生。我们介绍过几种恢复原始文件的办法，可要是能够从根本上预防类似事故的发生岂不是更好？该怎么办呢？

可以用“**C-x C-q**”组合键把编辑缓冲区设置为只读属性。找个编辑缓冲区试试，状态行左边会出现两个百分号（%%），它们就位于在表示编辑缓冲区已被修改的星号（**）位置处。这两个百分号表明这个编辑缓冲区是只读的。如果试图对一个只读的编辑缓冲区进行输入，Emacs 将发出蜂鸣并在辅助输入区里显示一条出错信息“Buffer is read-only（编辑缓冲区是只读的）”。如果改变主意，又想对只读编辑缓冲区进行编辑该怎么办呢？好办，再按一次“**C-x C-q**”组合键即可。这个命令的作用是切换编辑缓冲区的只读状态 —— 也就是说，反复按下“**C-x C-q**”组合键将使编辑缓冲区交替进入只读和读写状态。

记住：切换只读状态不会改变UNIX文件的权限。如果正在编辑的编辑缓冲区里包含的是别人的文件，“**C-x C-q**”组合键将无法改变其只读状态。编辑别人文件的方法之一是先用**write-file**命令给自己复制一份它的副本，然后再对副本进行修改。如果想对属于别人的电话号码簿文档进行修改，应该先读入这个文件，然后用“**C-x C-w**”组合键把这个文件存盘一次，再用“**C-x C-q**”组合键把它从只读状态改为读写状态。以上操作不会改变原来的电话号码簿文档中的内容；但会得到一份能够随意修改的副本。如果想把某个只读文件里的一小部分内容复制到另外一个文件里，应该先标记好文本块，然后用“**ESC w**”组合键复制它，再移动到要插入这些文本的地方，按下“**C-y**”组合键粘贴它。

使用窗口进行编辑

在X窗口系统出现以前，许多人都把Emacs用做惟一的窗口化系统。如今，X和Emacs已经集成在一起了（在X下运行Emacs的有关内容请参考第十四章），但尽管如此，还是有很多人坚持使用Emacs的窗口功能，甚至在X下也不改初衷——他们看中的是Emacs窗口能够完成许多工作并大大提高工作效率，而不是计较Emacs在具体做法上与X的不同。不管是否使用X，了解Emacs窗口的使用方法都是很有必要的。

前面已经讲过，窗口是屏幕上的一些区域，而Emacs会把正在编辑的编辑缓冲区显示在这些窗口里。可以同时在屏幕上打开许多个窗口，每个窗口显示不同的编辑缓冲区。但打开的窗口越多，每个窗口的面积也就越小——Emacs窗口不能重叠，这是它与X窗口的一个不同之处。随着打开的窗口数目的增加，原有窗口的尺寸将会缩小。屏幕就像是一块馅饼，可以把它分成许多份，但分的份数越多，它们也就越小。可以让窗口左右排列、上下排列，或者混合排列。每个窗口都有它自己的状态行，上面给出了编辑缓冲区的名字、所处的编辑模式、所在编辑缓冲区里的前后位置等。为了表明窗口之间的边界，状态行通常呈反显状态。

我们还说过，窗口与编辑缓冲区是两回事。事实上，可以用一个以上的窗口来显示同一个编辑缓冲区里的内容。如果想同时查看某个大文件不同位置上的内容，这个功能将非常有用。甚至可以让编辑缓冲区的同一块内容同时显示在两个窗口里，在其中一个窗口里所做的任何修改都将出现在另一个窗口里。

想对文本进行标记、剪切和粘贴等操作的时候，编辑缓冲区和窗口之间的区别就变得非常重要了。文本标记是与编辑缓冲区而不是窗口关联的，每个编辑缓冲区只能有一个文本标记。如果在同一编辑缓冲区上的另外一个窗口里又设置了文本标记，Emacs会把文本标记移到新的位置上，在前一个窗口里设置的文本标记将不复存在。

再来说说光标的事。无论何时，屏幕上只能有一个光标；也就是说，无论何时，都只能待在一个窗口里。虽然只有一个光标，可每个窗口都能记住其中的当前编辑位置——也就是说，如果把光标从一个窗口移到另外一个窗口并做了一些编辑工作，那么当再次回到原来的窗口里时，仍会回到刚才离开的位置。在窗口里的当前编辑位置（不管光标是否在这个窗口里）叫做“插入点（point）”。每个窗口都有自己的插入点。插入点和光标这两个术语可以互换使用——但我们将尽量做到对它们有所区分。

编辑缓冲区和窗口之间的区别听起来很混乱，但在实际工作中却很容易把它们区分开。在使用多个窗口对同一个编辑缓冲区进行编辑方面，我们有着多年的经验，至今还没有犯过糊涂。

创建上下排列的窗口

如果想把当前窗口以水平方式分割为两个，请按下“**C-x 2**”组合键。重复这个操作将分割出更多的窗口。如下所示：

初始状态：

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way-in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.

-----Emacs:dickens          (Text Fill)--L1--All----------------------
```

按下：**C-x 2**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--L1--All----------------------
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--L1--All----------------------
```

屏幕被分割为两个水平窗口；每个窗口都有自己的状态行。

可以让Emacs在开始编辑工作时就把窗口安排好。如果想用水平窗口同时对两个文件进行编辑，可以在启动Emacs时同时给出这两个文件的名字。比如，如果想编辑*dickens*和*joyce*这两个文件，就应输入“**emacs** *dickens joyce*”；Emacs将把这两个文件分别显示在两个水平窗口里。如果想打开两个以上的文件，Emacs将显示两个水平窗口：在一个窗口里显示着某个文件，在另一个窗口里给出一份编辑缓冲区的清单。

创建左右排列的窗口

如果想把窗口垂直地分割为两个相邻的窗口，请按下“**C-x 3**”组合键。重复这个操作将分割出更多相邻的窗口。如下所示：

按下：**C-x 3**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the w$|It was the best of times,it was $
wisdom,it was the age of foolishness $|wisdom,it was the age of foolish$
was the epoch of incredulity,it was t$|was the epoch of incredulity,it $
season of Darkness,it was the spring $|season of Darkness,it was the sp$
despair,we had everything before us,$|despair,we had everything before$
-----Emacs:dickens          (Text Fill ----Emacs:dickens (Text Fill
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--L1--All----------------------
C-x 3
```

这个操作将把屏幕分割为两个垂直窗口。

对窗口进行了垂直分割之后，屏显画面的宽度通常就不够Emacs显示一个完整的文本行。因为垂直窗口一般显示不出完整的文本行（行尾的美圆符号“$”表示这一行没有被全部显示出来），所以大家必须知道怎样才能让文本做左、右卷动。

如果想把当前显示的文本向左推（好让自己看到右边的文本），请按下“**C-x <**”组合键。如果想把当前显示的文本向右推（这样就能看到左边的文本），请按下“**C-x >**”组合键。不管是不是发生在窗口里，只要有文本行超出了屏幕画面的宽度，就可以使用这两个命令（特别是在关闭了自动换行功能的场合）。

在窗口之间移动

按下“**C-x o**”组合键（这个命令里的字母“**o**”代表“*other*”，中文意思是“另一个”）就可以从一个窗口移动到另一个窗口。Emacs 在窗口之间的移动情况如图4-1所示。如果屏幕上显示有两个以上的窗口，Emacs将沿顺时针方向移动到下一个窗口去。由于Emacs没有提供移动到指定窗口去的方法，因此，如果屏幕上显示有两个以上的窗口，那么可能要按几次“**C-x o**”组合键才能到达想去的窗口。不过，可以用“**ESC** *n* **C-x o**”（“*n*”是一个数字）命令沿顺时针方向一次移动 *n* 个窗口。

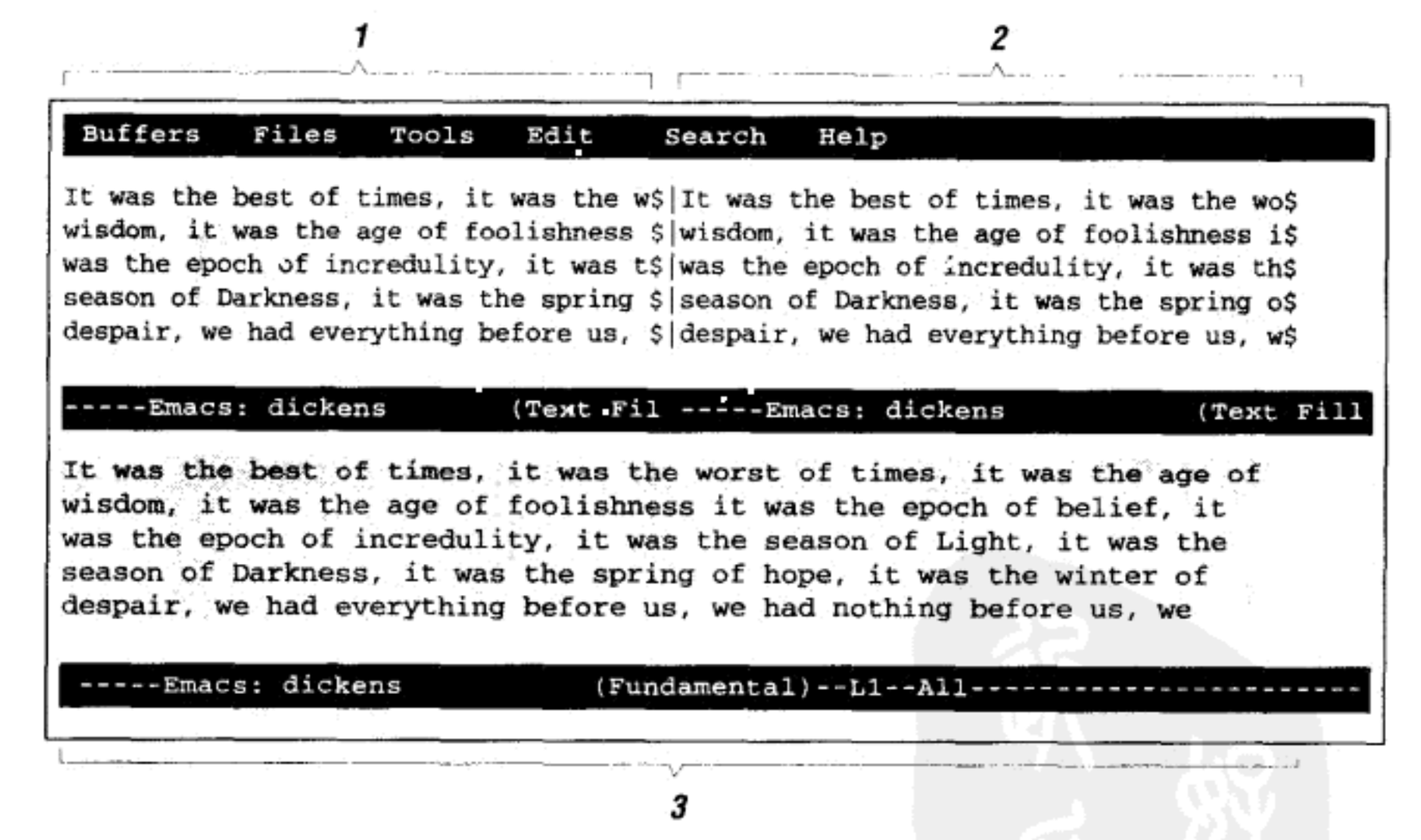


图 4-1：按顺时针方向在窗口之间移动

现在，既能创建出两个窗口，又知道了如何在它们之间移动，还能做些什么？回答是任何事情。看看下面的例子。首先，打开两个dickens窗口，两个窗口呈水平排列。刚开始时，这两个窗口里显示的是同一个文件。

按下：**C-x 2**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--L1--Top----------------------
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--L1--Top----------------------
```

在 *dickens* 上打开了两个窗口。

可以在任意一个窗口里执行任何编辑命令。在一个窗口里的前后移动不会对另一个窗口产生任何影响。下面来看看编辑另外一个文件时会发生什么事。

输入：**C-x C-f blake**

```
Buffers Files Tools Edit Search Help
Mock on Mock on Voltaire Rousseau
Mock on Mock on tis all in vain
You throw the sand against the wind
And the wind blows it back again.

-----Emacs:blake            (Text Fill)--L1--All----------------------
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
-----Emacs:dickens          (Text Fill)--Top--------------------------
```

现在有两个窗口、两个编辑缓冲区和两个文件。

利用“**C-x o**”组合键可以依次编辑这两个文件。可以在一个编辑缓冲区里删除一段文字，然后在另一个编辑缓冲区里把它恢复（即粘贴）回来。比如，准备把Blake（一位英国诗人）的诗歌的第一行移动到dickens编辑缓冲区的最顶部。如下所示：

输入：**C-k C-x o ESC < C-y RETURN**

```
Buffers Files Tools Edit Search Help

Mock on Mock on tis all in vain
You throw the sand against the wind
And the wind blows it back again.

--**-Emacs:blake          (Text Fill)--L1--All----------------------
Mock on Mock on Voltaire Rousseau
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
--**-Emacs:dickens        (Text Fill)--L2--Top----------------------
```

Blake 的诗歌中的文字被复制到 dickens 编辑缓冲区里。

能够在不同的窗口里对不同的编辑缓冲区进行编辑是一种很有用的功能，特别是想把一个文件的内容复制到另外一个文件里，或者想一边查阅着某个内容为参考资料的文件，一边编辑着另一个文件的时候。程序员经常需要同时查看好几个文件，比如同时查看一个头文件和一个源代码文件，或者同时查看某个函数调用点和那个被调用的例程等。一旦熟悉并掌握了在不同窗口之间进行移动的命令，就会经常在屏幕上同时打开两个、三个甚至四个窗口进行工作。

窗口的删除

删除一个窗口只意味着它将不再出现在屏幕上，而它里面的任何内容和没有存盘的任何修改都不会丢失；与之对应的编辑缓冲区也还在那里，用“**C-x b**”组合键就能切换过去。如果想删除所在的窗口，请按下“**C-x 0**”（数字 0）组合键。如果想把其他的窗口都删除，只保留自己正在其中进行工作的窗口，请按下“**C-x 1**”（数字 1）组合键——意思是“把当前窗口作为我惟一的窗口”；或者从“**Files**”菜单里选择执行“**One Window**（一个窗口）”操作。保留下来的窗口将“膨胀”到填满剩余的空间。还可以用“**ESC x delete-windows-on RETURN** *buffername* **RETURN**”命令来删除某个特定编辑缓冲区上的全部窗口。

调整窗口的大小尺寸

Emacs 在对窗口进行分割时，永远会把它分割成同样大小的两块。这样的分割在多

数情况下会正好满足要求，但有时候却未必如此，特别是在打开了很多窗口的时候。如果屏幕上同时有四个、五个甚至六个窗口，适当控制各个窗口的大小就将变得很重要。否则，最感兴趣的窗口可能会变得非常之小；而当只能看到文件的五、六行内容时，想对它进行有效的编辑就几乎是不可能的。如果想加高正在其中工作的窗口，请按下“**C-x ^**”组合键；Emacs 将加高当前窗口，它下方的窗口将被相应地压低。如果你想加宽当前窗口，请按下“**C-x }**”组合键；Emacs 将加宽这个窗口，它右方的窗口将相应地变窄。

如果想让窗口变小一点，则可以收缩它们。如果想在垂直方向上收缩窗口，请输入“**ESC x shrink-window**”命令（注 3）。Emacs 将压低当前窗口一行，屏幕上的其他窗口将被相应地加高。如果想在水平方向上收缩窗口，请使用“**C-x {**”命令。这个命令将压窄当前窗口一列，其他窗口将在水平方向上做相应地扩展。

人们通常更喜欢以比较大的步子，而不是一次一行或一次一列地来调整窗口的大小尺寸。如果在这些命令的前面加上一个“**C-u**”，就能以一次四行或一次四列的方式来执行这些命令。举个例子，用“**C-u C-x ^**”来加高 dickens 窗口。

按下：**C-u C-x ^**

```
Buffers Files Tools Edit Search Help
Mock on Mock on Voltaire Rousseau
Mock on Mock on tis all in vain
--**-Emacs:blake          (Text Fill)--L1--Top----------------------

It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
-----Emacs:dickens        (Text Fill)--L1--All----------------------
C-u C-x ^
```

正如大家所预期的，当让一个窗口变大的时候，将会有一些原来看不见的编辑缓冲区文本填补进来。下面再介绍几个用来调整窗口尺寸的快捷操作。如果有一个非常

注 3：　如果你需要经常做这个操作，可以考虑把它绑定到一个组合键上去。具体步骤请参考第十一章。

小的编辑缓冲区（假设一天下来，用来定义简写词条目的编辑缓冲区里只有一行文本），就可以用“**C-x -**”（命令名是**shrink-window-if-larger-than-buffer**）组合键让窗口收缩到编辑缓冲区那么小。如果编辑缓冲区的尺寸比窗口大，这条命令就什么也不做。按下“**C-x +**”（命令名是**balance-windows**）组合键将使窗口的尺寸全都变成同样的大小。（“**C-x +**”命令在打开的窗口个数是一个奇数时也很有用，它会让那些窗口之间的显示都呈同样的大小。）

窗口大小的上、下限

Emacs里的窗口最大不能超过屏幕，而最小则要看Emacs变量**window-min-height**（默认值是4行）和**window-min-width**（默认值是10个字符）的取值情况。当扩大某个窗口的时候，如果其他窗口被压缩到宽度小于10个字符或高度小于4行，Emacs就会删除它们。可以根据自己的具体情况对这两个变量的值进行设置。变量设置方面的详细讨论请参考第十一章。

对其他窗口进行操作的快捷命令

Emacs提供了几个对其他窗口进行操作的命令，这样不必移动到另一个窗口里就能对它进行操作。这些操作里最常用的是卷动其他窗口里的文本内容。如果想卷动（按顺时针方向）下一个窗口里的内容，请使用“**ESC C-v**”命令。

有几个“其他窗口”命令其实是一些中间插有数字“4”的普通命令。比如，如果想在另外一个窗口里查找并打开一个文件，就要按下“**C-x 4 f**”组合键。（如果当前只打开了一个窗口，那么Emacs会自动打开一个新窗口。）如果想在另外一个窗口里选择另一个编辑缓冲区，就要按下“**C-x 4 b**”组合键。与正常的“**C-x C-f**”和“**C-x b**”命令相比，在进行多窗口操作的时候，许多用户更喜欢这里介绍的命令，因为它们能够省略一个操作步骤——不需要移动到那个窗口，给出一命令，然后再移动回来。

对比两个窗口中的文件

如果想找出两个大文件之间细微的不同之处——比如想看看自己正在编辑的文件

和它的自动保存文件有什么不同，**compare-windows**命令就可以派上用场。先找到并打开有关文件，然后用**compare-windows**命令找出它们之间的第一个不同之处。

要想使用**compare-windows**命令，必须先把将要进行比较的两个编辑缓冲区分别在两个窗口里打开，它们可以左右并排，也可以水平排列（注4）。然后把两个编辑缓冲区的插入点分别移动到各自的开头，然后输入“**ESC x compare-windows**”或者从“**Tools**（工具）”菜单里选择“**Compare**（比较）”再选择“**This Window And Next Window**（本窗口和下一窗口）”。Emacs会把这两个编辑缓冲区都翻卷到出现不同之处的地方，并且会把两个编辑缓冲区各自的插入点，都放到出现不同之处的位置。这样，只需按下“**C-x o**”组合键在两个编辑缓冲区之间移动光标，就能看出这两个文件到底有哪些不同。

上面的操作只能找到两个编辑缓冲区之间出现的第一个不同之处。如果想找出第二个、第三个或者更多的不同之处，就得动些脑筋了。**compare-windows**命令只有在两个编辑缓冲区的插入点都精确地处于位置时才能工作，所以必须把它们在两个编辑缓冲区里分别移过第一个不同之处以后才能再次输入“**ESC x compare-windows**”。在找出两个文件之间的不同之处方面，UNIX操作系统的**diff**命令提供了更有效的办法（尽管它的输出看着有点费劲）。Emacs还准备了一个面向**ediff**的接口，在“**Tools**”菜单的“**Compare**”子菜单里也有一个对应的操作选项。

窗口操作方面的重要命令我们就介绍这么多，表4-1对它们进行了汇总。

表4-1：窗口命令速查表

键盘操作	命令名称	动作
C-x 2 *Files→Split Window*	**split-window-vertically**	把当前窗口分割为上、下排列的两个窗口
C-x 3	**split-window-horitontally**	把当前窗口分割为左、右排列的两个窗口
C-x >	**scroll-right**	窗口内容右卷
C-x <	**scroll-left**	窗口内容左卷

注4：在屏幕上可以有两个以上窗口，但只有两个窗口能够参加比较：光标所在的那个窗口和按顺时针方向的下一个窗口。

表 4-1: 窗口命令速查表（续）

键盘操作	命令名称	动作
C-x o	**other-window**	移动到其他窗口；如果有多个窗口，按顺时针方向移动到下一窗口
C-x 0	**delete-window**	删除当前窗口
C-x 1 *Files→One Window*	**delete-other-windows**	删除所有窗口，只保留当前窗口
（无）	**delete-windows-on**	删除某个给定编辑缓冲区上的所有窗口
C-x ^	**enlarge-window**	加高当前窗口
（无）	**shrink-window**	压低当前窗口
C-x }	**enlarge-window-horizontally**	加宽当前窗口
C-x {	**shrink-window-horizontally**	压窄当前窗口
C-x -	**shrink-window-if-larger-than-buffer**	如果编辑缓冲区比窗口小，就压缩窗口面积
C-x +	**balance-windows**	把所有窗口调整为同样大小
ESC C-v	**scroll-other-window**	对其他窗口做卷屏操作
C-x 4 f	**find-file-other-window**	在其他窗口里查找并打开一个文件
C-x 4 b	**switch-to-buffer-other-window**	在其他窗口里选择一个编辑缓冲区
（无） *Tools→Compare→This Window And Next Window*	**compare-windows**	对两个编辑缓冲区的内容进行比较，并显示它们之间的第一个不同之处

查看编辑缓冲区清单

Emacs允许在一次编辑工作中创建任意多个编辑缓冲区，可要是打开的编辑缓冲区太多，就不一定都能记住。好在随时都能查到一个编辑缓冲区的清单表。这个清单

表能够提供许多重要的信息，比如上次存盘后是否又对某个编辑缓冲区进行过修改等。

无论何时，只要按下“**C-x C-b**”组合键，Emacs就会把编辑缓冲区清单表列出来。它会在屏幕上创建出一个新的“*Buffer List*”窗口，所有的编辑缓冲区就列在里面。从“**Buffers**”菜单里选择执行“**List All Buffers**（列出所有的编辑缓冲区）”操作也可以调出编辑缓冲区清单。

按下：**C-x C-b**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
season of Darkness,it was the spring of hope,it was the winter of
despair,we had everything before us,we had nothing before us,we
were all going direct to Heaven,we were all going direct the other
way—in short,the period was so far like the present period,that
some of its noisiest authorities insisted on its being received,for
good or evil,in the superlative degree of comparison only.
-----Emacs:dickens          (Text Fill)--L1--All----------------------

MR Buffer        Size Mode        File
-- ------        ---- ----        ----
. dickens        608  Text        /home/deb/dickens
*  blake          135 Text        /home/deb/fiction/blake
   joyce            0 Text
   *scratch*        0 Lisp Interaction
*  *Messages*     884 Fundamental
   *mail*         386 Mail
 % RMAIL        78648 RMAIL       /home/deb/RMAIL
   *Completions*  107 Completion List
*% *Buffer List*  414 Buffer Menu

--%*-Emacs:*Buffer List*      (Buffer Menu)--L3--All-------------------
```

Emacs显示编辑缓冲区的清单。

一般来说，这个画面将至少会列出两个不是自己亲自创建的编辑缓冲区。“*Buffer List*”是按下“**C-x C-b**”组合键查看编辑缓冲区清单时由Emacs创建的；而“*scratch*”缓冲区则是每次启动Emacs做文字处理工作时，都会由Emacs自动创建的一个空白编辑缓冲区。可以把这个清单用做一个提示性的资料窗口（“我的编辑缓冲区”），也可以从这个清单开始对编辑缓冲区进行处理，就像将在下一小节里介绍的那样。图4-2给出了编辑缓冲区清单里各种符号的含义。

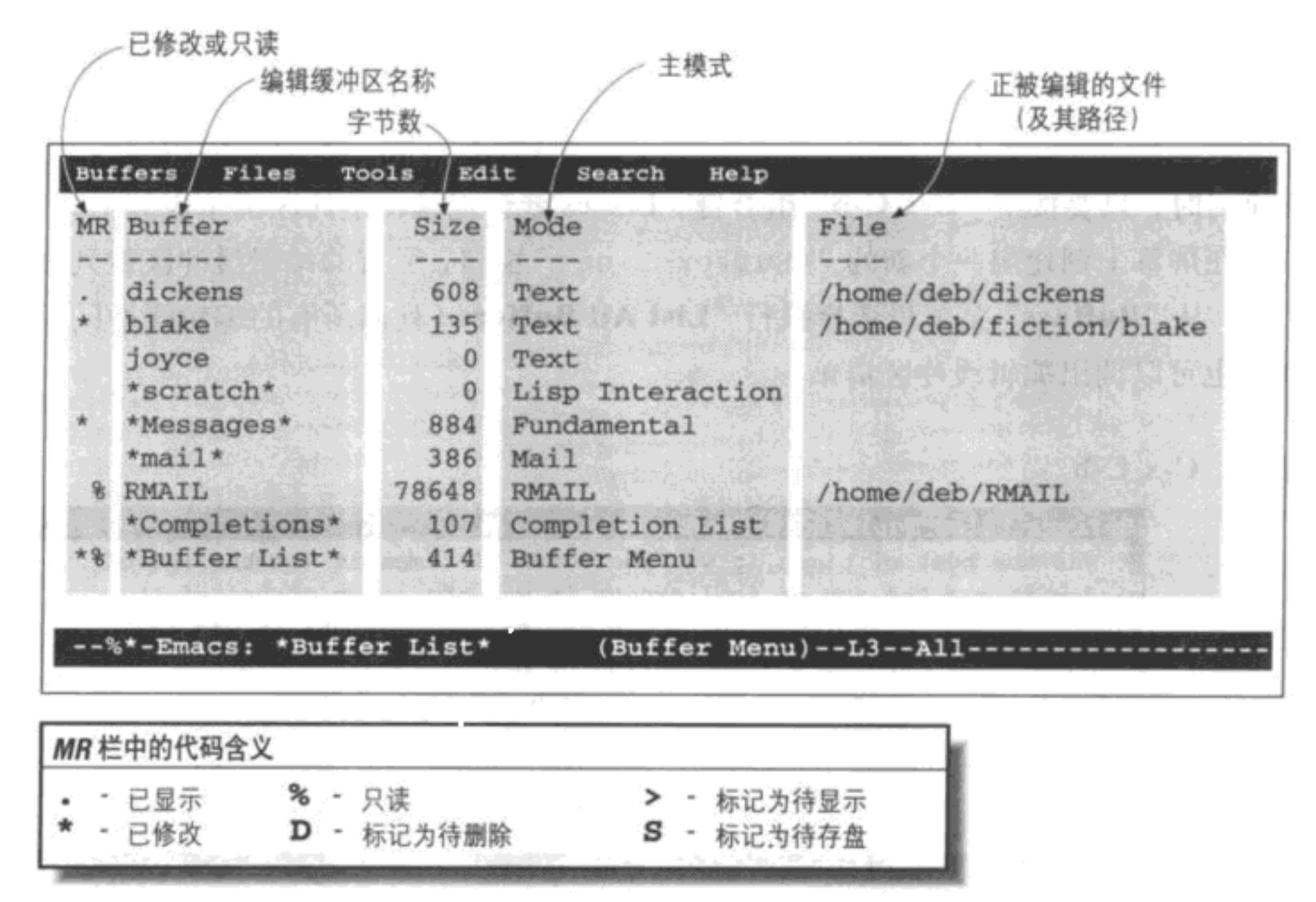


图 4-2: 了解编辑缓冲区清单

与编辑缓冲区清单有关的操作

编辑缓冲区清单可不仅仅是一个提示性的资料窗口，可以从编辑缓冲区清单入手对编辑缓冲区进行显示、删除和保存等操作。按下“**C-x o**”组合键就能进入编辑缓冲区清单窗口。Emacs 将把光标放在第一列的位置上。在编辑缓冲区清单里，按“**C-n**”下移一行，按“**C-p**”上移一行，也可以按空格键来下移一行。在这种情形里（或者类似的其他情形里，比如将在下一章介绍的子目录编辑功能），空格键和“**C-n**”的作用是相同的。

可通过一系列单字符命令来对列在清单里的编辑缓冲区进行操作。如果想删除某个编辑缓冲区，则先把光标移动到该编辑缓冲区所在的那一行，然后按下“**d**”或“**k**”键；大写字母“**D**”将出现在第一列里。可以给任意多个编辑缓冲区加上待删除标记。这些编辑缓冲区并不会立刻被删除；只有在完成编辑缓冲区的标记工作并按下“**x**”（表示“execute”执行）键之后，它们才会被真正地删除。如果准备删除的编辑缓冲区与某个文件有关联，Emacs 会询问在继续执行任何操作之前，是否想保存

所做的修改。(注意：如果编辑缓冲区与文件没有关联，Emacs就不会进行提问，所以在删除它们之前千万别忘了要先存盘。)

如果在按下“**x**”键之前又不想删除某个编辑缓冲区了，可以移动到相应的那一行并按下“**u**”键去掉待删除标记。更简便的办法是用**DEL**键来去掉上一个编辑缓冲区上的操作标记。为什么要这样做呢？原因很简单：“**d**”命令会自动前进到下一行。如果刚给一个编辑缓冲区加上待删除标记就改了主意，按一下**DEL**键当然要比按“**C-p u**”（移动到上一行，再去掉待删除标记）组合键简单。

如果想对某个编辑缓冲区进行存盘操作，则先把光标移动到该编辑缓冲区所在的那一行，然后按下“**s**”键；大写字母“**S**”将出现第一列里；再按下“**x**”完成真正的存盘动作。这样，可以一边查看着编辑缓冲区清单，一边给编辑缓冲区分别加上待删除标记或待存盘标记，最后按下“**x**”一次完成全部的工作。注意：如果你改了主意，“**u**”或**DEL**键也可以用来去掉待存盘标记。

有个命令是一经使用就会立刻作用于编辑缓冲区的，它就是“~（波浪号）”。按下“~”键将给编辑缓冲区加上一个未修改标记。这个符号的作用是告诉Emacs不要自动保存这个编辑缓冲区里的修改（既然这个编辑缓冲区没有被修改过，Emacs也就没有必要通过它的自动保存功能来保存对它的修改）。如果此前真的做过一些修改，它们当然还都在编辑缓冲区里（只是在“欺骗”Emacs说没有对那个编辑缓冲区做过修改而已）。如果在给编辑缓冲区加上未修改标记之后又对它进行了修改，Emacs将觉察到它又被进行过改动，并且会自动地把它保存到一个备份文件里去。备份文件的文件名格式是“*filename*~”。

用百分号“%”键可以把一个编辑缓冲区的状态从读－写改变为只读。按下“%”键将立刻改变编辑缓冲区的读写状态。当编辑缓冲区处于只读状态时，它的状态行上将出现两个百分号（%%）。当正在对编辑缓冲区进行编辑的时候，可以用“**C-x C-q**”组合键来切换它的读－写和只读状态（我们在前面曾经介绍过）。

还可以通过编辑缓冲区清单把多个编辑缓冲区显示到多个窗口里去。如果想让某个编辑缓冲区满屏显示，则先要把光标移动到显示着编辑缓冲区清单的窗口里，再用“**C-p**”或“**C-n**”组合键移动到想要的编辑缓冲区条目上，然后按下“**1**”（数字1）键。Emacs将把这个编辑缓冲区满屏显示。

如果想用某个编辑缓冲区替换编辑缓冲区清单，可以按下“**f**”键。如果想把某个编辑缓冲区放到另外一个（不是编辑缓冲区清单所占据的）窗口里，要按下“**o**”键；Emacs将把这个编辑缓冲区显示到另外一个窗口里，并且会把光标也放到那个窗口里。按下“**C-o**”组合键的结果稍有些不同：Emacs将把这个编辑缓冲区显示到另外一个窗口，但不会把光标也放到那里。

最后，还有一个编辑缓冲区显示命令。可以让Emacs动态地创建多个窗口来显示多个编辑缓冲区。先挑选出准备在窗口里显示的编辑缓冲区，方法是在你想要的编辑缓冲区条目按下“**m**”（意思是“mark”，标记）键；Emacs在用“**m**”键标记的编辑缓冲区旁显示一个大于号“**>**”。挑选完毕后按下“**v**”键，Emacs将自动创建一些水平窗口来显示那些加上“**>**”标记的编辑缓冲区。

删除“`*Buffer List*`”窗口的办法是：如果在编辑缓冲区清单窗口里，按下“**C-x 0**”组合键（数字0）；如果在某个编辑缓冲区的窗口里，按下“**C-x 1**”组合键（数字1）。

表4-2对与编辑缓冲区操作有关的命令进行了汇总。

表4-2：编辑缓冲区操作命令速查表

键盘操作	命令名称	动作
C-x b	**switch-to-buffer**	移动到指定的编辑缓冲区
C-x C-b *Buffers→List All Buffers*	**list-buffer**	显示编辑缓冲区清单
C-x k *Files→Kill Current Buffer*	**kill-buffer**	删除指定的编辑缓冲区
（无）	**kill-some-buffers**	以问答方式删除各个编辑缓冲区
（无）	**rename-buffer**	把编辑缓冲区的名字改为指定的名字
C-x s	**save-some-buffers**	以问答方式存盘各个修改过的编辑缓冲区

表4-3对与编辑缓冲区清单有关的操作命令进行了汇总。

表 4-3: 编辑缓冲区清单操作命令速查表

键盘操作	动作	执行情况
C-x n	移动到清单里的下一个编辑缓冲区（即编辑缓冲区清单里的下一行）	立即执行
SPACE	移动到清单里的下一个编辑缓冲区	立即执行
C-p	移动到清单里的上一个编辑缓冲区（即编辑缓冲区清单里的上一行）	立即执行
d	给编辑缓冲区加上待删除标记	按下“**x**”键时执行
k	给编辑缓冲区加上待删除标记	按下“**x**”键时执行
s	给编辑缓冲区加上待存盘标记	按下“**x**”键时执行
u	去掉编辑缓冲区上的操作标记	立即执行
x	对加有操作标记的所有编辑缓冲区执行相应的操作	立即执行
DEL	去掉上一个编辑缓冲区上的操作标记	立即执行
~	给编辑缓冲区加上未修改标记	立即执行
%	转换编辑缓冲区的只读状态	立即执行
1	把编辑缓冲区满屏显示	立即执行
2	把这个编辑缓冲区和下一个编辑缓冲区显示到两个水平窗口里	立即执行
f	在原本显示编辑缓冲区清单的窗口里显示此编辑缓冲区的内容	立即执行
o	把此编辑缓冲区显示到另外一个窗口里	立即执行
m	给编辑缓冲区加上待显示标志。参见“**v**”命令的说明	按下“**v**”键时执行
v	显示用“**m**”命令加上待显示标志的编辑缓冲区。Emacs 将动态地创建足够的窗口来显示加有这类标志的编辑缓冲区	立即执行
q	退出编辑缓冲区清单	立即执行

PDG

在文档中使用书签

用多个文件进行工作的时候，要想记住自己刚才正在什么地方进行编辑是一件很不容易的事情。幸好Emacs准备了书签功能，这是一种记录在文件中位置的简便办法，通过它容易就能回到书签所在的位置。假设正在编辑的文件有一个很长的路径名。每次启动Emacs时都必须输入这么长的路径名当然不是一件愉快的事，可以在这个文件里设置一个书签——假设给书签起的名字叫“*current project*”。利用这个书签，就可以让Emacs自动查找并打开这个文件，然后把光标放到当初设置这个书签时所指定的位置。

书签使文件中的定位工作变得简单了。特别是当前项目在主目录以下几层深的某个子目录里，或者在另外一个完全不同的文件系统里时，在文件里设置一些书签能够快速地到达那里。我们将在第七章里介绍如何通过ange-ftp模式来查找远程系统上的文件，用户完全可以给这些文件也加上书签。这样，只需一个简单的查书签操作，就能迅速回到最喜欢的FTP档案。

请注意，如果是从Emacs里访问World Wide Web（这部分内容也将在第七章里讨论）的，就不能互换使用Emacs书签和W3收藏夹条目。这主要是因为某些主流的浏览器把它们的收藏夹条目也叫做“书签”，弄清楚这一点是很重要的。

创建书签的时候，Emacs将在主目录里创建一个名为“*.emacs.bmk*”的书签文件。它会在退出Emacs时把新书签都自动添加到这个文件里。

书签是按用户分别保存的。如果你和其他人都在阅读同一篇在线文档（比如Emacs在线手册），你将用你的书签来保存你的阅读位置，别人则用他们自己的书签来保存他们自己的阅读位置，各自的阅读是不会相互干扰的。

从“**Search**”菜单选择“**Bookmarks**（书签）”也能到达“**Bookmarks**”菜单，它列出了全部的书签命令以供选用。我们认为Emacs的书签菜单界面设计得特别好；即使你通常不使用菜单，也可能会对它情有独钟——至少在你学会有关的键盘操作命令之前会是如此。书签功能很容易让人上瘾，频繁使用它们的时候输入键盘命令总要比通过菜单进行操作要快捷一些。

书签的设置

如果想在光标位置设置一个书签，需要按下“**C-x r m**”（命令名是**bookmark-set**）组合键或者从“**Bookmarks**”菜单选择执行“**Set Bookmark**（设置书签）”操作。Emacd提示给书签输入一个名字，书签名的长度从理论上讲没有限制（但实践中不能长过屏幕的宽度），而且里面还可以有空格（例如“current project”、“Moore proposal's greatest flaw”或者“Othello Act 2 Scene 4”等）。如果还没有在这次编辑工作中使用过书签，Emacs会把文件名放置在一对括号里作为默认书签（如果使用过书签，括号里放的就是书签的名字）。直接按回车键将接受默认的书签名，或者先输入一个书签名再按回车键。现在，就有了一个可以在任何Emacs编辑工作当中随时跳转过来的书签。

注意，如果在给新书签起名字时使用了一个曾经用过的书签名，Emacs将认为是想用老书签来指示一个新位置，不管它原来是不是在另外一个文件里。所以，如果不是真的想移动书签的话，就一定要给书签起一个独一无二的名字。

移动到书签指示的位置

要想移动到书签指示的位置，请按下“**C-x r b**”（命令名是**bookmark-jump**）组合键。接着输入书签的名字，或者输入它的前几个字母再按下**TAB**键；Emacs将自动补足书签名，或者打开一个窗口把可能的书签名都列在其中。书签名出现后按下回车键。Emacs将查找并打开包含有这个书签的文件，并把光标放到书签位置上；不管文件的路径名有多么复杂，Emacs都能找到它。用几个“远点的文件”来试试就能掌握这个技巧（注5），说不定今后会几乎完全放弃使用文件**find-file**（查找文件）命令“**C-x C-f**”呢。

利用菜单也可以迅速移动到某个书签所指示的位置。从“**Bookmarks**”菜单（它本身是“**Search**”查找菜单里的一个选项）里选择执行了“**Jump to Bookmark**（跳转至书签）”操作时，Emacs会在一个窗口里把可用的书签都列出来。选择想要的书签，Emacs会把文件显示在屏幕上，而光标将被放在那个书签所指示的位置上。

注5：这里所说的“远点的文件”指的是那些路径名相当复杂的文件，它们可能位于你的本地系统上、位于本地网络的其他系统上，或者位于可以通过FTP检索到的因特网主机上。使用Emacs来访问因特网的有关内容请参考第七章中的讨论。

书签的重新命名和删除

有时可能会觉得书签名字太一般，如果手里有好几个工作项目，“current project”（中文意思是“当前工作项目”）这样的名字未免有点含糊其词，日子长了就难免忘记它指的到底是哪个东西了。对书签重新进行命名的方法是：输入“**ESC x bookmark-rename**”或者从“**Bookmarks**”菜单里选择执行“**Rename Bookmark**（重新命名书签）”操作。如果用键盘给书签改名，就需要在Emacs提示的“`Old bookmark name:`（旧书签名）”处输入书签旧名再按下回车键。（如果使用的是X操作界面，就可以从一个窗口里选取旧名。）Emacs接着会提示“`New name:`”（新名字），输入新的书签名再按下回车键。给书签重新命名就这么简单，它既不会改变书签的指示位置，也不会改变它的内容，只是给书签改了个名字而已。

删除书签的方法是：输入“**ESC x bookmark-delete**”或者从“**Bookmarks**”菜单里选择执行“**Delete Bookmark**”操作。输入准备删除的书签的名字或者用鼠标选取它。删除书签对它所标记的文件不会造成任何影响。

这里又引出一个有趣的问题：如果在文件中删除了原来放有一个书签的文本时，会发生什么样的事情？因为书签指向文件里的一个位置而不是一段文本，所以文本被删除后，书签仍旧指向原来的位置——就像删除几个段落之后光标位置不会发生任何变化一样。自己实践一下就能明白了。用删除书签位置上文本的办法是删不掉书签的。

那如果删除一个原来包含着书签的文件时又会发生什么样的事情？如果删除了整个文件，或者哪怕只是给文件改了个名字，那么当再次访问关联于那个文件的书签时，Emacs会显示一条如下所示的出错信息：

```
filename nonexistent. Relocate "bookmark name"? (y or n)
```

如果只是对进行文件进行了重命名或移动而不是删除操作，回答“**y**”之后就可以把文件的新路径输入进去。如果回答“**n**”，Emacs会再显示一条提示信息：

```
Bookmark not relocated, consider removing it
```

换句话说，Emacs认为没有人会需要指向并不存在的文件的书签，我们也这样认为。

与书签清单有关的操作

还记得刚才介绍过的编辑缓冲区清单吗？书签也有一个类似的清单。与书签清单有关的操作也都是通过一些单字符命令完成的，这些命令能够一次处理好全部的书签。

开始对书签清单进行编辑的方法是：按下“**C-x r l**”（小写字母“L”）组合键或者从“**Bookmarks**”菜单里选择执行“**Edit Bookmark List**”操作。“*Bookmark List*”编辑缓冲区将出现在屏幕上。如下所示：

按下：**C-x r l**

```
Buffers Files Tools Edit Search Help
% Bookmark
- --------
R6source information            /source/README
  RMAIL                          /home/deb/RMAIL
  blake                          /home/deb/fiction/blake
  dickens                        /home/deb/fiction/dickens
  heidegger                      /home/deb/fiction/heidegger
  hjames                         /home/deb/fiction/hjames
  joyce                          /home/deb/fiction/joyce
  capitalization                 /online/ref/chicago

--%*-Emacs: *Bookmark List*    (Bookmark Menu)--L3--All----------------
Loading bookmark...done
```

Emacs 把所有的书签及其关联文件的路径都列在这个清单里。

在这个清单里，“**d**”键给书签加上待删除标记，“**x**”键真正删除掉它们（与编辑缓冲区清单不同的是，书签清单里只有删除操作需要二次使用“**x**”命令）。如果改变了主意，在按下“**x**”键之前还可以用**DEL**键去掉“**d**”命令设置的待删除标记。按“**r**”键对书签进行重新命名，Emacs 会提示输入一个新的名字。如果想把书签全都保存起来，请按“**s**”键。如果想查看与书签关联着的文件，可以先用“**m**”键给书签加上待显示标记，书签旁边将出现一个大于号“**>**”标记。给所有想查看的书签都加好标记后，按“**v**”键，Emacs 会把与这些书签关联着的文件都打开在多窗口里（光标的位置不用说都应该在书签所指示的位置上）。如果只想进入某一个带书签的文件，可以不用先给它加上待显示标记，直接按“**v**”键即可，单按“**f**”键也行。

按“**t**”键可以对书签清单的显示方式稍加改变。在默认的情况下，这个清单先列出书签名，然后是其关联文件的完整路径。如果按下了“**t**”键，它就只列出书签名。

表 4-4 对与书签清单有关的命令进行了汇总。

表 4-4: 书签清单编辑命令速查表

命令	动作
d	给书签加上待删除标记
r	对书签重新命名
s	保存清单里的全部书签
f	显示光标位置上的书签
m	给书签加上待显示标记。
v	显示加有待显示标记的书签。如果没有加上待显示标记的，就显示光标所在处的那个书签
t	切换书签关联文件的路径的显示 / 不显示状态
w	显示书签关联文件的存放位置（即路径名）
x	删除加有待删除标记的书签
u	去掉书签上的待操作标记
DEL	去掉上一行书签上的待操作标记
q	退出书签清单

与书签有关的其他编辑命令

除了前面介绍的这些操作以外，书签还有一些奇妙的操作命令。其中包括：**bookmark-insert**，用来把书签文件中的文本插到光标位置上；**bookmark-write**，提示输入一个用来保存书签的新文件名；**bookmark-load**，加载其他书签文件等等。这些命令不常用，不过也许各位会替它们想出一些合适的用途来。通过“**Bookmarks**”菜单也能访问这些命令。

表 4-5 对书签命令进行了汇总，包括“**Bookmarks**”菜单里的有关选项。

表 4-5: 书签命令速查表

键盘操作	命令名称	动作
C-x r m *Search→Bookmarks→ Set Bookmark*	**bookmark-set**	在当前光标位置处设置一个书签
C-x r b *Search→Bookmarks→ Jump to Bookmark*	**bookmark-jump**	跳转到书签指示位置
（无） *Search→Bookmarks→ Rename Bookmark*	**bookmark-rename**	重新命名一个书签
（无） *Search→Bookmarks→ Delete Bookmark*	**bookmark-delete**	删除一个书签
（无） *Search→Bookmarks→ Save (in default file)*	**bookmark-save**	把书签全都保存到默认的书签文件里
C-x r l *Search→Bookmarks→ Edit Bookmark List*	**bookmark-menu-list**	进入“*Bookmark List*”编辑缓冲区
（无） *Search→Bookmarks→ Insert Content*	**bookmark-insert**	插入与给定书签关联着的文件的完整内容
（无） *Search→Bookmarks→ Write (to another file)*	**bookmark-write**	把书签全都保存到一个指定的文件里
（无） *Search→Bookmarks→ Load a Bookmark file*	**bookmark-load**	从指定文件里加载书签

临时性地挂起 Emacs

在大多数情况下，都能用“**C-z**”组合键把 Emacs 的运行暂停下来。如果使用的是

一个字符终端，就会到达UNIX操作系统的命令行提示符。重新返回Emacs的方法有两种，一种是在UNIX的命令行提示符处输入“**fg**”命令（如果shell有作业控制功能），另一种是在UNIX的命令行提示符处输入“**exit**”命令（如果shell要使用一个子shell来仿真作业控制功能）。如果不能肯定自己的shell有没有作业控制功能，或者是一个对作业控制还不太了解的新用户，可以两个命令都试试，不会有什么害处。肯定会有一个命令能工作，而另一个命令会给你一条无害的出错信息。

如果在运行X窗口系统时按下“**C-z**”组合键，结果就会稍有不同。Emacs窗口将缩小为一个图标（图标可能是一只非洲羚羊的图案，也可能是一个流着水的水槽图案）。如果用鼠标打开这个图标（注6），Emacs窗口将恢复为原状。

重新回到Emacs时，一切都和刚才离开时一样。原来的文件还以原来的状态显示在屏幕上。甚至在挂起Emacs之前都用不着先把文件存盘（重新返回后再存盘也不迟）。如果不想让别人看到正在编辑的文字内容（比如，你正在为朋友安排一个惊喜聚会，可恰好有个朋友走进了你的办公室），你可以立刻把Emacs挂起来，不让他看到。话虽如此，我们还是建议大家在挂起Emacs之前最好把文件存盘，因为没人知道离开的时候都会发生哪些事情：忘记自己挂起过Emacs而去吃午饭了，可计算机恰好就在这时候出了问题，而你前面编辑的东西也就都找不回来了。

能够临时性地把Emacs挂起来，去做一些其他工作是很有用的，特别是当工作在一台老式终端，而不是一台现代化的图形工作站上时更是如此。我们再次告诫大家：人们很容易忘记自己有一个“暂停”的Emacs编辑工作而又重新开始一次，而编辑的还是原来的那个文件。这样做很容易导致严重的混乱，如果比较懒于对文件进行存盘，这种情况就更容易发生。可能会有多个Emacs任务在同时运行，而各个任务编辑的却是同一个文件稍有不同的几个版本。可以用UNIX操作系统的**ps**命令（虽然它的输出不太容易看得懂）来查看是否有遗忘的Emacs任务。下面的输出情况告诉我们机器里运行着两个Emacs任务：

```
% ps
  PID TT STAT  TIME COMMAND
  129 co IW    0:01 sunview
  131 co IW    0:01 selection_svc
  134 co IW    2:39 clock -Wp 497 32 -Ws 210 47 -WP 704 0 -Wi
```

注6：打开一个图标所需要的鼠标动作将取决于使用的窗口管理器；它可能是一次鼠标左键单击、一次鼠标左键双击或者是一次鼠标中键单击。

```
 135 co IW    1:01 mailtool -Wp 492 71 -Ws 670 770 -WP 908 0 -Wi
 137 co IW    0:00 Mail -N -B -f /tmp/MTda00135
3474 p2 S     1:16 emacs ch01.2.mS
3585 p2 T     0:01 emacs others
3484 p3 S     0:00 -bin/csh -i (csh)
3586 p3 R     0:00 ps
```

如果使用的shell支持**jobs**命令（比如**csh**和**ksh**），就能得到下面这种更清楚的输出：

```
% jobs
[1]  + Stopped              emacs ch01.2.mS
[2]  - Stopped              emacs others
```

如果不知道自己用的是哪一种shell，或者根本就不知道shell是什么，试试**jobs**命令也不会对你的系统造成任何损害，最多不过是看到一条“Command not found”（命令没有找到）的出错信息。

注意，**ps**命令和**jobs**命令都只能告诉Emacs任务是如何启动的，它们不能告诉正在进行的工作是什么。这两个命令给出的文件名，即例子里的*ch01.2.mS*和*others*，只是启动Emacs时指定的文件名。因为Emacs允许在编辑工作中随时打开和切换不同的文件，所以现在编辑的可能已经不是这两个文件了。

最后一句忠告：几乎永远没有任何理由离开Emacs。即使自由软件基金会明天就抛弃了“**C-z**”命令，也不要丢掉它。我们现在还不能说明这是为什么，学到第五章时自然就会明白。

使用多个X窗口进行编辑

如果使用X窗口系统，那么就不仅限于Emacs窗口，还可以使用多个X窗口进行工作。Emacs把X窗口称为“窗格（frame）”，这是为了把它们与前面已经介绍过的Emacs窗口区分开来。

Emacs有一套完整的用来对窗格进行操作的命令，这些命令与Emacs窗口操作命令很相似。比如，如果想在另一个窗口里查找一个文件，相应的命令是“**C-x 4 f**”；而如果想在另外一个窗格里查找一个文件，相应的命令则是“**C-x 5 f**”。

窗格操作命令

打开一个新窗格的方法是：从“**Files**”菜单里选择执行“**Make New Frame**”制作新窗格操作或者按下“**C-x 5 2**”（命令名是 **make-frame**）组合键。Emacs 会创建一个包含当前编辑缓冲区的新窗格，并把它放到当前窗格的上面。

如果新窗格完全遮盖住当前窗格，则可能需要对新窗格的大小进行调整以区分它们。作为一个相对一劳永逸的解决方案，可以把下面这几条语句添加到“*.emacs*”文件里：

```
(setq initial-frame-alist '((width . 75) (height . 34)))
(setq default-frame-alist '((width . 60) (height . 20)))
```

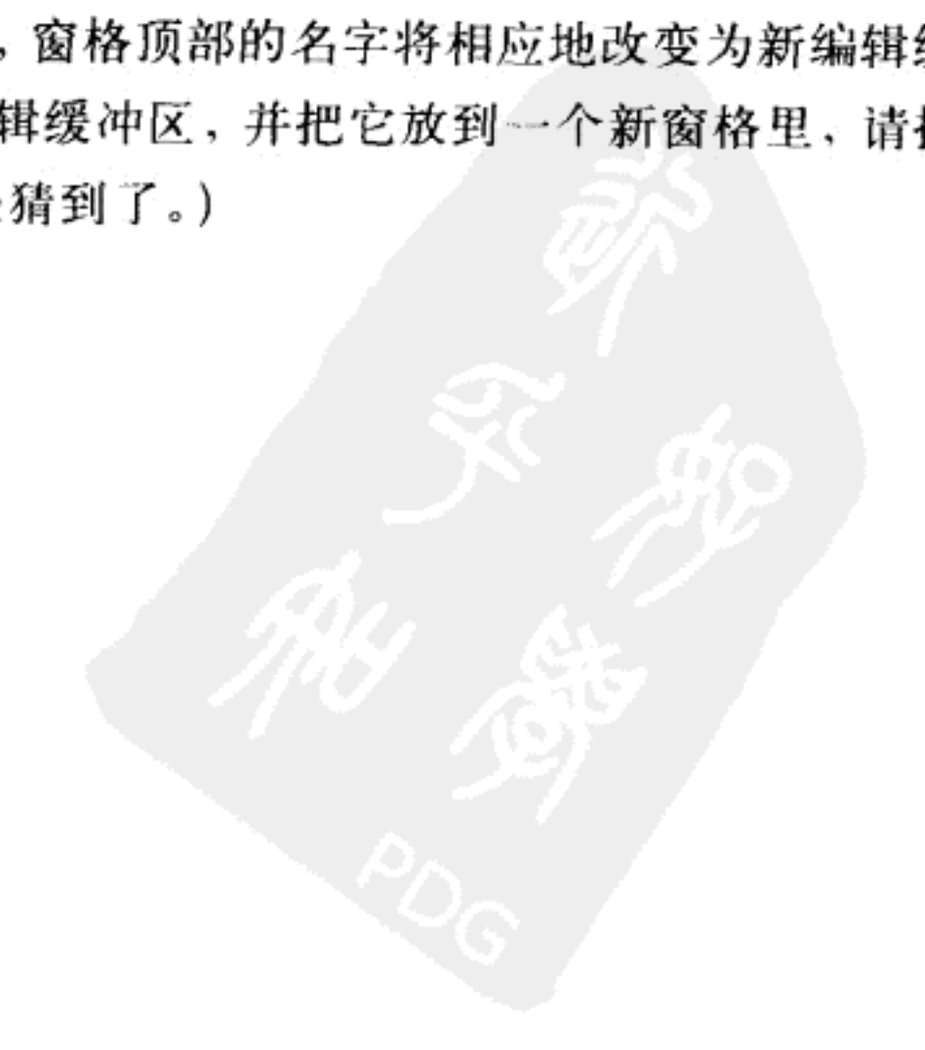
这些语句对 Emacs 窗格尺寸的宽度和高度进行了设置。第一个窗格的尺寸将采用 **initial-frame-alist** 中的设置值（在上例中，宽度是 75 个字符，高度是 34 行），后续的窗格将采用 **default-frame-alist** 中的设置值，即 60 个字符宽、20 个字符高。可以根据显示器屏幕的尺寸对这些数字进行上、下调整。

按下：**C-x 5 2**（如图 4-3 所示）：

请注意新窗格的顶部。Emacs 在默认的情况下会把编辑缓冲区的名字放在那里（如果只有一个窗格，Emacs 会把系统的名字放在其顶部。）。因为这里的情况是在同一个编辑缓冲区上有两个窗格，所以 Emacs 会把同样的名字放到新窗格的顶部。下面我们来在我们的 `dickens` 编辑缓冲区上打开一个窗格。如下所示：

输入：**C-x 5 f dickens RETURN**（如图 4-4 所示）：

新窗格的名字是 `dickens`，与它的编辑缓冲区同名。如果用“**C-x b**”组合键移动到另外一个编辑缓冲区，窗格顶部的名字将相应地改变为新编辑缓冲区的名字。（如果想移动到另外一个编辑缓冲区，并把它放到一个新窗格里，请按下“**C-x 5 b**”组合键 —— 大家可能已经猜到了。）



```
Buffers Files Tools Edit Search Help
My second correspondent, also a woman, sends me the following
statement:
     Life seemed difficult to me at one time.  I was always breaking
     down, and had several attacks of what is called nervous
     prostration, with terribl insomnia...
Here i                    hjames
these  Buffers Files Tools Edit Search Help
state  My second correspondent, also a woman, sends me the followi\
       ng
       statement:
            Life seemed difficult to me at one time.  I was always\
        breaking
            down, and had several attacks of what is called nervou\
       s
            prostration, with terrible insomnia...

       Here is another case, more concrete, also that of a woman. \
        I read you
       these cases without comment-- they express so many varietie\
       s of the
       state of mind we are studying.

       -----Emacs: hjames          9:59am 0.17    (Nroff Abbrev Fill)
-----Emacs: hjames          9:59am 0.17    (Nroff Abbrev Fill)--Bil----------
```

图 4-3: Emacs 打开了一个标题为“hjames”的新窗格

```
Buffers Files Tools Edit Search Help
My second correspondent, also a woman, sends me the following
statement:
     Life seemed difficult to me at one time.  I was always breaking
     down, and had several attacks of what is called nervous
     prostration, with terrible insomnia...
Here i                    dickens
these  Buffers Files Tools Edit Search Help
state          It was the best of times, it was the worst of times\
       , it was
               the age of wisdom, it was the age of foolishness, i\
       t was the
               epoch of belief, it was the epoch of incredulity, i\
       t was the
               season of Light, it was the season of Darkness, it \
       was the
               spring of hope, it was the winter of despair, we ha\
       d
               everything before us, we had nothing before us, we \
       were all
               going direct to Heaven, we were all going direct th\
       e other
               way--in short, the period was so far like the prese\
       nt period,
               that some of its noisiest authorities insisted on i\
       -----Emacs: dickens         10:05am 0.14    (Nroff Abbrev Fill
-----Emacs: hjames          10:05am 0.14    (Nroff Abbrev Fill)--Bil----------
```

图 4-4: Emacs 打开了一个标题为“dickens”的新窗格

有好几种办法可以用来在窗格之间移动。可以用鼠标选取窗格，也可以通过“**C-x 5 o**”组合键进入另外一个窗格。如果想查看所有当前窗格的一个清单列表，请在“**Buffer**”菜单里选择执行相应的操作。如果打开了一个以上的窗格，Emacs 会弹出

一个窗口让用户对编辑缓冲区或窗格进行选择。如果选择的是“**Frames**（窗格）”，Emacs就显示一个当前活动窗格的清单列表，可以用鼠标左键从中进行挑选。可以在“**Buffer**”菜单下来回切换显示编辑缓冲区清单和窗格清单。

如果想删除一个窗格，请按下“**C-x 5 0**”组合键或者从“**Files**”菜单里选择执行“**Delete Frame**（删除窗格）”操作；Emacs将删除所在的窗格。删除一个窗格和删除一个窗口都只影响屏幕上的显示情况。原先的编辑缓冲区依然是活动的，可以用“**C-x b**”组合键来删除编辑缓冲区。

假设你需要经常在线查阅一本参考书——比如说《Chicago Manual of Style》吧。你可以用“**C-x 5 r**”命令以只读方式打开一个窗格，来查阅这份文件的一个副本（因为你并不是想对这本参考书的内容进行编辑修改），如下所示：

输入：**C-x 5 r chicago**（如图4-5所示）：

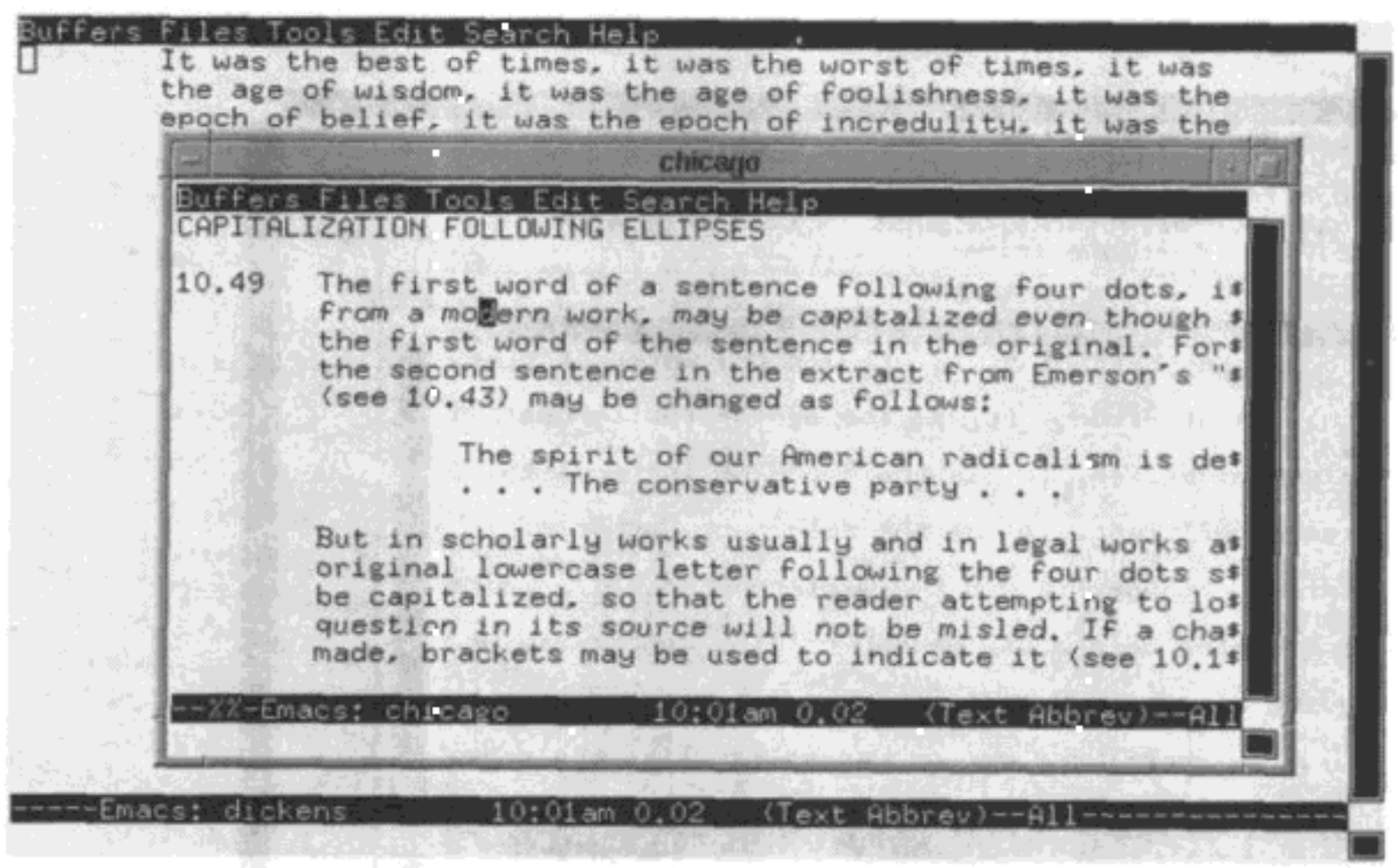


图4-5：Emacs打开一个新窗格查阅《Chicago Manual of Style》的内容

如果窗格遮住了屏幕上的其他东西怎么办？举个例子，只是偶尔查一下《Chicago Manual of Style》一书中的内容。那么，如果这个窗格遮住了屏幕上的其他东西，则可以把它挪到其他窗格的下面，或者就用“**C-z**”（命令名是**iconify-or-deiconify-frame**）组合键把它缩小为一个图标，可以随时用鼠标来重新打开它。利用“**Buffer**”

菜单来选取一个窗格，也会把它从图标恢复为原来的大小。（一旦窗格被缩小为图标，就不能再通过“**C-x 5 o**”组合键切换到它里面去；Emacs 会认为用户不想查看一个被缩小为图标的窗格。）

窗格与编辑缓冲区的关系

窗格和编辑缓冲区有着相当紧密的联系。Emacs会给窗格起一个与编辑缓冲区同名的名字，这我们已经看到了。如果想查看编辑缓冲区清单，那么所有的编辑缓冲区都会被列出来，显示在其他窗格里的那些编辑缓冲区也将列在这个清单里。窗格与窗口有很多相似之处，它们都只是用来显示编辑缓冲区内容的一种手段。如果准备退出 Emacs，它会询问是否想保存被修改过的编辑缓冲区。回答完这些问题之后，所有的窗格都将被关闭。

与多编辑缓冲区、多窗格和多窗口有关的操作，到这里就介绍得差不多了。下一章将介绍它们的一些实际应用情况，有些操作（比如，子目录编辑器的使用和在Emacs里启动 UNIXshell 等）已经在这一章里一笔带过地提到。表 4-6 对窗格命令进行了汇总。

表 4-6：窗格命令速查表

键盘操作	命令名称	动作
C-x 5 o *Buffers→Frames*	**other-frame**	移动到其他窗格
C-x 5 0 *Files→Delete Frame*	**delete-frame**	删除当前窗格
C-x 5 2 *Files→Make New Frame*	**make-frame**	在当前编辑缓冲区上创建一个新窗格
C-x 5 r	**find-file-read-only-other-frame**	创建新窗格并查找文件、把编辑缓冲区设置为只读的（用来查阅你不小心修改了的文件）
C-x 5 f	**find-file-other-frame**	在一个新窗格里查找文件
C-x 5 b	**switch-to-buffer-other-frame**	创建新窗格并在其中显示另外一个编辑缓冲区

疑难解答

- **所有窗格的名字都一样**。如果在对不同名字的多个编辑缓冲区进行编辑时遇到了这个问题，那么最可能的原因是“*.Xdefaults*”文件给Emacs窗格定义了一个统一的名字。需要对“*.Xdefaults*”文件进行编辑，删除出现“`emacs.title`”字样的那一行。保存“*.Xdefaults*”文件，退出Emacs，然后退出登录，再重新登录上机。Emacs将会根据正在编辑的编辑缓冲区给窗格加上标题。详细资料请参考第十四章。

- **菜单栏不见了**。菜单栏只有在使用Emacs 19.30或以上的版本，或者是在X窗口系统里才能工作。如果确实有X窗口系统而菜单栏真的不见了（这一般出现在使用窗格进行工作的时候），输入“**ESC x menu-bar-mode**”就可以把它找回来。

- **书签文件的名字是**“*.emacs-bkmrks*”而不是“*.emacs.bmk*”。这其实算不上是一个问题，这说明使用的Emacs版本低于19.29。如果升级到19.29或者更高的版本，Emacs将自动对这个文件进行转换，并且把它的名字改为“*.emacs.bmk*”。

本章内容：

- 在shell编辑缓冲区里执行UNIX命令
- 文件和目录操作
- Emacs中的打印操作
- 用Emacs查阅UNIX的在线文档
- 时间管理工具的使用
- 用好Emacs工作环境

第五章

Emacs工作环境

需要用UNIX的shell完成的许多工作，用Emacs也能完成得很好。诸如执行UNIX命令、对目录进行操作、打印文件之类的事情根本用不着离开Emacs。想转去进行另外一项工作吗？很简单，切换到相应的编辑缓冲区或窗口就行了。如果屏幕大得足以容纳多个窗口，就可以用“**C-x o**”组合键在任务之间进行切换。

这有什么重要的吗？当然很重要，能够在任务之间快速切换本身就很有好处；更重要的是不管正在做些什么，都会拥有一个统一的编辑环境：通过各种Emacs编辑命令，能够对文件进行文字处理、发出shell命令和启动目录编辑器做一些文件维护方面的工作等等。把文本从一个窗口移动到另外一个窗口，只能算是一种很简单的操作。可以执行一条UNIX命令，然后用Emacs命令把它的结果复制到某个文件里。虽然现代的窗口系统有很多优点，但就许多日常工作而言，Emacs所提供的集成性是最好的。在此后的各章里，我们将讨论Emacs在电子邮件、Usenet新闻、Telnet、FTP和WWW等方面的使用方法。这一章只能算是一个入门，目的是介绍怎样才能把各种日常工作集成到Emacs里去。

在shell编辑缓冲区里执行UNIX命令

Emacs的重要特色之一是它能够在一个编辑缓冲区里运行一个UNIX的shell。只要进入shell编辑缓冲区，就可以在Emacs里完成各种常见的UNIX工作。这样做有哪些好处呢？

- 不必离开Emacs就能得到UNIX命令行提示符。如果想对正在编辑的文件进行打印或者编译，完全可以立即动手。
- 可以利用Emacs的编辑功能写出自己的命令。
- 可以利用Emacs的编辑功能“备份”自己的命令清单、复制旧命令，修改它，然后再执行它。
- 可以保存自己的shell编辑缓冲区，即保留一份自己编辑工作的日志——它将自动把运行的每一条命令的输出情况包括在其中。
- 可以把命令的执行结果复制到其他的窗口（其他文件、电子邮件消息等）里去。
- 可以把复杂的命令保存到一个文件里，然后把这个文件插入到命令行提示符处——这样就不必反复输入那些命令了。

习惯了使用Emacs进行工作之后，就会发现它越来越多的用途。

Emacs还提供了几种能够不通过shell访问UNIX操作系统的功能。Emacs有一个对目录进行（列文件清单和删除文件等）操作的工具，它比**ls**和**rm**等命令用起来还要方便。只要愿意，在执行有关操作的时候完全可以抛开那些UNIX命令。它还有一些用来对文件进行编译（即运行**make**命令）和打印的内置命令。

在这一小节里，我们将对shell模式进行讨论。在本章的后半部分，我们将讨论目录编辑、打印以及在Emacs里对个人日程进行简单管理的日历和日记等功能。现在从shell模式的一个简单变体开始讲起，这个功能能够一次执行一条UNIX命令。

一次执行一条UNIX命令

如果想在一次Emacs编辑工作的过程当中执行一条UNIX命令，请按下“**ESC !**”组合键。Emacs将提示输入一条准备执行的命令。输入一个命令并按下回车键，Emacs将打开一个名为“`*Shell Command Output*`”的窗口，并把该命令的执行结果显示在其中。按下“**C-x 1**”组合键可以关闭这个窗口。

按下：**ESC !**

```
Buffers Files Tools Edit Search Minibuf Help
Stately,plump Buck Mulligan came from the stairhead,bearing a bowl
of lather on which a mirror and a razor lay crossed.A yellow
dressing gown,ungirdled,was sustained gently behind him by the mild
morning air.He held the bowl aloft and intoned:
  -Introibo ad altare Dei
-----Emacs:joyce            (Text Fill)--L1--All----------------------
Shell command:
```

Emacs 提示输入一条准备执行的命令。

输入：**diff joyce joyce2**

```
Buffers Files Tools Edit Search Help
Stately,plump Buck Mulligan came from the stairhead,bearing a bowl
of lather on which a mirror and a razor lay crossed.A yellow
dressing gown,ungirdled,was sustained gently behind him by the mild
-----Emacs:joyce            (Text Fill)--L1--All----------------------
1,5c1,6
<Stately,plump Buck Mulligan came from the stairhead,bearing a bowl
---
>Stately,plump Duck Mulligan came from the stairhead,bearing a bowl
--**-Emacs:*Shell Command Output*      (Fundamental)--L6--Top----------
```

Emacs 执行 **diff** 命令并把其输出放到一个“*Shell Command Output*”编辑缓冲区里。

因为 **diff** 命令的输出是在一个编辑缓冲区里，所以可以对它进行编辑、保存、或者做任何想做的操作。

Emacs的shell命令工具有一个很有意思的功能，即可以用某个编辑缓冲区中的一个文本块，而不是一个传统的文件作为命令的输入。比如，假设对一个电话号码表进行排序。首先，要把光标放到电话号码表里的某个位置（比如“Eric”的头一个字符上）；然后，发出 **mark-paragraph** 命令（“**ESC h**”组合键）。这个命令将把电话号码表定义为一个文本块，光标在这个段落的开头，文本标记在这个段落的末尾。在下面的示例中，加阴影的区域代表准备进行排序的文本块范围。在选取好文本块之后，按下“**ESC |**”（命令名是 **shell-command-on-region**）组合键，Emacs 提示输入将要执行的 shell 命令。

按下：**ESC h ESC !**

```
Buffers Files Tools Edit Search Minibuf Help
Name                Number

Barb                827-1059
Leigh               353-4131
Randy               878-2945
Eric                442-3312
--**-Emacs:phones            (Text Fill)--L2--All----------------------
Shell command on region:
```

Emacs 提示输入一条准备执行的命令。

接着，发出 **sort** 命令，但不必指定任何输入文件（注 1）。提供输入数据的工作将由 Emacs 负责。

输入：**sort RETURN**

```
Buffers Files Tools Edit Search Help
Barb                827-1059
Leigh               353-4131
Randy               878-2945
Eric                442-3312
--**-Emacs:phones            (Text Fill)--L4--All----------------------
Barb                827-1059
Eric                442-3312
Leigh               353-4131
Randy               878-2945
--**-Emacs:*Shell Command Output*      (Fundamental)--L4--All----------
```

Emacs 对刚才定义的文本块进行了排序。

Emacs 对电话号码表（即文本块中的全部内容）进行了排序。“**ESC !**”有一个很有用的变体，即允许把命令的执行结果直接放到当前编辑缓冲区而不是“`*Shell Command Output*`”编辑缓冲区里。这需要在“**ESC !**”命令的前面加上一个“**C-u**”；比如，“**C-u ESC !**”将执行一条 shell 命令，并把其输出放到当前编辑缓冲区里。如下所示：

注 1：　事实上，Emacs 自己的“**ESC x sort-lines**”命令执行起来会更快一些。我们那么做是为了找个简单点的例子。

输入：**C-u ESC ! ls RETURN**

```
Buffers Files Tools Edit Search Help
Here's a list of files in my current directory:

INDEX
News
RMAIL
RMAIL~
addresses
albert
broken.ps
--**-Emacs:directory          (Text Fill)--L1--Top----------------------
```

Emacs 执行 **ls** 命令并把其执行结果放到当前编辑位置。

在一般情况下，Emacs 使用默认的 shell 来执行各种命令。如果你平时使用的是 C shell(**csh**)，它使用的也将是C shell；如果你平时使用的是Bourne shell(**sh**)，Emacs 也将使用Bourne shell来执行各种命令。依次类推，可以用修改变量**shell-file-name** 取值的方法把它换成另外一种不同的 shell。这个变量的取值必须是新 shell 的可执行文件的完整路径。比如，如果想用 GNU 项目下的 Bourne-again shell（**bash**）（注 2）来执行命令，就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq explicit shell-file-name "/bin/sh")
```

使用 shell 模式

从现在开始，我们将介绍Emacs编辑器的shell模式，用来执行UNIX命令的交互式功能组件。启动 shell 编辑缓冲区的方法是给出“**ESC x shell**”命令，这将创建出一个名为“*shell*”的编辑缓冲区。在这个编辑缓冲区里会出现 UNIX 操作系统的 shell 的提示符。如下所示：

输入：**ESC x shell**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
%

--**-Emacs:*shell*          (Shell:run)--L2--All----------------------
Loading shell...done
```

现在，可以输入 shell 命令了。

注 2：　对不同的计算机系统而言，**bash**的存放位置是会变化的。它甚至有可能在你的系统上根本就不可用。

除了可以在输入命令时用Emacs对之进行编辑以外，就其大部分而言，shell模式与正常的UNIX命令界面几乎是一模一样的。可以把命令从一个地方复制到另一个地方，或者把命令的执行结果复制到某个文件里等等。

有几个注意事项是必须知道的。比如，通常会通过按下“**C-c**”组合键来中止一个作业；但如果想在shell模式里用按下“**C-c**”组合键的办法来中止一个程序（因为有许多Emacs命令都是以“**C-c**”开头），Emacs将认为这个“**C-c**”是它自己的编辑命令的一部分。因此，如果在Emacs中想通过“**C-c**”组合键来中止当前作业，就必须按下“**C-c C-c**”组合键才行。类似地，必须按下“**C-c C-z**”组合键而不是“**C-z**”组合键才能停止一个作业的运行；必须按下“**C-c C-d**”组合键而不是“**C-d**”组合键等等（严格地说，不一定非得按下“**C-c C-d**”组合键，因为Emacs能够根据上下文领会“**C-d**”的含义。如果是在编辑缓冲区的末尾，“**C-d**”的含义将是“文件尾”；如果是在其他地方，“**C-d**”将删除一个字符）。另外，如有必要，可以从“**Signals**（信号）”菜单里选择适当的选项而不使用控制字符（比如，通过菜单选择“**EOF**”而不是按下“**C-d**”组合键）。

shell模式还准备了一些简化操作的快捷键。比如，命令“**ESC p**”把输入的最后一个命令放到shell模式的命令提示符处，不管已经在编辑缓冲区里移动了多远。连续按下“**ESC p**”组合键将把更早的命令找回来。如下所示：

按下：**ESC p**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
coatimundi.txt   fish.not     lemurs.c      quetzal.c     zooanimals.txt
detroit.ms       fossa.ms     ocelot.txt    whales.txt    zymurgy.ms
<ruby>
% ls zoostuff
--**-Emacs:*shell*          (Shell:run)--L4--Bot-----------------------
History item: 1
```

“**ESC p**”把你输入的最后一个命令放到shell模式的命令提示符处，即使它已经不在屏幕上了。

在上面的例子里，输入的最后一条命令是“**ls zoostuff**”。它已经不在屏幕上了，它的执行输出（一长串文件清单）已经把它从屏幕顶部推出去了。可命令“**ESC p**”又把这条**ls**命令找了回来，但不会执行它；在按下回车键之前，还有机会对这个命令进行编辑。如果想把再早一些的命令找回来，可以按下“**ESC n**”组合键。

这些命令可能很面熟，的确如此。它们就是所谓的“历史记录”命令，与在第三章里讨论的辅助输入区的历史记录命令是完全一样的。Emacs 专门准备了一个完备的“**In/Out**（输入 / 输出）”菜单，来负责命令的历史记录方面的工作。

在 shell 模式里，回车键（**RETURN**）和制表键（**TAB**）有特殊的功用。按下回车键将执行当前命令行上的命令，即使移动到某条老命令所在的那一行上。按下回车键的时候，Emacs 会把这条命令复制到 shell 编辑缓冲区的末尾并执行它。在按下回车键之前，还有机会对任何命令进行编辑修改。

按下制表键（**TAB**）将启用 Emacs 的自动补足功能。自动补足的对象可以是操作系统命令、文件名和变量等等。可以从“**Complete**（自动补足）”菜单开始访问这些命令，但按 **TAB** 键通常更简便一些。

如果输入了一个会产生大量输出的命令，并因此而妨碍了编辑工作，那么有一个简便的办法可以预防这种情况：键入命令后按下“**C-c C-o**”（命令名是 **comint-kill-output**）组合键，或者从“**In/Out**”（输入 / 输出）菜单里选择执行“**Kill Current Output Group**”（清除当前输出组）操作，不要按回车键。如下所示：

按下：**C-c C-o**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
% ls zoostuff
*** output flushed ***
%
--**-Emacs:*shell*          (Shell:run)--L15--Bot----------------------
```

“**C-c C-o**”自动删除上一个命令的执行输出。

前一条命令（**ls zoostuff**）还在屏幕上，可它那一长串文件清单输出却没有出现在屏幕上。“**C-c C-o**”只能删除刚键入的那条命令的输出；如果以前的命令都是按回车键执行的，那么命令的执行结果将出现在屏幕上，“**C-c C-o**”是不能删掉这类命令输出的。“**C-c C-o**”特别适用于给出了一个可能会产生大量输出的命令，但却并不关心它们的情况。“**C-c C-o**”还提供了一种把 UNIX 命令的执行结果，复制到另外一个编辑缓冲区里去的便利手段。只要移动到另外一个编辑缓冲区里再按下“**C-y**”（命令名是 **yank**），就可以把那些输出结果找回来。

shell 模式另外一个很有用的命令是“**C-c C-r**”（命令名是 **comint-show-output**）。这个命令特别适用于这样的情况：命令产生大量的输出而输出内容的前几行卷出了屏幕。“**C-c C-r**”会对窗口进行调整，使最后一条命令的输出内容的前几行出现在窗口的最顶部。如果想查看输出内容的最后几行，请按下“**C-c C-e**”（命令名是 **comint-show-maximum-output**）组合键或者从“**In/Out**”菜单里选择执行“**Show Maximum Output**（显示最大输出）”操作。这个命令的作用是把输出内容的最后一行挪到窗口的最底一行。

编写书刊的时候，希望能够以段落为单位进行移动是一种很合理的要求；但在使用 shell模式的时候，以输出组为单位进行移动可能更有用一些。一个输出组由一条命令和它的执行输出内容构成。如果想移动到前一个输出组，请按下“**C-c C-p**”组合键或者从“**In/Out**”菜单里选择执行“**Backward Output Group**”（后退到上一输出组）操作。如果想移动到后一个输出组，请按下“**C-c C-n**”组合键或者从“**In/Out**”菜单里选择执行“**Forward Output Group**（前进到下一输出组）”操作。

shell模式的一个突出优点是可以在一条命令开始执行之后，在这条命令的执行过程中继续对其他一些编辑缓冲区进行编辑。shell 编辑缓冲区甚至可以不出现在屏幕上；只要输入“**ESC x shell**”就可以再把 shell 编辑缓冲区调回到屏幕上。（与此形成鲜明对照的是“**ESC !**”和“**ESC |**”，它们将强制性地等到 UNIX 命令执行完毕后，才能继续进行其他编辑工作。）如果 shell 不具备作业控制功能，这种允许在命令执行期间继续进行编辑的做法无疑是一种福音。即使 shell 具备作业控制功能，与让命令运行在后台而继续编辑工作的情况相比，shell模式也还是要方便得多。有了 Emacs，就用不着担心后台作业的输出会干扰到正在进行的工作了。它的输出会完整地保存到 shell 编辑缓冲区里。可以用“**ESC x rename-uniquely**”命令重新命名shell编辑缓冲区，而这将同时拥有多个shell编辑缓冲区。可以重复、重复、再重复地这样做，直到 shell 编辑缓冲区的个数满足工作需要为止。

当前目录

在本书的开头，我们介绍了默认目录的概念。具体规定是：如果光标在某个文件里，则默认目录就是该文件所在的那个目录。启动一个shell的时候，Emacs会自动进入它的默认目录。也就是说，创建shell编辑缓冲区时所在的文件目录就是最初的工作目录。

此后的事情也许会变得很复杂。Emacs 会根据 **cd**、**popd** 和 **pushd** 等命令（C shell 中用来改变目录的命令）来改变其默认目录的表示方法。如果还没有被弄糊涂，就该知道用来查找文件的默认目录永远都和当前目录一样。比如，如果执行完“**cd /home/src/emacs/lisp**”命令之后，又准备用“**C-x C-f**”编辑一个文件，则用来查找文件的默认目录就将是“*/home/src/emacs/lisp*”。

这个功能存在着以下几个问题:

- Emacs 只有在把 **cd**、**popd** 和 **pushd** 等命令输入到命令行的最开始时才能检测到它们。
- Emacs 理解不了假名或其他任何会改变目录路径的程序。可以用修改变量 **shell-cd-regexp**、**shell-pushd-regexp**、**shell-popd-regexp** 的方法来帮助它识别这些命令。这几个变量的值都是正则表达式，它们匹配的东西是用来完成 **cd**、**popd** 和 **pushd** 等操作的命令。对正则表达式的详细讨论见第十三章。
- 最后，有些用户（包括我们几位作者在内）不喜欢自己的默认目录变来变去。如果各位也像我们一样不喜欢它，请把下面这几条语句添加到“*.emacs*”文件里:

```
(setq-default shell-cd-regexp nil)
(setq-default shell-pushd-regexp nil)
(setq-default shell-popd-regexp nil)
```

shell 的初始化

Emacs是怎样知道应该启动哪个shell的呢？它首先查看的是变量**explicit-shell-file-name**；接着查看的是一个名为 **ESHELL** 的 UNIX 环境变量；最后查看的是一个名为 SHELL 的环境变量。因此，如果在 Emacs 里时想另外运行一个特定的 shell（比如 Bourne shell），就应该把下面这条语句添加到“*.emacs*”文件里:

```
(setq explicit-shell-file-name "/bin/sh")
```

当 Emacs 启动一个交互式的 shell 时，它会在 shell 正常的启动文件之后再额外运行一个初始化文件。这个文件的名字是“*.emacs_shell-name*”，其中的“*shell-name*”是打算在 Emacs 里使用的 shell 的名字。它必须保存在主目录里。比如，如果打算使用 C shell，就必须把它们放到文件“`&.emacs_csh`”里作为 Emacs 专用的启动命

令。再比如，如果想把Emacs的shell模式的命令行提示符设置为“emacs:%”，还想把一个名为**WITHIN_EDITOR**的环境变量设置为“**T**”，就必须在“*.emacs_csh*”文件里加上如下所示的语句:

```
set prompt="emacs:% "
setenv WITHIN_EDITOR T
```

在shell编辑缓冲区里，Emacs还会把环境变量**EMACS**设置为“**t**”，把终端类型（即环境变量**TERM**）设置为“**emacs**”。

远程shell的问题

如果是在一个网络上，那么可能会想先运行Emacs，进入它的shell编辑缓冲区，然后在这个窗口里运行**rlogin**来访问其他的系统。这种情况经常会在屏幕显示方面出现一个小问题——已输入的UNIX命令在屏幕上出现了两次，而且大多数命令行的后面还多出了一个“**^M**”字样。而除此之外，其他一切都很正常。如下图所示:

初始状态:

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
% rlogin remhost
remhost % pwd
pwd ^M
/home/remhost/deb ^M
remhost % ls
ls ^M
file1 file2 file3^M
remhost%
--**-Emacs:*shell*        (Shell:run)--L37--Bot--------------------
```

每个命令都会重复显示一遍，而且还有一些不知从哪儿冒出来的“**^M**”。

这个问题很容易解决。在远程系统的shell编辑缓冲区里，输入UNIX命令“**stty -echo nl**”。如下图所示:

输入：**stty -echo nl**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
% rlogin remhost
remhost% stty -echo nl
stty -echo nl ^M
remhost%pwd
/home/remhost/deb
remhost%
--**-Emacs:*shell*          (Shell:run)--L39--Bot----------
```

使用 **stty** 命令加以改正之后的来自远程系统的输出情况。

如果这条 **stty** 命令能自动执行，那当然就更理想了。可以做到这一点，方法是对打算使用的所有远程系统上的“*.cshrc*”文件（适用于C shell 用户）和“*.profile*”文件（适用于 Korn shell 或 Bourne shell 用户）进行修改。具体方法如下：

如果是 csh:

```
if ($TERM == emacs) stty -echo nl
```

如果是 ksh 或 sh:

```
if [ $TERM = "emacs" ]
then
      stty -echo nl
fi
```

在上面那个针对 **ksh** 或 **sh** 进行设置的例子里，千万要在括号左右多加一个空格。忘记这些空格是一种经常出现的错误。

用 shell 模式预防安全隐患

在默认的情况下，shell模式将把输入的一切内容显示在屏幕上。如果输入了用来转换到另外一个 UNIX 账户去的“**su**”命令，并输入了相应的口令字，则那个口令字也将显示在屏幕上，这无疑是一个巨大的系统安全漏洞。不过，这个问题是有办法解决的。在输入口令字之前，需要先输入“**ESC x send-invisible**”，然后 Emacs 提示输入不再显示的文本。输入一个字符时，Emacs会在辅助输入区里显示一个星号，按下回车键，Emacs 把口令字送出去。整个过程中，口令字始终没有以明文的形式显示在屏幕上。如果想让 Emacs 在输入口令字的时候，总是把口令字隐藏起来，请把下面这两行语句添加到“*.emacs*”文件里：

```
(add-hook 'comint-output-filter-functions
          'comint-watch-for-password-prompt)
```

以后，只要屏幕上再出现口令字提示符，Emacs 就会要求把不显示的文字输入到辅助输入区里，这就保证了口令字将再也不会出现在屏幕上。但要注意的是，这个功

能是在最近才添加到Emacs 19里去的。Emacs 19.22就没有这项功能，Emacs 19.28里有。（用“**ESC x version**”命令可以查出使用的Emacs版本。）

表5-1对shell模式里使用的各种命令进行了汇总。

表5-1：shell模式命令速查表

键盘操作	命令名称	动作
（无）	**shell**	进入shell模式
C-c C-c *Signals → BREAK*	**comint-interrupt-subjob**	中断当前作业；相当于UNIX的shell中的“**C-c**”组合键
C-d	**comint-delchar-or-maybe-eof**	如果是在编辑缓冲区的末尾，送出EOF字符；如果是其他位置，删除一个字符
C-c C-d *Signals → EOF*	**comint-send-eof**	送出EOF字符
C-c C-u	**comint-kill-input**	删除当前行；相当于UNIX的shell中的“**C-u**”组合键
C-c C-z *Signals → STOP*	**comint-stop-subjob**	对非X用户，挂起或者停止一个作业；相当于UNIX的shell中的“**C-z**”组合键
ESC p *In/Out → Previous Input*	**comint-previous-input**	检索此前的上一个命令（可以重复执行以找回更早的命令）
ESC n *In/Out → Next Input*	**comint-next-input**	检索此后的下一个命令（可以重复执行以找回更近的命令）
RETURN	**comint-send-input**	送出输入在当前行上的命令
TAB *Complete → Complete Before Point*	**comint-dynamic-complete**	自动补足当前命令、文件名或者变量名
C-c C-o *In/Out → Kill Current Output Group*	**comint-kill-output**	删除最后一条命令的输出
C-c C-r	**comint-show-output**	把输出内容的第一行移到窗口的顶部

表 5-1: shell 模式命令速查表（续）

键盘操作	命令名称	动作
C-c C-e *In/Out →* *Show Maximum Output*	**comint-show-maximum-output**	把输出内容的最后一行移到窗口的底部
C-c C-p *In/Out →* *Backward Output Group*	**comint-previous-prompt**	移动到前一条命令
C-c C-n *In/Out →* *Forward Output Group*	**comint-next-prompt**	移动到后一条命令

文件和目录操作

Dired 模式（directory editing mode，目录编辑模式）是 Emacs 最引人注意的功能之一，它提供了对文件目录进行编辑的有效手段。可以查看目录里所有文件的一个完整清单，对它们进行删除、重命名、复制以及其他各种基本的文件操作。一旦用熟了Dired，就再也不需要UNIX下的**cp**、**rm**或**mv**等命令了。尤其重要的是，Dired能够提高工作效率。可以同时对一组文件进行操作，比如删除、移动和压缩等等，甚至可以对文件里的字符串进行查询－替换。

进入目录编辑模式的方法有好几种。如果还没有进入Emacs，可以在启动Emacs的时候用一个目录名作为其命令行参数，比如:

```
% emacs fiction
```

则Emacs将开始对目录*fiction*进行编辑 —— 出现一个单独的窗口，里面是*fiction*目录里的文件清单。用“**C-x C-f**”命令（或任何可以打开文件的其他编辑命令）也能进入目录编辑器，但前提是给出的必须是一个目录名而不是一个文件名。比如，输入“**C-x C-f fiction**”将开始对*fiction*目录进行编辑。按下“**C-x d**”（命令名是**dired**）组合键或者从“**Files**”菜单里选择执行“**Open Directory**”操作也能启动Dired（还得再输入一个目录名）。不管用什么办法启动目录编辑器，结果都将是一样的。

输入：**C-x C-f fiction RETURN**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
 /home/deb/fiction:
 total 641
 drwxrwxr-x  2 deb          512  Jun 23 13:22 .
 drwxrwxr-x 13 deb         7680  Jun 23 13:22 ..
 -rwxrwxrwx  1 deb        54576  Apr 22 18:55 ch01
 -rwxrwxrwx  1 deb        79915  Apr 22 18:55 ch02
 -rwxrwxrwx  1 deb        82809  Apr 22 18:56 ch03
 -rwxrwxrwx  1 deb        79434  Apr 22 18:56 ch04
 -rwxrwxrwx  1 deb        85805  Apr 22 18:56 ch05
 -rwxrwxrwx  1 deb       109282  Apr 22 18:56 ch06
 -rwxrwxrwx  1 deb        38191  Apr 22 18:57 ch07
 -rwxrwxrwx  1 deb        91235  Apr 22 18:57 ch08
 -rw-rw-r--  1 deb          611  Aug 25  1995 dickens
 -rw-rw-r--  1 deb          315  Aug 26  1995 elephant
 -rw-rw-r--  1 deb          418  Aug 26  1995 hjames
--%%-Dired:fiction              (Dired by name)--L5--Top----------
Reading directory /home/deb/fiction/...done
```

目录编辑器的初始画面将出现在屏幕上。

正如大家所看到的，Dired的初始画面与在UNIX提示符处输入**ls-l**命令后看到的内容基本相同。在文件名的前面依次列出了文件的存取权限、文件的属主、系统名、文件长度和最近一次修改日期等项目。这个目录里所有的文件和下级子目录都列在里面，包括那些文件名以句点打头的文件。光标的初始位置在一个文件名上，不在第一列里。

在默认的情况下，文件清单将根据文件名进行排序，但也可以让它根据日期来排序。再来看看状态行，它上面将出现“(Dired by name)”字样。如果想对画面中的文件清单重新进行排序，请按“**s**”键（命令名是**dired-sort-toggle-or-edit**）。这个命令将把最新的文件放在清单的最顶部，很容易地就解决了“我昨天编辑的文件是哪一个?”的问题，状态行将相应地出现“(Dired by date)”字样。再按一次“**s**”键，将回到按文件名的字母顺序排序显示文件清单的画面。

如果各位还记得用来对编辑缓冲区清单进行编辑的命令(参见第四章)，就会发现它们与将要介绍的用在目录编辑器里的命令几乎都是一样的。不过，虽然命令还是那些基本的命令，但能做的事情可大不一样。

注意：请记住，目录编辑器是直接对文件而不是编辑缓冲区进行工作的。如果用Dired删除了一个文件，它可就永久性地“消失”了。

在 Dired 里有好几种移动办法。“**SPACE**”、“**C-n**”和“**n**”等键盘命令都会移动到文件清单中的下一个文件上；而“**DEL**”、“**C-p**”和“**p**”等键盘命令都会移动到前一个文件上。还可以使用任何一种查找命令（递增查找、单词查找等等）来找出某个特定的文件。

查看和编辑文件

在查看某个目录的文件清单时，可能需要快速查看一下那些文件的内容。Dired 的“**v**”命令就是用来完成这项工作的：把光标放到想查看的文件上，然后按下“**v**”（命令名是 **dired-view-file**）键或者从“**Immediate**（立即）”菜单里选择执行“**View This File**”（查看这个文件）操作即可。Emacs会把这个文件以查看模式（view mode）显示在屏幕上，此时的文件内容只能看，不能改。按下“**C-c**”或“**q**”键（哪个方便按哪个），将返回到目录的文件清单窗口。在查看文件内容的时候，可以用“**s**”键开始一次递增查找操作，或者按下回车键前进到下一行。按下“**=**”键，Emacs 会告知光标在哪一行。还有一些其他 Emacs 命令（比如标记文本块等）的快捷键，但老实说，常规命令都能正确工作 —— 如果已经掌握的那些命令都能工作得很好，再去另外记忆一套特殊的命令似乎就没什么必要了。

如果想从 Dired 编辑缓冲区开始去编辑某个文件，请按下字母“**f**”或者从“**Immediate**”菜单里选择执行“**Find This File**（找出这个文件）”操作。Emacs 会把这个文件找出来以便对它进行编辑。这可完全是一个普通的编辑缓冲区可以做任意的编辑修改、保存、查看其他文件等各种操作。按下“**C-x b**”组合键后，再输入正在工作的子目录名将回到 Dired 编辑缓冲区，也可以通过编辑缓冲区清单（“**C-x C-b**”组合键）重新回到 Dired 编辑缓冲区。

能够从 Dired 编辑缓冲区出发查看和编辑文件当然是件好事，不过编辑方面的事情我们都已经向大家介绍过了。下面来学点我们还不知道的事情 —— 如何删除文件。

文件的删除、复制、重命名操作

已经讲过，文件的删除与通过编辑缓冲区清单进行的编辑缓冲区删除在操作上是一样的。只要知道怎样删除编辑缓冲区，就等于已经掌握了通过 Dired 删除文件的基本方法。首先，需要给文件加上一个待删除标记 —— 把光标移动到那个文件的名

字上并按下“**d**”键，这会在该文件名的左边放上一个大写的字母“D”，而光标将移动到文件清单里的下一个文件上。可以随心所欲地给任意多个文件加上待删除标记；如果此时改变了主意、还来得及用“**u**”键去掉文件的待删除标记。然后，按下“**x**”键把加有待删除标记的全部文件一次性地都删除掉（具体情况马上就要讲到）。下面是给几个文件加上待删除标记后 Dired 编辑缓冲区的显示画面示例：

按下：**d**（可以给任意多个文件加上待删除标记）

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  /home/deb/fiction:
  total 641
  drwxrwxr-x  2 deb          512 Jun 23 13:22 .
  drwxrwxr-x 13 deb         7680 Jun 23 13:22 ..
  -rwxrwxrwx  1 deb        54576 Apr 22 18:55 ch01
  -rwxrwxrwx  1 deb        79915 Apr 22 18:55 ch02
  -rwxrwxrwx  1 deb        82809 Apr 22 18:56 ch03
  -rwxrwxrwx  1 deb        79434 Apr 22 18:56 ch04
  -rwxrwxrwx  1 deb        85805 Apr 22 18:56 ch05
  -rwxrwxrwx  1 deb       109282 Apr 22 18:56 ch06
  -rwxrwxrwx  1 deb        38191 Apr 22 18:57 ch07
  -rwxrwxrwx  1 deb        91235 Apr 22 18:57 ch08
  -rw-rw-r--  1 deb          611 Aug 25  1995 dickens
D -rw-rw-r--  1 deb          315 Aug 26  1995 elephant
D -rw-rw-r--  1 deb          418 Aug 26  1995 hjames
D -rw-rw-r--  1 deb          865 Jun 23 13:01 iostuff.c
--%*-Dired:fiction          (Dired by name)--L17--Top----------
```

本例中给 3 个文件加上了待删除标记。

前面讲过，可以随时利用“**u**”命令去掉某个文件前面的待删除标记。按下“**u**”键，将移动到上一个加有待删除标记的文件处。也可以用**DEL**键去掉文件前面的待删除标记 —— 这个命令的作用是去掉文件清单里前一个文件上的待删除标记并上移一行（即移动到刚被去掉待删除标记的那个文件处）。

因为Emacs会创建备份文件和自动保存文件，所以可能会不时地需要对它们进行一番清理。Emacs提供了一些快捷命令来给这些文件加上待删除标记。按下“**#**”键会给所有的自动保存文件加上待删除标记（自动保存文件的文件名、其首尾各有一个“#”字符），Emacs 会给它们都加上待删除标记，并报告总共有多少个文件被加上了待删除标记。按下“**~**”键会给所有的备份文件加上待删除标记（备份文件的文件名都以“~”字符结尾）。因为这些文件只是被加上了待删除标记，所以如果还想保留某个备份文件（比如最近编辑的文件的备份拷贝），现在还来得及去掉其上的待删除标记。

等到真的想从磁盘上把文件删掉的时候，请按下“**x**”键。Emacs将把所有被加上待删除标记的文件列出来，然后询问是否真的想删除掉它们。如下所示：

按下：**x**

```
Buffers Files Tools Edit Search Minibuf Help
  -rwxrwxrwx 1 deb                91235  Apr 22 18:57 ch08
  -rw-rw-r-- 1 deb                  611  Aug 25  1995 dickens
D -rw-rw-r-- 1 deb                  315  Aug 26  1995 elephant
D -rw-rw-r-- 1 deb                  418  Aug 26  1995 hjames
D -rw-rw-r-- 1 deb                  865  Jun 23 13:01 iostuff.c
  -rw-rw-r-- 1 deb                  280  Jun 23 13:02 joyce
  -rw-rw-r-- 1 deb                  280  Aug 25  1995 joyce~
  -rw-rw-r-- 1 deb                   53  Jan 19 09:12 letterb
  -rw-rw-r-- 1 deb                  226  Aug 26  1995 phone
--%*-Dired:fiction                (Dired by name)--L17--Bot------------
elephant hjames iostuff.c
--**-Emacs:*Deletions*        (Fundamental)--L1--All--------------------
Delete D [3 files] (yes or no)
```

“**x**”键的作用是把文件真正地删除掉；回答“**yes**”表示“开动”。

回答“**yes**”将把它们都删除掉，而回答“**no**”将不做删除并返回Dired编辑缓冲区。

这是删除文件的常用办法，但如果想立刻删掉某个文件，请输入大写字母“**D**”。Emacs将询问是否真的想删掉这个文件（yes或no）。回答“**yes**”将立刻删掉这个文件，回答“**no**”则不做删除。在Dired里，这是少数几种命令的小写字母形式（比如“**d**”用来设置待删除标记）和大写字母形式（比如“**D**”用来立刻删除某个文件）有不同含义的情况。

如果想通过Dired来复制某个文件，请在它旁边按下“**C**”（必须是大写字母）键或者从“**Operate**（操作）”菜单选择执行“**Copy to**（复制到）”操作。Emacs会要求提供一个文件名作为复制操作的目标文件名。输入这个文件名再按下回车键，Emacs报告“Copied: 1 file（完成复制一个文件）”。如果想对文件清单中的多个文件进行复制，可以在大写字母“C”的前面加上一个数字。比如，输入“**3 C**”将复制光标位置上的文件和紧随其后的两个文件。

如果想通过Dired来重新命名某个文件（类似于UNIX中的**mv**命令），请在其文件名的旁边按下“**R**”键或者从“**Operate**”菜单选择执行“**Rename**”操作。Emacs会询问新文件名是什么。输入它并按下回车键，Emacs报告“Moved: 1 file（完成移动一个文件）”。

文件的压缩和解压缩操作

对文件进行压缩可以节省出大量的磁盘空间，而Dired使压缩文件这种操作变得非常简单。把光标放到想压缩的文件所在的那一行上，然后按下“**Z**”（命令名是**dired-do-compress**）键或者从“**Operate**”菜单选择执行“**Compress**（压缩）”操作。Emacs提问：

```
Compress or uncompress filename? (y or n)
```

如果文件还没有被压缩过，那么Emacs将对它进行压缩操作；如果已经被压缩过了，则Emacs将对它进行解压缩操作（注3）。回答“**y**”将开始对文件接进行压缩或解压缩操作。压缩工作是立刻进行的，所以，能够在Emacs对文件进行压缩的同时观察到文件扩展名的改变和文件长度的变化。当然，不能直接对压缩文件进行编辑，所以在对它进行编辑之前必须先对它进行解压缩操作（注4）。

对文件进行比较

上一章对分处两个窗口的文件之间的比较操作进行了讨论。Emacs在Dired里提供了一种利用UNIX的**diff**命令进行这类比较的方法。先给想用**diff**命令进行比较的文件之一设置操作标志，再把光标放到另一个文件上，然后按下“**=**”键或者从“**Immediate**”菜单里选择执行“**Diff**（比较）”操作。Emacs将比较这两个文件，并且会打开一个窗口——窗口中的“`*diff*`”编辑缓冲区里包含该命令的执行结果。

Emacs还提供了一个对某个文件及其备份文件进行比较的操作选项。把光标放在想

注3：Emacs只能识别**compress**和**gzip**格式，对PKZIP或其他专利性的文件压缩算法无能为力。对文件进行解压缩操作的时候，Emacs能够识别出后缀为“*.z*”、“*.Z*”或“*.gz*”的文件并正确地完成解压缩操作。在对文件进行压缩时，Emacs将使用FSF的**gzip**程序，其结果文件将以“*.gz*”为后缀名。

注4：Emacs有一个能够自动完成压缩/解压缩操作的自动压缩模式（auto-compress mode）。这个编辑模式在默认情况下是处于启用状态的，所以用不着特意去启用它。但是，在我们写这本书的时候，它的工作情况还不太正常。为了让它工作，请输入“**ESC x auto-compress-mode RETURN**”命令，这条命令的作用是关闭自动压缩功能。然后，请输入“**C-x ESC ESC RETURN**”重复执行一次这条命令。现在，自动压缩模式才真正被激活了。不过，要是让所有的文件都能被自动地压缩（以节约磁盘空间）和解压缩（以便对它们进行编辑），即使需要把这个命令重复输入两次也算不上是多大的麻烦。Emacs甚至会在完成文件的编辑工作之后再次自动对它进行压缩。

与其备份文件进行比较的文件上，然后按下“**ESC =**”组合键或者从“**Immediate**”菜单里选择执行“**Compare With Backup**（与备份文件进行比较）”操作。Emacs 将把两个文件之间的差异显示在一个“*diff*”编辑缓冲区里。

在文件上运行 UNIX 命令

在 Dired 里实现的 **diff** 命令当然很有用（Dired 里还有 **chmod**、**grep** 和 **find** 等命令实现），但从更广义的方面讲，只要按下“!”键，就可以在文件上执行任何 UNIX 命令。举个例子，用 **sort** 命令对前面的电话号码表按字母表顺序进行排序。如下所示：

移动到 *phone* 文件并按“!”键。

```
Buffers Files Tools Edit Search Minibuf Help
  -rwxrwxrwx 1 deb                91235  Apr 22 18:57 ch08
  -rw-rw-r-- 1 deb                  611  Aug 25  1995 dickens
  -rw-rw-r-- 1 deb                  280  Jun 23 13:02 joyce
  -rw-rw-r-- 1 deb                  280  Aug 25  1995 joyce~
  -rw-rw-r-- 1 deb                   53  Jan 19 09:12 letterb
  -rw-rw-r-- 1 deb                  226  Aug 26  1995 phone
--%*-Dired:fiction                (Dired by name)--L18--Bot------------
!on phone:
```

Emacs 询问要运行的命令。

输入：**sort**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  -rwxrwxrwx 1 deb                91235  Apr 22 18:57 ch08
  -rw-rw-r--1 deb                   611  Aug 25  1995 dickens
  -rw-rw-r--1 deb                   280  Jun 23 13:02 joyce
  -rw-rw-r--1 deb                   280  Aug 25  1995 joyce~
  -rw-rw-r--1 deb                    53  Jan 19 09:12 letterb
  -rw-rw-r-1 deb                    226  Aug 26  1995 phone
--%*-Dired:fiction                (Dired by name)--L15--Bot------------
Allen      (814)723-8799
Alvin      (412)833-7833
Bob        (412)238-7732
Eileen     (717)355-5894
Ellen      (904)735-5894
Fred       (814)353-3325
Horace     (814)676-5476
Nicole     (305)977-7721
Olivia     (617)822-3532
--**-Emacs:*Shell Command Output*      (Fundamental)--L10--All---------
```

Emacs 在另外一个窗口里给出这个命令的输出结果。

稍微复杂点的例子是给出的命令要求使用一个以上的文件作为参数。请看下面这个例子：想把排好序的电话号码表保存到一个新文件里。如下所示：

把光标移动到 *phone* 文件并按“!”键。

```
Buffers Files Tools Edit Search Minibuf Help
  -rwxrwxrwx 1 deb                91235  Apr 22 18:57 ch08
  -rw-rw-r-- 1 deb                  611  Aug 25  1995 dickens
  -rw-rw-r-- 1 deb                  280  Jun 23 13:02 joyce
  -rw-rw-r-- 1 deb                  280  Aug 25  1995 joyce~
  -rw-rw-r-- 1 deb                   53  Jan 19 09:12 letterb
  -rw-rw-r-- 1 deb                  226  Aug 26  1995 phone
--%*-Dired:fiction                (Dired by name)--L18--Bot------------
!on phone:
```

Emacs 询问要运行的命令。

现在，告诉 Emacs 要对 *phone* 文件进行排序并把其结果输出保存到一个名为 *newphone* 的新文件里。因为光标就在 *phone* 文件上，所以不需要在命令里重复输入它的名字。可以用一个星号（*）来代表这个文件的名字，如下所示：

输入：**sort** * > *newphone*

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  -rwxrwxrwx 1 deb                91235  Apr 22 18:57 ch08
  -rw-rw-r-- 1 deb                  611  Aug 25  1995 dickens
  -rw-rw-r-- 1 deb                  280  Jun 23 13:02 joyce
  -rw-rw-r-- 1 deb                  280  Aug 25  1995 joyce~
  -rw-rw-r-- 1 deb                   53  Jan 19 09:12 letterb
  -rw-rw-r-- 1 deb                  226  Aug 26  1995 phone
--%*-Dired:fiction                (Dired by name)--L17--Bot------------
(Shell command completed with no output)
```

操作系统对 *phone* 文件进行了排序，然后把执行结果输出到新文件 *newphone* 里。

我们已经创建了 *newphone* 文件，但它没有出现在屏幕画面里。要想看到 *newphone*，请按下“**g**”键。如下所示：

按下：**g**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  -rwxrwxrwx 1 deb                91235 Apr 22 18:57 ch08
  -rw-rw-r-- 1 deb                  611 Aug 25  1995 dickens
  -rw-rw-r-- 1 deb                  280 Jun 23 13:02 joyce
  -rw-rw-r-- 1 deb                  280 Aug 25  1995 joyce~
  -rw-rw-r-- 1 deb                   53 Jan 19 09:12 letterb
  -rw-rw-r-- 1 deb                  211 Jun 23 13:29 newphone
  -rw-rw-r-- 1 deb                  211 Jun 23 13:27 phone
--%*-Dired:fiction                (Dired by name)--L18--Bot------------
Reading directory /home/deb/fiction/...done
```

Emacs 刷新了 Dired 的显示情况，把 *newphone* 文件赫然列在其中。

那是不是必须先按下“**g**”键，Dired 才刷新屏显画面呢？这不一定。有些命令（马上就要讲到）会立刻刷新屏显画面；而另外一些命令（比如部分在文件上运行的 shell 命令）就不是这样。希望大家记住下面这条操作规则：如果没有在屏幕上看到预期应该出现的东西，那最好按下“**g**”键试试。

对文件组进行操作

到目前为止，我们讨论的都是一次对一个文件进行操作 —— 给出的命令只能对光标位置上的文件进行操作。Dired 的真正强大之处在于它能同时对多个文件进行操作。一旦掌握了几个快捷键，就能毫不费力地把目录管理得更好。先介绍几种用来选取文件的操作方法，然后再说说对那些被选中的文件都能做些什么操作。

选取文件

到目前为止，我们只介绍过如何给文件加上待删除标记的方法。当想对一组文件进行一些其他的操作时，必须先给它们加上一个星号作为待操作标记。按下“**m**”键给光标位置上的文件加上待操作标记，在以前会出现待删除标记，即字母“D”的地方将出现一个星号。输入“**3m**”将给本文件和它下面的两个文件加上待操作标记。给有关文件都加上星号标记之后，Emacs 会认为此后的操作都将是以这些加有操作标记的文件为对象的。也就是说，如果给 3 个文件加上了待操作标记并按下了“**Z**”键做压缩操作，Emacs 会认为想对那 3 个文件进行压缩。但是，在压缩操作结束后，那 3 个文件上的待操作标记还会保留在那里。那么，怎样才能在完成了这些文件上的操作之后去掉星号标记呢？

去掉星号待操作标记的方法是：按下“**ESC DEL**”（命令名是 **dired-unmark-all-files**）组合键，或者从“**Mark**（标记）”菜单里选择执行“**Unmark All**（去除全部标记）”操作。Emacs 会询问想去掉哪些标记。按回车键，Emacs 将去掉所有的待操作标记。

用正则表达式来选取文件

经常会有需要对一组彼此相关的文件进行选取，然后再对它们进行归档、移动、压缩或者删除等操作。在 UNIX 操作系统（以及大多数其他的操作系统）里，要想选取多个文件必须使用通配符；但在 Dired 里，得使用正则表达式。用正则表达式选取多个文件的方法是：先输入一个百分号“%”，再用“**m**”给文件加上待操作标记；或者再用“**d**”给文件加上待删除标记。

请看下面的例子。选取文件名以“*ch*”打头的所有文件。根据第三章里的正则表达式快速教程，我们知道用来匹配单词开头的特殊字符应该是“^”；因此，正则表达式“**^ch**”将把文件名以“*ch*”打头的所有文件都找出来。如下所示：

输入：**%m**

```
Buffers Files Tools Edit Search Minibuf Help
 /home/deb/fiction:

 total 639
 drwxrwxr-x  2 deb          512  Jun 23 13:29 .
 drwxrwxr-x 13 deb         7680  Jun 23 13:26 ..
 -rwxrwxrwx  1 deb        54576  Apr 22 18:55 ch01
 -rwxrwxrwx  1 deb        79915  Apr 22 18:55 ch02
 -rwxrwxrwx  1 deb        82809  Apr 22 18:56 ch03
 -rwxrwxrwx  1 deb        79434  Apr 22 18:56 ch04
 -rwxrwxrwx  1 deb        85805  Apr 22 18:56 ch05
 -rwxrwxrwx  1 deb       109282  Apr 22 18:56 ch06
 -rwxrwxrwx  1 deb        38191  Apr 22 18:57 ch07
 -rwxrwxrwx  1 deb        91235  Apr 22 18:57 ch08
 -rw-rw-r--  1 deb          611  Aug 25  1995 dickens
 -rw-rw-r--  1 deb          280  Jun 23 13:02 joyce
 -rw-rw-r--  1 deb          280  Aug 25  1995 joyce~
 -rw-rw-r--  1 deb           53  Jan 19 09:12 letterb
 -rw-rw-r--  1 deb          211  Jun 23 13:29 newphone
 -rw-rw-r--  1 deb          211  Jun 23 13:27 phone
--%*-Dired:fiction        (Dired by name)--L18--All----------
Mark files (regexp):
```

Emacs 要求输入一个正则表达式以便给文件加上待操作标记。

输入：**^ch**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  /home/deb/fiction:
  total 639
  drwxrwxr-x  2 deb                512  Jun 23 13:29 .
  drwxrwxr-x 13 deb                7680  Jun 23 13:26 ..
* -rwxrwxrwx  1 deb               54576  Apr 22 18:55 ch01
* -rwxrwxrwx  1 deb               79915  Apr 22 18:55 ch02
* -rwxrwxrwx  1 deb               82809  Apr 22 18:56 ch03
* -rwxrwxrwx  1 deb               79434  Apr 22 18:56 ch04
* -rwxrwxrwx  1 deb               85805  Apr 22 18:56 ch05
* -rwxrwxrwx  1 deb              109282  Apr 22 18:56 ch06
* -rwxrwxrwx  1 deb               38191  Apr 22 18:57 ch07
* -rwxrwxrwx  1 deb               91235  Apr 22 18:57 ch08
  -rw-rw-r--  1 deb                 611  Aug 25 1995 dickens
  -rw-rw-r--  1 deb                 280  Jun 23 13:02 joyce
  -rw-rw-r--  1 deb                 280  Aug 25  1995 joyce~
  -rw-rw-r--  1 deb                  53  Jan 19 09:12 letterb
  -rw-rw-r--  1 deb                 211  Jun 23 13:29 newphone
  -rw-rw-r--  1 deb                 211  Jun 23 13:27 phone
--%*-Dired:fiction                (Dired by name)--L18--All-----------
8 matching files marked.
```

Emacs给所有以“*ch*”打头的文件都加上选取标记，并且会告知它到底选取了多少个文件。

现在，已经有了一些加上了选取标记的文件，下面来看看都能对它们进行哪些操作。

文件组上的操作

在日常工作中，某个目录经常会因为文件过多而显得乱七八糟的。于是，决定按项目划分对它们进行归置。先建立一些子目录，再把文件分别移动到相关的子目录里去。这两件事都可以在 Dired 里完成。

假设文件名以“*ch*”打头的文件是一部小说的各个章节。需要一个名为“*novel*”的下级子目录来保存它们。于是，先按下“**+**”键（命令名是**dired-create-directory**）或者从“**Immediate**”菜单里选择执行“**Create Directory**（创建目录）”操作以创建一个目录。如下所示：

按下：+

```
Buffers Files Tools Edit Search Minibuf Help
  /home/deb/fiction:
  total 639
  drwxrwxr-x  2 deb                512  Jun 23 13:29 .
  drwxrwxr-x 13 deb                7680  Jun 23 13:26 ..
* -rwxrwxrwx  1 deb               54576  Apr 22 18:55 ch01
* -rwxrwxrwx  1 deb               79915  Apr 22 18:55 ch02
* -rwxrwxrwx  1 deb               82809  Apr 22 18:56 ch03
* -rwxrwxrwx  1 deb               79434  Apr 22 18:56 ch04
* -rwxrwxrwx  1 deb               85805  Apr 22 18:56 ch05
* -rwxrwxrwx  1 deb              109282  Apr 22 18:56 ch06
* -rwxrwxrwx  1 deb               38191  Apr 22 18:57 ch07
* -rwxrwxrwx  1 deb               91235  Apr 22 18:57 ch08
  -rw-rw-r--  1 deb                 611  Aug 25  1995 dickens
  -rw-rw-r--  1 deb                 280  Jun 23 13:02 joyce
  -rw-rw-r--  1 deb                 280  Aug 25  1995 joyce~
  -rw-rw-r--  1 deb                  53  Jan 19 09:12 letterb
  -rw-rw-r--  1 deb                 211  Jun 23 13:29 newphone
  -rw-rw-r--  1 deb                 211  Jun 23 13:27 phone
--%*-Dired:fiction        .      (Dired by name)--L18--All----------
Create directory:~/fiction/
```

Emacs 要求输入一个目录名。

输入：**novel RETURN**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
  /home/deb/fiction:
  total 639
  drwxrwxr-x  2 deb                512  Jun 23 13:29 .
  drwxrwxr-x 13 deb                7680  Jun 23 13:26 ..
* -rwxrwxrwx  1 deb               54576  Apr 22 18:55 ch01
* -rwxrwxrwx  1 deb               79915  Apr 22 18:55 ch02
* -rwxrwxrwx  1 deb               82809  Apr 22 18:56 ch03
* -rwxrwxrwx  1 deb               79434  Apr 22 18:56 ch04
* -rwxrwxrwx  1 deb               85805  Apr 22 18:56 ch05
* -rwxrwxrwx  1 deb              109282  Apr 22 18:56 ch06
* -rwxrwxrwx  1 deb               38191  Apr 22 18:57 ch07
* -rwxrwxrwx  1 deb               91235  Apr 22 18:57 ch08
  -rw-rw-r--  1 deb                 611  Aug 25  1995 dickens
  -rw-rw-r--  1 deb                 280  Jun 23 13:02 joyce
  -rw-rw-r--  1 deb                 280  Aug 25  1995 joyce~
  -rw-rw-r--  1 deb                  53  Jan 19 09:12 letterb
  -rw-rw-r--  1 deb                 211  Jun 23 13:29 newphone
  drwxrwxr-x  2 deb                 512  Jun 23 13:31 novel
  -rw-rw-r--  1 deb                 211  Jun 23 13:27 phone
--%*-Dired:fiction              (Dired by name)--L18--All----------
```

Emacs 创建一个新的子目录并把它显示在屏幕上。

接着，再把刚才加上待操作标记的那些“*ch*”文件移到这个新子目录里。用重命名命令“**R**”来做这件事。记住，Dired 的重命名命令与 UNIX 下的 **mv** 命令在执行效果方面是完全一样的。因为已经给一些文件加上星号标记了，所以，当按下“**R**”键的时候，Emacs 将认为是想对这些加有待操作标记的文件进行操作。如下所示：

按下：**R**

```
Buffers Files Tools Edit Search Minibuf Help
* -rwxrwxrwx  1 deb              109282  Apr 22 18:56 ch06
* -rwxrwxrwx  1 deb               38191  Apr 22 18:57 ch07
* -rwxrwxrwx  1 deb               91235  Apr 22 18:57 ch08
  -rw-rw-r--  1 deb                 611  Aug 25 1995 dickens
  -rw-rw-r--  1 deb                 280  Jun 23 13:02 joyce
  -rw-rw-r--  1 deb                 280  Aug 25  1995 joyce~
  -rw-rw-r--  1 deb                  53  Jan 19 09:12 letterb
  -rw-rw-r--  1 deb                 211  Jun 23 13:29 newphone
  drwxrwxr-x  2 deb                 512  Jun 23 13:31 novel
  -rw-rw-r--  1 deb                 211  Jun 23 13:27 phone
--%*-Dired:fiction              (Dired by name)--L18--Bot------------
ch01 ch02 ch03 ch04 ch05 ch06 ch07 ch08
*--**-Emacs:*Marked Files*      (Fundamental)--L1--All-----------------
Move *[8 files] to::~/fiction/
```

Emacs 询问想把加上选取标记的那些文件移到何处。

输入：**novel RETURN**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
/home/deb/fiction:
 total 639
 drwxrwxr-x  2 deb                 512  Jun 23 13:29 .
 drwxrwxr-x 13 deb                7680  Jun 23 13:26 ..
 -rw-rw-r--  1 deb                 611  Aug 25 1995 dickens
 -rw-rw-r--  1 deb                 280  Jun 23 13:02 joyce
 -rw-rw-r--  1 deb                 280  Aug 25  1995 joyce~
 -rw-rw-r--  1 deb                  53  Jan 19 09:12 letterb
 -rw-rw-r--  1 deb                 211  Jun 23 13:29 newphone
 drwxrwxr-x  2 deb                 512  Jun 23 13:31 novel
 -rw-rw-r--  1 deb                 211  Jun 23 13:27 phone
--%*-Dired:fiction              (Dired by name)--L10--All------------
Move:8 files
```

Emacs 完成文件移动操作。

这样，那些文件就都被移到新子目录里。用正则表达式来给文件加上待操作标记的方法，能够快速选中一组文件并对它们进行各种操作。

在允许对文件组进行的操作里还有一些更令人感兴趣的东西，其中之一是只需一条命令就能对它们的内容进行全面的查询－替换。对大型项目来说，所谓“最后一分

钟”的修改，往往需要对很多文件里的某些特定文本进行全面的查找和替换。这个操作的具体步骤是：先把想包括在查询－替换操作里的文件都选上，然后按下“**Q**”（命令名是**dired-do-query-replace**）键；接下来，先后输入查找字符串和替换字符串（这两个字符串既可以是普通文本，也可以是一个正则表达式）；Emacs将开始一次查询－替换操作，依次进入被选中的各个文件里（注5）。但有一点必须注意：如果在这次查询－替换操作的过程中另外开始了一次递归编辑，这次查询－替换操作就将到此为止，只能重新回到Dired编辑缓冲区里才能再次继续这个查询－替换操作。

正如大家所看到的，文件维护和清理方面的大部分工作都可以在Dired里轻松地完成。表5-2对Dired命令进行了汇总，其中有些命令或者没有介绍，或者介绍得不够细致。Dired里需要学习的东西确实还有很多，不过，既然大家已经入了门，以后的修行还是靠自己吧。

表5-2：Dired命令速查表

键盘操作	命令名称	动作
C-x d *Files → Open Directory*	**dired**	启动Dired
C *Operate → Copy to*	**dired-do-copy**	复制文件
d *Mark → Flag*	**dired-flag-file-deletion**	给文件加上待删除标记
D *Operate → Delete*	**dired-do-delete**	以问答方式立刻删除某个文件
e *Immediate → Find This File*	**dired-find-file**	编辑文件
f	**dired-advertised-find-file**	查找文件（以对它进行编辑）
g *Files → Revert Buffer*	**revert-buffer**	从磁盘上重新读入目录
G *Operate → Change Group*	**dired-do-chgrp**	改变文件的组权限

注5：这项功能是在版本19.29里增加进来的。

表 5-2: Dired 命令速查表（续）

键盘操作	命令名称	动作
k	**dired-do-kill-lines**	从画面上删除光标所在的那一行（不是删除文件）
m *Mark → Mark*	**dired-mark**	给文件加上“*”待操作标记
n	**dired-next-line**	移动到下一行
o *Immediate → Find in Other Window*	**dired-find-file-other-window**	在另外一个窗口里查找文件，移动到新窗口
C-o *Immediate → Display in Other Window*	**dired-display-file**	在另外一个窗口里查找文件，但不移动到新窗口
P *Operate → Print*	**dired-do-print**	打印文件
q	**dired-quit**	退出 Dired
Q	**dired-do-query-replace**	在加有待操作标记的文件里对字符串进行查找－替换操作
R *Operate → Rename to*	**dired-do-rename**	重新命名文件
u *Mark → Unmark*	**dired-unmark**	去掉待操作标记
v *Immediate → View This File*	**dired-view-file**	查看文件内容
x	**dired-do-flagged-delete**	删除加有待删除标记“**D**”的文件
Z *Operate → Compress*	**dired-do-compress**	对文件进行压缩或解压缩操作
ESC DEL *Mark → Unmark All*	**dired-unmark-all-files**	把文件上的各种待操作标记都去掉
~ *Mark → Flag Backup Files*	**dired-flag-backup-files**	给备份文件加上待删除标记；去掉这些标记的命令是“**C-u ~**”

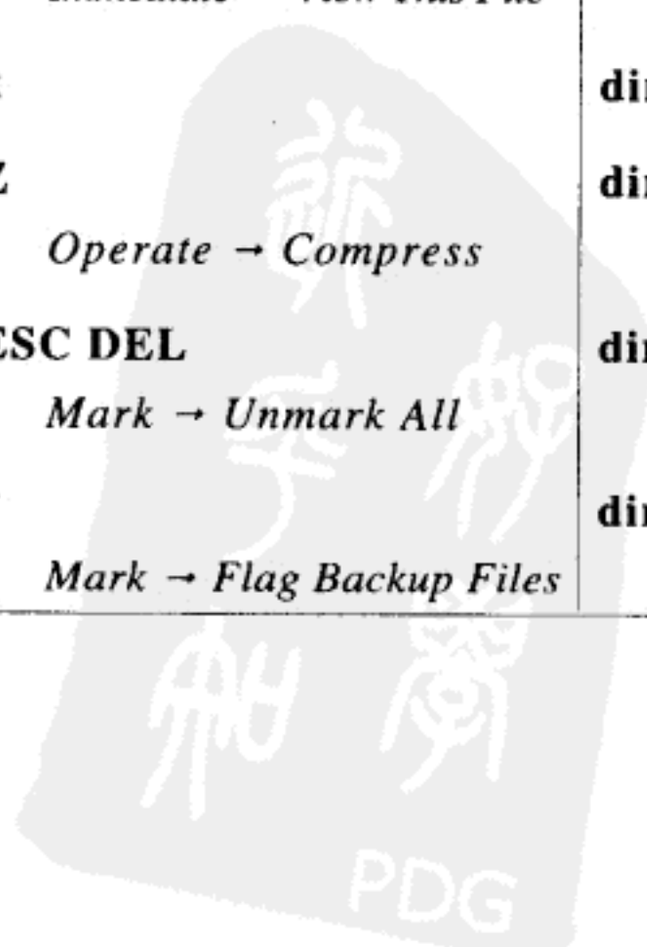


表 5-2: Dired 命令速查表（续）

键盘操作	命令名称	动作
* *Mark → Mark Executables*	**dired-mark-executables**	给可执行文件加上“*”标记；去掉这些标记的命令是“**C-u** *”
# *Mark → Flag Auto-save Files*	**dired-flag-auto-save-files**	给自动保存文件加上待删除标记；去掉这些标记的命令是“**C-u #**”
. *Mark → Mark Old Backups*	**dired-clean-directory**	给带编号的备份文件（如果有的话）加上待删除标记
/ *Mark → Mark Directories*	**dired-mark-directories**	给目录加上“*”标记；去掉这些标志的命令是“**C-u /**”
= *Immediate → Diff*	**dired-diff**	把这个文件与（文本标记处的）另一个文件进行比较
ESC = *Immediate → Compare With Backup*	**dired-backup-diff**	把这个文件与它的备份文件进行比较
! *Operate → Shell command*	**dired-do-shell-command**	以问答方式执行 shell 命令；命令的操作对象是加有待操作标记的文件
ESC } *Mark → Next Marked*	**dired-next-marked-file**	移动到加有“*”或“**D**”标记的下一个文件
ESC { *Mark → Previous Marked*	**dired-prev-marked-file**	移动到加有“*”或“**D**”标记的上一个文件
%d *Regexp → Flag*	**dired-flag-files-regexp**	给匹配这个正则表达式的文件加上待删除标记
%m *Regexp → Flag*	**dired-mark-files-regexp**	给匹配这个正则表达式的文件加上待操作标记
+ *Immediate → Create Directory*	**dired-create-directory**	创建一个目录
> *Subdir → Next Dirline*	**dired-next-dirline**	移动到下一个目录
< *Subdir → Prev Dirline*	**dired-prev-dirline**	移动到上一个目录

表 5-2: Dired 命令速查表（续）

键盘操作	命令名称	动作
s	**dired-sort-toggle-or-edit**	对Dired编辑缓冲区里的文件清单按日期或按文件名重新排序（在两者之间切换）

简单的目录清单

有两个命令可以把目录的文件清单放到一个编辑缓冲区里以供查看：命令“**C-u C-x C-d**”给出的是一个详细的目录文件清单，类似于“**ls -l**”命令的执行结果；而命令“**C-x C-d**”给出的是一个简单的目录文件清单，类似于“**ls -F**”命令的执行结果 —— 文件清单里的目录名后面跟有一个斜线字符（/），可执行文件名后面跟有一个星号（*），符号链接名后面跟有一个“@”字符。

与直接对文件进行操作的 Dired 模式不同，这些文件清单都只是些临时性的编辑缓冲区，可以像对待其他 Emacs 编辑缓冲区那样对它们做任意编辑 —— 在屏幕上做的修改对底层的文件结构不会有任何影响。这个功能很适合用来打印文件目录清单 —— 可以给文件加上一些注释或者根据打印出来的文件清单决定需要删除哪些东西。

Emacs 中的打印操作

Emacs提供了好几个用来打印编辑缓冲区和文本块内容的命令。如果想在打印编辑缓冲区内容时加上页码、页眉和文件名，请输入“**ESC x print-buffer RETURN**”或者从“**Tools**”菜单里选择执行“**Print Buffer**（打印编辑缓冲区）”操作。这个命令会先把整个编辑缓冲区的内容送往pr程序（这是一个对文本进行简单排版的程序），然后再送往 **lpr** 程序（这个程序负责把文件内容送往打印机）（注 6）。如果想直接打印出文件，不需要 **pr** 程序提供的页眉和页码，那么请输入“**ESC x lpr-buffer**”命令。还可以用这些命令来打印文件中某个被选定的区域。具体做法是：先定义一个文本块，在文本块的一端设置一个文本标记，然后把光标移动到文本块的另一端去；给出命令“**ESC x print-region RETURN**”（或命令“**ESC x lpr-region**

注 6: 如果使用的是 System V UNIX，与此对应的 UNIX 命令可能会稍有不同。

RETURN”）或者从“**Tools**”菜单里选择执行“**Print Region**（打印文本块）”操作。

lpr-buffer和**lpr-region**命令会检查变量**lpr-switches**的取值，以判断是否还有需要向UNIX的**lpr**命令传递的其他选项参数。这些选项参数可以用来指定具体完成打印工作的打印机或者用来对许多其他事项进行设定；详细资料请参考UNIX使用手册里对**lpr**命令的介绍。比如，如果想让Emacs中的打印工作都使用名为“lpt1”的打印机来完成，就需要把变量**lpr-switches**设置为“-Plpt1”，也就是说，需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq lpr-switches '("-Plpt1")
```

请注意这条语句中的单引号、括号和字符串“`-Plpt1`”。这种奇怪的写法是LISP语法所要求的，详细资料请参考第十三章中的有关讨论。

也可以直接从Dired里开始打印。如果想打印光标所在位置上的文件，按下“**P**”或者从“**Operate**”菜单里选择执行“**Print**”操作。Emacs将把默认的打印命令及其各种必要的附加参数（比如用来指定特定打印机的参数）放到辅助输入区里。在按下回车键之前，还有机会对打印命令做一些调整。

Emacs还提供把编辑缓冲区当做一个PostScript文件打印输出的命令。如果在编辑缓冲区里使用了粗体字之类的文本属性（这方面内容可以在第十四章查到），并且想在打印这个编辑缓冲区时带上这些属性，请输入“**ESC x ps-print-buffer-with-faces**”命令或者从“**Tools**”菜单里选择执行“**Print Postscript Buffer**（按Postscript格式打印编辑缓冲区）”操作。

表5-3对Emacs在打印方面的操作命令进行了汇总。

表5-3：打印命令速查表

键盘操作	动作
ESC x print-buffer *Tools → Print Buffer*	打印编辑缓冲区（类似于UNIX中的“**pr	lpr**”命令）
ESC x print-region *Tools → Print Region*	打印文本块（类似于UNIX中的“**pr	lpr**”命令）
ESC x lpr-buffer	打印编辑缓冲区，但不带页码（类似于UNIX的**lpr**命令）

表 5-3: 打印命令速查表（续）

键盘操作	动作
ESC x lpr-region	打印文本块，但不带页码（类似于 UNIX 的 **lpr** 命令）
ESC x dired-do-print *Operate → Print*	在 Dired 里，把默认的打印命令放到辅助输入区里；在按回车键执行它之前还可以对它进行修改
ESC x ps-print-buffer-with-faces *Tools → Postscript Print Buffer*	打印带有文本属性的编辑缓冲区

用 Emacs 查阅 UNIX 的在线文档

Emacs 还可以用来查阅 UNIX 的在线文档（即“manpages”——UNIX 命令的使用手册页），具体做法是输入“**ESC x man**”或者从“**Help**”菜单里选择执行“**Man**”操作。这个命令将把排好版的使用手册页，放到一个新创建的编辑缓冲区里，用 Emacs 命令前后翻阅（或者复制其中的内容）。做法很简单：只要输入“**ESC x manual -entry RETURN** *UNIX-command-name* **RETURN** or”或者从“**Help**”菜单里选择执行“**Man**”操作即可。

说到UNIX命令的名字，既可以使用简单如“**ls**”这样的名字，也可以使用“**ttytab(5)**”之类的使用手册页章节号。

使用**manual-entry**命令的好处是，可以随心所欲地前后翻阅使用手册页，这在大部分UNIX版本下都不容易做到。再有，如果准备在shell模式下查阅使用手册页，如果设置不当，它们就会显示得乱七八糟；而**manual-entry**命令就不会出现这样的问题，它永远能显示出整齐的文本。但要注意的是：**manual-entry** 命令将运行在后台——这可能会让人觉得 Emacs 似乎没有听到指示。不过，别着急，一个显示使用手册页输出的窗口最终会在屏幕上弹出，并且能够像对待任何Emacs编辑缓冲区那样前后翻阅之。

时间管理工具的使用

Emacs 说不定会成为最离不开的个人信息助理。它可以把时间显示在状态行上，而

Emacs的日历和日记功能要大大优于它们的UNIX对手。Emacs的日历功能可以同时显示3个月，而它的日记功能则可以显示每天的活动和日程安排记录。

显示时间

有时工作可能需要能够随时看到时间。输入“**ESC x display-time**”命令就可以在状态行上显示一个时钟。如果想让时钟自动出现，请把下面这条语句添加到“*.emacs*”文件里:

```
(display-time)
```

这样，当下次进入Emacs的时候，状态行上就会自动出现一个时钟。

显示日历

如果想显示日历，请输入“**ESC x calendar**”命令。Emacs会在一个日历窗口里显示3个月（上个月、本月、下个月）的日历。如下所示:

输入: **ESC x calendar**

```
Buffers Files Tools Edit Search Help
It was the best of times,it was the worst of times,it was the age of
wisdom,it was the age of foolishness,it was the epoch of belief,it
was the epoch of incredulity,it was the season of Light,it was the
-----Emacs:dickens        (Text Fill)--L1--All----------------------
      March 1997                April 1997                May 1997
 Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa      Su Mo Tu We Th Fr Sa
                    1                 1  2  3  4  5                 1  2  3
  2  3  4  5  6  7  8      6  7  8  9 10 11 12      4  5  6  7  8  9 10
  9 10 11 12 13 14 15     13 14 15 16 17 18 19     11 12 13 14 15 16 17
 16 17 18 19 20 21 22     20 21 22 23 24 25 26     18 19 20 21 22 23 24
 23 24 25 26 27 28 29     27 28 29 30 25 26 27     28 29 30 31
 30 31
C-x <    Calendar   ? info/o other/. today    Fri,Apr 18,1997    C-x >-
```

Emacs将把光标放在今天的日期上，而这个日期也将同时出现在状态行上。日历上没有书写的空间（这也就是为什么会有日记功能的原因，我们马上就会讲到日记了）。

在默认情况下，星期是从星期日开始的。如果想从星期一开始，请输入“**ESC x set-variable calendar-week-start RETURN 1 RETURN**”命令。再次进入日历时这个

设置就会生效。如果想让日历永远从星期一开始，请把下面这条语句添加到“*.emacs*”文件里:

```
(setq calendar-week-start-day 1)
```

如果想在每次进入Emacs的时候都能看到日历，请把下面这条语句添加到“*.emacs*”文件里:

```
(calendar)
```

在日历中移动

身处日历中的时候，Emacs 会明智地按日期而不是按字符移动。“**C-f**”组合键会移动到后一天;“**C-b**”组合键会移动到前一天;“**C-n**”组合键会移动到下一星期的同一天;而“**C-p**”组合键则会移动到上一星期去。键盘上的方向键有与此相同的作用。“**ESC }**”和“**ESC {**”以月份为单位做前后移动;“**C-x]**”和“**C-x [**”以年为单位做前后移动。“**C-v**”前卷三个月，“**ESC v**”后卷三个月。

以上讨论的这些移动命令都是相对于光标位置的移动。如果今天是星期二，按下“**C-n**”将移动到上一个星期二。如果今天是 1 月 25 日，按下“**ESC }**”将移动到 2 月 25 日。如果今天是 1997 年 8 月 15 日，按下“**C-x [**”将移动到 1996 年 8 月 15 日。

其他一些命令可以移动到星期、月份、年的开始或者结尾。“**C-a**”和“**C-e**”能够分别移动到星期的开始或结束;“**ESC a**”可移动到月份的开始;而“**ESC <**”则移动到年的开始。表 5-4 对这些日历中的移动命令进行了汇总。

如果想直接到达某个特定的日期，请使用“**g d**”命令。Emacs 将依次提示输入年、月、日，然后会移动到选定的日期(这个命令用来回答诸如“我在2002年的生日是星期几?”之类的问题是再合适不过了)。

表 5-4: 日历移动命令速查表

键盘操作	命令名称	动作
. *Goto → Today*	**calendar-goto-today**	移动到今天的日期
C-f	**calendar-forward-day**	向前移动一天
C-b	**calendar-backward-day**	向后移动一天

表 5-4: 日历移动命令速查表（续）

键盘操作	命令名称	动作
C-n	**calendar-forward-week**	向前移动一星期
C-p	**calendar-backward-week**	向后移动一星期
ESC }	**calendar-forward-month**	向前移动一个月
ESC {	**calendar-backward-month**	向后移动一个月
C-x]	**calendar-forward-year**	向前移动一年
C-x [	**calendar-backward-year**	向后移动一年
C-a *Goto → Beginning of Week*	**calendar-beginning-of-week**	移动到本星期的开始
C-e *Goto → End of Week*	**calendar-end-of-week**	移动到本星期的结束
ESC a *Goto → Beginning of Month*	**calendar-beginning-of-month**	移动到本月的开始
ESC e *Goto → End of Month*	**calendar-end-of-month**	移动到本月的结束
ESC < *Goto → Beginning of Year*	**calendar-beginning-of-year**	移动到本年的开始
ESC > *Goto → End of Year*	**calendar-end-of-year**	移动到本年的结束
C-u *n*	**universal-argument**	重复执行随后命令 ***n*** 次
g d *Goto → Other Date*	**calendar-goto-date**	移动到指定日期
o	**calendar-other-month**	把指定月份放在日历画面的中间
C-x < *Scroll → Forward 1 Month*	**scroll-calendar-left**	前卷一个月
C-x > *Scroll → Backward 1 Month*	**scroll-calendar-right**	后卷一个月
SPACE	**scroll-other-window**	卷动另外一个窗口

显示节假日

下面再来说说人们都很感兴趣的一个话题：节假日。如果想查看当前日历画面中的

3个月里都有哪些节假日，请按下“a”键（命令名是**list-calendar-holidays**）或者从“**Holidays**（节假日）”菜单里选择执行“**3 Months**”（3个月）操作。如下所示：

按下：**a**

```
Buffers Files Tools Edit Search Help
Monday,March 17,1997:St.Patrick's Day
Thursday,March 20,1997:Vernal Equinox 8:58am (EST)
Friday,March 28,1997:Good Friday
Sunday,March 30,1997:Easter Sunday
Tuesday,April 1,1997:April Fool's Day
Sunday,April 6,1997:Daylight Savings Time Begins 2:00am (EST)
Tuesday,April 22,1997:Passover
Friday,May 9,1997:Islamic New Year 1418
Sunday,May 11,1997:Mother's Day
Monday,May 26,1997:Memorial Day
--------------------Notable Dates from March to May,1997----------------
      March 1997                 April 1997                 May 1997
  Su Mo Tu We Th Fr Sa       Su Mo Tu We Th Fr Sa       Su Mo Tu We Th Fr Sa
                     1                1  2  3  4  5                   1  2  3
   2  3  4  5  6  7  8        6  7  8  9 10 11 12        4  5  6  7  8  9 10
   9 10 11 12 13 14 15       13 14 15 16 17 18 19       11 12 13 14 15 16 17
  16 17 18 19 20 21 22       20 21 22 23 24 25 26       18 19 20 21 22 23 24
  23 24 25 26 27 28 29       27 28 29 30                25 26 27 28 29 30 31
  30 31
C-x <    Calendar   ? info/o other/. today     Fri,Apr 18,1997    C-x >
Looking up holidays...done
```

Emacs列出屏显日历期间内的节假日。

正如大家所看到的，Emacs知道的节假日还真不少呢。如果在日历中的其他地方却想查看本月前后的节假日，请输入“**ESC x holidays**”命令；Emacs会把它们列出来。如果想查看今天是不是一个节假日，请按下“**h**”键或者从“**Holidays**（节假日）”菜单里选择执行“**One Day**（当天）”操作。

按下“**x**”键会让节假日显示为特殊形式。如果可能，它们就会显示为另外一种字体或者颜色；如果不能，Emacs将会在节假日的右边显示一个星号（*）。再按下“**u**”键将去掉这些标记。

这里介绍的只是一些基本的日历命令。Emacs还能告知日出、日落时间和月相。可以选择查看其他种类的日历，比如玛雅历、法国革命历、伊斯兰历和犹太历等等。不过这些东西还是留给大家自己去探索吧。

日记功能的使用

日记功能与日历功能是紧密相关的，它可以给某个特定的日期加上一些文字记录。用户可以输入一个完整的日程安排，也可以只注明重大的事件。文字记录的详细程度完全由自己掌握。

创建一个日记文件

要想使用日记，必须有一个用来记录重大事件或者日程安排的“*.diary*”文件。可以用它来提醒自己每个星期四要对系统进行备份，每两星期领一次薪水，七月的前两个星期是休假，或者母亲的生日是8月6日等等。

这个文件的名字必须是“*.diary*”，并且必须存在于自己的主目录里。把自己打算记住的日期插入到这个文件里，或者说是让Emacs把它们记在里面。“*.diary*”文件不需要全部是统一的格式，也不需要按什么顺序进行排序。日期的格式允许混用：可以把“1997年12月19日”写成“12/19/97”、“Dec 19 97”，或者“dec 19 1997”。下面是从某个“*.diary*”文件里节选出来的几行记录，用它来说明一下日记条目的具体用法：

```
11/14 My birthday
July 17 1997 Company picnic
March 18 1997 Annual report due
January 8 1997 Hair appointment
&Saturday Tea with Queen Elizabeth
Friday Payday
```

如果没有指定年份，Emacs将认为想把每一年的这个日期都记下来，比如“birthday”（生日）那一行。如果指定的是星期几而不是某个确切的日期（比如“Tea with Queen Elizabeth（与女王喝午茶）”的那一行），Emacs就会在每个星期六显示这个日记条目。在某个日记条目之前放上一个“&”字符的意思是让Emacs不要在日历上把该条目显露出来（不想每个星期六上都有一个记号，或者不想让别人知道自己与皇家有什么瓜葛）。

Emacs对欧式日期格式（“DD/MM/YY”或“9 October 1997”）和美式日期格式

（“MM/DD/YY”或“October 9, 1997”）都接受，但在使用“*.diary*”文件之前（注 7），必须先把自己想用的日期格式确定下来。

添加日记条目

既可以自己写日记条目，也可以让 Emacs 帮助把条目放入。如果想让 Emacs 帮助，需要先用“**ESC x calendar**”命令进入日历功能；然后用“**g d**”命令移动到某个特定的日期；再按下“**i d**”（命令名是**insert-diary-entry**）两键或者从“**Diary**”菜单里选择执行“**Insert Daily**（插入日记）”操作。Emacs 将进入日记窗口，窗口里面已经写好了日期；可以接在日期的后面写出一个日记条目。如果写的东西不止一行，请在第二行和以后每一行的开始留出一个空格，这是为了让 Emacs 明白它是一个续行。在写完这个日记条目之后，Emacs 会留在 diary 编辑缓冲区里以便用户继续写出其他条目。如果想离开 diary 编辑缓冲区进入另一个编辑缓冲区，请按下“**C-x b**”组合键。

insert-diary-entry命令会认为只想写一条一次性的日记条目。如果想写出一个周期性的日记条目，就必须使用另外几个命令。如果想插入以一星期为一个周期的日记条目，请按下“**i w**”两键或者从“**Diary**”菜单里选择执行“**Insert Weekly**（插入周记）”操作。Emacs 将进入 diary 编辑缓冲区，那里已经写好了是星期几。输入以一星期为一个周期的活动（比如员工会等），然后保存“*.diary*”文件。如果想插入以一年为一个周期的日记条目，请按下“**i y**”两键或者从“**Diary**”菜单里选择执行“**Insert Yearly**（插入年记）”操作。Emacs 进入 diary 编辑缓冲区，那里已经写好了月份和日期，然后就可以输入以一年为一个周期的事情了。

还可以输入以任意时间段为循环周期的日记条目，这些事件将按固定的时间间隔反复发生，比如用来提醒自己每 3 个月给汽车更换一次机油的备忘条这类的东西。具体操作方法是：先移动到上一次更换机油的日子，然后按下“**i c**”或者从“**Diary**”菜单里选择执行“**Insert Cyclic**（插入循环性事件）”操作。Emacs 提问“Repeat every how many days:（每隔多少天重复一次）”，输入更换机油的天数。Emacs 将写出一个负责处理此次循环时间段的 LISP 函数，并把它插入到 diary 编辑缓冲区，接在这个 LISP 函数的后面把事项写出来。比如提醒更换机油的备忘条内容。

注 7：默认的日期格式是美式日期格式。如果你想指定使用欧式日期格式，请在“*.emacs*”文件里加上“(setq european-calendar-style 't)”这条语句。

Emacs插入的日记条目如下所示（笔者的日记里就有这样一个提醒更换机油的备忘条目）:

```
%%(diary-cyclic 90 12 23 1997) Change the oil
```

这个日记条目的意思是：从插入这个条目的那一天 —— 1997 年 12 月 23 日开始，每隔 90 天更换一次机油。

可以把一个日期段都标记下来，比如持续一个星期的会议或者假期等。具体做法是：先把光标移动到日期段的第一天并按下“**C-SPACE**”组合键以设置一个时间标记（注 8）；然后移动到日期段的最后一天（要使用“**C-f**”、“**C-n**”等日历移动命令）并按下“**i b**”两键或者从“**Diary**”菜单里选择执行“**Insert Block**”操作。Emacs 将进入 diary 编辑缓冲区，那里将已经插入有代表在日历上选取的那个星期的 LISP 函数。接在 Emacs 插入的那个 LISP 函数的后面把想要的文本写进去即可。Emacs 插入的日记条目可能如下所示:

```
%%(diary-block 3 15 1997 3 20 1997) Trip to Alabama
```

这个条目的意思是从 3 月 15 日到 3 月 20 日，要去 Alabama 州旅行。

如果需要在每个月的15号提交开支报告该怎么办？Emacs允许在日期里使用星号通配符（*），就像在按下“**i m**”（命令名是 **insert-monthly-diary-entry**）两键时将会看到的那样。Emacs 会先插入一个星号来代替月份，然后是日子。比如，如果想对每个月的 15 号都做出安排，日记条目里的日期设置就将是“* 15”；日记内容还像往常一样写在日期项的后面。

现在，你们已经知道Emacs是如何构造日记条目的了，大家不妨参照Emacs所做的设置试着写几条自己的日记条目。说到底，“*.diary*”文件与任何其他的Emacs 文件是一样的，大家可以随心所欲地对它进行修改、插入条目或者删除条目。惟一的要

注 8：如果平时把**set-mark**命令绑定到另外一个键盘动作上或者通常拼写出那个命令，那么要想在日历里标记一个时间段就会遇到麻烦。在日历里，“**C-SPACE**”和“**C-@**”组合键将运行**calendar-set-mark**命令而不是**set-mark**命令。也就是说，所加的标记将以时间日期为对象，而不再以屏幕上的文字内容为对象。为了在日历里正确地选取一个时间段（即选取对象是时间日期而不是屏幕上的文字内容），必须用“**C-SPACE**”、“**C-@**”或者“**ESC x calendar-set-mark**”来设置时间标记。

求是必须在完成设置工作之后保存这个文件。下面，我们来看看怎样才能在到期的时候把日记条目显示出来。

显示日记条目

如果想查看每个给定日期的日记条目，请在日历画面里按下“**d**”键。如果想查看整个的“*.diary*”文件，请在日历画面里按下“**s**”键。如果想在启动Emacs的时候让今天的日记条目自动显示出来，请把下面这条语句添加到“*.emacs*”文件里：

```
(diary)
```

这样，当启动Emacs的时候，如果当天有日记条目，那么此日记条目就将自动显示出来。举个例子，假设前些时候已经给好友的生日加上了标记，而今天恰好就是这一天。那么，当启动Emacs的时候，屏幕画面就可能是下面这样的：

启动Emacs。

```
Buffers Files Tools Edit Search Help

-----Emacs:*scratch*        (Lisp Interaction)--L16--Bot--------------

Jan 20 Jeanne's birthday
---------------------Diary for Tuesday, January 20, 1998---------------
```

Emacs把关于朋友生日的日记条目显示在屏幕上。

如果某一天没有日记条目，Emacs也就不会显示什么日记了。如果在启动Emacs的时候给出了两个文件名，也就是说将在两个窗口里进行编辑工作，那么日记也不会被显示出来。

如果已经在“*.emacs*”文件里添加了“(calendar)”项，以便日历自动显示在屏幕上，那么日历会阻止日记的显示；如果想查看日记，就必须删除日历。

如果想把有日记条目的日期在日历上标记出来——也许是使用另外一种字体、另外一种颜色或者是在日期后面加上一个加号（+），那么请在日历画面里按下“**m**”键或者从“**Diary**”菜单里选择执行“**Mark All**”操作。如果想去掉这些标记，请按下“**u**”键或者从“**Holidays**”菜单里选择执行“**Unmark**”操作（这个命令将去掉用来突出日记条目和节假日的有关标记）。

表 5-5 对与日历和日记操作有关的命令进行了汇总。

表 5-5: 日历和日记命令速查表

键盘操作	命令名称	动作
p d	**calendar-print-day-of-year**	显示今天是本年度的第几天（比如 365 天里的第 364 天）
SPACE	**scroll-other-window**	卷动另一个窗口
q	**exit-calendar**	退出日历功能
a *Holidays → 3 Months*	**list-calendar-holidays**	显示日历画面中的 3 个月里的节假日情况
h *Holidays → 1 Day*	**calendar-cursor-holidays**	显示今天的节假日（如果有节假日）
x *Holidays → Mark*	**mark-calendar-holidays**	突出显示节假日，节假日将被显示为另外一种字体、另外一种颜色或者在它们旁边加上一个星号（*）
u *Holidays → Unmark*	**calendar-unmark**	去掉用来突出节假日的标记（与 **x** 命令的功能正好相反）
i w *Diary → Insert Weekly*	**insert-weekly-diary-entry**	根据此时是星期几添加一项以一星期为循环周期的日记条目
i y *Diary → Insert Yearly*	**insert-yearly-diary-entry**	添加一项以一年为循环周期的日记条目
i d *Diary → Insert Daily*	**insert-diary-entry**	为指定日期添加一项日记条目
i m *Diary → Insert Monthly*	**insert-monthly-diary-entry**	添加一项以一个月为循环周期的日记条目
i c *Diary → Insert Cyclic*	**insert-cyclic-diary-entry**	添加一项以 *n* 天为循环周期的日记条目
i a *Diary → Insert Anniversary*	**insert-anniversary-diary-entry**	添加一项以一年为循环周期的日记条目
i b *Diary → Insert Block*	**insert-block-diary-entry**	添加一项日期段条目

表 5-5: 日历和日记命令速查表（续）

键盘操作	命令名称	动作
m	**mark-diary-entries**	突出显示日记条目，它们将被显示为另外一种字体、另外一种颜色或者在它们旁边加上一个加号（+）
d	**view-diary-entries**	显示当前日期的日记条目
s *Diary → Show All*	**show-all-diary-entries**	显示“*.diary*”文件的内容
ESC =	**calendar-count-days-region**	计算某个时间段里的天数
M *Moon → Lunar Phases*	**calendar-phases-of-moon**	显示 3 个月期间的月相情况
S	**calendar-sunrise-sunset**	根据给定的经度和纬度，显示当前日期的日出、日落时间
C-SPACE 或 **C-@**	**calendar-set-mark**	以时间日期为对象设置（时间）块标记，而不是按普通情况设置文本标记

用好 Emacs 工作环境

现在，我们已经知道有些工作不用离开Emacs也仍然能够完成。接下来的两章将对能够在Emacs里完成的工作做更多的介绍。第六章将介绍如何在Emacs里收/发电子邮件，以及张贴/阅读Usenet新闻。第七章讨论如何使用Emacs的Telnet、FTP和WWW等功能访问因特网。通过学习使用这些Emacs里的常用工具，大家将能够把自己更多的工作集成到Emacs工作环境中。

子邮件地址很复杂的人发信，可以在本章“使用假名”一节里找到一个能够减轻这类负担的方法。

用 Emacs 发送邮件

从 Emacs 里向外发送电子邮件是很简单的。如果想发送一个邮件，那么请按下“**C-x m**”组合键，进入由 Emacs 创建的一个名为“*mail*”的编辑缓冲区。如下所示：

按下：**C-x m**

```
Buffers Files Tools Edit Search Headers Mail Help
To:
Subject:
--text follows this line--

-----Emacs:*mail*        (Mail Fill)--L1--All----------------------
```

准备开始输入邮件的内容。

在“To:(收信人)”栏里填上邮件接收人的电子邮件地址；然后在下一行填上这封邮件的“Subject（主题）”——这一行可以不填。如果在“To:”栏填写了多个地址，就可以把邮件同时发送给好几个人；但不要忘了用逗号把各个地址隔开。如果收信人不止一行，别忘记在第二行和以后各行的开始加上一个空格（只要是用来填写电子邮件地址的地址栏，比如马上就要介绍到的“CC:”、“BCC:”栏等就都适用于这条规则）。

完成上述书写邮件内容前的准备工作之后，把光标移动到“--text follows this line--”一行的下面开始输入邮件内容，邮件内容可以用任何 Emacs 命令来编辑。千万不要删除或者修改“--text follows this line--”标记；Emacs 要靠它来区分邮件信头和邮件信体，如果删除，Emacs 的电子邮件功能就无法正常工作。这一行不会随邮件一起发送，它的作用只是告诉邮件的信头（收信人、主题等）在哪里结束，邮件的信体又是从哪里开始的。

在书写邮件内容的时候，可以使用 Emacs 任何一种编辑功能，比如拼写检查、单词简写模式等等。写好邮件之后，按下“**C-c C-c**”组合键或者从“**Mail**”菜单里选

择执行“**Send Mail**（发送邮件）”操作就可以把它发送出去。我们后面还会介绍一些更花哨的东西，但上面介绍的是发送电子邮件最基本的做法。被发送的所有邮件都会标明其发送者，并且会自动加上当时的日期和时间 —— 虽然这些都没有在“*mail*”编辑缓冲区里显示出来。

编写邮件消息的正常编辑画面：

```
Buffers Files Tools Edit Search Headers Mail Help
To: deb, ralph, fred@grom.com, tom@jones.net, victor@hugo.org,
 dorothy@sayers.com, randal@pierre.com
Subject: Upcoming Naysayers Meeting
--text follows this line--
The naysayers will meet Thursday at 4:00 p.m.in the Green Room.

--**-Emacs:        *mail*(Mail Fill)--L4--All----------------------
```

写好了一条消息，现在就可以把它发送出去。

按下：**C-c C-C**

```
Buffers Files Tools Edit Search Help
Stately, plump Buck Mulligan came from the stairhead, bearing a bowl
of lather on which a mirror and a razor lay crossed. A yellow
dressing gown, ungirdled, was sustained gently behind him by the mild
morning air. He held the bowl aloft and intoned:
  --Introibo ad altare Dei
--          (Text Fill)        --L1--All--------------------------------
Sending...done
```

已经把消息发送出去，Emacs 又回到刚才编辑的编辑缓冲区里。

把邮件发送出去以后，Emacs 将把邮件编辑缓冲区从屏幕上删除，然后回到刚才正在进行编辑的编辑缓冲区里 —— 在这个例子里，就是回到那个叫做“joyce”的编辑缓冲区里。

有一件事需要特别提醒大家注意：因为“**C-c C-c**”或“**C-c C-s**”（这个命令的作用是送出邮件但仍然停留在邮件编辑缓冲区里）组合键在“*mail*”编辑缓冲区外很常用，所以很容易出现因不小心按下了这两个组合键，而错误地把一封没有写完的邮件发送出去的情况。如果在这方面总是遇到麻烦，可以先在另外一个编辑缓冲区里写好邮件的内容，等准备发送它们的时候再把它们复制到“*mail*”编辑缓冲区里。

从窗口或窗格里发送邮件

把邮件编辑缓冲区放到另外一个窗口里，而不是放到正在做其他编辑工作的当前窗口里往往更方便，也更容易在编辑缓冲区之间复制它们。在另外一个窗口里发送电子邮件的操作方法是：按下“**C-x 4 m**”组合键，填好“To:”和“Subject:”栏，然后移到下面输入邮件内容。如下所示：

按下：**C-x 4 m**

```
Buffers Files Tools Edit Search Headers Mail Help
Stately, plump Buck Mulligan came from the stairhead, bearing a bowl
of lather on which a mirror and a razor lay crossed. A yellow
dressing gown, ungirdled, was sustained gently behind him by the mild
-----Emacs:joyce          (Text Fill)--L1--All-----------------------
To:
Subject:
-text follows this line-

-----Emacs:*mail*         (Mail Fill)--L1--All-----------------------
```

现在，在另一个窗口里开始输入一条新消息。

写好邮件内容后，按“**C-c C-c**”组合键把它发送出去（并退出电子邮件功能）。Emacs 会像前面介绍的那样把邮件从屏幕上删除，但它会把那个窗口留在屏幕上。如果想删掉那个此时已属多余的窗口，请按下“**C-x 0**”组合键。如果又不想发出那封邮件了，按“**C-x k**”组合键将关闭那个编辑缓冲区，同时也删掉了那封邮件。

从另一个窗格里发送电子邮件的命令对X用户来说也大致相同。如果想在另外一个窗格里书写邮件内容，请按下“**C-x 5 m**”组合键。发送邮件仍要像其他情况那样使用“**C-c C-c**”组合键。如果想在送出邮件之后删除那个窗格，请按下“**C-x 5 0**”组合键。

对邮件内容进行拼写检查

电子邮件是一种非常方便的交流手段。为了让工作尽快地完成，人们在书写邮件时可能会不那么正规。因此，对邮件进行拼写检查就成为一种值得培养的好习惯。如果使用的是 Ispell，请输入“**ESC x ispell-message**”命令或者从“**Spell**（拼写）”菜单里选择执行“**Check Message**（检查邮件）”操作。如果现在用的不是 Ispell，那么可以试试这种方法。虽然 UNIX 的拼写检查器 spell 也能对邮件进行拼写检查，但从使用效果上讲还是 Ispell 更好用一些。**ispell-message** 命令很聪明，它会跳过在

邮件里引述的他人邮件内容和电子邮件地址，只对书写内容进行拼写检查。与拼写检查器有关的更多内容请参考第三章中的讨论。

把邮件抄送给其他收信人

如果还想把邮件的副本发送给没有被列在“TO:”（收信人）名单里的其他人，就需要在信头部分加上一个“CC:（抄送）”行；但正常的邮件编辑模板里是不带这个“CC:”行的。因此，需要通过按下“**C-c C-f C-c**”组合键让Emacs创建一个“CC:”栏、或者从“**Headers**（信头）”菜单里选择执行“**CC**”操作、或者自己动手输入“CC:”行。自己动手输入“CC:”行的方法是：在“TO:”栏里把收信人的名字都输好以后，按回车键创建一个空白行。然后在这个空白行上先按下“CC:”组合键，再把想让其他也收到这封邮件的人的电子邮件地址输进去——别忘了要在两个地址之间加上一个逗号。如果因地址太多而在一行里放不下，请按回车键，再先留出一个空格，然后继续输入地址。

把邮件密抄给其他收信人

如果想把邮件发送给某些人，却又不想让这些人的名字出现在信头里（即想发一封“密抄”邮件），请在信头里增加一个“BCC:（密抄）”行，然后把密抄邮件的收信人地址写到这一行上。也可以按下“**C-c C-f C-b**”组合键或者从“**Headers**”菜单里选择执行“**Bcc**（密抄）”操作，Emacs会在信头里增加一个“BCC:”栏。

使用密抄的原因有很多（有的好，有的坏），什么场合该用、什么场合不该用完全由自己来决定。不过，密抄至少能够很有效地解决一个传统邮件上存在的问题——发出去的邮件没有存底。一般说来，人们很容易忘记给自己留一份外发信件的存底；这样日积月累下来，当真的需要引用以前发出的某封信件时，手里却根本没有它。把Emacs设置为自动给本人密抄一份邮件就可以解决这个问题。

如果想利用这种密抄功能给所有的邮件都留一份存底，那么请把**mail-self-blind**变量设置为“**t**”。这样，Emacs就会在每一封邮件的信头部分自动加上一个“BCC:”栏，而且里面已经有你自己的用户名了。此后，如果不想收到某特定邮件的副本，可以手动删除该邮件的“BCC:”栏。如果想永久性地把**mail-self-blind**变量设置为“**t**”，请把下面这条语句添加到“*.emacs*”文件里：

```
(setq mail-self-blind t)
```

然后，保存“*.emacs*”文件再重新进入 Emacs，好让这条语句生效。

如果做了这样的设置，以后只要一发送邮件，Emacs 就会自动把下面这一行：

```
BCC: yourname
```

添加到发出邮件的信头部分，其中的 *yourname* 是你的用户名。

把邮件发送到一个文件

利用“BCC：”密抄功能，能自动收到一份自己邮件的副本。还有一种方法是让Emacs自动地把一份邮件副本发送到某个文件——只要在信头里输入“**FCC:**”，再紧跟着输入一个文件名(当然还要有这个文件的完整路径)，邮件就被追加到这个指定的文件里。如果使用了这项功能，则既能给自己发出的每封邮件保留一份存底，又不会因给自己发送的邮件而把邮箱弄得乱七八糟。

类似于密抄功能的设置情况，也可以让Emacs把邮件的副本自动发送到指定的文件里——只要设置好变量**mail-archive-file-name**即可。比如，如果想把自己所有外发邮件的副本都保存在主目录里一个名为“*mycopies*”的文件里，就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq mail-archive-file-name 'mycopies')
```

然后，退出 Emacs 再重新进入。下次发送电子邮件的时候，就会看到：

```
FCC: mycopies
```

出现在邮件的信头。如果发送了很多的邮件，那么这个文件将会变得非常大。如果磁盘空间比较紧缺，就需要定期清理自己的外发邮件，好让这个文件不至于增长得太大。

在邮件里插入一个文件

Emacs 不支持把文件用做电子邮件附件的做法，但允许把ASCII 文本文件插入到邮件信体里。具体操作步骤是：先按下“**C-x m**”组合键并填写好信头；在输入信体

的时候按下“**C-x i**”组合键，然后根据提示给出一个文件名。Emacs将把这个文件插入到信体里。可以对它进行必要的编辑，最后用“**C-c C-c**”组合键把它发送出去。

给邮件加上签名

如果想把一个标准的签名放到邮件消息里，那么先要把它保存到主目录里一个名为“*.signature*”的文件里，写完邮件内容后再按下“**C-c C-w**”组合键或者从“**Mail**”菜单里选择执行“**Insert Signature**（插入签名）”操作，就可以把这个文件插入到邮件消息里。（注意：有的系统可能已经设置成会自动给电子邮件加上一个“*.signature*”签名文件的情况，如果真是这样，当再按下“**C-c C-w**”组合键时，邮件里的签名就会变成两个。）许多人喜欢用一些有意思的图案来作为自己的签名（注1），在Emacs里绘制图形的方法请参考第八章“绘制简单的图形”一节里对图形模式（picture mode）的讨论。另外一些人喜欢使用文字形式的签名，下面就是一种签名的格式：

```
-----------------------------------------------------
Your name.                          A brief quotation.
Your work  address.
Your email   address.               Your phone number.
-----------------------------------------------------
```

签名中的“brief quotation”通常是某种形式的声明，表明作者的看法不代表其雇主的观点。这些话一般说得都很巧妙，比如“If anyone speaks for Btsfphbl Grommets, Inc., it's certainly not me.（如果有人是Btsfphbl Grommets公司的发言人，那肯定不是我）”之类。

下面的例子给出了签名的使用方法。

注1： 但我们希望大家知道，把大块的ASCII图案用做自己的签名并不是一个好习惯。

按下：**C-c C-w**

```
Buffers Files Tools Edit Search Headers Mail Help
To: david
Subject: grommets
-text follows this line-
Your ideas about grommets are completely losing.

Sincerely,

Jo
-----------------------------------------------------
Joe Btsfphbl                    "It's not my fault-
Btsfphbl Grommets, Inc.          I was born obnoxious"
email: joe@grom.com             phone: (777)666-4444
-----------------------------------------------------
--**-Emacs: *mail*        (Mail Fill)--L8--All----------------------
```

"**C-c C-w**"组合键把签名加在邮件的末尾。

当然，如果还没有创建"*.signature*"文件，那么再怎么按"**C-c C-w**"组合键也没有用。

使用假名（别名）

如果经常要给一些地址很长的人去信，那么使用假名将会更轻松些。比如，假设想给一位名叫 Fred 的朋友写封信，而他的电子邮件地址是：

```
fred@plan9fromouterspace.galaxy.universe.com
```

如果每次写信都要输入这么一大堆字符就实在是太受罪了，但用不着！邮件假名允许给常用的地址或地址组定义一个缩略语。定义缩略语的方法是输入"**ESC x define-mail-alias RETURN.**"命令。Emacs 将在辅助输入区里显示一个"Define mail alias（定义邮件假名）"提示符；输入假名的名字（比如 **fred**），然后按下回车键。Emacs 将给出下面这样的提示：

```
Define fred as mail alias for:
```

接着输入 Fred 的完整地址，如下所示：

```
fred@plan9fromouterspace.galaxy.universe.com
```

如果是为一个地址组定义邮件假名，顺序输入那些地址即可（地址之间要用空格隔开）。

用这种方法定义的假名只能用在当前的Emacs工作中，如果退出了这次Emacs，假名就不会被保留下来。把假名永久性地定义下来当然会更有用，只需用下面这样的语句把假名命令添加到“*.emacs*”文件里即可：

```
(define-mail-alias 'fred" "fred@plan9fromouterspace.galaxy.universe.com')
(define-mail-alias 'club" "kathy ray allen fred nadine esther charles")
```

从根本上讲，邮件假名只是一个很简单的概念；但在实际应用中，它们却可能会非常复杂。问题的根源在于假名可以在很多地方进行定义：在本地的“*.emacs*”文件里、在其他邮件软件的定制文件里（比如“*.mailrc*”），或者在某个系统级文件里（它通常是“*/etc/aliases*”文件，但也可能有别的名字）。

那么，Emacs会到哪里去找假名呢？Emacs会读取“*.mailrc*”文件寻找假名，但“*.emacs*”文件里的假名更优先。举个例子，如果“*.mailrc*”文件说“*francis*”是“*francis@assisi.org*”但“*.emacs*”文件却说“*francis*”是“*francis@usedcars.com*”，则Emacs将使用后一项定义。

要想了解假名的工作原理，可以先定义一两个假名，然后用它们来发送几个邮件（别忘了把自己加到“CC：”栏里），看它们生成的信头是什么样的。比如，如果准备把一组地址定义为一个假名并给它起名为“*club*”，记得要让自己是其中的一员；然后，给“*club*”发一封信，看看自己收到的邮件会是什么样子。收信人“To：”栏里根本就不会出现“*club*”这个词，出现的是包括在这个假名定义里的各个地址。假名可能会被扩展好几次：把“*nadine*”列为假名“club”的一个成员，可能会有另外一个假名把“*nadine*”转换为“*nadine_fuller*”，还可能会再有一个假名把“*nadine_fuller*”转换为“*nxf@orbis.com*”。

回信地址（Reply-to）栏的用法

邮件消息可以用“Reply-to:（回信地址）”栏来保证回信能够被发送到正确的地址。一般用不着关心这一栏，邮件软件基本上都能正确地生成一个供回信使用的地址。但在某些情况下，明确地设定“Reply-to:”栏是很有必要的。比如，如果系统的邮件配置有毛病，别人的回信老是莫名其妙地收不着。这种问题当然得纠正，但与邮件系统打交道并不是件容易的事情，而且这很可能根本就不在自己的管辖范围内。该怎么办呢？如果知道自己用来接收他人邮件的地址永远是“*george@*

futons.com"，就完全可以自己动手解决这个问题——把这个地址填写在外发邮件的"Reply-to:"栏上。

再比如，如果遇到需要临时使用别人的账户（或者别人的计算机）来发送电子邮件，但想让回信还是回到你那里去的情况，"Reply-to:"栏也会很有用。举个例子，假设你的计算机坏了，现在用的是一个朋友的计算机。但当你从朋友的机器上发电子邮件时，你多半还是想让回信回到自己的系统——因为在两个不同的系统上接收邮件很不方便，也许因为你估计自己的计算机不至于宕那么长的时间。

要想插入"Reply-to:"栏，可以按下"**C-c C-f C-r**"组合键或者从"**Headers**"菜单里选择执行"**Reply-To**（回信地址）"操作。如果需要经常使用这个功能，可以用设置变量**mail-default-reply-to**的方法把"Reply-to:"和回信地址自动加到外发邮件的信头里。比如，如果想把自己外发邮件里的"Reply-to:"栏全部设置为"*george@futons.com*"，就需要把下面这条语句添加到"*.emacs*"文件里：

```
(setq mail-default-reply-to "george@futons.com")
```

如果只想给某一个邮件单独加上"Reply-to:"栏，可以使用**set-variable**命令进行设置：输入"**ESC x set-variable RETURN mail-default-reply-to RETURN george@futons.com RETURN**"命令即可。

邮件发送命令汇总

表6-1列出了可以在Emacs里使用的各种邮件发送命令，"**Headers**"和"**Mail**"菜单里的有关操作选项也包括在其中。这个表里的某些命令没有在以上各小节里介绍，希望大家认真地看一看。

表6-1：邮件发送命令速查表

键盘操作	命令名称	动作
C-x m	**mail**	打开"*mail*"编辑缓冲区，以邮件模板为基础填写有关资料和邮件内容
C-x 4 m	**mail-other-window**	在一个新窗口里打开"*mail*"编辑缓冲区

表6-1：邮件发送命令速查表（续）

键盘操作	命令名称	动作
C-x 5 m	**mail-other-frame**	在一个新窗格里打开“*mail*”编辑缓冲区
C-c C-f C-t *Headers → To*	**mail-to**	移动到“To:”栏
C-c C-f C-c *Headers → Cc*	**mail-cc**	移动到“CC:”栏（如果没有就创建之）
C-c C-f C-b *Headers → Bcc*	**mail-bcc**	移动到“BCC:”栏（如果没有就创建之）
C-c C-f C-f *Headers → Fcc*	**mail-fcc**	提示输入一个文件名，然后向这个文件发送一份邮件的副本
C-c C-f C-r *Headers → Reply-To*	**mail-reply-to**	指定一个地址，而这封邮件的回信都将被发到这个地址上去
C-c C-f C-s *Headers → Subject*	**mail-subject**	移动到信头中的主题栏
C-c C-t *Headers → Text*	**mail-text**	移动到可以开始输入信体的地方去
C-c C-w *Mail → Insert Signature*	**mail-signature**	插入“*.signature*”文件的内容
C-c C-c *Mail → Send Message*	**mail-send-and-exit**	发送邮件并退出“*mail*”编辑缓冲区
C-c C-s *Mail→Send, Keep Editing*	**mail-send**	发送邮件，但不退出“*mail*”编辑缓冲区
（无）	**define-mail-alias**	为某个名字或某个邮件表定义一个缩写的假名
（无） *Mail → Cancel*	**mail-dont-send**	取消正在书写的邮件消息

表6-2列出了可以加到邮件信头里去的各个信息栏。在默认的情况下，“*mail*”编辑缓冲区里只显示“To:(收信人)”和“Subject:(主题)”两栏。

表 6-2: 邮件信头中的信息栏

信头中的信息栏	作用
To:	收信人: 此邮件将被发送给这些人
CC:	抄送: 这些人将收到此邮件的一个副本
FCC:	文件抄送: 此邮件的一个副本将被追加到这个文件的末尾
BCC:	密抄: 这些人将收到此邮件的一个副本，但他们的名字不会出现在此邮件的信头里
Subject:	主题: 此邮件信息的主题
From:	发信人: 发出这封邮件的人（如果与自己的用户名不一样）: 这一栏只有在用别人的账户发邮件消息时才需要填写
Reply-to:	回信地址: 此邮件的回信应该被发送到的地址

用 Emacs 读取邮件

如果不能接收电子邮件，那么发出再多的电子邮件也没有什么用处。Emacs 有一个内置的编辑模式可以用来读取接收到的电子邮件。在Emacs里读取电子邮件的方法是: 输入“**ESC x rmail RETURN**”或者从“**Tools**”菜单中选择执行“**Read Mail**（读取电子邮件）”操作。Emacs 将检查收信箱（inboxes）和用来保存旧邮件的文件（默认文件名是“*~/RMAIL*”文件）（注 2）。如果想用其他文件而不是“*~/RMAIL*”文件保存邮件消息，请把变量**rmail-file-name**设置为该文件的名字。比如，如果要把邮件消息都保存在“*~/mail/inbox.rmail*”文件里，就需要把下面这条语句添加到“*.emacs*”文件里:

```
(setq rmail-file-name "~/mail/inbox.rmail")
```

顺便告诉大家，*RMAIL* 文件的备份保存在“*~/RMAIL~*”文件里。

注 2: 收信箱（inbox）是系统用来保存新接收到的电子邮件的地方，一般用不着去操心它。Emacs 会去“*/var /mail/username*”、“*/usr/spool/mail/username*”、“*/usr/mail/username*”或者“*/var/spool/mail/user*”等地方检查有没有新邮件。如果这些文件都不存在，就说明邮件系统的设置情况与标准的不一样——问问系统管理员自己的收件箱到底在什么地方。一旦找到了它，请用变量**rmail-primary-inbox-list**为外来邮件指定一个完整的路径名。

如果没有邮件，那么RMAIL会显示一条相应的信息告知。如果确实有邮件，在读完收件箱之后，Emacs会把邮件显示在一个如图6-1所示的画面里。

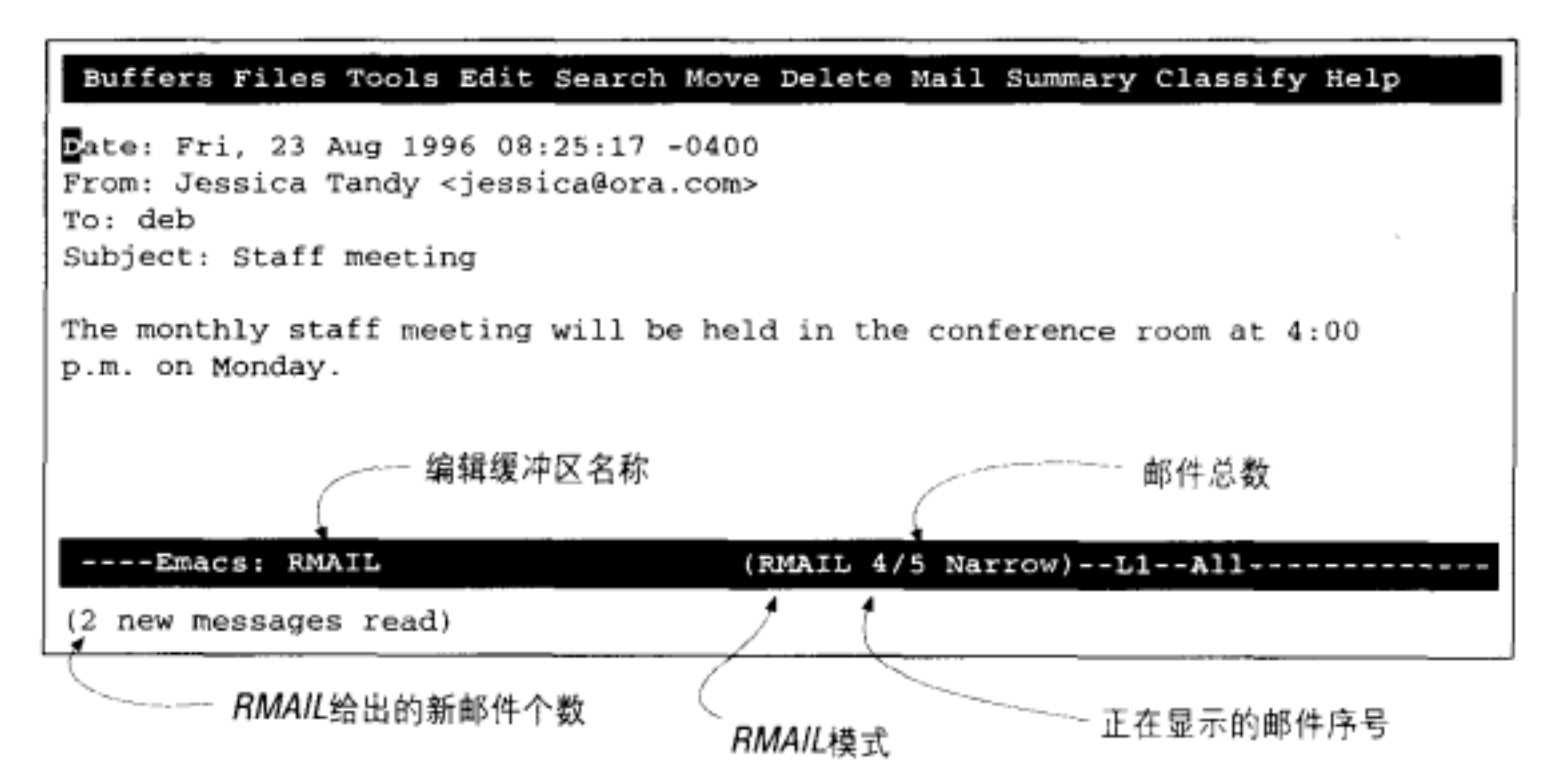


图6-1：用RMAIL读过邮件之后

从状态行上可以看出：当前编辑缓冲区叫做“RMAIL”，并且处于RMAIL模式中。这是一个专门用来读取邮件内容的编辑模式。上图中的“4/5”表示正在读取总共5条消息里的第4条。这个数字会根据当时有多少尚未读过的邮件而变化。最底下的那一行表示：自从上次读过邮件之后，又新收到2条消息，而其中的一条正显示在屏幕上。

如果没有新邮件，那么Emacs会把最近收到的那条消息显示在屏幕上，同时会在辅助输入区里给出一条信息“(No new mail has arrived)(没有新邮件)”。

如果收到大量的邮件，那么，当进入RMAIL时，就会有10条、20条或者更多的新邮件在等着去读取。在这种情况下，可能需要先查看一下对各条消息进行了简单汇总的邮件清单。这方面的详细讨论请参考本章后面的“与邮件清单有关的操作”一节。

“RMAIL”编辑缓冲区除了是只读的以外，与其他Emacs编辑缓冲区没有什么区别。可以用“**C-x b** *buffername*”命令切换到其他的编辑缓冲区，或者按下“**b**”键把

"RMAIL"编辑缓冲区暂时"埋"在其他编辑缓冲区的下面。退出RMAIL的方法有两种：按下"**q**"键或者像平时那样退出Emacs（按"**C-x C-c**"组合键）。

在 RMAIL 里移动

大部分标准的Emacs移动命令都可以用在RMAIL模式里，可如果需要频繁地使用RMAIL，那么还是掌握一些用来完成面向邮件处理工作的单字符命令比较好。比如，完全能够像在其他Emacs文件里那样用"**C-v**"组合键来卷动查看一封邮件。但在RMAIL里，只按空格键也能达到同样的效果。

在默认情况下，RMAIL会把所有的邮件都保存在一个文件里，其路径是"*~/RMAIL*"。Emacs会把这个文件里的每条消息看做是彼此独立的事物，并且会在状态行上显示一个术语"Narrow"来表明这一点。即使所有的邮件都在该文件里，Emacs也会把屏显画面收缩到一次只显示一条消息的程度。也就是说，即使消息的长度只有一行，它也会显示在一个独立的画面里，而信头部分将永远出现在画面的最顶部。这种设置会让我们把注意力依次集中在各条消息上。

表6-3列出了用来在RMAIL里四处移动的单字符命令，以及这些命令的用处，"**Move**（移动）"菜单里的有关操作选项也包括在其中。

表6-3：RMAIL命令速查表

键盘操作	命令名称	动作
SPACE	**scroll-up**	卷屏，查看此消息的下一个画面
DEL	**scroll-down**	卷屏，查看此消息的上一个画面
.	**rmail-beginning-of-message**	移动到此消息的开头
n *Move→Next*	**rmail-next-undeleted-message**	移动到下一条消息
p *Move→Previous*	**rmail-previous-undeleted-message**	移动到上一条消息
< *Move→First*	**rmail-first-message**	移动到第一条消息

表 6-3: RMAIL 命令速查表（续）

键盘操作	命令名称	动作
> *Move→Last*	**rmail-last-message**	移动到最后一条消息
j	**rmail-show-message**	如果这个命令的前面有一个数字“*n*”，跳到第 *n* 条消息。

获取新邮件

在 RMAIL 里，可以在任何时候按下“**g**”键或者从“**Mail**”菜单里选择执行“**Get New Mail**（获取新邮件）”操作以检查自己有没有新邮件。Emacs将检查收信箱，如果有新邮件，它就会在 *RMAIL* 文件存盘之后把它显示出来。

删除邮件

有用来在RMAIL消息之间移动的单字符命令，同样也有用来删除邮件的简单命令。

类似于删除编辑缓冲区或书签的情况（请参考前几章中的讨论），删除邮件也要分两步走。首先，得用“**d**”或者“**C-d**”给邮件加上待删除标记（“**d**”将移动到下一条消息，而“**C-d**”将移动到前一条消息），这一步也可以通过从“**Delete**（删除）”菜单里选择执行“**Delete**”操作来完成。这并不表示邮件消息已经无影无踪了——还来得及改变主意，用“**u**”命令去掉邮件消息上的待删除标记（按下“**u**”键将回到上一条加有带删除标记的消息位置上，从“**Delete**”菜单里选择执行“**Undelete**（撤销删除）”操作也能到达同样的目的）。当确实想把已经加上带删除标记的邮件都删掉的时候，再只用一个操作就可以让Emacs把所有已经加上带删除标记的邮件都删掉。可以采取以下几种方法之一：

- 输入“**x**”命令。
- 从“**Delete**”菜单里选择执行“**Expunge**（整理）”或“**Expunge/Save**（整理/保存）”操作。
- 按下“**s**”键，保存 *RMAIL* 文件。
- 按下“**q**”键，退出 RMAIL。

一旦执行了上述几种操作中的任何一个，那些邮件就都将被删除，再也无法用撤销删除命令来恢复了。

表6-4对用来进行删除邮件操作的命令进行了汇总，“**Delete**”菜单里的有关操作选项也包括在其中。

表 6-4：邮件删除命令速查表

键盘操作	命令名称	动作
d *Delete→Delete*	**rmail-delete-forward**	给邮件加上待删除标记，然后移动到下一个
C-d	**rmail-delete-backward**	给邮件加上待删除标记，然后移动到上一个
ESC n *Move→Next*	**rmail-next-message**	移动到下一条消息；不管它是否已经加上待删除标记
ESC p *Move→Previous*	**rmail-previous-message**	移动到上一条消息；不管它是否已经加上待删除标记
u *Delete→Undelete*	**rmail-undelete-previous-message**	去掉邮件消息上的待删除标记
x *Delete→Expunge*	**rmail-expunge**	删除已经加有待删除标记的全部消息
s *Delete→Expunge/Save*	**rmail-expunge-and-save**	删除加有待删除标记的消息并保存 *RMAIL* 文件

回复别人的邮件

电子邮件通常被看做是电话、书信或者会议的一种替代品。它并不仅仅是一种发送给他人的备忘录，它也是一种双向的交流手段，可以有回信，甚至可以有回信的回信。知道如何对收到的邮件进行回复是很重要的。

如果想对正显示在屏幕上的消息进行回复，请按下“**r**”键或者从“**Mail**”菜单里选择执行“**Reply**（回信）”操作。Emacs 将打开一个“*mail*”窗口，并且会把发出回信所需要的信头信息都填写好，可以根据需要对之进行编辑以增减收信人。

在默认情况下，RMAIL 不会在回信中包括原来的信息。如果想这样做，那也很容易，只要按下“**C-c C-y**”组合键或者从“**Mail**”菜单里选择执行“**Cite Original**（引用原文）”操作即可。Emacs 会把正准备回信的那封邮件的内容，以段落缩进的格式插入到回信里。（段落缩进可以让人们分清哪些东西是原来的邮件内容，哪些东西是回复。）这种格式可能会使某些行超出画面，所以可能需要用“**C-c C-q**”组合键或者从“**Mail**”菜单里选择执行“**Fill Citation**（重排原文）”操作对引用的原文进行段落排版。如果愿意，也可以用“**C-u C-c C-y**”组合键以没有段落缩进的格式插入原邮件内容。

假设刚刚接收到一封关于召开员工会议的邮件，并准备发出一封回信。在读完会议通知邮件之后按下“**r**”键，Emacs 将打开一个如图 6-2 所示的窗口以便输入回信。

原来的邮件

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help

Date: Fri, 23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m. on Monday.

----Emacs: RMAIL                    (RMAIL 4/5 Narrow)--L1--All-----------

To:  jessica@ora.com
In-reply-to:  <jessica@ora.com> Jessica Tandy
Subject:  Re: Staff meeting
--text follows this line--
 
----Emacs: *mail*              (Mail Fill)--L5--All--------------------------
```

回信

图 6-2：打开一个回信窗口

按下“**C-c C-y**”组合键插入邮件原文，如图 6-3 所示。

插入邮件原文的时候，Emacs 会删除“RMAIL”窗口，没有必要同时在屏幕上显示同一个邮件的两份副本。

```
Buffers Files Tools Edit Search Headers Mail Help

To:  jessica@ora.com
In-reply-to:  <jessica@ora.com> Jessica Tandy
Subject:  Re: Staff meeting
--text follows this line--
   Date: Fri, 23 Aug 1996 08:25:17 -0400
   From: Jessica Tandy <jessica@ora.com>
   To: deb
   Subject: Staff meeting

   The monthly staff meeting will be held in the conference room at 4:00
   p.m. on Monday.

-----Emacs: *mail*          (Mail Fill)--L13--All-----------------------
```

按下*C-c C-y*组合键插入原邮件内容

图 6-3: 原邮件的内容被插入到回信里

在插入了原邮件内容之后，可以输入回信内容，编辑原来的邮件内容，或者做任何书写新邮件所需要的事情。去掉原邮件里一些不必要的内容以便回信不至于太长，是一种很好的做法。比如，如果邮件的始发者列出了 5 点意见，而你只想回复其中的 2 点，就可以把不想回复的那 3 点删掉。如果是在回复一封回信的回信（可以如此反复多次），太多层次的缩进段落反而有可能会把对方弄糊涂。只要在回信里能够把谁说过什么讲明白就足够了。引用原文的时候，可以让 Emacs 用一些更直观的东西，而不仅仅是空格来缩进正准备回复的消息。如果设置了变量**mail-yank-prefix**，就可以让 Emacs 插入一些其他的东西来代替缩进。如果想试试这个方法，请输入“**ESC x set-variable RETURN mail-yank-prefix RETURN**”，然后输入准备用在原文引述中的缩进字符串，比如一个大于号加一个空格之类的东西。此后，当再在回信里引用原文的时候，每一行原文的前面就都会加上“> ”字样，从而把它们与回信的内容区别开。如果想让 **mail-yank-prefix** 在每次 Emacs 工作中都生效，就需要把下面这条语句添加到“*.emacs*”文件里，在引号里可以放入任何想使用的字符串：

```
(setq mail-yank-prefix "> ")
```

保存文件再重新进入 Emacs 以使这条语句生效。

写好回信之后，按下“**C-c C-c**”组合键，就可以把这封邮件发送出去。Emacs 将

回到原先的“RMAIL”编辑缓冲区，刚才回复的邮件仍显示在屏幕上；但此时的状态行上会出现“answered（已回复）”的字样，表示已经发出过一封回信。

邮件的转发

有时候，会想把自己收到的邮件再转发给另外一个人。转发邮件的方法是：按下“**f**”键或者从“**Mail**”菜单里选择执行“**Forward**（转发）”操作。Emacs会打开一个邮件编辑缓冲区，其信体是当前邮件的内容，而光标则位于“To:”栏处——直接填上准备把这份副本转发给谁即可。可以在消息里任意加上自己的评论（比如转发原因等等），然后按下“**C-c C-c**”组合键把这封邮件发送出去。Emacs将回到原先的“RMAIL”编辑缓冲区，并且会在这条消息的状态行上显示“forwarded（已转发）”字样。

把邮件保存到一个文件里

阅读邮件时，也许会想把某些消息保存起来以供今后参考。这件事在RMAIL里是很容易做到的，但在此之前，还得先做一个决定：是把这封邮件保存为Emacs的RMAIL格式，还是把它保存为ASCII格式。（ASCII文件就是普通的文本文件，在Emacs里编辑的大部分文件都是这种格式。）

如果只使用Emacs阅读邮件，那么RMAIL格式就是最合适的。它就是为让用RMAIL来阅读保存的邮件而设计的。比如，可以先把一封邮件保存到一个文件里，然后以后再对它进行回复。RMAIL格式能够在今后再使用RMAIL对邮件进行处理。再看ASCII格式，它更适用于那些需要把邮件转移到其他邮件系统，或者其他计算机上去的人们。其他的邮件系统也许消化不了RMAIL格式，但读取ASCII格式应该不会有问题。把邮件保存为RMAIL格式的方法是按下“**o**”键或者从“**Classify**（分类）”菜单里选择执行“**Output (RMAIL)**(输出为RMAIL格式)”操作。Emacs将提示给出一个文件名；输入文件名，再按下回车键。如果省略了文件名，Emacs将把邮件保存到主目录里一个名为“*XMAIL*”的文件里。

PDG

按下：**o**

```
Buffers Files Tools Edit Search Minibuf Help
Date:Fri, 23 Aug 1996 08:25:17 -0400
From:Jessica Tandy <jessica@ora.com>
To:deb
Subject:Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
---Emacs: RMAIL          (RMAIL 4/5 Narrow)--L1--All----------------
Output message to Rmail file:(default XMAIL)~/
```

准备输入输出文件的名字。

把邮件保存为ASCII格式的方法是按下“**C-o**”键或者从“**Classify**”菜单里选择执行“**Output (inbox)**(输出收信箱)”操作。Emacs将提示给出一个文件名：输入文件名，再按下回车键。如果省略了文件名，Emacs将把ASCII格式的邮件保存到主目录里一个名为“*xmail*”的文件里。

如果是把邮件输出到一个现有的文件里，那么Emacs会把这条消息追加到那个文件的末尾，不会覆盖该文件的当前内容。如果因为某些原因使（比如磁盘已满）Emacs无法保存这封邮件，它就会发出告知。

保存到文件里的任何消息都会在RMAIL里留一份副本。如果想让已经被保存到文件里的消息在退出RMAIL的时候自动删除，那么请把变量**rmail-delete-after-output**的值设置为“**t**”。

许多UNIX用户会用一个特殊的子目录（通常叫做“*Mail*”）来保存邮件消息。根据个人组织目录的习惯，有人可能会觉得这种做法很方便，也有人可能会觉得把邮件与其他文件混在一起更适应。

如果因为不小心按错了“**o**”和“**C-o**”，而把RMAIL格式和ASCII格式弄颠倒了，那么会出什么事？不会出什么事。Emacs会先检查文件的格式，然后再把新邮件按与文件原有内容相同的格式追加到文件的末尾。（我们说过Emacs很了不起。）

此外，如果改了主意，事后又想用另外一种格式保存邮件，那么可以再把RMAIL格式的文件转换为ACSII，或者再把ASCII格式的文件转换为RMAIL格式。输入“**ESC x unrmail**”命令就能给*RMAIL*文件另外创建一个ASCII版本（要为ASCII文件新输入一个文件名）；而用RMAIL读入一个ASCII文件（这个操作马上就要讲

到）就能把它转换为RMAIL格式。（注意：如果已经对ASCII格式的邮件文件进行过编辑，那么RMAIL能不能再次读出它来可就不能保证了。）

读取邮件文件

如果是用“**o**”命令把邮件保存为RMAIL格式的，那么启动RMAIL，按下“**i**”键，Emacs将在辅助输入区里提示：

```
Run rmail on RMAIL file:
```

输入想读入的文件名字。在读完这个文件之后，Emacs会把最后一条消息显示在屏幕上。接下来，可以用“**n**”和“**p**”来前后移动、用“**r**”来回复旧邮件、用“**f**”来转发邮件等等。

要想查看RMAIL文件的内容是很容易的。如果用“**C-x C-f**”组合键找到的是一个RMAIL格式的文件，Emacs会识别出它来并自动进入RMAIL模式里。

阅读以ASCII格式保存的邮件文件

如果想查看ASCII格式的邮件内容并让它保留ASCII格式，那么可以像对待任何其他文件那样用“**C-x C-f**”（命令名是**find-file**）组合键打开它。如果用RMAIL读入了一个ASCII格式的文件，Emacs会把它转换为RMAIL格式。如果想让它回到ASCII版本，可以使用刚才介绍的**unrmail**命令。

表6-5对文件邮件的操作命令进行了汇总，“**Classify**”菜单里的有关操作选项也包括在其中。

表6-5：邮件文件操作命令速查表

键盘操作	命令名称	动作
o *filename* **RETURN** *Classify→Output (Rmail)*	**rmail-output-to-rmail-file**	把邮件消息保存为RMAIL文件格式
C-o *filename* **RETURN** *Classify → Output (inbox)*	**rmail-output**	把邮件消息保存为UNIX邮件文件格式（一个标准的ASCII文本文件）

表 6-5: 邮件文件操作命令速查表（续）

键盘操作	命令名称	动作
i *filename* **RETURN**	**rmail-input**	从文件里读出邮件消息并把该文件转换为RMAIL格式
（无）	**unrmail**	创建RMAIL 文件的ASCII版本

RMAIL 的查找功能

在RMAIL 里，每个邮件都单独放在它自己的编辑缓冲区里，就好像它们是彼此分开的文件一样。在阅读邮件时，可以像平常一样使用查找命令。递增查找命令“**C-s**”也能正常工作，但它只能在当前邮件消息中进行查找操作，多一点也不行。

RMAIL组件提供了一个特殊的查找命令，以便在整个邮件文件的范围内进行查找。这种扩展性的查找往往是很有用的。举个例子，假设邮件文件里有120条消息（如果真有这么大的文件，那就应该考虑把它分割为几个小一点的文件，不过这是另外一个话题了），现在想把与“grommets”有关的消息找出来，但却想不起它是哪一条了。查看了邮件清单（我们马上就要讲到这个操作），可它并没有什么帮助——没有一条消息在它的主题栏里提到“grommets”。惟一的解决方案就剩下查找整个文件了。

于是，按下“**ESC s**”组合键或者从“**Move**（移动）”菜单里选择执行“**Search**（查找）”操作。Emacs将提示输入一个正则表达式。第三章对正则表达式做过一个简要的介绍，不过即使你跳过了它或者忘了它，都没有多大关系。就这个例子而言，输入一个正常的单词就够用了。输入查找字符串，Emacs就会把包含有这个单词的邮件消息找出来。如下所示:

按下：**ESC s**

```
Buffers Files Tools Edit Search Minibuf Help
Date:Fri, 23 Aug 1996 08:25:17 -0400
From:Jessica Tandy <jessica@ora.com>
To:deb
Subject:Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
 ---Emacs: RMAIL                    (RMAIL 4/5 Narrow)--L1--All-----------
Rmail search (regexp):
```

准备输入查找字符串。

输入：**grommets RETURN**

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help
Date: Fri, 23 Aug 1996 08:29:37 -0400
From: George Jetson <george@grom.com>
To: deb
Subject: your recent order

We received your order for grommets. When shall we ship them?
 ---Emacs: RMAIL                    (RMAIL 5/5 Narrow)--L1--All-----------
Rmail search for grommets...done
```

找到内容里有单词“grommets”的邮件。

如果再次按下“**ESC s**”组合键，Emacs 会记得刚才的查找字符串，因而给出下面这样的提示：

```
Rmail search (regexp): (default grommets)
```

如果还想对字符串“grommets”进行查找，请直接按下回车键。Emacs 会在剩下的邮件里继续查找这个单词。如果想对其他文字进行查找，输入新字符串，再按下回车键。

在开始进行查找操作之前，最好先用“**j**”命令跳转到第一条消息，以确保能够这次查找操作能够从头至尾地搜索整个邮件文件。但因为 RMAIL 里提供有逆向搜索的命令，所以这倒不是绝对必要的。如果想从当前消息开始逆向查找到文件的开始，请按下“**C-u – ESC s**”组合键或者从“**Move**”菜单里选择执行“**Search Back**（逆向查找）”操作。

这种查找功能有一个很大的缺点，即正向查找和逆向查找都不会对当前消息进行查找。这往往会造成一些困惑：明明看见想要找的字符串显示在屏幕上，可“**ESC s**”

命令就是找不到它。这可不是你的过错；“**ESC s**”总是从当前消息的结尾开始向文件末尾方向进行查找；而“**C-u - ESC s**”却又总是从当前消息的开头开始向文件头方向进行查找。不过有办法解决这个问题——递增查找（“**C-s**”）能够对当前消息进行查找，这样就不会造成遗漏了。

在第三章和第十三章里，你可以找到更多关于递增查找和正则表达式查找的内容。

与邮件清单有关的操作

与许多其他的邮件系统一样，RMAIL提供了对邮件消息清单进行操作的功能。查看邮件清单的操作方法是：利用“**h**”命令或者从“**Summary**（邮件清单）”菜单里选择执行“**All**（查看全部）”操作。Emacs将打开一个名为“RMAIL-summary”的窗口，如图6-4所示。

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help

Date: Fri, 23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m. on Friday.

----Emacs: RMAIL                    (RMAIL 4/5 Narrow)--L1--All-----------

  1  23-Aug              land@ora.com  Photo wanted
  2  23-Aug              mark@ora.com  PO number
  3  23-Aug               sue@ora.com  Corporate policy
  4  23-Aug           jessica@ora.com  Staff meeting
  5  23-Aug           george@grom.com  your recent order

-%*-Emacs: RMAIL-summary          (RMAIL Summary)--L4--All------------------
Computing summary lines...done
```

*Emacs*给出的邮件消息汇总清单

图 6-4：RMAIL 的邮件清单画面

从这份清单开始，只需利用几个单字符命令就能完成对邮件消息进行查看、删除和撤销删除，以及在邮件之间移动等操作。（如果你没有跳过第四章里对编辑缓冲区清单上的操作命令的讨论，对这些命令就不应该感到陌生。）比如，如果输入用来移动到下一条消息的“**n**”命令，Emacs就会把光标移动到邮件清单的下一行，并且会

把选中的消息显示在“RMAIL”窗口里。类似地，“**p**”将使光标移动到邮件清单的上一行，并把选中的消息自动显示出来。如果想同时退出邮件清单和RMAIL功能，请按下“**q**”键。

最新版本的Emacs在这些现有单字符命令的基础上又前进了一步，使光标移动命令基本上都能用在邮件清单上。更方便的是，递增查找命令“**C-s**”也能用在邮件清单里。这样，如果需要在一个很长的邮件清单里进行查找，那么用标准的命令就能对各种地址名、人名和邮件主题进行递增查找了。

按下“**d**”键给消息加上待删除标记，邮件清单里该条消息的前面会出现一个大写字母“**D**”，表示已经给这条消息加上了待删除标记。但这条消息并不会立刻被删除，还有机会用“**u**”来撤销它身上的待删除标记。Emacs将在按下“**x**”键时才会真正删除那些加有待删除标记的消息。按“**q**”键退出RMAIL时也会删除已加上标记的邮件。当按下“**q**”键时，Emacs将退出邮件清单模式和RMAIL；而RMAIL会在退出时把加有待删除标记的邮件都删掉。这种先给邮件加上待删除标记然后再进行删除的做法，与在Dired中从编辑缓冲区清单里删除编辑缓冲区，或者从书签表里删除书签的情况是相同的，这种不同功能操作之间的一致性使Emacs更容易学习和掌握。

在邮件清单和邮件内容之间跳来跳去可不方便，因此，Emacs允许在不离开邮件清单的情况下查看邮件的内容。如果在邮件清单里按一下“SPACE”键，Emacs就会把显示在“RMAIL”窗口里的邮件内容卷到下一页；如果按下的是“**DEL**”键，Emacs就会把邮件内容卷到上一页。记住：这些命令对邮件清单窗口本身不会有任何影响。在邮件消息之间移动要使用“**n**”和“**p**”键，在邮件消息的内容里前后移动要使用“**SPACE**”键和“**DEL**”键。总之，邮件清单模式有阅读邮件所需要的全部东西。

表6-6列出了能够在邮件清单窗口里使用的命令。

表6-6：邮件清单操作命令速查表

键盘操作	命令名称	动作
SPACE	**rmail-summary-scroll-msg-up**	向前卷动RMAIL窗口里的邮件消息
DEL	**rmail-summary-scroll-msg-down**	向后卷动RMAIL窗口里的邮件消息

表 6-6: 邮件清单操作命令速查表（续）

键盘操作	命令名称	动作
d *Delete→Delete*	**rmail-summary-delete-forward**	给消息加上待删除标记（在消息序号前出现字母“D”标记）
e	**rmail-summary-edit-current-message**	编辑当前消息（完成时要按下“**C-c C-c**”组合键）
u *Delete→Undelete*	**rmail-summary-undelete**	去掉当前消息上的待删除标记
n	**rmail-summary-next-msg**	移动到下一条消息并把它显示在 RMAIL 窗口里
p	**rmail-summary-previous-msg**	移动到上一条消息并把它显示在 RMAIL 窗口里
x *Delete→Expunge*	**rmail-summary-expunge**	删除所有加有待删除标记的消息
q	**rmail-summary-quit**	退出 RMAIL
w	**rmail-summary-wipe**	删除 RMAIL 邮件清单窗口

邮件的整理归类

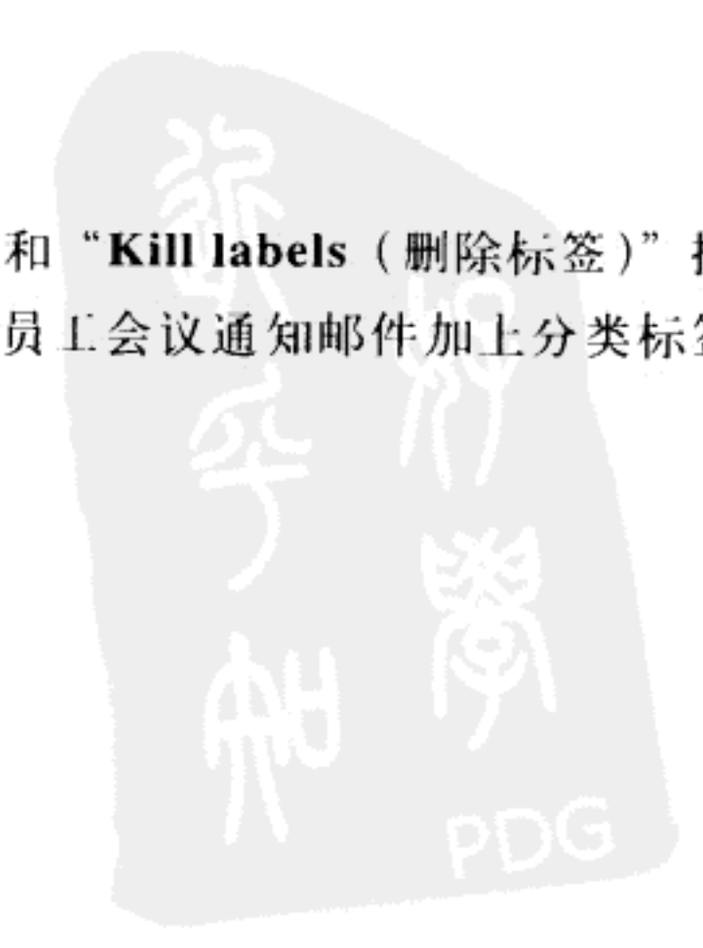
如果邮件很多，那么就可以利用分类标签（label）对它们进行整理和归类。要想给一封邮件加上分类标签，需要输入:

```
a label RETURN
```

要想删除一个分类标签，请输入:

```
k label RETURN
```

“**Classify**”菜单里的“**Add labels**（添加标签）”和“**Kill labels**（删除标签）”操作也能完成同样的工作。我们来看一个例子: 给员工会议通知邮件加上分类标签“meetings”。如下所示:



按下：**a**

```
Buffers Files Tools Edit Search Minibuf Help
Date:Fri,23 Aug 1996 08:25:17 -0400
From:Jessica Tandy <jessica@ora.com>
To:deb
Subject:Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
---Emacs: RMAIL                (RMAIL 4/5 Narrow)--L1--All----------
Add label:
```

准备添加一个分类标签。

看到辅助输入区里的提示信息后，输入一个分类标签作为响应。如果此前使用过“**a**”命令，那么Emacs会记得以前的分类标签，并询问是否想用它做为默认的设置值。这很适合用来给一组邮件消息加上分类标签。

添加好分类标签之后，Emcs将把它显示在状态行上。如下所示：

输入：**meetings RETURN**

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help
Date: Fri,23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
---Emacs: RMAIL        (RMAIL 4/5;meetings'Narrow)--L1--All----------
```

这条邮件消息加上了“meetings”分类标签。

如果现在查看RMAIL的邮件清单，就会发现这个标签已经在那里了。如下所示：

按下：**h**

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help
Date: Fri,23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
--- Emacs: RMAIL          (RMAIL 4/5; meetings Narrow)--L1--All----------
  1  23-Aug              land@ora.com  Photo wanted
  2  23-Aug              mark@ora.com  PO number
  3  23-Aug               sue@ora.com  Corporate policy
  4  23-Aug           jessica@ora.com  { meetings,} Staff meeting
  5  23-Aug           george@grom.com  your recent order
--%*-Emacs:RMAIL-summary          (RMAIL Summary)--L1--All--------------
```

邮件清单已经给第 4 条消息加上了“meetings”分类标签。

可以创建一个特殊的邮件清单，让它只列出带某些特殊分类标签的邮件消息。与此对应的命令是“**l**”，后面必须跟着打算在邮件清单里列出的分类标签（可以有好几个）。如下所示：

按下：**l**

```
Buffers Files Tools Edit Search Minibuf Help
Date: Fri,23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
--- Emacs: RMAIL          (RMAIL 4/5; meetings Narrow)--L1--All----------
  1  23-Aug              land@ora.com  Photo wanted
  2  23-Aug              mark@ora.com  PO number
  3  23-Aug               sue@ora.com  Corporate policy
  4  23-Aug           jessica@ora.com  {meetings,}Staff meeting
  5  23-Aug           george@grom.com  your recent order
--%*-Emacs: RMAIL-summary         (RMAIL Summary)--L1--All--------------
Labels to summarize by:
```

可以创建一个信头清单，让它里面只列出带有某些特定分类标签的邮件。

输入：**meetings RETURN**

```
Buffers Files Tools Edit Search Move Delete Mail Summary Classify Help
Date: Fri, 23 Aug 1996 08:25:17 -0400
From: Jessica Tandy <jessica@ora.com>
To: deb
Subject: Staff meeting

The monthly staff meeting will be held in the conference room at 4:00
p.m.on Monday.
--- Emacs: RMAIL      (RMAIL 4/5; meetings Narrow)--L1--All----------
 4 23-Aug        jessica {meetings }Staff
10 1-Sep         george {meetings }Can't make it
15 1-Sep         deb {meetings }You lazy bum
--%*-Emacs: RMAIL-summary      (RMAIL Summary)--L1--All---------------
```

得到的邮件清单里只列出了那些带有“meetings”分类标签的邮件消息。

这个清单与其他邮件清单没有什么区别——可以在清单里上下移动、删除邮件等等。用不着去担心那些没有列在这份清单里的邮件。也就是说，分类标签功能能够方便地对邮件消息进行归类，并且能够方便地对邮件消息组进行处理。可以依次查看所有与会议有关的消息，决定对它们的取舍。而这一切对邮箱里的其他东西不会有任何影响。

可以给一条消息加上任意多个分类标签。有时候，如果一条消息上的分类标签太多，就可能无法把它们都显示在状态行上；但只要愿意，Emacs 并不会因此而限制使用更多的分类标签。

如果想去掉某个邮件消息上的分类标签，请使用“**k**”命令。比如，为了去掉某个邮件消息上的“meetings”分类标签，先要用 RMAIL 把那条邮件消息显示在屏幕上，然后输入命令“**k meetings RETURN**”。

有几个内建的分类标签是由 Emacs 自动管理的，它们都列在表 6-7 里。

表 6-7：内置的邮件消息分类标签

分类标签	含义
filed	此消息已经被保存到一个邮件文件里
unseen	还没有读过这条消息
answered	已经（用“**r**”命令）给这条邮件消息发了一封回信
forwarded	已经把这条消息转发给别人
deleted	已经给这条消息加上了待删除标记

虽然这些内置的分类标签不会出现在信头清单里，但它们会出现在状态行上，大家可能已经注意到了。它们与其他分类标签并没有什么不同，只是不必费心去设置和删除它们而已。Emacs 会根据对邮件消息所做（或者没有做）的操作而自动设置好它们——比如，新接收到的邮件消息会自动加上“unseen”分类标签，直到对它们进行了读取；而一旦读取过，Emacs 就会自动删除其“unseen”分类标签。把邮件消息保存到每个文件里时，它们会自动加上“filed”分类标签。删除“deleted”分类标签与去掉邮件消息上的待删除标记效果完全相同。“**d**”和“**u**”命令所做的事情其实就是添加和去除分类标签。（不信可以自己去试试。）类似地，给一条消息加上“unseen”分类标签，等于是告诉Emacs你还没有阅读过这条消息，以此类推。

对邮件清单进行排序

分类标签提供了一种对邮件进行整理和归类的办法，但Emacs还提供了其他一些办法。可以只查看邮件清单的一个子集，比如按收信人、正则表达式、主题或分类标签等分别查看邮件清单的某个部分。还可以用不同的办法对邮件文件本身进行排序。把这些技术组合在一起，就能轻松地对爆满的邮箱进行“减肥”。

我们刚刚讲过，按下“**l**”键（命令名是 **rmail-summary-by-labels**）就能按分类标签来查看邮件清单。还可以按主题（实际就是一个关键词或关键短语）来查看邮件清单。要想找出与某个主题有关的全部邮件，请按下“**ESC C-t**”（命令名是 **rmail-summary-by-topic**）组合键或者从“**Summary**”菜单里选择执行“**By Topic**（按主题）”操作。Emacs 将提示输入一个主题，把出现在邮件“Subject:”行里的某个单词或者短语做为关键字提供给它。Emacs 将查看全部邮件的“Subject:”行，把里面有那个关键字的邮件都挑出来列成一个邮件清单。（它不搜索邮件消息的内容，它还没那么能干。）

使用正则表达式筛选到的邮件清单就更有用了（对正则表达式的简单介绍请参考第三章）。如果想用正则表达式来查看邮件清单，可以按作者（比如所有来自 John 的邮件）、按日期（比如所有十月份的邮件），或者按主题行里的某个字符串等等来进行查找匹配。用正则表达式来查看邮件清单的方法是：按下“**ESC C-s**”（命令名是 **rmail-summary-by-regexp**）组合键或者从“**Summary**”菜单里选择执行“**By Regexp**（按正则表达式）”操作。按照 Emacs 的提示输入一个正则表达式（单词也

可以看做是正则表达式)，然后按下回车键。Emacs将把信头匹配上此正则表达式的所有邮件都挑出来列成一个邮件清单。这个清单会把无法从邮件清单里查看的某些信头信息也显示出来，比如有哪些用户收到过该邮件的抄送副本等。也就是说，如果对与John有关的邮件进行查找，除了John发送给的邮件外，向John发送过抄送副本的邮件也会被找出来。

这几个命令虽然能够看到RMAIL邮件清单的某个子集，却不能改变*RMAIL*文件的排序情况。可有时候，确实需要对*RMAIL*文件本身进行排序。在默认情况下，它是按照邮件到达的先后顺序来排序的，最新收到的排在最后一个。但可以按行数、作者或者主题等项目重新进行排序。按不同顺序对邮件进行排序听起来似乎很吓人，其实不然；即使按其他顺序对*RMAIL*文件本身进行了排序，也可以通过按日期重新排序的办法随时让它恢复原状。注意：即使按其他顺序对*RMAIL*文件进行排序之后放在那里不管，Emacs也仍然会按自己的办法对它进行管理和维护——它仍然会把最新收到的邮件追加在这个文件的末尾，不管是否按其他顺序对该文件进行过排序。当然，最好是在完成操作之后用“**ESC x rmail-sort-by-date**”命令对它重新进行排序。

假设*RMAIL*文件大得有点过头，它里面有一个很大的邮件——比如别人邮来的一本书，它和其他邮件混在一起。如果想把最长的邮件找出来，请输入“**ESC x rmail-sort-by-lines**”命令，Emacs将按从小到大的顺序对邮件进行排序。(这个顺序也很容易倒过来，只要在任何排序命令前加上一个“**C-u**”，就可以按逆序进行排序。在这种情况下，最长的邮件将位于邮件清单最顶部。)

还可以按作者或按主题进行排序。按主题排序的命令是“**ESC x rmail-sort-by-subject**”；按作者排序的命令是“**ESC x rmail-sort-by-author**”。Emacs将按字母表顺序对邮件清单进行排序；先是Fred发来的所有邮件，然后是Maureen发来的所有邮件，再后面是Zane发来的所有邮件等等。

表6-8对用来列出邮件清单的各种命令进行了汇总，“**Summary**”菜单里的有关操作选项也包括在其中。

表 6-8: 邮件排序命令速查表

键盘操作	命令名称	动作
h *Summary→All*	**rmail-summary**	显示邮件清单
ESC C-t *Summary→By Topic*	**rmail-summary-by-topic**	按指定主题列出一个邮件清单；相应的查找操作是对“Subject:”栏进行的
ESC C-s *Summary→By Regexp*	**rmail-summary-by-regexp**	按指定正则表达式列出一个邮件清单；相应的查找操作是对信头的所有信息栏进行的
ESC C-r *Summary→By Recipients*	**rmail-summary-by-recipients**[a]	按用户列出一个邮件清单
l *Summary→By Labels*	**rmail-summary-by-labels**	按分类标签列出一个邮件清单
（无）	**rmail-sort-by-date**	按日期对 *RMAIL* 文件进行排序
（无）	**rmail-sort-by-subject**	按主题对 *RMAIL* 文件进行排序
（无）	**rmail-sort-by-author**	按作者对 *RMAIL* 文件进行排序
（无）	**rmail-sort-by-recipient**	按收信人对 *RMAIL* 文件进行排序
（无）	**rmail-sort-by-lines**	按消息长度对 *RMAIL* 文件进行排序（从短到长）
（无）	**rmail-sort-by-correspondents**	按对应关系对 *RMAIL* 文件进行排序
C-u	**universal-argument**	颠倒排序顺序

a. 这里用“*recipient*（收信人）”一词有些误导成分。这个命令给出的邮件清单其实是按用户排列的，不管他是发出邮件的人、收到邮件的人、还是收到邮件抄送副本的人。

熟悉邮件的使用方法之后，下一步就是Usenet新闻：根据不同的用户，它可能是一

种信息财富，也可能是一种时间和磁盘空间的巨大浪费。Emacs自带一个名为Gnus的新闻阅读器，它就是下面将要介绍的东西。

用 Gnus 读取 Usenet 新闻

Usenet最初是作为美国北卡罗莱纳州两所大学之间的一个公告板系统出现的。随着时间的推移，Usenet在运营站点数量和讨论组数量两方面都有非常巨大的增长。人们通常把Usenet中的讨论组称为“新闻组（newsgroup）”，它们现在的数量要以千来计算，讨论的话题从超人到超级计算机，可以说是五花八门，应有尽有。

新闻在某些方面与邮件很相似，但它们彼此之间有一些重要的区别。首先，它们用的术语就不一样：电子邮件说的是消息，新闻则说的是文章（国人经常称之为“帖子”）。发送的是邮件消息，张贴的则是自己的文章（“张贴”暗含着“这是一个公告板系统”的意思，用来描述Usenet正合适）。其次，新闻的受众和传播形式与电子邮件是截然不同的。电子邮件只会发到发信人指定的那个人手里，而Usenet新闻却会被世界各地众多的因特网站所接收——它是一个属于大众的讲坛。一篇文章哪怕只张贴在某一个新闻组里，它也可能会被成千上万的人读到。（这可能会让人们犹豫该不该向Usenet张贴稿件，但它确实是这样的。）

人们经常把新闻说成是Usenet新闻，但严格说来，这两种叫法都不准确。与Usenet没有关系却依然属于新闻范畴的新闻组是存在的。基于这个原因，有些人把Usenet新闻组更准确地称为网络新闻。“新闻”这个词本身含义就比较含糊。大多数新闻组其实都是一些讨论组，其内容与人们心目中像《华尔街日报》或《晚间新闻》之类传统意义上的新闻没有什么瓜葛。确实有少数新闻组会提供传统意义上的新闻——比如Clarinet新闻组，只能在支付服务费用的站点上才能获得。

在Emacs里就能通过Gnus阅读和张贴新闻。在Emacs 19.30里，GNUS 4.x已经升级为Gnus 5.x。有改动的地方还真不少——至少它名字里的大小写改了。此外，Gnus仍在进一步地开发当中。它是一个完备的新闻阅读器软件。如果你还在使用GNUS 4.x——那么别管什么原因，请找系统管理员讨论升级问题。

这一小节里将要介绍Gnus的基本用法。详细资料请参考Gnus的在线手册——它的主页地址是*http://www.ifi.uio.no/~larsi/ding-manual/gnus_toc.html*

（注 3），或者去查阅由 Mark Harrison 撰写、O'Reilly & Associates 出版公司出版的《The USENET Handbook》（USENET 手册）一书。

启动 Gnus

Gnus 5.x 的重大改进之一是它的首次启动容易了许多。Gnus 的早期版本在首次启动时会给用户订阅所有可订阅的新闻组（光是这项工作就得用掉一个多小时的时间），而如今的Gnus会从先给用户订阅少量的新闻组开始，不再像以前那样让人等得着急了。如果以前阅读过新闻，那么Gnus将使用原有的新闻初始化文件“*.newsrc*”启动，这个文件存放在主目录里（也就是说，它的完整路径是“*~/.newsrc*”）。

启动Gnus的方法是输入“**ESC x gnus RETURN**”命令。Gnus将尝试读取“*.newsrc*”文件。如果这个文件不存在，那么 Gnus 就会创建它。如下所示：

输入：**ESC x gnus RETURN**

Emacs 显示的 Gnus 初始画面。

注 3：这堆字符串是一个 URL（Uniform Resource Locator，统一资源定位器）地址，这是 World Wide Web 所使用的地址形式。用我们将在第七章介绍的 Emacs 的 W3 模式就可以查阅这个手册。

画面会自动前进，用不着按任何键。可能还会出现一条信息“Garbage collecting”。这条信息并不意味着有什么新闻审查软件出现，它的意思是说内部清理工作正在进行，而这些事是用不着用户去操心的。

```
Buffers Files Tools Search Misc Groups Group Help
      4: news.announce.newusers
    111: news.groups.questions
     13: gnu.emacs.gnus
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L1--All-------
Checking new news...done
```

Emacs 给出的“Group”编辑缓冲区。

“Group”编辑缓冲区列出订阅的（如果是一位新用户，那么就是 Gnus 挑选的）所有新闻组。如果已经读过订阅的所有新闻组里的所有新闻，那么 Emacs 就会显示一条谚语“No news is good news”做为提示。

这个编辑缓冲区给出各新闻组的一些情况。新闻组名称前面的数字是这个新闻组里尚未阅读过的新闻条数。有些新闻组里会有成千上万篇文章，而有些却连 50 篇也到不了。

选读一个新闻组

在“Group”编辑缓冲区里，首先得移动到想要阅读的新闻组。有以下几种方法。如果想选读的新闻组就显示在屏幕上，那么把光标移动到这一行再按下回车键即可。如果知道想要选读的新闻组的名字，那么请按下“**j**”键（命令名是 **gnus-group-jump-to-group**）。Emacs 提示输入准备阅读的新闻组的名称。可以逐字输入它完整的名称，也可以按 **TAB** 键以利用 Emacs 的自动补足功能。它允许用户挑选任何一个新闻组，即使以前并没有订阅也不要紧。下面以阅读 *gnu.emacs.help* 新闻组为例来进行说明。

按下：**j**

```
Buffers Files Tools Edit Search Minibuf Help
      4: news.announce.newusers
    111: news.groups.questions
     13: gnu.emacs.gnus

-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L1--All-------
Group:
```

Emacs 提示输入一个新闻组名称。

输入：**gnu.emacs.help RETURN**

```
Buffers Files Tools Search Misc Groups Group Help
      4: news.announce.newusers
    111: news.groups.questions
     13: gnu.emacs.gnus
K    40: gnu.emacs.help
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L4--All------
```

Emacs 移动到 *gnu.emacs.help* 新闻组。

注意新闻组名称旁边的大写字母“**K**”，它意味着这个新闻组已经被某个初始化文件排除（注4）；这是为了保证屏幕画面不至于被过多的新闻组弄得乱七八糟。如果想订阅这个新闻组，请使用“**u**”（命令名是**gnus-group-unsubscribe-current-group**）命令。这个命令其实是一个切换开关（如果尚未订阅，它就进行订阅；如果已经订阅，就取消订阅）。虽然“**u**”命令看上去不那么直观，但用起来却没问题。请继续往下看：

按下：**u**

```
Buffers Files Tools Search Misc Groups Group Help
      4: news.announce.newusers
    111: news.groups.questions
     13: gnu.emacs.gnus
     40: gnu.emacs.help
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L4--All------
```

Emacs 订阅这个新闻组，并去掉它旁边的大写字母“**K**”标记。

要想阅读新闻组里的新闻，请按“**SPACE**”（命令名是**gnus-group-read-group**）键。如果新闻组里的新闻很多，Emacs 会让用户来决定想阅读多少篇文章。

Emacs现在会进入“Summary”编辑缓冲（这个编辑缓冲区将在下一小节介绍）。表6-9 对“Group”编辑缓冲区里使用的部分操作命令进行了汇总。

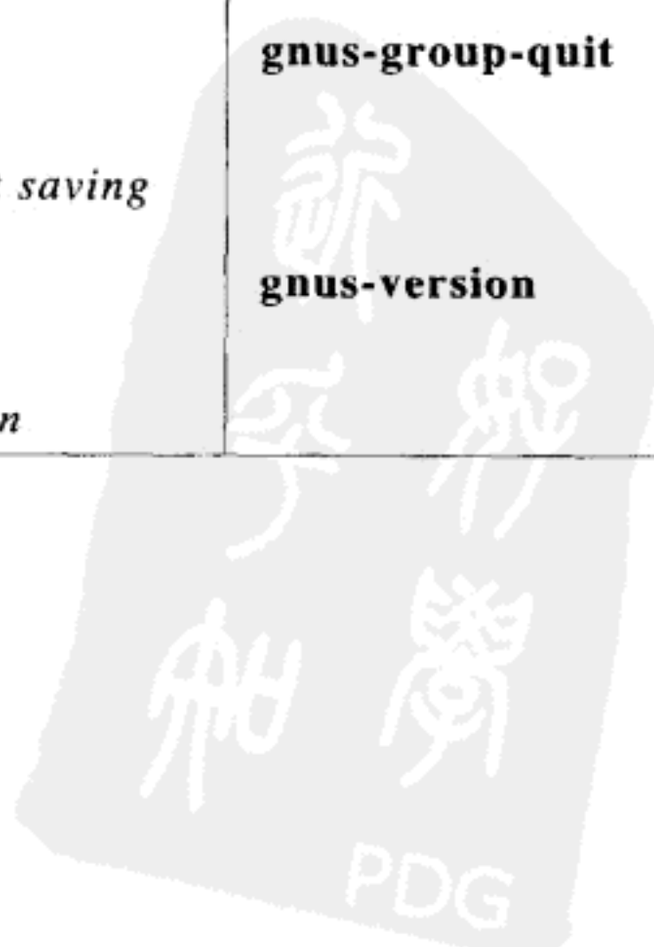
注 4：Gnus 的启动过程是这样的：如果你有一个“*.newsrc*”文件，它就会读入；如果你没有“*.newsrc*”文件，就说明你可能没有在这台机器上阅读过新闻。Gnus 现在是不会去创建“*.newsrc*”文件的（因为这可能要花上好几个小时的时间），它会先在一个名为“*.newsrc.eld*”的辅助启动文件里把其中的大部分新闻组排除掉，只保留少数几个。然后再把很少的几个新闻组放到你的“*.newsrc*”文件里——你今后可以按自己的想法随意增加。用户自己是不能直接修改“*.newsrc.eld*”文件的，有些命令能自动对它做出修改。

表 6-9: “Group” 编辑缓冲区操作命令速查表

键盘操作	命令名称	动作
(无) *Tools→Read Net News*	**gnus**	启动 Gnus
SPACE *Group→Read*	**gnus-group-read-group**	阅读光标位置处的新闻组里的文章
j *Groups→* *Jump to group*	**gnus-group-jump-to-group**	提示输入一个新闻组名称以转到它那里去（可以转到未曾订阅的新闻组）
n	**gnus-group-next-unread-group**	移动到下一个有尚未阅读过的新闻的新闻组
p	**gnus-group-prev-unread-group**	移动到上一个有尚未阅读过的新闻的新闻组
N	**gnus-group-next-group**	移动到下一个新闻组
P	**gnus-group-prev-group**	移动到上一个新闻组
<	**beginning-of-buffer**	移动到编辑缓冲区的开始
>	**end-of-buffer**	移动到编辑缓冲区的末尾
u *Group→* *Toggle subscription*	**gnus-group-unsubscribe-current-group**	订阅或者撤销订阅这个新闻组
U	**gnus-group-unsubscribe-group**	订阅或者撤销订阅一个指定的新闻组
c	**gnus-group-catchup-current**	给这个新闻组里的所有文章都加上已阅读标记并删除它们
C *Group→* *Catch up all articles*	**gnus-group-catchup-current-all**	给这个新闻组里的所有文章都加上已阅读标记并删除它们，包括那些带惊叹号标记（表示文章已保存）的文章
A k	**gnus-group-list-killed**	列出那些被“*.newsrc.eld*”文件里的语句所排除掉的新闻组
l	**gnus-group-list-groups**	列出已订阅并且有新闻可读的新闻组

表 6-9: “Group” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
L	**gnus-group-list-all-groups**	列出此服务器上的全体新闻组
g *Misc→* *Check for new news*	**gnus-group-get-new-news**	取回启动 Gnus 后新收到的新闻
R *Misc→* *Restart Gnus*	**gnus-group-restart**	读“*.newsrc*”文件并重新启动 Gnus（同时取回最新的新闻）
b *Misc→* *Delete bogus groups*	**gnus-group-check-bogus-groups**	找出不存在的新闻组并删除它们
a *Misc→* *Post an article*	**gnus-group-post-news**	为这个新闻组写一篇新文章
C-x C-t	**gnus-group-transpose-groups**	交换当前行和上一行的位置
s *Misc→* *Save .newsrc files*	**gnus-group-save-newsrc**	保存“*.newsrc*”文件
z *Misc→* *Suspend Gnus*	**gnus-group-suspend**	临时性地挂起 Emacs 和 Gnus
q *Misc→* *Exit from Gnus*	**gnus-group-exit**	退出新闻功能并刷新“*.newsrc*”文件
Q *Misc→* *Exit without saving*	**gnus-group-quit**	退出新闻功能但不刷新“*.newsrc*”文件
V *Misc→* *Gnus version*	**gnus-version**	显示 Gnus 的版本号



阅读新闻消息

选定一个新闻组之后，Emacs 屏幕画面也将随之改变。画面的上半部分是一个“Summary”编辑缓冲区，下半部分则是一个“Article”编辑缓冲区。如下图所示：

```
Buffers Files Tools Search Post Threads Article Score Misc Help
R  [ 50: Greg Kedge            ] Re: Strip signature
     <29: Richard Pieri          >
   [ 33: Kevin K. Lewis         ] Re: ksh-mode.el
      < 29: Zach Leber           >
- Gnus gnu.emacs.help/31001 {39 more}                    (Summary)-L1-
From: Greg Kedge<gkedge@ycc.Kodak.COM>
Subject: Re: Strip signature
Newsgroups: gnu.emacs.help,comp.emacs
Date: 23 Jan 1996 12:05:32 -0500
Organization: Eastman Kodak Co.Photo CD
Reply-To: gkedge@ycc.kodak.com


    Greg>When I reply to a message in which I am going to yank(C-c C-y)
    Greg>the original into the reply,I am interested in first stripping
- Gnus gnu.emacs.help/31001 Re: Strip signature          (Article)-L1-
Loading gnus-cite...done
```

本画面分为一个“Summary”编辑缓冲区和一个“Article”编辑缓冲区。

“Summary”编辑缓冲区用来给出各篇文章的基本情况。已经读过的文章的左边会出现一个大写的字母“R”标记，表示已经读过（读过的文章会在这次新闻操作结束时从未阅读文章清单里删除）。方括号里给出的是文章里的文本行数和（如果有）作者的名字（有时候只有作者的电子邮件地址）。接着是文章的标题。在上面这个画面里，当前文章的标题是“Re: Strip signature”。类似于电子邮件方面的情况，这个“Re:”表示这篇文章是对此前一篇同名文章的回复。新闻组的名字会显示在状态行上，它后面跟着一个用斜线字符隔开的数字；这个数字是当前文章的编号，其作用相当于每一篇 Usenet 帖子的身份代码。

为什么有些行是缩进的？因为它们是“线索（thread）”，也就是回复别人文章的文章。新闻是一场永无休止的对话，事实上，它是许多场同时进行的对话。阅读不带线索的新闻就好像是在一间非常大的屋子里听很多人同时在交谈一样，没有头绪；而沿着一条交谈线索去听就方便多了。Gnus是一个线索化的新闻阅读器，线索是被默认启用的——可以用“**ESC C-t**”（命令名是**gnus-summary-toggle-threads**）组合键关闭它们。因为线索的缘故，某篇文章的所有回复都会附在该文章的后面。有

时候，甚至会有针对回复文章的回复。在上面的例子里，Richard Pieri 针对 Greg Kedge 的帖子写了一篇回复文章。

用不着移动到“Article”编辑缓冲区就可以读到新闻，而事实是也不应该那样做。Emacs 为“Summary”编辑缓冲区准备的命令，可以不用移动到“Article”编辑缓冲区，就能看到新闻的内容。“Article”编辑缓冲区里只有很少几条命令可供使用（比如，用来后翻新闻内容的**SPACE**键，或用来前翻新闻内容的**DEL**键等），而这些命令在“Summary”编辑缓冲区里使用时也能达到同样的效果。

在正常情况下，按“**n**”键（命令名是**gnus-summary-next-unread-article**）移动到下一篇文章，按“**p**”键（命令名是**gnus-summary-prev-unread-article**）移动到上一篇文章；这两个命令的作用是在未阅读过的文章中间移动。如果因“**n**”键按得太快而跑过了头，就需要向回移动去阅读此前的文章，哪怕Emacs认为已经读过它了。返回并阅读前一篇文章的方法是按下“**P**”键（命令名是**gnus-summary-prev-article**）；这条命令将不管是否已经阅读过前一篇文章而返回到那里。另外一个方法是按下“**l**”键（小写的字母“L”，命令名是**gnus-summary-goto-last-article**），它的作用是返回到刚才读过的最后一篇文章。现在，我们准备阅读下一篇文章。

按下：**n**

```
Buffers Files Tools Search Post Threads Article Score Misc Help


R  [   50: Greg Kedge            ] Re: Strip signature
R       <  29: Richard Pieri        >
   [   33: Kevin K.Lewis         ] Re: ksh-mode.el
        <  29: Zach Leber           >
- Gnus gnu.emacs.help/31013 {38 more}                    (Summary)-L2-

From: Richard Pieri <ratinox@unilab.dfci.harvard.edu>
Subject: Re: Strip signature
Newsgroups: gnu.emacs.help,comp.emacs
Date: 24 Jan 1996 11:27:34 -0500
Organization: Dana-Farber Cancer Institute
-----BEGIN PGP SIGNED MESSAGE-----
>>>>>"g" === gkedge <Greg>writes:
g>This better should look for "^- $$" (note the space after the
g>dashes).
I'd have to double-check the RFC (yes, there *IS*an RFC that defines
the signature delimiter) but I believe that "^- 0(rq is a better regular
expression to match on.
- Gnus gnu.emacs.help/31013 Re:Strip signature          (Article)-L20-
```

Emacs 把下一篇文章显示到屏幕上。

现在，第一篇和第二篇文章都已经加上已阅读标记“R”；离开新闻功能的时候它们将会被删除。光标则向下移动到当前文章处。

读过几篇文章之后，你可能会认为其余的文章都不值得一读了。这时，可以按下“**c**”键（命令名是 **gnus-summary-catchup-and-exit**）结束这次阅读新闻工作。Emacs 将提问：

```
Mark all unread articles as read?
```

如果回答“**y**”，Emacs 就将给清单里所有还未曾阅读过的文章（仅限于这个新闻组）都加上已阅读标记，然后回到“Group”编辑缓冲区。

另外一种情况是发现有些文章很有意思，值得保存下来。可是，Gnus 会给阅读过的文章都加上已阅读标记，并且会在退出时删除它们。要想保存某篇文章，必须明确地给它加上未阅读标记——即告诉 Gnus 要把它留在文章清单里等待阅读。保存当前文章的方法是按下“**u**”键（命令名是 **gnus-summary-tick-article-forward**）。Emacs 将把字母“**R**”标记替换为一个惊叹号。这样，当下次进入这个新闻组时，这篇文章就还会保留在那里。如果想把文章保存到一个文件里（而不是让它留在新闻组里），请按下“**C-o**”组合键把它保存到一个 ASCII 文件里，或者按下“**o**”键把它保存为 RMAIL 格式。

如果已经把某个新闻组里的文章都读过了，Emacs 会询问是否还想前进到下一个新闻组；如果已经没有别的新闻组可读，Emacs 就会询问是否想退出。可以用“**q**”键（命令名是 **gnus-summary-exit**）随时返回到“Group”编辑缓冲区。

下一小节将讨论怎样才能查看到所有可供阅读的新闻组，过滤（kill）文件又该怎样使用，以及如何向新闻组投稿等事项。表 6-10 列出了一些可以在“Summary”编辑缓冲区里使用的命令。

表 6-10：“Summary”编辑缓冲区操作命令速查表

键盘操作	命令名称	动作
SPACE	**gnus-summary-next-page**	文章前卷
DEL	**gnus-summary-prev-page**	文章后卷
RETURN	**gnus-summary-scroll-up**	文章前卷，每次一行

表 6-10: “Summary” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
.	**gnus-summary-first-unread-article**	移动到此新闻组里尚未读过的第一篇文章处
< *Article→* *Beginning of the article*	**gnus-summary-beginning-of-article**	移动到当前文章的开始
> *Article→* *End of the article*	**gnus-summary-end-of-article**	移动到当前文章的末尾
n	**gnus-summary-next-unread-article**	移动到下一篇文章
N	**gnus-summary-next-article**	移动到下一篇文章（即使已经读过）
p	**gnus-summary-prev-unread-article**	移动到上一篇文章
P	**gnus-summary-prev-article**	移动到上一篇文章（即使已经读过）
l	**gnus-summary-goto-last-article**	移动到刚读过的最后一篇文章（如果移动得太快，可以用这条命令返回去）
H f *Misc→Fetch group FAQ*	**gnus-summary-fetch-faq**	取回这个新闻组的常见问题答疑文件
ESC C-t *Threads→* *Toggle threading*	**gnus-summary-toggle-threads**	打开 / 关闭线索功能
ESC C-k *Threads→* *Mark thread as read*	**gnus-summary-kill-thread**	排除当前线索，包括其子线索
ESC C-d *Threads→* *Go down thread*	**gnus-summary-down-thread**	移动到这个线索的下一篇文章处

表 6-10: “Summary” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
ESC C-u *Threads→* *Go up thread*	**gnus-summary-up-thread**	移动到这个线索的上一篇文章处
ESC C-f *Threads→* *Go to next thread*	**gnus-summary-next-thread**	移动到下一个线索（即移动到另外一个主题）
ESC C-b *Threads→* *Go to previous thread*	**gnus-summary-prev-thread**	移动到上一个线索（即移动到另外一个主题）
ESC C-h *Threads→Hide thread*	**gnus-summary-hide-thread**	隐藏当前线索
ESC C-s *Threads→* *Display hidden thread*	**gnus-summary-show-thread**	显示当前线索
q	**gnus-summary-exit**	返回“Group”编辑缓冲区
Q	**gnus-summary-exit-no-update**	返回“Group”编辑缓冲区，但不删除读过的文章
c *Group→Catch up*	**gnus-summary-catchup-and-exit**	给此新闻组里的文章都加上已阅读标记并返回“News-group”编辑缓冲区
u	**gnus-summary-tick-article-forward**	给当前文章加上未阅读标记，在它旁边放上一个惊叹号以保留它供今后阅读。如果重复输入此命令，则保存下一篇文章
U	**gnus-summary-tick-article-backward**	给当前文章加上未阅读标记，在它旁边放上一个惊叹号以保留它供今后阅读。如果重复使用此命令，则保存上一篇文章
C-o	**gnus-summary-save-article-mail**	以 UNIX 格式保存当前文章

PDG

表 6-10: “Summary” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
o	**gnus-summary-save-article**	以RMAIL格式保存当前文章
d	**gnus-summary-mark-as-read-forward**	给当前文章加上已阅读标记，从当前行开始向下移动
D	**gnus-summary-mark-as-read-backward**	给当前文章加上已阅读标记，从当前行开始向回移动
j	**gnus-summary-goto-subject**	要求指定在屏幕上显示的文章篇数
ESC n	**gnus-summary-next-unread-subject**	移动到下一个尚未读过的主题
ESC p	**gnus-summary-prev-unread-subject**	移动到上一个尚未读过的主题
ESC C-n	**gnus-summary-next-same-subject**	移动到同一主题的下一篇文章
ESC C-p	**gnus-summary-prev-same-subject**	移动到同一主题的上一篇文章
m *Post→Send a mail*	**gnus-summary-mail-other-window**	打开一个电子邮件消息的编辑缓冲区
C-c C-f	**gnus-summary-mail-forward**	把这篇文章的副本发送给某人
=	**gnus-summary-expand-window**	扩展“Summary”窗口，让它充满整个屏幕
g *Article→Redisplay*	**gnus-summary-show-article**	显示当前文章（特别适用于扩展了的“Summary”窗口或者需要在这个窗口里移动的情况）
s *Article→Isearch article*	**gnus-summary-isearch-article**	用isearch功能查找当前文章中的文本
ESC s *Article→* *Search all articles*	**gnus-summary-search-article-forward**	对当前文章进行正方向的正则表达式查找

表 6-10: “Summary” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
ESC r	**gnus-summary-search-article-backward**	对当前文章进行反方向的正则表达式查找
t	**gnus-summary-toggle-header**	打开/关闭文章信头的详细显示状态
w	**gnus-summary-stop-page-breaking**	不在文章中出现换页符的地方进行分页
x	**gnus-summary-remove-lines-marked-as-read**	把带已阅读标记的文章全部删除
C-c TAB	**gnus-info-find-node**	显示 Gnus 的 **Info** 帮助信息
C-c C-r	**gnus-summary-caesar-message**	加密 / 解密这篇文章
C-x C-s	**gnus-summary-reselect-current-group**	重启这个新闻组，不给文章加上已阅读标记
ESC t	**gnus-summary-toggle-mime**	进入 MIME 模式
ESC U	**gnus-summary-clear-mark-backward**	清除当前行上所有标记，包括连字符、排除标记“K”、待删除标记“D”。如果重复输入这个命令，则清除前一行上的标记
ESC u	**gnus-summary-clear-mark-forward**	清除当前行上所有标记。如果重复使用这个命令，则清除后一行上的标记
C-t *Misc→* *Toggle line truncation*	**gnus-summary-toggle-truncation**	打开/关闭针对长文本行的截断功能
& *Misc→Filter articles*	**gnus-summary-execute-command**	在信头部分里查找指定的正则表达式
|	**gnus-summary-pipe-output**	把这篇文章经管道输出到一个子进程去，比如送去打印等

表 6-10: “Summary” 编辑缓冲区操作命令速查表（续）

键盘操作	命令名称	动作
C-d *Article→* *Enter digest buffer*	**gnus-summary-rmail-digest**	如果这篇文章是一份文摘，分析并阅读之

列出所有的新闻组

在默认情况下，Gnus会把新闻组的名单从（典型的）10,000个以上裁减为只剩下几个。可好奇的用户可能想去看看到底有多少新闻组是可用的。为此，你必须在“Group”编辑缓冲区里（如果还待在“Summary”编辑缓冲区里，请输入“**C-x b *Group***”命令进入“Group”编辑缓冲区）进行操作。查看全部新闻组名单的方法是连续输入“**A k**”（命令名是 **gnus-group-list-killed**）。如下所示:

输入: **A k**

```
Buffers Files Tools Search Misc Groups Group Help
K     *: alt.Cajun.info
K     *: alt.abortion.inequity
K     *: alt.abuse.offender.recovery
K     *: alt.abuse.recovery
K     *: alt.activism
K     *: alt.activism.d
K     *: alt.activism.death-penalty
K     *: alt.adoption
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L1--Top------
A k
```

Gnus 把全部可用的新闻组都列出来。

要想找出自己感兴趣的新闻组，可以用“**C-s**”组合键对之进行查找，或者就老老实实地翻看这份名单。如果想订阅某个新闻组，请按下“**u**”键。如果想返回自己已经订阅的新闻组名单（那份名单现在已经包括刚刚添加的那些新闻组了），请按“**l**”（小写的“L”字母）键，它对应的命令名是 **gnus-group-list-groups**。

过滤文件（kill file）的使用

Usenet 里的帖子反映了各种各样的人性。新闻组里充斥着各种各样的文章，它们有的来自智者，有的来自普通大众，有的来自幽默大师，有的来自讨厌鬼，甚至还有

些是来自心理阴暗的人。几乎每个新闻组里都有那么一两个臭名昭著的人，他们不停地张贴文章，一会儿是想卖些谁也不想买的东西，一会儿是散布一些与这个新闻组里大多数人观点对立的言论。这就是为什么会有过滤文件的原因——它们能有选择地对某些人或话题说“到此为止!”。过滤文件是用LISP命令写成的，它会告诉Gnus哪些东西是不想看到的。Gnus会自动排除那些不想看见的文章。因为不是每个人都知道如何编写LISP语句，所以Gnus提供了一些命令来帮你完成这件事情。

阅读文章的时候（在“Summary”编辑缓冲区里才能阅读新闻），即使没有使用过滤文件（我们马上就要对它进行详细的讨论），也有一条快速命令可以选用。如果关于某个话题的文章来得非常之多并且对它们却都不感兴趣，就需要在其中某篇文章显示在屏幕上时按下“**C-k**”（命令名是**gnus-summary-kill-same-subject**）组合键排除掉它们。如果只输入了一个字母“**k**”（命令名是**gnus-summary-kill-same-subject-and-select**），Emacs将排除掉与这个主题有关的全部文章，并选择下一个主题以供阅读。在“**k**”或“**C-k**”的前面加上“**C-u**”等于是进行相应的反操作，并且会把各种标记清除掉。Emacs会给用这种方法排除掉的文章旁边加上一个“**K**”标记，然后在用户阅读其他新闻的时候越过它们。

这些命令只对本次工作有效，而这往往也正是希望达到的效果。过滤文件是一种更具永久性的解决方案；只要不去修改它们，它们就会一直起作用。可以先用“**C-k**”命令在某次阅读新闻时试试，如果“垃圾”文章还是源源不断地涌现出来，就应该考虑在某个过滤文件里增加一条语句以彻底地解决它们。

可以创建一个局部过滤文件专门来对付某个新闻组，也可以创建一个全局过滤文件，在全部新闻组里把来自某个作者或者关于某个主题的帖子都排除干净。下面来看一个给局部过滤文件添加语句的例子。编辑一个局部过滤文件的方法是按下“**ESC k**”（命令名是**gnus-summary-edit-local-kill**）组合键；创建一个全局过滤文件的方法是按下“**ESC K**”（命令名是**gnus-summary-edit-global-kill**）组合键。

当想排除的文章出现在屏幕上时，按下“**ESC k**”组合键。Emacs将在屏幕上打开一个新的编辑缓冲区，它（显示在状态行上）的名字是那个新闻组的名字再加上一个“*.KILL*”后缀。（全局过滤文件的名字就简简单单地是“*KILL*”。）。请看下图:

按下：**ESC k**

```
Buffers Files Tools Search Post Threads Article Score Misc Help
R       [  18:Tom "Tom " Harrington ]
R  [  24: Joe Btsfphbl          ] Great Deals on Grommets!
   [   7: Sidney Schwartz       ] NG charter archives?
   [  15: jules.verne@a1.nl     ] Why the HD-led lights w/no W/R-action?
- Gnus news.groups.questions/15927 {64 more}          (Summary)-L44-

--**-Emacs: news.groups.questions.KILL   (Kill)--L4--All-------------
```

Emacs 打开一个新的编辑缓冲区，它的名字是新闻组名加一个“*.KILL*”后缀。

如果想把这位作者的所有文章都排除掉，请按下“**C-c C-k C-a**”（命令名是 **gnus-kill-file-kill-by-author**）组合键。如下所示：

按下：**C-c C-k C-a**

```
Buffers Files Tools Search Post Threads Article Score Misc Help
R       [  18: Tom "Tom " Harrington ]
R    [ 24: Joe Btsfphbl          ] Great Deals on Grommets!
     [  7: Sidney Schwartz       ] NG charter archives?
     [ 15: jules.verne@a1.nl     ] Why the HD-led lights w/no W/R-action?
-Gnus news.groups.questions/15927 {64 more}   .          (Summary)-L44-
(gnus-kill "From " "joe@grom \.com (Joe Btsfphbl)")

--**-Emacs: news.groups.questions.KILL   (Kill)--L4--All-------------
```

Emacs把用来在这个新闻组里排除这位作者全部文章的LISP代码，插入到编辑缓冲区里。

如果想按主题而不是按作者进行过滤，请按下“**C-c C-k C-s**”（命令名是 **gnus-kill-file-kill-by-subject**）组合键。Emacs 将把与这个主题有关的所有文章都排除掉，但也仅限于这个主题。如果真正想排除的是包含有某些特定单词（比如“*grommets*”）的全部文章，就需要使用正则表达式，而且还得有点LISP编程经验才行（详细讨论请参考第十三章）。

永久保存这些设置并退出“kill”编辑缓冲区的方法是按下“**C-c C-c**”（命令名是 **gnus-kill-file-exit**）组合键。以后，每次进入Gnus时，Emacs 都会自动加载这个过滤文件，并给那些被排除的文章加上一个大写字母“X”标记。如果没有用“**ESC u**”（命令名是 **gnus-summary-clear-mark-forward**）组合键去掉“X”标记，Gnus 就会自动越过这些文章。

另外，如果用其他新闻阅读器创建过滤文件，那么Gnus也会使用它们。表6-11列出了与过滤文件有关的操作命令。

表6-11：与过滤文件有关的操作命令速查表

键盘操作	命令名称	动作
C-k *Group→Kill*	**gnus-summary-kill-same-subject**	排除这个主题下的所有文章；其反操作是“**C-u C-k**”组合键
k	**gnus-summary-kill-same-subject-and-select**	排除这个主题下的所有文章并选择下一个主题；其反操作是“**C-u k**”组合键
ESC k *Group→Edit kill file*	**gnus-summary-edit-local-kill**	编辑一个局部过滤文件（只对当前新闻组有影响）
ESC K *Misc→* *Edit global kill file*	**gnus-summary-edit-global-kill**	编辑一个全局过滤文件（对所有的新闻组都有影响）
C-c C-k C-a	**gnus-kill-file-kill-by-author**	插入排除此作者全部文章的LISP代码
C-c C-k C-s	**gnus-kill-file-kill-by-subject**	插入排除此主题全部文章的LISP代码
C-c C-c	**gnus-kill-file-exit**	保存过滤文件并退出“kill”编辑缓冲区

获取一份常见问题答疑（FAQ）文件

在读了一些新闻之后，有人可能想更积极地参与到其中，比如张贴一篇新文章，给文章的作者发一封邮件、或者把回应文章张贴到新闻组里等等。在向Usenet投稿之前，应该先阅读一下那个新闻组的常见问题答疑（frequently asked questions，简称FAQ）文件。Gnus专门为此准备了一个快速命令，只要连续按下“**H f**”（命令名是**gnus-summary-fetch-faq**）两键即可。Gnus会自动建立一条FTP协议（file transfer protocol，FTP）通路连接到存有该新闻组FAQ文件的某个服务器上。在Emacs里使用FTP功能的详细讨论请参考第七章。

在“Summary”编辑缓冲区按下“**H f**”两键。如下所示：

按下：**H f**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
R   [  12: Teddy Hogeborn      ] Re: "Couldn't open server " only on first t
       <    20: Michael K.Sanders  >
    [  49: Glenn Barry         ] Re: xemacs using xv to display xface inst$
       <    19: Pete Forman        >
- Gnus gnu.emacs.gnus/6641 {10 more}                          (Summary)-L1-
 /ftp@mirrors.aol.com:/pub/rtfm/Usenet/gnu.emacs.gnus:
 total 292
 drwxr-xr-x   2 root     sys         512 Jan 22 02:38 .
 drwxr-xr-x  1498 root   sys       44544 Jan 30 02:13 ..
 -r--r--r--   1 root     sys       28617 Jan 20 00:36 gnus_(Emacs_Newsrea\
der)_FAQ
 --%%-Dired: gnu.emacs.gnus          (Dired by name)--L5--All--------------
Reading directory /ftp@mirrors.aol.com:/pub/rtfm/usenet/gnu.emacs.gnus/
 ...done
```

Gnus 自动建立一条通路连接到存有该新闻组 FAQ 文件的某个服务器上。

如果那个FTP服务器因为有很多用户在登录而不能再接纳登录用户，那么就会显示一条说明这一情况的信息，请过几分钟再试试。按下“**H f**”键会进入一个 Dired 编辑缓冲区里，我们在第五章里已经解释过这一问题。它与平常使用的其他Dired 编辑缓冲区没什么两样，但实际列在其中的却是某个远程服务器上的一些FAQ 文件。（进入 Dired 编辑缓冲区里而非显示FAQ 文件内容的原因是：很多FAQ 都是由多个相对小一点的文件组成的，必须根据需要进行选择。）用 Dired 命令查看那些文件，移动到想要的FAQ 文件所在的那一行，按“**f**”键。如下所示：

按下：**f**

```
Buffers Files Tools Edit Search Help
R [  12: Teddy Hogeborn        ] Re: "Couldn 't open server " only on first t
     <   20: Michael K.Sanders     >
  [  49: Glenn Barry           ] Re: xemacs using xv to display xface inst$
newsserver          n.mit.edu!          < 19: Pete Forman        >
-Gnus gnu.emacs.gnus/6641 {10 more}                          (Summary)-L1-
 ........th: senator-bedfellow.mit.edu!b.pixel.kodak.com
From: Steven L Baur <steve@miranova.com>
Newsgroups: gnu.emacs.gnus
Subject: Gnus (Emacs Newsreader)FAQ
Supersedes: <m291k8aoaq.fsf@diana.miranova.com>
Followup-To: poster
Date: 19 Jan 1996 11:26:09 -0800
Organization: Miranova Systems,Inc.
Lines: 841
kugtq m2ag3Approved: news-answers-request@MIT.EDU
Expires: 21 Feb 1995 00:00:00 GMT
Message-.fsf@miranova.com>
NNTP-Posting-Host: 204.212.162.100
Mime-Version: 1.0 (generated by tm-edit 7.40)
-----Emacs: Gnus_(Emacs_Newsreader)_FAQ   (Text Fill)--L1--Top-------
```

Emacs 把该 FAQ 文件的内容显示在屏幕上。

接下来，可以在线阅读这份FAQ，也可以把它保存到一个本地文件上以后再看。本地保存这份文件的方法是先按下“**C-x C-w**”组合键，再给出一个本地系统上的文件名。（如果需要从头开始输入一个新的路径名，先用“**C-a**”命令移动到路径名的最开始，再用“**C-k**”命令删除它。）输入一个本地文件名，就可以把那份FAQ复制下来。这只是查看和在本地保存远程文件的方法之一，用 Dired 的复制命令（“**C**”键）也可以同样容易地把它复制下来。有关操作请参考第五章中对 Dired 模式的介绍。既然知道获取FAQ文件是这么的容易，那还不赶快去下载并阅读它们。可以把它看做是一所 Usenet 大学 —— 远程教育和学习的终极途径。

用 Gnus 向 Usenet 投稿

读过FAQ之后，有人可能还想往Usenet投稿。但考虑到Usenet的看客是那么的多，如果对是否有必要投稿没把握，那最好先忍一忍，或者请周围的人给文章把把关。Usenet 的看客实在是太多了，多加小心总是应该的。

给文章加上签名

许多用户会给自己的文章加上一个签名，本章前半部分已经介绍过给电子邮件加上签名的办法。签名保存在一个名为“*.signature*”的文件里。对电子邮件而言，“*.signature*”文件在默认情况下是不会被包括在其中的。但对Gnus来说，它们却会被默认地包括在稿件里。因此，如果在练习使用电子邮件时创建了一个可笑的“*.signature*”文件，千万要记得删除它或者对它进行修改，否则它就会被默认地夹在稿件里发送到 Usenet 上 —— 这可是我们的经验之谈。

在输入文章内容的时候，“*.signature*”文件是不会出现屏幕上的，但它实际上已经被追加在文章末尾 —— 如果从新闻组里调出自己张贴的文章，就会看到情况确实是这样的。如果不想让“*.signature*”文件被自动地包括进去，请把下面这条语句添加到“*.emacs*”文件里：

```
(setq gnus-signature-file nil)
```

张贴一篇新文章

往Usenet张贴一篇新文章的方法是按下“**a**”键（命令名是**gnus-group-post-news**）。如下所示:

按下: **a**

```
Buffers Files Tools Search Misc Groups Group Help
            64: news.groups.questions
            12: gnu.emacs.gnus
            39: gnu.emacs.help
             6: alt.religion.emacs
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L3--All-------
Are you sure you want to post to all of Usenet?(y or n)
```

Emacs 要求确认是否想向 Usenet 投稿。

回答“**y**”是指投出稿件，回答“**n**”是指取消它。如果回答的是“**y**”，Emacs 将提示给文章输入一个主题。如下所示:

按下: **y**

```
Buffers Files Tools Search Misc Groups Group Help
            64: news.groups.questions
            12: gnu.emacs.gnus
            39: gnu.emacs.help
             6: alt.religion.emacs
-- Gnus List of groups   {nntp:news.ora.com}   (Group)--L3--All-------
Subject:
```

Emacs 提示给文章输入一个主题。

输入: **Peace summit for vi and emacs users RETURN**

```
Buffers Files Tools Edit Search News Fields Help

Newsgroups: gnu.emacs.help
Subject: Peace summit for emacs and vi users
-text follows this line -

--**-Emacs: *post-news*    (News Reply Fill)--L2--All-----------------
```

可以开始写文章了。

请注意: Emacs 已经把“Newsgroups:(新闻组)”栏默认地设置好了，这一栏的作用有点像电子邮件中的“To:(收信人)”栏。经过考虑，这篇文章还是投给 *alt.*

religion.emacs 新闻组比较好一些 —— 它在那里可能会引起比较大的反响；因此，应该把新闻组那一栏改过来。

不要删除“-text follows this line-”这一行，它是不会出现在稿件里的。把文章内容写出来，或者，如果它已经在另外一个文件里，那么先按下“**C-x i**”组合键，再输入那个文件名，然后按回车键。

做好发送文章的准备工作之后，按下“**C-c C-c**”（命令名是 **news-inews**）组合键。Emacs 将给文章加上签名（如果有“*.signature*”文件）并把它张贴到 Usenet 上。

回应他人的文章

对一篇文章的回应有两种：一种是把回应张贴到Usenet，另一种是通过电子邮件直接与文章的作者进行交流。如果没有必要让全世界都知道，那么还是与文章的作者做一对一的交流比较好。直接回应文章作者的方法是按下“**r**”键（命令名是 **gnus-summary-reply**）。如果还想在自己的回应里引用那篇文章的原文，就需要按下“**R**”键。Emacs 将打开一个“mail”编辑缓冲区，写好回应后按“**C-c C-c**”就可以把它发送出去。

如果想把回应张贴到新闻组里，请按下“**f**”键（命令名是**gnus-summary-followup**）。Emacs会先问是否真的想把回应文章投给整个Usenet，然后它会打开一个编辑缓冲区，那里面已经把必要的地址信息都填写好了。如果还想在自己的回应里引用那篇文章的原文，就需要按下“**F**”键（命令名是 **gnus-summary-followup-with-original**）；Emacs 将会把原文插入到回应文章里。可能需要对原文进行一些删节，好让它只包括准备回应的要点。写好回应之后，请按下“**C-c C-c**”组合键把它发送出去。

撤回投稿

如果在投出稿件之后又改变了主意，就得赶快动手。撤回投稿的方法是在“Summary”编辑缓冲区里按下“**C**”键（命令名是**gnus-summary-cancel-article**）；Emacs将尝试撤回这篇稿件。能否成功地撤回这篇文章在很大程度上要看动作有多快。即使是立刻就进行了操作，也可能会有人读到这篇文章了。

表 6-12 对向 Usenet 投稿用的命令进行了汇总。

表 6-12: 投稿命令速查表

键盘操作	命令名称	动作
a *Post→Post an article*	**gnus-summary-post-news**	张贴一篇新文章
r	**gnus-summary-reply**	直接回复文章的作者
R *Post→Reply and yank*	**gnus-summary-reply-with-original**	直接回复文章的作者并附带一份原文的副本
f	**gnus-summary-followup**	把对当前文章的回应张贴到 Usenet 上
F *Post→* *Followup and yank*	**gnus-summary-followup-with-original**	把对当前文章的回应张贴到 Usenet 上并附带一份原文的副本
C-c C-f C-n	**news-reply-newsgroups**	移动到“News”编辑缓冲区的“Newsgroups:”栏；如果它不存在，就创建它
C-c C-f C-s	**mail-subject**	移动到“News”编辑缓冲区的“Subject:”栏；如果它不存在，就创建它
C-c C-f C-f	**news-reply-followup-to**	移动到“News”编辑缓冲区的“Followup-To:”栏；如果它不存在，就创建它
C-c C-f C-k	**news-reply-keywords**	移动到“News”编辑缓冲区的“Keywords:”栏；如果它不存在，就创建它
C-c C-f C-d	**news-reply-distribution**	移到“News”编辑缓冲区的“Distribution:”栏；如果它不存在，就创建它
C-c C-f C-a	**news-reply-summary**	移到“News”编辑缓冲区的“Summary:”栏；如果它不存在，就创建它
C-c C-y	**news-reply-yank-original**	在“News”编辑缓冲区里插入原始文章

表 6-12: 投稿命令速查表（续）

键盘操作	命令名称	动作
C-c C- q	**mail-fill-yanked-message**	对插入到“News”编辑缓冲区里的原始文章进行段落重排（以便它们有统一的文本行长度）
C-c C-r	**gnus-summary-caesar-message**	在“News”编辑缓冲区里对文章进行简单的加密（采用rot13算法）
C-c C-c	**news-inews**	把文章张贴到Usenet上（如果在“News”编辑缓冲区里）或把它作为电子邮件发出去（如果在“mail”编辑缓冲区里）
C	**gnus-summary-cancel-article**	在“Summary”编辑缓冲区里，撤回投稿

退出 Gnus

退出 Gnus 请按下“**q**”键；根据当时所处的位置情况，可能得按两次。如果处在“Summary”编辑缓冲区里，按一次“**q**”键将退回到“Group”编辑缓冲区里。如果想完全离开 Gnus，你还得在“Group”编辑缓冲区里再按一次“**q**”键。Emacs 将提问：

```
Are you sure you want to quit reading news? (y or n)
```

回答“**y**”将退出 Gnus。Emacs 将保存“*.newsrc*”文件，也就是把你这次的阅读情况记录下来。

电子邮件和新闻方面的基本操作到这里就介绍得差不多了。下一章将对诸如Telnet、FTP 和 WWW 之类的 Emacs 因特网组件进行讨论。

疑难解答

- **对电子邮件进行排序的时候，画面内容没有变化。**使用的可能是 Emacs 19 的一个早期版本。请按“**h**”键让 Emacs 刷新 RMAIL 的“Summary”画面。
- **Gnus不工作。**可能是系统没有连接到因特网，或者是没有本地新闻服务器。另一种可能是使用着 Emacs 一个比较早期的版本（比如 Emacs 18）——它们没有支持 Gnus 的功能。
- **Gnus 的工作情况与这里介绍的完全不一样。**使用的可能是 Gnus 4.x，它是 Emacs 19.29或更早的版本里配备的新闻组件。这一章里介绍的是Gnus 5.1（配备在 Emacs 19.30 或更高级的版本里）。

第七章

Emacs 的因特网工具箱

本章内容:

- Emacs 的 Telnet 模式
- Emacs 的 Ange-ftp 模式
- 用W3模式浏览Web主页

有许多工具可以用来访问因特网。事实上，这些工具的数量每天都在增加。那么，我们凭什么说 Emacs 是 Telnet、FTP 和 World Wide Web 等因特网应用方面一种值得拥有的工具呢？虽然我们一直在说它并不是最好的工具，但Emacs提供的集成性却无与伦比。如果你正使用着Netscape之类的图形化浏览器，我们并不打算让你放弃它们。可一旦你学会了如何通过Emacs来访问因特网资源，就肯定会把它添加到你的百宝箱里去。

即使不离开Emacs，也能telnet到另外一台计算机、浏览Web或者使用FTP功能下载并编辑文件。如果在网上发现了一些很不错的因特网资源，会不会把它的URL（uniform resource locator，统一资源定位器）地址发送给朋友？像这样的电子邮件你又收到过多少？可以把它们用笔和纸记下来，或者直接把它们存到某个文件里去。但更好的办法是去快速访问一下那个站点，如果它值得保留，就把它添加到自己的收藏夹里。Emacs 提供给我们的正是这些东西。

本章主要介绍几种因特网工具:

- Telnet 模式 —— 访问远程计算机

- Ange-ftp 模式 —— 用 FTP 协议（file transfer protocol、文件传输协议）传输文件
- W3 模式 —— 访问 World Wide Web

Telnet 模式和 ange-ftp 模式是 Emacs 的内置编辑模式，而 W3 模式则是一个需要用户自己从因特网上下载和安装的LISP软件包。有的站点上可能已经有人把它们下载下来了。如果还没有，请继续阅读，我们会在后面说明在因特网上的什么地方可以找到它。

本章将介绍一些用Emacs来访问因特网的基本操作。如果希望多学一些关于因特网自身的知识，那么《The Whole Internet User's Guide & Catalog》会是一个很好的开端，这本书作者是 Ed Krol，由 O'Reilly & Associates 出版公司出版。

Emacs 的 Telnet 模式

Telnet 提供一种在网上使用其他计算机的方法。在 Emacs 里，完全能用 Telnet 访问本地网络或者因特网上的任何一台主机并登录到该系统上 —— 简单地输入“**ESC x telnet**”命令即可。我们现在就尝试用 Telnet 访问一个文档服务器试试。（文档服务器能够帮助在因特网上寻找程序和文件。找到之后，再用 ange-ftp 模式把它们复制下来。具体操作将在后面介绍。）

输入：**ESC x telnet**

```
Buffers Files Tools Edit Search Minibuf Help
It was the best of times,it was the worst of times,it
was the age of wisdom,it was the age of foolishness,
it was the epoch of belief,it was the epoch
of incredulity,it was the season of Light,it was
the season of Darkness,it was the spring of hope,it
was the winter of despair,we had everything before us,
-----Emacs:dickens          (Text Fill)--L1--80%----------------------
Open telnet connection to host:
```

Emacs 询问想连接到哪台计算机上去。

输入：**archie.unl.edu**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
telnet>Trying 128.167.254.195 ...
Trying 129.93.1.14 ...
Connected to crcnis2.unl.edu.
Escape character is   ] .

SunOS UNIX (crcnis2)
login:

--**-Emacs:*archie.unl.edu-telnet*      (Telnet:run)--L8--All----------
```

文档服务器提示输入一个登录名。

输入：**archie RETURN**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
Trying 129.93.1.14 ...
Connected to crcnis2.unl.edu.
Escape character is   ] .

SunOS UNIX (crcnis2)

login:archie
Password:

--**-Emacs:*archie.unl.edu-telnet*     (Telnet:run)---L9----All--------
```

文档服务器继续提示输入一个口令。

这个文档服务器要求输入一个口令。许多服务器不要求输入口令，但这台需要（如果某个公共服务器要求输入一个口令——比如现在，那么请把电子邮件地址当做口令输入进去）。进入Telnet模式的这一步骤存在着这样一个问题（如果各位没有跳过本书前面对shell模式进行讨论的部分，就应该已经见过这个问题）：输入的口令会显示在屏幕上，而这是不安全的。如果想让Emacs把输入的口令隐藏起来——不管在Emacs里的什么地方——请把下面这两行语句添加到“*.emacs*”文件里：

```
(add-hook 'comint-output-filter-functions
          'comint-watch-for-password-prompt)
```

把这两行语句添加到“*.emacs*”文件里之后，保存该文件，退出Emacs，然后再重新进入。现在，只要屏幕上再出现要求输入口令提示信息，Emacs就会到辅助输入区里去输入口令。不管输入的是什么，Emacs都将显示为一个星号。在这个例子里，我们将把Joe Btsfphbl的电子邮件地址用做口令。如下所示：

输入：**joe@grom.com RETURN**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
        #   # # #       #     # #   #   #######       # #  #   #######
        #    ## #       #     # #    #  #     # #     # #   #  #     #
        #     # ####### ######  #     # #     #  #####  #    # #     #
     Welcome to the ARCHIE server at the University of Nebraska -Lincoln
    If you need further instructions,type help at the unl-archie>prompt.
##################################################################
# Bunyip Information Systems, 1993
# Terminal type 'emacs' is unknown to this system.
# 'erase' character is '^?'.
# 'search' (type string)has the value 'sub'.
unl-archie>
--**-Emacs:*archie.unl.edu-telnet*      (Telnet:run)--L34--Bot---------
```

已经登录到这台文档服务器上。

虽然这台文档服务器不知道“emacs”代表的是哪一种终端类型，但却允许继续工作。（如果遇到因无法把 Emacs 识别为一种终端类型，而不允许以 Telnet 模式登录上机的服务器，就只能到 shell 下去使用 Telnet 了。）现在，我们来查找 GNU 压缩程序 *gzip*。如下所示：

输入：**prog gzip**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
Host ftp.cc.rochester.edu     (128.151.224.6)
Last updated 09:10  8 Sep 1994
    Location: /pub/Solaris-2.1
       FILE   -rwxr-xr-x  139264 bytes  01:00 25 Mar 1993  gzip
Host primost.cs.wisc.edu     (128.105.36.61)
Last updated 11:06  5 Aug 1994
    Location: /bin
      FILE    ---x--x--x   90112 bytes  12:42 16 Mar 1994  gzip
unl-archie>
--**-Emacs: *archie.unl.edu-telnet*     (Telnet:run)--L87--Bot---------
```

稍等一会儿，文档服务器返回一份清单，里面列出可以找到 *gzip* 程序的各个地点。

既然清单已经显示在自己的屏幕上，那么就可以结束这次 Telnet 会话了。

输入：**bye**

```
Buffers Files Tools Edit Search Complete In/Out Signals Help
Host ftp.cc.rochester.edu    (128.151.224.6)
Last updated 09:10  8 Sep 1994
    Location:/pub/Solaris-2.1
      FILE -rwxr-xr-x 139264 bytes 01:00 25 Mar 1993 gzip
Host primost.cs.wisc.edu (128.105.36.61)
Last updated 11:06 5 Aug 1994
    Location:/bin
      FILE ---x--x--x 90112 bytes 12:42 16 Mar 1994 gzip
unl-archie>bye
#Bye.
Connection closed by foreign host.

--**-Emacs:*archie.unl.edu-telnet*    (Telnet:no process)--L102--Bot---
```

Emacs 关闭这个 Telnet 连接，但这次会话的输出内容仍留在屏幕上。

文档服务器的输出内容会留在这个编辑缓冲区里，既可以把它保存到一个文件里去，也可以把它留在那里。学会使用 ange-ftp 模式之后，甚至可以马上把 FTP 地址复制到辅助输入区里并下载那个文件。

Telnet 模式就是这么简单；不过还必须再学几个键盘命令。如果想发送一个“**C-c**”字符到 Telnet 服务器去，就必须按下“**C-c C-c**”。如果想发送一个“**C-z**”字符，就必须也给它加上一个“**C-c**”前缀。如果各位没有跳过 shell 模式部分的学习内容，就应该还记得与此类似的 shell 命令也必须加上“**C-c**”前缀。表 7-1 列出了 Telnet 模式下的各种命令。

表 7-1：Telnet 命令速查表

键盘操作	命令名称	动作
（无）	**telnet**	进入 Telnet 模式
C-d	**comint-delchar-or-maybe-eof**	根据上下文，发送 EOF 字符或删除光标位置下的字符
RETURN	**telnet-send-input**	处理 Telnet 输入
C-c C-c	**telnet-interrupt-subjob**	中断当前作业，相当于 shell 中的“**C-c**”组合键
C-c C-q	**send-process-next-char**	发送紧随其后的控制字符，相当于 shell 中的“**C-q**”组合键
C-c C-d	**comint-send-eof**	发送 EOF（文件尾）字符

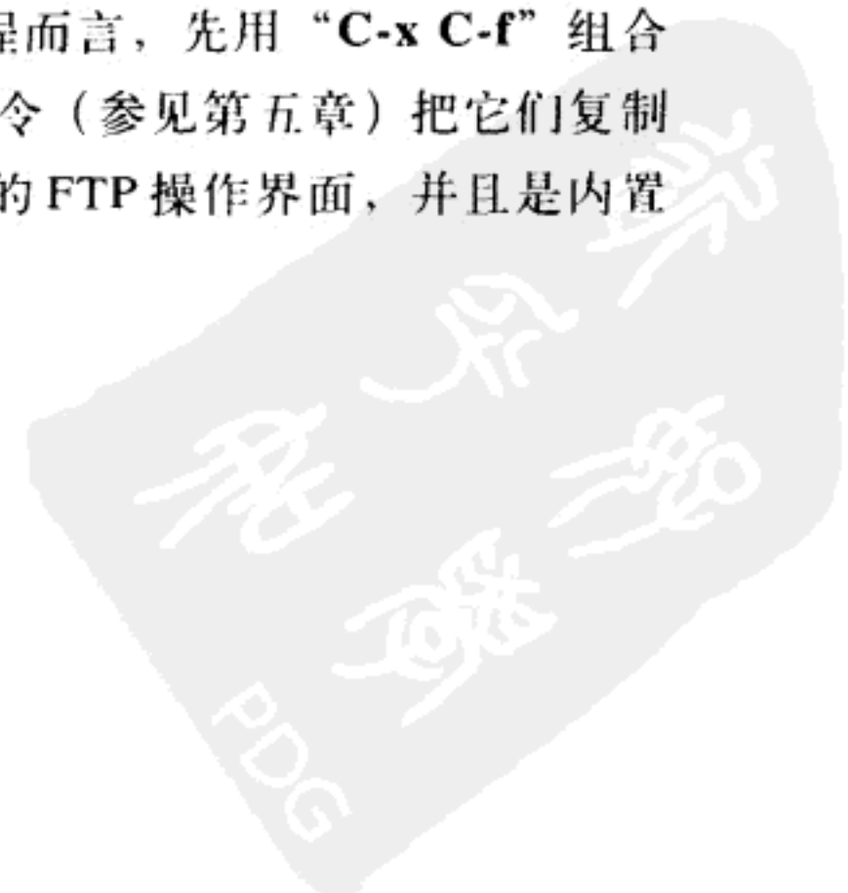
表7-1: Telnet命令速查表（续）

键盘操作	命令名称	动作
C-c C-r	**comint-show-output**	让输出内容的第一行显示在窗口的顶部
ESC C-l	**comint-show-output**	让输出内容的第一行显示在窗口的顶部
C-c C-e	**comint-show-maximum-output**	让输出内容的最后一行显示在窗口的底部
C-c C-o	**comint-kill-output**	删除上一个命令的输出内容
C-c C-z	**telnet-c-z**	挂起或暂停一个作业，相当于shell中的“**C-z**”组合键
C-c C-w	**backward-kill-word**	删除前一个单词
C-c C-u	**comint-kill-input**	删除当前行，相当于shell中的“**C-u**”组合键
ESC n	**comint-next-input**	查看此后输入的命令（重复时可查看到更靠后的命令）
ESC P	**comint-previous-input**	查看此前输入的命令（重复时可查看到更靠前的命令）

Emacs 的 Ange-ftp 模式

FTP是一种在因特网上或者在任何一个TCP/IP网络上挪动文件的标准办法。因特网上有无数“匿名的”FTP服务器允许从它们那里复制文件。可以先利用刚才介绍的档案服务器找出某个特定文件的保存地点，然后再用FTP把文件复制到本地服务器上。

Ange-ftp模式使得不用再学FTP命令了。就操作过程而言，先用“**C-x C-f**”组合键“找到”某远程系统上的文件，然后再用Dired命令（参见第五章）把它们复制下来即可（参见第五章）。Ange-ftp模式是一个透明的FTP操作界面，并且是内置在Emacs里的。



用 Ange-ftp 模式复制文件

从用户的角度看，ange-ftp 模式实际上是个增加了一点语法、扩展了一些功能的 **find-file** 命令（键盘操作仍是按下“**C-x C-f**”组合键）。那么，当你按下“**C-x C-f**”组合键时，Emacs 又是怎样知道你想启动的是 Ange-ftp 模式呢？如果同时满足下面 3 个条件，Emacs 就会启动 ange-ftp 模式：

1. 文件名以一个斜线字符（/）打头
2. 斜线字符后紧跟着 *username@systemname*
3. 系统名之后是一个冒号（:），然后是子目录名或文件名（如果有）。例如 */anonymous@rtfm.mit.edu:/pub*

比如，输入“*/anonymous@rtfm.mit.edu:/pub*”将打开一个到主机 *rtfm.mit.edu* 的 FTP 连接，并把 */pub* 子目录里的文件清单显示在屏幕上。注意：容易忘记打头的斜线字符，而系统名和文件路径名之间的冒号就更容易漏掉。如果 Emacs 提示：

```
New file
```

或者

```
Use M-x make-dir RET RET to create a directory
```

就得用“**C-x C-v**”组合键（命令名是 **find-alternate-file**）来纠正这个错误。当然也有可能是把文件名、子目录名或系统名拼写错了，请注意检查它们。另外，当访问某个因特网资源的时候，如果它被更新过或者发生了其他变化，你可以去掉路径名中的一两个元素再多试几次。比如，如果在“*/anonymous@rtfm.mit.edu:/pub/usenet-by-group*”位置上没有找到自己想要的东西，可以去试试“*/anonymous@rtfm.mit.edu:/pub/*”；如果这样还是不行，可以再去试试“*/anonymous@rtfm.mit.edu:*”。

要想下载本章前面提到的 *gzip* 程序，请输入“**C-xC-f/anonymous@ftp.cc.rochester.edu:/pub /Solaris-2.1/gzip**”命令。如果给出了文件名，Emacs 就将把那个文件检索出来；如果省略了文件名，Emacs 就会把远程子目录里的文件清单显示在屏幕上，可以看看里面有没有自己感兴趣的其他东西。

PDG

输入：**C-x C-f /anonymous@ftp.cc.rochester.edu:/pub /Solaris-2.1**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
 /anonymous@ftp.cc.rochester.edu:/pub/Solaris-2.1:
 total 1005
 drwxrwsr-x  4 0            512 Mar 25  1993 .
 drwxr-sr-x 20 0            512 Jun 14 20:54 ..
 drwxr-sr-x  6 1142         512 Mar 25  1993 Migration_Kit
 -rw-r r   1 1142          760 Mar 25  1993 README
 drwxr-sr-x  2 1142         512 Mar 25  1993 Solaris_Info
 -rwxr-xr-x  1 1142       139264 Mar 25  1993 gzip
 -rw-r r   1 1142       866103 Mar 25  1993 migration.tar.z
--%%-Dired:Solaris-2.1            (Dired by name)--L5--All--------------
Reading directory /anonymous@ftp.cc.rochester.edu:/pub/Solaris-2.1/
...done
```

Emacs 把远程系统上的子目录显示在屏幕上。

把光标移动到 *gzip* 条目上，然后按下：**C**

```
Buffers Files Tools Search Operate Mark Regexp Immediate Subdir Help
 /anonymous@ftp.cc.rochester.edu:/pub/Solaris-2.1:
 total 1005
 drwxrwsr-x  4 0            512 Mar 25  1993 .
 drwxr-sr-x 19 0           1024 Apr  5 11:25 ..
 drwxr-sr-x  6 1142         512 Mar 25  1993 Migration_Kit
 -rw-r r   1 1142          760 Mar 25  1993 README
 drwxr-sr-x  2 1142         512 Mar 25  1993 Solaris_Info
 -rwxr-xr-x  1 1142       139264 Mar 25  1993 gzip
 -rw-r r   1 1142       866103 Mar 25  1993 migration.tar.z
--%%-Dired:Solaris-2.1            (Dired by name)--L8--All--------------
Copy [-p] gzip to: /anonymous@ftp.cc.rochester.edu:/pub/Solaris-2.1/
```

Emacs 询问想把文件复制到何处；默认设置是复制到那个远程系统上。

Emacs 认为想把文件复制到远程系统上的某个地方。删除这个路径（先用“**C-a**”组合键移动到路径的开始，再用“**C-k**”组合键删除它），然后输入想把文件复制过去的路径。就这个例子来说，可以输入“*~/gzip*”。

文件复制：二进制格式与 ASCII 格式

FTP 用户常犯的错误之一是试图以 ASCII 格式来复制一个二进制文件。Ange-ftp 模式会尽量以正确的格式来下载文件 —— 如果它发现必须以二进制格式来进行下载，就会自动进行切换。它使用一个正则表达式来判断是否需要把文件作为一个二进制文件来进行传输（程序、图形和 PostScript 文件等都属于二进制文件）。Ange-ftp 会

把以“*.Z*”、“*.lzh*”、“*.arc*”、“*.zip*”、“*.zoo*”、“*.tar*”、“*.dvi*”、“*.ps*”、“*.elc*”、“*.gif*”和“*.EXE*”等后缀结尾的文件看做是二进制文件。这些扩展名在变量 **ange-ftp-binary-file-name-regexp** 的默认值里已经设置好。如果想增加其他的文件扩展名，就需要自己去修改这个正则表达式。

想由自己来直接发出FTP命令也可以，但必须先进入一个ange-ftp通常不显示的编辑缓冲区才行。每个FTP连接都有它自己的一个名为“`*ftp` *systemname*`*`”的编辑缓冲区，按下“**C-x C-b**”组合键就能在编辑缓冲区清单里看到此缓冲区。如果在另外一个窗口里把这个编辑缓冲区显示出来，就能观察到ange-ftp模式的工作情况。如果在shell提示符下使用FTP，就可以自己来发出FTP命令（比如切换到二进制模式的 **bin** 命令等）。

用 Ange-ftp 模式查找文本文件

FTP最普通的用途就是下载文件。在访问FTP站点的时候，我们经常会看到一些名为“*README*”或者“*Index*”的文件。如果想查看这些文件的内容，在命令行方式的FTP中是很麻烦的。要先把这些文本文件下载回来，关闭FTP连接，用 **more** 命令或者Emacs阅读这些文本文件的内容，然后再启动FTP连接到刚才的地方，才能下载自己想要的文件。

但在ange-ftp模式里，这一切只需一个操作步骤就能完成——只要像对待任何其他文件那样用ange-ftp模式找到并打开*README*文件即可。Emacs将把它显示在屏幕上。下面来看一个查找并打开某个文本文件的例子，准备查找的是*gnu.emacs.help*新闻组的FAQ文件（注1）。

输入：**C-x C-f /anonymous@rtfm.mit.edu:/pub /usenet-by-hierarchy/gnu/emacs/help/GNU_Emacs_Frequently_Asked_Questions_(FAQ),_part_1_5**（注2）

注1：我们当然知道现在可以通过“**Help**”菜单来查阅这份FAQ文件的某个版本，我们只是用它来举个例子。再说，这个地址上的版本总是最新的。

注2：就这个例子来说，其实用不着输入这么长的文件名。可以先打开它的子目录，移动到这个文件，然后利用“**f**”命令打开它。

```
Buffers Files Tools Edit Search Help
Path: senator-bedfellow.mit.edu!the-tech!reuven
From: Reuven M.Lerner <reuven@the-tech.mit.edu>
Newsgroups: gnu.emacs.help,comp.emacs,comp.answers,news.answers
Subject: GNU Emacs Frequently Asked Questions (FAQ), part 1/5
Supersedes: <GNU-Emacs-FAQ-1_802998241@the-tech.mit.edu>
Followup-To: gnu.emacs.help
Date: 12 Jun 1995 23:02:50 GMT
Summary: Questions and answers having to do with GNU Emacs
                    GNU Emacs FAQ: Introduction

Emacs:GNU_Emacs_Frequently_Asked_Questions_(FAQ),_part_1_5   (Text) Top
```

Emacs 检索出这个文件并把它显示在屏幕上。

如果想保留一份这个文件的副本，则可以用“**C-x C-w**”组合键加文件名的方法把它写到一个本地文件里。如果只是随便看看，并不想留一份副本，就没有必要把它存到本地系统上。给它设置一个书签（参见第四章中的讨论内容），以后再需要查阅它时，只需用一个简单的命令就能再次找到并打开它了，磁盘空间可以留给更重要的东西（比如从 Internet Talk Radio 网站下载的一个很大的音乐文件）。

用 W3 模式浏览 Web 主页

如果有William Perry编写的基于Emacs的浏览器W3（注3），那么即便是浏览World Wide Web，也不需要离开Emacs。（它不是Emacs软件的默认配备，但要想下载到它也非常容易，稍后再对此做详细介绍。）W3的最新版本现在还没有图形化操作界面（但听说它正在朝这方面努力）。与Emacs的其他功能组件一样，我们看中的是W3与其他工作的集成性。假设收到的电子邮件里有一个指向某个Web站点，或FTP站点的URL地址，于是想去那里看一看（注4）。移动到该URL地址处并输入“**ESC**

注 3：在本书付印期间，有消息说W3将改名为Gnuscape。与其他优秀的Web浏览器一样，W3也在不断地改进，所以W3现在的功能很有可能已经超越这本书里介绍的情况了。

注 4：URL地址的格式是“*protocol: //resourcename*”。比如，Gnu Bulletin的URL地址就是“*http: //www.cs.pdx.edu/~trent/ gnu/gnu.html*”。W3的FTP站点的URL地址是“*ftp: //cs.indiana.edu:/pub/elisp/w3*”。任何想让W3去识别的因特网地址都必须是这种格式，因此必须把“*ftp.microsoft.com*”写成“*ftp: //ftp.microsoft.com*”。而如果你遇到的是像“*URL: //http: //www.openmarket.com*”这样的URL地址时，则又必须把最前面的“*URL: //*”去掉。

x w3-follow-URL-at-point”，就会转到那个站点上去。如果对那个站点上的东西感兴趣，则可以把该站点添加到收藏夹里，或者用“**C-x C-w**”组合键把该主页上的内容保存到一个文件里去。Emacs 在保存这类文件时会先去掉主页上的 HTML（Hypertext Markup Language，超文本标记语言）语言标记。

也可以在不退出 Emacs 的情况下浏览自己编写的 HTML 文档，输入“**ESC x w3-open-local RETURN filename RETURN**”命令即可。Emacs会用W3把这个HTML文件显示在屏幕上。（第九章将介绍如何利用 html-helper 模式编写 HTML 文档。）

有了W3，基本上用不着再自己去输入URL地址了。可以把URL地址复制到删除环（kill ring，见第二章）、粘贴它们插入到编辑缓冲区里，把它们添加到收藏夹里，或者去浏览它们，而这一切都不需要输入工作。此外，想下载URL地址指示的资源时，还可以在辅助输入区里按下 **TAB** 键，以利用 Emacs 的自动补足功能。如果想看看某个给定的站点上都有什么好东西，这不失为一个好方法。

W3 支持以下几种 URL 地址类型:

- Usenet 新闻
- HTTP .9 或 HTTP 1.0
- Gopher 或 Gopher+
- FTP
- 本地文件
- Telnet
- TN3270
- 电子邮件

虽然，在大多数情况下 —— 至少是我们写这本书的时候，W3 还不能显示多媒体对象，但用户可以对 W3 进行设置，让它在选中一个多媒体超链接时，启动其他的浏览器进行浏览。比如，X 窗口系统的用户就能指定各种各样的外部程序，以便让 Emacs 显示图形。

让 W3 成为启动工作的一部分

如果还没有安装W3，那么可以从“*ftp: //cs.indiana.edu:/pub/elisp/w3*”处下载。如果想让它成为Emacs启动工作的一部分，请把下面这些语句添加到“*.emacs*”文件里，别忘了把第一行里的“FULLPATHNAME FOR W3”替换为真正的路径名。

```
(setq load-path (cons "FULLPATHNAME FOR W3" load-path))
(autoload 'w3-preview-this-buffer "w3" "WWW Previewer" t)
(autoload 'w3-follow-url-at-point "w3" "Find document at pt" t)
(autoload 'w3 "w3" "WWW Browser" t)
(autoload 'w3-open-local "w3" "Open local file for WWW browsing" t)
(autoload 'w3-fetch "w3" "Open remote file for WWW browsing" t)
(autoload 'w3-use-hotlist "w3" "Use shortcuts to view WWW docs" t)
(autoload 'gopher-dispatch-object "gopher" "Fetch gopher dir" t)
```

默认启始主页

输入“**ESC x w3 RETURN**”命令就能启动W3。状态行上将出现“**WWW**”字样，而W3将会把默认的启始主页“*http://www.cs.indiana.edu/elisp/w3/docs.html*”（注5）加载到屏幕上。

输入：**ESC x w3 RETURN**

```
Buffers Files Tools Edit Search Help
 The Emacs World Wide Web Browser
-----------------------------------------------------------------------
[CAUTION] [[Current version is 2.2.0.]] If you are using an earlier
version, please upgrade now. The latest version is [[available from
our anonymous ftp site]]
-----------------------------------------------------------------------
W3 is an Emacs subsystem that allows the user to browse the wonderful
World Wide Web. Like most packages written for Emacs, it is covered
by the [[GNU copyleft]]. The World Wide Web was started at the CERN
--%%-Emacs:The Emacs World Wide Web Browser      (WWW)--L1--Top--------
```

Emacs进入W3模式并加载默认的启始主页。

按**SPACE**键向下翻页，按**DEL**键向上翻页。如果想移动到页面的开始，按“<”键，

注5：　如果想另外指定默认启始主页，请把下面这条语句添加到“.emacs”文件里：

```
(setq w3-default-homepage "http:// your URL here")
```

比如，如果想用Global Network Navigator做为主页，就得用下面这条语句进行设置：

```
(setq w3-default-homepage 'http://www.gnn.com/')
```

如果想移动到页面的结尾，按“>”键。如果想按超链接在页面中做跳跃式前进而不是一页一页地翻阅，请按“**f**”或**TAB**制表键（命令名是**w3-forward-link**），Emacs将前进到下一个超链接处，并把它的URL地址显示在辅助输入区里。再按**TAB**制表键则Emacs将前进到再下一个链接处。按下“**b**”键将使Emacs返回到上一个链接。在很多情况下，按照超链接移动要比逐页翻卷有用得多。

在WWW页面之间移动

可以用输入URL地址的方法来漫游WWW，但更典型的方法是选择超链接，这些超链接可能有各种各样的显示手段，比如呈高亮反显状态等。在下面给出的画面里，超链接显示在双方括号里。要想跟随超链接在WWW页面之间进行转移，需要把光标移动到超链接上再按下回车键。如下所示：

按下：**TAB TAB TAB**

```
Buffers Files Tools Edit Search Help
The Emacs World Wide Web Browser
-----------------------------------------------------------------------
[CAUTION] [[Current version is 2.2.0.]] If you are using an earlier
version, please upgrade now. The latest version is [[available from
our anonymous ftp site]]
-----------------------------------------------------------------------
W3 is an Emacs subsystem that allows the user to browse the wonderful
World Wide Web. Like most packages written for Emacs, it is covered
by the [[GNU copyleft]]. The World Wide Web was started
--%%-Emacs:The Emacs World Wide Web Browser      (WWW)--L1--Top---------
```

Emacs前进到“GNU copyleft”超链接处。

按下：**RETURN**

```
Buffers Files Tools Edit Search Help
********
PREAMBLE
********
The license agreements of most software companies try to keep users at
the mercy of those companies. By contrast, our General Public License is
intended to guarantee your freedom to share and change free software---
to make sure the software is free for all its users. The General Public
License applies to the Free Software Foundation s software and to any
other program whose authors commit to using it. You can use it for your
programs, too.
--%%-Emacs:GNU Copyleft Preamble      (WWW)--L1--Top-------------------
```

Emacs进入“GNU Copyleft”Web主页。

那么，当转到另外一个主页之后，怎样才能再返回来呢？按“**l**”（小写的字母“L”代表“上一个”）键或者直接切换编辑缓冲区。W3会把每一个页面分别放在它们各自的编辑缓冲区里，用“**C-x b**”组合键就能在它们之间进行切换。按下“**C-x C-b**”组合键将调出编辑缓冲区清单，里面会把访问过的所有主页都列出来——其他的编辑缓冲区也包括在内。在某个主页的显示画面里按“**q**”键将删除那个主页，用来删除编辑缓冲区的“**C-x k**”组合键也能完成同样的事情。

除用来对Emacs编辑缓冲区进行操作的命令以外，还可以使用历史记录进行操作。历史记录不仅记录了这次上网所访问过的地方，此前的W3会话中访问过的站点也会被记在里面。在默认情况下，W3不保存历史记录。如果想让它这样做，就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq url-keep-history t)
```

Emacs会把访问过的所有URL地址都添加到一个history（历史记录）编辑缓冲区里。如果想返回到前一页，请按下“**B**”键（命令名是**w3-backward-in-history**）；如果想前进到后一页，请按下“**F**”键。要想查看本次会话的历史记录，请按下“**C-c C-b**”（命令名是**w3-show-history-list**）组合键；Emacs将以超链接的形式把这次会话的历史记录清单列出来，如果想访问其中的某个链接，把光标移动到那儿再按下回车键即可。退出的时候，Emacs将把这次会话的历史记录以追加方式写到一个名为“*.mosaic-global-history*”的文件里。（根据个人兴趣和磁盘空间的情况，有的用户可能不想让开关变量**url-keep-history**总是处于“开”的状态。可以定期删除这个文件或者从“*.emacs*”文件里删除这条语句。）

当返回到前一页的时候，会发现刚访问过的那个超链接的显示情况又发生了变化。括在访问过的超链接两端的将不再是方括号，而是花括号就像“{{GNU copyleft}}”这样。括号方面的这种变化能帮助回忆已经访问过哪些链接。

W3还有一个很有用的功能，就是它不仅能记住已经访问过哪些主页，还能记住在这些主页上的位置，就像Emacs能够记住在各个编辑缓冲区里的编辑位置一样。因此，如果在阅读某个主页时去其他地方转了一圈，再利用历史记录功能回到这个主页上时还能接着刚才的地方继续往下看。

用 W3 模式查找 URL 地址

W3最让人喜欢的功能是它允许在Emacs里随时去访问任何一个URL地址。假设你在阅读自己用Emacs的ange-ftp模式取回来的Gnu Emacs FAQ时，看到了一个URL并且想到那里去看看，那么你根本不需要退出Emacs去启动一个浏览器。只要把光标移动到那个URL地址处，然后再输入“**ESC x w3-follow-URL-at-point RETURN**”命令就能让Emacs把该URL指示的资源取回来。（如果你经常使用W3，就应该考虑把这个命令绑定到键盘的某个按键上。具体步骤请参考第十一章内容。）

初始化状态：

```
Buffers Files Tools Edit Search Help
 A collection of past GNU s Bulletins (as well as other GNU-related
 information) is available at

   http://www.cs.pdx.edu/ trent/gnu/gnu.html

21:Where can I get help in installing Emacs?

  See question 81 for some basic installation hints, and question 83 if
  you have problems with the installation.

 Emacs: GNU_Emacs_Frequently_Asked_Questions_(FAQ),_part_1_5      (Text)
```

在阅读 *gnu.emacs.help* 新闻组的FAQ时，看到了一个有意思的URL。

把光标移动到那个URL地址处并输入：**ESC x w3-follow-URL-at-point RETURN**

```
Buffers Files Tools Edit Search Help
***************
[IMAGE(gnu.xbm)] GNU S NOT UNIX!
***************

This is an unofficial archive of GNU information. It is mainly a
collection of information that I have collected over the years. Much
of it is either  historical  information or information that FSF does
not keep in the software archives.

        o [[GNU s Bulletins]]. Or you can skip the index and get the
          [[latest]] edition.
        o [[GNU HURD information]]
        o RMS  paper on [[Why you should not use TCL]]
--%%-Emacs:GNU s Not Unix!        (WWW)--L1--Top--------------------------
```

Emacs将自动启动W3，并把该URL指示的资源取回来。

这种取回URL所指资源的功能可以随时使用。在阅读文件或者电子邮件的时候，我们经常会遇到一些有意思的因特网地址。有了W3，只需把光标移过去，再输入“**ESC x w3-follow-URL-at-point RETURN**”命令就可以满足自己的好奇心了。如果你经常上网，就会发现几乎遍地都是URL；如果先得记下它们、过后才能进行访问，就未免太麻烦了。有了W3，随时都能到其他站点上去溜达一圈，等转够了再回到手里的工作上来。

我们在第六章介绍了Emacs的新闻阅读器Gnus，在它里面取URL所指资源就更容易了。只要把光标移动到文章里的某个URL地址上，再按下**RETURN**回车键，Gnus就会启动W3把那个URL指示的资源给取回来。

收藏夹的使用

收藏夹是想多次访问的URL地址，为了能够方便今后的使用，它们都被保存为超链接的形式。在默认情况下，Emacs将使用Mosaic的收藏夹，它的文件名是“*~/.mosaic-hotlist-default*”。如果想导入Netscape的收藏夹，请输入“**ESC x w3-import-netscape-bookmarks**”（注6）命令。Emacs会复制那个收藏夹文件，把它转换为W3格式，然后把它保存为现有收藏夹的一部分。

请注意：收藏夹与Emacs书签是不一样的。Emacs书签可以设置在普通文件或者远程文件里（通过FTP），只需用一个简单的操作就能到达文件中书签的特定位置。收藏夹虽然在效果上与此类似，但它们只能工作在W3模式下，并且必须指向URL地址。

要想把当前浏览页面的URL添加到收藏夹里，请按下“**a**”键（命令名是**w3-hotlist-add-document**），W3将把该URL保存到收藏夹里，而它的名字就是当前主页的标题名称。如果不喜欢这个名字，可以用“**ESC x w3-rename-hotlist-entry**”命令给它另外取个名字。

在没有取回某个URL的情况下也能把它添加到收藏夹里去。只要移动到该URL地址上，再输入“**ESC x w3-hotlist-add-document-at-point**”命令即可。（如果已经在W3模式下，按“**A**”键也能完成同样的事情。）

注6：　如果使用的是W3的一个早期版本，这个命令可能不管用。但可以把HTML文件“*.netscape-bookmarks.html*”当做一个本地的Web主页来打开。

要想访问收藏夹里的某个站点，请按下“**h**”键（命令名是**w3-use-hotlist**）。Emacs将提示给出想访问的收藏夹条目的名称。可以在输入该条目的第一个或者前几个字符之后按**TAB**键，Emacs的自动补足功能将把该名称的剩余部分填写出来。

要想查看收藏夹的内容，请按下“**H**”键（命令名是**w3-show-hotlist**）。Emacs会把收藏夹中各个主页的标题列出来。当移动到某个标题并按下回车键时，Emacs就会把那个主页取回来。

W3与多媒体对象

根据系统的软、硬件配置情况，有的用户也许能够用W3浏览多媒体数据对象。如果在某个图片链接上按下“**ESC RETURN**”组合键（命令名是**w3-follow-inlined-image**），Emacs将会尝试启动系统上的某个看图器程序。Emacs会检查系统里是否有一个把该图形数据对象，与某个能对其进行正确处理的程序关联起来的定义。Emacs先检查环境变量**MIMETYPES**是否存在，如果存在，就使用该变量所做的定义；如果不存在，Emacs将再去检查主目录里是否有一个名为“*~/.mime-types*”的文件。如果这个文件也不存在，Emacs将依次检查以下几个系统文件是否存在，它们是“*/etc/mime-types*”文件、“*/usr/etc/mime-types*”文件、“*/usr/local/mime-types*”文件以及“*/usr/local/www/conf/mime-types*”文件。

查看URL

许多浏览器——例如Mosaic及其同系列产品，会把当前主页的URL显示在屏幕顶部的某个地方。可W3却不一定会把URL显示在屏幕画面里。如果想查看自己正在浏览的主页的URL，请按下“**v**”键（命令名是**url-view-url**）。如果想查看某个给定超链接的URL，就需要移动到该超链接上再按下“**V**”键（命令名是**w3-view-this-url**）。Emacs将把该URL地址显示在辅助输入区里。

复制URL

自从出现了WWW，人们不再需要先学习一大堆关于因特网的知识，就能去自己感兴趣的地方进行访问。但这也给Web带来这样一个缺陷：URL地址必须输入正确。为了避免不必要的错误，最好的办法是尽可能不亲自去输入它们。

我们已经介绍过一个可以避免亲手输入 URL 的方法：移动到 URL 地址上再输入“**ESC x w3-follow-URL-at-point**”命令。可如果想把 URL 地址发送给自己的朋友该怎么办呢？或者，如果遇到需要在自己设计的主页里插入一个指向某URL的交叉引用的情况时又该怎么办呢？方法有两种：如果想把当前主页的URL地址保存到删除环（kill ring，参见第二章）里，请按下“**k**”键或“**C-k**”组合键（命令名是 **w3-save-url**）；如果是想把光标位置上的 URL 地址复制到删除环里，请按下“**K**”键（命令名是 **w3-save-this-url**）。粘贴 URL 地址的键盘操作是按下“**C-y**”组合键。

按下“**ESC TAB**”组合键（命令名是 **w3-insert-this-url**）可以节省一个步骤。Emacs 要求指定一个用来放置该 URL 的编辑缓冲区，然后把该 URL 地址插到该编辑缓冲区的光标位置处。

查看页面的 HTML 源代码

大部分 WWW 主页都是 HTML 文档。在编写 HTML 文档时，如果能够以其他各类主页的源代码做参考，借鉴他人安排 HTML 元素的经验，那么肯定有很大的帮助。如果想查看某个主页的 HTML 源代码，请按下“**ESC s**”或“**s**”键（命令名是 **w3-source-document**）。如果想查看某个URL所指主页的源代码，就得移动到那个URL地址上再按下“**S**”键（命令名是 **w3-source-document-at-point**）。Emacs将把HTML源代码文件显示在屏幕上。

打开一个本地文件

如果想检查自己编写的 HTML 文件是否正确，可以在 Emacs 的 W3 模式里打开它。在 W3 模式里打开一个本地文件的方法是按下“**o**”键（命令名是 **w3-open-local**）；Emacs 将提示输入文件名。输入文件名再按下回车键，W3 就将对该 HTML 文件进行检查，它会把其中的错误报告出来，并且会把该页面显示在屏幕上。

W3 模式下的文件存盘操作

在浏览 World Wide Web 的时候，可能会遇到一些想要保存起来的页面（你会发现值得保存的页面实在是太多了，不见得有足够的磁盘空间把它们都保存或记录下来 —— 不过这是另外一个问题了）。如果想把文件保存为不带 HTML 标记的形式

（以便在 W3 以外也能顺利地阅读它们），请输入“**C-x C-w filename RETURN**”命令。Emacs 将把文件保存为普通的 ASCII 文本，把其中的 HTML 标记（以及各种超链接）都去掉。

如果想在保存文件时保留 HTML 标记，需要先按下“**s**”键查看它的源代码，然后再输入“**C-x C-w filename RETURN**”命令保存它。Emacs 会认为要把文件写到主目录里，但你可以根据自己的具体情况对路径进行修改。

表 7-2 对 W3 模式下的各种操作命令进行了汇总。

表 7-2：W3 命令速查表

键盘操作	命令名称	动作
（无）	**w3-follow-URL-at-point**	取回光标位置处的 URL 并进入 W3（在 Emacs 的任何组件里都能使用这个命令进入 W3）
（无）	w3	进入 W3，并把默认起始页面显示在屏幕上
C-o	**w3-fetch**	提示输入一个 URL，然后把它取回来
SPACE	**scroll-up**	向后卷动屏显内容
DEL	**scroll-down**	向前卷动屏显内容
RETURN	**w3-follow-link**	选中一个超链接
<	**w3-start-of-document**	前进到当前页的最开始
>	**w3-end-of-document**	前进到当前页的最末尾
TAB 或 n	**w3-forward-link**	移动到下一个超链接
b	**w3-back-link**	移动到前一个超链接
m	**w3-complete-link**	提示输入一个要移动到的超链接
I	**w3-goto-last-buffer**	前进到访问过的最后页
B	**w3-backward-in-history**[a]	退回到上一页
F	**w3-forward-in-history**	前进到历史记录里的下一页
C-c C-b	**w3-show-history-list**	把历史记录显示为超链接

表 7-2: W3 命令速查表（续）

键盘操作	命令名称	动作
a	**w3-hotlist-add-document**	把当前页面添加到收藏夹里
A	**w3-hotlist-add-document-at-point**	把光标位置上的 URL 添加到收藏夹里
d	**w3-hotlist-delete**	提示指定一个收藏夹条目以删除它
H	**w3-show-hotlist**	把收藏夹显示为超文本
h	**w3-use-hotlist**	提示指定一个收藏夹条目以显示它
（无）	**w3-rename-hotlist-entry**	提示指定一个收藏夹条目以重新命名
ESC RETURN	**w3-follow-inlined-image**	尝试显示插入点处的链接图片
v	**url-view-url**	显示当前页的 URL 地址
V	**w3-view-this-url**	显示光标位置处超链接的 URL 地址
k 或 **C-k**	**w3-save-url**	把当前页的 URL 地址复制到删除环里
K	**w3-save-this-url**	把光标位置处的 URL 地址复制到删除环里
s	**w3-source-document**	显示这一页的 HTML 源代码
S	**w3-source-document-at-point**	显示光标位置处 URL 的 HTML 源代码
o	**w3-open-local**	在 W3 中打开一个本地 HTML 文件
U	**w3-use-links**	前进到下一个 <LINK> 标记处
?	**w3-help**	查阅 W3 的在线手册
Q 或 **u**	**w3-leave-buffer**	退出 W3，但不删除当前编辑缓冲区
q	**w3-quit**	退出 W3，并删除当前编辑缓冲区
g 或 **r**	**w3-reload-document**	删除编辑缓冲区，然后再重新取回这一页
l	**w3-goto-last-buffer**	前进到刚才访问过的前一个编辑缓冲区

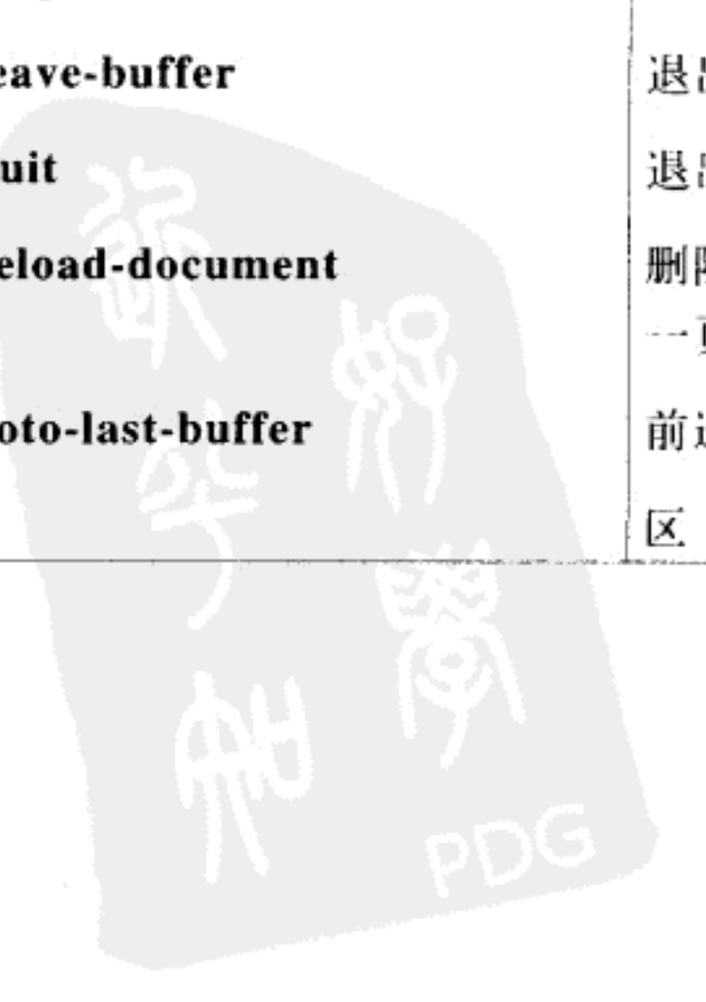


表 7-2: W3 命令速查表（续）

键盘操作	命令名称	动作
P	**w3-print-url-under-point**	以 HTML 或 LaTeX 格式把光标位置处 URL 指示的主页打印出来
p	**w3-print-this-url**	以 HTML 或 LaTeX 格式把当前主页打印出来
w	**w3-submit-bug**	提交一份在 W3 里发现的程序漏洞报告
C-c C-v	**w3-version**	显示 W3 的版本号
ESC s	**w3-search**	查找（如果这是一个可查找数据项）。用 Emacs 的查找命令进行本地查找
ESC m	**w3-mail-current-document**	把这个主页以电子邮件方式发送给某人，具体格式由用户自己选定
ESC M	**w3-mail-document-under-point**	取回光标位置处 URL 指示的主页并把它发送给某人
ESC TAB	**w3-insert-this-url**	提示指定一个用来放置此 URL 的编辑缓冲区；如果带有“**C-u**”前缀，就把 URL 插入到光标位置处
（无）	**w3-import-netscape-bookmarks**	导入Netscape的收藏夹文件并把它转换为W3格式（原始文件不会被修改）

a. 只有在变量 **url-keep-history** 被设置为“t”时，与历史记录有关的命令才能用。要想对该变量做它永久性的设置，请把下面这条语句添加到“*.emacs*”文件里：

```
(setq url-keep-history t)
```

疑难解答

- **Telnet**、**Ange-ftp** 模式和 **W3** 不工作。造成这种问题的原因可能是计算机没有连接在一个 TCP/IP 网络上。如果情况确是如此，任何 TCP/IP 功能都将无法使用。第二种可能是 Emacs 在编译时没有把 TCP/IP 支持功能包括在内。请向系统管理员做进一步查询。顺便说一句，Telnet 和 Ange-ftp 模式是 Emacs 自带的功能组件，而 W3 则必须安装为一个 LISP 增值软件包。

本章内容:

- 文本的缩进
- 文本的居中
- 插入分页符
- 矩形编辑
- 绘制简单的图形
- Emacs 的大纲模式

第八章

简单的文字排版和特效编辑

Emacs在本质上是文件编辑器而不是字处理器。它是一种用来创建文件以容纳屏幕上显示内容的工具，而不是一种用来美化文本文件打印效果的工具。但Emacs仍具备以下几种排版方面的功能:

- 以各种方式对文本进行缩进。
- 使单词、文本行和文本段落居中。
- 绘制简单的框图、草图和图片，并且能够把它们插入文件或者电子邮件（其他字处理器和图形工具包大都不具备这项功能）。
- 按列而不是按行来进行编辑（特别适合用来创建和修改表格）。在Emacs里，这项功能被称为矩形编辑（rectangle editing）。
- 用大纲模式（outline mode）隐藏和显示文档的某些部分，以便能够对文档的整体布局进行安排。大纲模式能够简化按大标题、小标题、草稿和正式稿的步骤完成文件的起草和编辑的工作。

除了以上这些将在本章介绍的功能以外，如果显示器还具备显示不同字体和颜色的能力，那么Emacs也允许在编辑缓冲区里使用它们。关于这方面的详细讨论请参考

第十四章。Emacs 还为 **troff**、TEX和 Scribe 等排版程序准备了专用的编辑模式，这些编辑模式将在第九章里介绍。

文本的缩进

在编写大纲、正式文件、技术文件或者其他任何东西的时候，经常需要对描写文本进行缩进处理。Emacs 为此准备了好几种解决方法。我们将从制表位讲起——它是最简单的文本缩进方法。然后我们将介绍缩进前导字符串（fill prefix），它既可以用来缩进文本，也可以用来给每行文本都增加一个前导字符串（特别适用于非正式文档中的注解或者程序中的注释）。Emacs里最精巧的文本缩进工具是文本缩进模式（indented text mode）。如果文档里有各种各样的缩进层次，那么用它来写是最合适不过的了。

制表位的使用

在Emacs里，**TAB**键的使用情况与各位预期的情况相当吻合，即把光标移动到下一个制表位处。光标后的文本也将向文件尾方向移动，文本就像是被推向了右方，结果是在这一行留出了一些空格——即使处在改写模式（overwrite mode）（注1）也会如此。比如，如果想缩进某个段落的第一行，就可以先把段落的内容输入进去，然后移动到它的第一行按下 **TAB** 键。如下所示：

按下：**TAB**

```
Buffers Files Tools Edit Search Help
    It was the best of times, it was the worst of times, it was the age of
wisdom, it was the age of foolishness, it was the epoch of belief, it
was the epoch of incredulity, it was the season of Light, it was the
season of Darkness, it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we
--**-Emacs: dickens        (Text Fill)--L1--All----------------------
```

TAB 键缩进的段落的第一行，其他文本也做相应的移动。

注1：　之所以要强调这一点，是因为某些编辑器中的制表位与它在打字机上的工作情况完全一样：按下 **TAB** 键将把光标向后移，但不改变文本的排列情况。

如果想手动缩进整个段落，当然没有问题：在每一行的开头按**TAB**键就能做到。不过Emacs给我们准备了一些简单点的法子，我们稍后再对它们进行介绍。现在，还是假设将手动完成缩进排版工作。

缩进整个段落

可以用在每行开头按**TAB**键的方法，来对某个段落做整体上的缩进。如下所示：

在段落中每一行的开始按下**TAB**键：

```
Buffers Files Tools Edit Search Help
    It was the best of times, it was the worst of times, it was the age of
    wisdom, it was the age of foolishness, it was the epoch of belief, it
    was the epoch of incredulity, it was the season of Light, it was the
    season of Darkness, it was the spring of hope, it was the winter of
    despair, we had everything before us, we had nothing before us, we
--**-Emacs: dickens        (Text Fill)--L9--All----------------------
```

整个段落都被缩进。

但对段落进行这样的编辑之后，它会变得参差不齐，还能用“**ESC q**”（命令名是**fill-paragraph**）组合键对它进行段落重排吗？如果只对段落的第一行进行了缩进，这个命令就还能用——它会重新安排文本，在它们和右页边之间留出一个适当的间隔。它不会改变原来的缩进设置，也不会影响到此前的段落，第一行文本前面的缩进让Emacs明白这里开始了一个新的段落。

可如果对整个段落进行了缩进，“**ESC q**”命令就不奏效了。由于它们的前面都有一些空格，所以Emacs会把每一行文本当做是彼此独立的段落。要想对整体缩进了的段落进行重排，必须先把这个段落定义为一个文本块，然后输入“**ESC x fill-individual-paragraphs RETURN**”命令。**fill-individual-paragraphs**命令作用于一个文本块而不是一个单纯的段落，它会根据缩进情况重新安排文本块的段落（缩进程度相同的文本行将组成一个段落，缩进程度不同的文本行将被划分为不同的段落）；然后再在保留其缩进设置的前提下，对文本块中的各个段落进行重排。请看下面这个屏幕画面里的文本。

初始状态:

```
Buffers Files Tools Edit Search Help
My second correspondent, also a woman, sends me the following
statement:
     Life seemed difficult to me at one time.I was always breaking
     down, and had several attacks of what is called nervous
     prostration, with terrible insomnia...
Here is another case, more concrete, also that of a woman. I read you
these cases without comment -they express so many varieties of the
state of mind we are studying.
-----Emacs: hjames          (Text Fill)--L9--All----------------------
```

上图中的文本由3个段落组成，因为其中有3个缩进程度不同的区块。如果需要对这类文本进行段落重排，就需要先在它的一端（用“**C-@**”组合键）设置好文本块标记，然后移动到它的另外一端输入“**ESC x fill-individual-paragraphs RETURN**”命令。

设置制表位

在默认情况下，制表位之间的间隔是8个字符。但Emacs允许改变制表位的位置。要想改变制表位，请输入“**ESC x edit-tab-stops**”命令。屏幕上将出现一个“*Tab Stops*”编辑缓冲区。如下所示:

输入: **ESC x edit-tab-stops**

```
Buffers Files Tools Edit Search Help
      :       :       :       :       :       :       :                $
0         1         2         3         4         5         6         7  $
01234567890123456789012345678901234567890123456789012345678901234567890123$
To install changes,type C-c C-c

--**-Emacs:*Tab Stops*     (Fundamental Ovwrt)--L1--All--------------
```

屏幕显示一个制表位标尺，制表位的位置用冒号表示。

第一行上的冒号代表制表位的当前设置情况。随后两行构成了一把标尺，给出文本行上每一个字符的显示位置。要想插入一个制表位，需要用“**C-f**”组合键移动到预定的字符列位置，然后输入一个冒号字符（:）。要想删除一个制表位，需要先移动到预定的制表位位置，然后按下空格键。“*Tab Stops*”编辑缓冲区处于改写模式下，所以所做的修改不会影响到其他的制表位。记住: 一定要在画面的第一行进行编辑修改，在其他行上做的修改不会生效。

对制表位的设置情况感到满意之后，按下“**C-c C-c**”组合键让它们生效。如果没进行任何修改，按“**C-c C-c**”组合键将退出这个编辑缓冲区。如果做了一些修改但又不想让它们生效，可以用“**C-x k RETURN**”命令删除这个编辑缓冲区；表示继续使用默认的制表位设置。

所设置的制表位只能在这次Emacs会话里起作用，新设置的制表位将影响到此后创建的一切编辑缓冲区。

把制表位转换为空格

如果有人不明白“把制表位转换为空格”的原因，那么可以不去管它，直接跳过这一小节应该也没什么影响。这部分内容主要与那些需要把文件或软件包从一个系统挪到另外一个系统上的人们，或者与那些遇到打印问题的人们有关。

制表位提供了一种对文本进行某些简单排版的方法。但它们可能会引起大问题——因为它们不是“可移植的”。对于制表位之间的标准间隔设置没有统一的规定。制表位通常是用一个特殊的字符（ASCII 9或者“**C-i**”）来表示的。当某个终端、编辑器或者打印机看到这个字符的时候，就会把这个字符解释为“移动到下一个制表位”——就像在打字机上看到的情况一样。刚刚介绍过Emacs允许把制表位设置在任意位置上，但要想重新设置打印机上的制表位可就没有这么简单了。虽然文件在屏幕上看起来完美无缺，但要是打印机的制表位设置得与Emacs的制表位不一致，就无异于一场灾难。对**troff**和TEX文档来说，制表位更是一个特别容易出问题的因素。有不少用户为了弄明白“为什么Emacs显示的是一个样，可打印出来的东西却又是另外一个样”而花费了大量的时间——根源全在几个隐藏着的制表位身上。

GNU Emacs特意准备了一种既能去掉文件里的制表符，又能用制表位进行简便排版的方法。这样，可以在编辑工作中使用制表位，最后把各制表位统一转换为正确个数的空格，而文件的版面布局丝毫不会改变。与制表符不同，对空格的定义总是很明确的。去掉制表符的命令是“**ESC x untabify**”。把空格转换为制表符的相应命令是“**tabify**”，但我们建议大家不要记这个命令。

untabify命令作用于文本块。也就是说，要想使用这个命令，必须先在编辑缓冲区里的某个地方（最好是编辑缓冲区的开头）设置一个文本块标记，再把光标移动到编辑缓冲区里的另外一个地方（最好是编辑缓冲区的结尾），最后再输入“**ESC x**

untabify RETURN”命令。“**C-x h**”组合键（命令名是 **mark-whole-buffer**）能够自动把光标放到编辑缓冲区的开头、把文本块标记放到其结尾。这将使去掉制表位的工作更容易进行一些，因为只需用一个简单的按键组合“**C-x h ESC x untabify RETURN**”就能把事情做好。

但我们必须提醒大家有一种文件是不能进行去除制表位操作的，它就是制作文件（makefile）。制作文件里的制表位都有着特殊的含义和作用，把它们转换为空格会引起一系列错误。（作为UNIX最有用的程序之一，出现这样一个设计缺陷真让人觉得遗憾。）假使不知道什么是制作文件也没有关系，只要记住不要对任何名字是“*makefile*”或“*Makefile*”的文件进行去除制表位的操作即可。

表 8-1 对与制表位有关的命令进行了汇总。

表 8-1：制表位命令速查表

键盘操作	命令名称	动作
（无）	**edit-tab-stops**	打开一个名为“*Tab Stops*”的编辑缓冲区以便改变制表位的设置情况
（无）	**tabify**	在不影响文本版面的情况下把 3 个及 3 个以上的空格视需要转换为制表位间隔
（无）	**untabify**	把全部制表位都转换为对应个数的空格
（无）	**fill-individual-paragraphs**	根据段落的缩进情况对它们进行段落重排

缩进前导字符串的使用

有好几种方法可以用来在段落前面自动增加一些空白。其中之一就是使用缩进前导字符串（fill prefix）——由 Emacs 自动放到每行输入文本开头的一个字符串。之所以会使用“*fill prefix*”这个术语，是因为 Emacs 把字换行（word wrap）功能称为“*auto-fill mode*（自动换行模式）”。换句话说，所谓缩进前导字符串就是这样一种字符串：当Emacs进行字环绕处理的时候，它会把这个字符串插入到每个文本行的开始（即给文本行加上一个“前缀”）。

缩进前导字符串最好是在自动换行模式下使用。如果状态行上有“Fill”字样，就

说明已经在自动换行模式中了。如果不是这样，请输入“**ESC x auto-fill-mode RETURN**”命令。

下面假设对一个备忘录进行缩进处理。在备忘录的第一行，手动输入缩进值——比如8个空格。然后按下“**C-x .**”组合键（命令名是**set-fill-prefix**）。Emacs将在辅助输入区里显示这样一条信息：“`fill prefix "  "`”。接下来，像平时一样开始输入。当输入文本超出右边界而使Emacs进行自动换行时，它会把8个空格的缩进设置插入到文本行的开头。

下面是一个更有意思的例子。没有理由让缩进前导字符串永远都是些空格，它们可以是任何东西。请看下面的例子：假设正在给同事发电子邮件，并且想给它加上一个抢眼的缩进前导字符串。

输入：**ELEPHANT RIDING PARTY!!! C-x .**

```
Buffers Files Tools Edit Search Help
ELEPHANT RIDING PARTY!!!

--**-Emacs: elephant        (Text Fill)--L1--All----------------------
fill prefix:"ELEPHANT RIDING PARTY!!!"
```

先输入一个字符串，再用“**C-x .**”组合键把它设置为缩进前导字符串。

设置好缩进前导字符串之后，像平时那样输入邮件消息的内容。

输入：**The time . . . the zoo.**

```
Buffers Files Tools Edit Search Help
ELEPHANT RIDING PARTY!!!The time for the annual elephant riding party
ELEPHANT RIDING PARTY!!!is nearly upon us. Plan to be at the San
ELEPHANT RIDING PARTY!!!Diego Zoo at 3:30 on Saturday,June 10. A
ELEPHANT RIDING PARTY!!!$5.00 donation per elephant ride is requested
ELEPHANT RIDING PARTY!!!to benefit the zoo.

--**-Emacs: elephant        (Text Fill)--L5--All----------------------
```

Emacs将把缩进前导字符串插入到邮件消息每一行的开头。

只需输入一遍“ELEPHANT RIDING PARTY!!!”就足够了，Emacs会自动把其余的插入进来。下面是一些在使用缩进前导字符串时需要了解的注意事项：

- Emacs不会在段落的第一行插入缩进前导字符串。很明显，不可能让Emacs从第一段的第一行就开始插入缩进前导字符串(总得在什么地方亲自输入这个字符串吧)。但Emacs在任何一段的第一行都不会插入缩进前导字符串。这么说吧，如果上面这个备忘录有两个段落，那你就必须在第二段的开头再输入（或者复制）一遍短语“ELEPHANT RIDING PARTY!!!”。

  不过，缩进前导字符串不用重新设置。Emacs会给后续段落除第一行之外的所有文本行都加上缩进前导字符串 —— 它只在段落的第一行上犯迷糊。

- 一旦开始使用缩进前导字符串，怎样才能让它停下来呢？没有专门为此准备的命令。得把光标放在左边界上，然后用“**C-x .**”组合键再定义一个新的缩进前导字符串。可这次的缩进前导字符串应该被定义为空字符串，即什么也没有。

- 最后，如果以后又想对这些段落重新进行排版，需先去掉以前的缩进前导字符串，重新定义一个新的，然后再按下“**ESC q**”组合键。这个命令虽然简单，却正是我们需要的，因为**fill-individual-paragraphs**之类的命令对付不了缩进前导字符串。

文本缩进模式

如果会大量使用缩进，就应该考虑使用Emacs的文本缩进模式（indented text mode)。这个特殊的编辑模式额外提供了一些对缩进功能的支持。它很适合用来拟订大纲（也可以考虑使用本章后面将要介绍的大纲模式“outline mode”)。如果书写的东西里需要大段的引用，那么文本缩进模式可以帮助把引用部分和其他文字区分开。

进入文本缩进模式的方法是输入“**ESC x indented-text-mode RETURN**”命令。状态行上将出现“`Indented`”字样。另外，如果状态行上没有“`Fill`”字样，还得再输入“**ESC x auto-fill-mode RETURN**”命令。如果自动换行模式没有启用，那么文本缩进模式是无法工作的。

知道如何进入文本缩进模式之后，我们再来看看如何使用它。先从屏幕画面的左边界开始用**TAB**键或者空格键设置出足够的缩进，然后开始录入内容。到达文本行末尾的时候，Emacs会自动对下一行进行缩进，使它与前一行匹配。下面是一个用文本缩进模式来排版一段引文的例子。

输入：**ESC x indented-text-mode RETURN**

```
Buffers Files Tools Edit Search Help

-----Emacs: dickensarticle     (Indented Text Fill)--L1--All-----------
```

状态行上将出现“Indented”字样。

输入文本的第一段。

```
Buffers Files Tools Edit Search Help
At the beginning of A Tale of Two Cities, Dickens penned the most
familiar opening lines found in any novel:

--**-Emacs: dickensarticle     (Indented Text Fill)--L2--All-----------
```

接下来将以缩进格式输入一段引文。

输入：**RETURN TAB**、然后输入一段引文：

```
Buffers Files Tools Edit Search Help
At the beginning of A Tale of Two Cities, Dickens penned the most
familiar opening lines found in any novel:
        It was the best of times, it was the worst of times, it was
        the age of wisdom, it was the age of foolishness, it was the
        epoch of belief, it was the epoch of incredulity, it was the
--**-Emacs: dickensarticle     (Indented Text Fill)--L5--All-----------
```

只要把第一行缩进，Emacs就能把整段引文都自动地缩进。

Emacs会认为想对所有的段落都做同样的缩进处理。如果想开始一个新的段落，需要连按两次回车键（留出一个空白行），然后再按一次**TAB**键（用来设置下一个段落的缩进量）。在设置好最初的缩进层次之后，再按一次**TAB**键将得到与前一段相同的缩进效果。在文本缩进模式里按一次**TAB**键会自动回到文本的第一列，不管制表位是如何设置的，也不管刚才设置的缩进距离是多大。

如果想减小缩进的距离，可以用**DEL**键删除一些制表位或空格（把它们全删除也行）；如果想加大缩进的距离，可以接着按**TAB**键或空格键。

知道了如何在文本缩进模式里录入文字之后，我们再来看看怎样才能对它们进行编辑。随着编辑工作的进行，文本将逐渐变得参差不齐，需要时不时地把它们重排为

整齐的段落。具体操作方法已经在前面介绍过：把打算重排的那部分内容标记为一个文本块，然后输入“**ESC x fill-individual-paragraphs**”命令。

退出文本缩进模式的方法是进入另外一个主模式，比如文本模式等。进入文本模式的方法是输入“**ESC x text-mode**”命令，状态行上的“Indented Text”字样将改变为“Text”字样。

文本缩进模式中的制表位

文本缩进模式不使用 Emacs 预先定义的制表位。该模式下的制表位情况是这样的：首先，Emacs 会找到上一个非空白行，然后，第一个制表位将对应于该行第一个非空白字符的那一列，第二个制表位则对应于该行下一个单词的第一个字母所在的那一列，以此类推。该行结束之后，正常的制表位将重新“占据”这一行的剩余部分。图 8-1 给出了 Emacs 在文本缩进模式下的制表位定义情况。

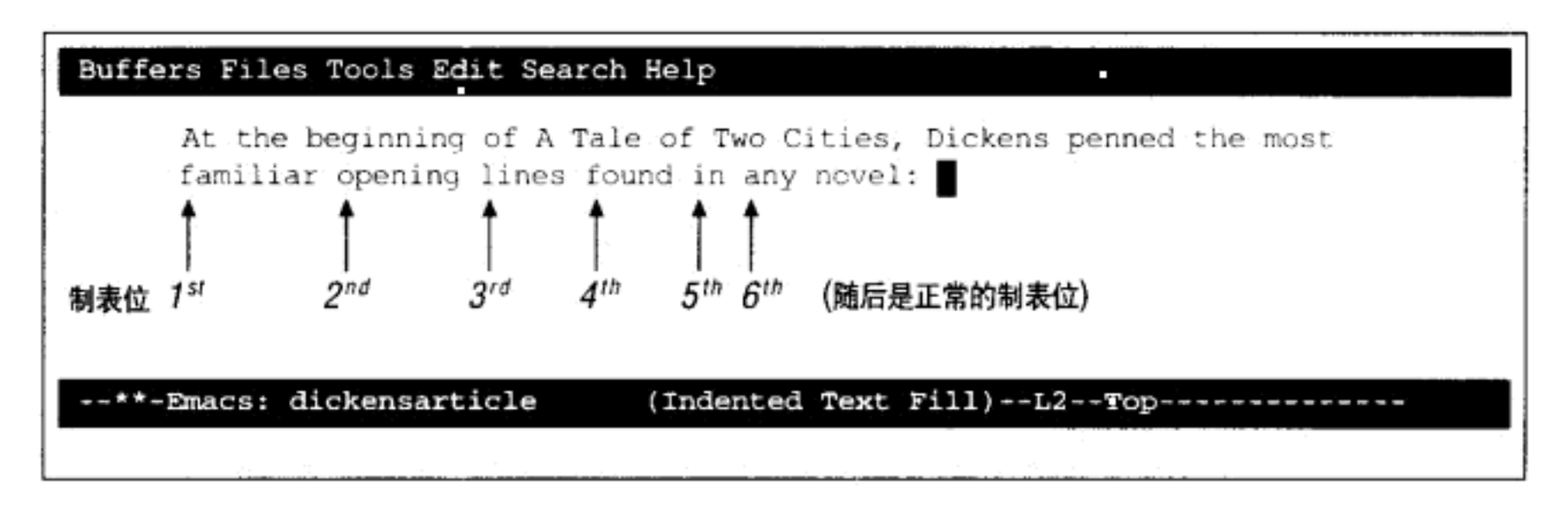


图 8-1：设定制表位

第一个制表位对应字母“f”所在的那一列，第二个对应字母“o”所在的那一列，以此类推。注意：这可能会带来很多的不便 —— 如果在文本缩进模式下拟订一份大纲，那么，按两次（或者更多次）**TAB** 键并不一定会把你带到文本行上的同一个位置。好在制表位并不是惟一的选择，可以用空格键对缩进距离做任意调整。

对文本块进行缩进

如果已经在没有进行缩进的情况下录入了文本，现在又想对它们进行缩进，该怎么办呢？请按以下步骤操作：

1. 进入文本缩进模式（输入“**ESC x indented-text-mode RETURN**”命令）。
2. 移动到准备缩进的文本块的末尾，设置文本块标记。
3. 移动到准备缩进的文本块的开头设置好缩进距离，用空格或制表位来设置都行。
4. 按下“**ESC C-**”组合键（命令名是**indent-region**）。Emacs将按第一行的格式对整个文本块进行缩进。
5. 按下“**ESC q**”组合键对文本块进行段落重排。

如果你打算缩进的区域有好几个缩进层次，就必须把不同缩进层次的每一块文本定义为不同的文本块，然后再按上述步骤进行设置。

其他缩进技巧

只要正使用着缩进（不管你是否在文本缩进模式里），那么随时可以用“**ESC m**”组合键（命令名是**back-to-indentation**）移动到文本行的第一个非空白字符上。在没有被缩进的文本行上，这个命令将把光标移动到这一行的开头。也就是说，“**ESC m**”命令将把光标移动到文本行的“逻辑”开始位置，也就是平时按下“**C-a**”组合键时到达的位置。

另一个缩进命令是“**ESC C-o**”（命令名是**split-line**）。可以用这个命令达到一种使文字呈台阶状排列的效果。这个命令也可以用在文本缩进模式以外的其他编辑模式里。先把光标移动到准备分拆到下一行去的文本处，然后按下“**ESC C-o**”组合键即可。如下所示：

初始状态：

```
Buffers Files Tools Edit Search Help
It was the best of times,  t was the worst of times

--**-Emacs:dickens        (Text Fill)--L1--All-----------------------
```

我们想分拆这一行。

按下：**ESC C-o**

```
Buffers Files Tools Edit Search Help
It was the best of times,
                         It was the worst of times

--**-Emacs: dickens          (Text Fill)--L1--All----------------------
```

“**ESC C-o**”在光标位置把这一行分拆为呈台阶状排列的两行。

虽然这个命令在文本模式下也能制造出这种有趣的效果——可能适合书写诗歌之类的东西，但它在本章后面介绍的图形模式（picture mode）里会更有用。

表 8-2 对前面介绍的缩进命令进行了汇总。

表 8-2：缩进命令速查表

键盘操作	命令名称	动作
C-x .	**set-fill-prefix**	把左边界到光标位置之间的字符串设置为加在段落每一行之前的缩进前导字符串；在第一列输入这个命令，表示此后将不再插入缩进前导字符串
（无）	**indented-text-mode**	进入文本缩进模式。在这个编辑模式里，按下 **TAB** 键将为某段落的后续文本行定义一个新的缩进层次
（无）	**text-mode**	退出文本缩进模式，让编辑缓冲区回到文本模式（进入其他的主编辑模式也能退出文本缩进模式）
ESC C-	**indent-region**	按第一行的缩进设置把文本块作整体上的缩进
ESC m	**back-to-indentation**	把光标移动到文本行的第一个字符
ESC C-o	**split-line**	在光标位置处分拆文本行，并把它的后半截缩进到光标所在的那一列
（无）	**fill-individual-paragraphs**	对缩进的段落进行重排，但保留缩进效果

文本的居中

另外一项常见的排版工作就是文本居中。比如，可能想让文档标题或者文档中的小标题居中。Emacs 提供有用来对文本行、段落和文本块等进行居中排版的命令。

首先要确认是在文本模式里。查看一下状态行，如果上面显示有“*Text*”字样，就说明已经在文本模式里；如果看不到这个单词，请输入“**ESC x text-mode RETURN**”命令以进入文本模式。接下来，录入想居中的文字（或者把光标移到某现有文本行的任意位置）之后，按下“**ESC s**”组合键。如下所示：

输入：**Annual Report**

```
Buffers Files Tools Edit Search Help
Annual Report

--**-Emacs: annualreport      (Text Fill)--L1--All----------------------
```

按下：**ESC s**

```
Buffers Files Tools Edit Search Help
                          Annual Report

--**-Emacs: annualreport  *  (Text Fill)--L1--All----------------------
```

Emacs 把这一行放到居中的位置。

还可以对段落或者文本块进行居中排版。Emacs将对段落或文本块中的每一行分别做居中处理，而不是把它们当做一个整体进行居中。要想使段落居中摆放，需要使用“**ESC S**”（命令名是 **center-paragraph**）组合键；要想使文本块居中，需要使用“**ESC x center-region**”命令。请看下面这个示例：让一段引文居中。

初始状态：

```
Buffers Files Tools Edit Search Help
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity..

--**-Emacs:dickens          (Text Fill)--L6--All----------------------
```

按下：**ESC S**

```
Buffers Files Tools Edit Search Help
                    It was the best of times,
                    it was the worst of times,
                    it was the age of wisdom,
                 it was the age of foolishness,
                  it was the epoch of belief,
               it was the epoch of incredulity ...
--**-Emacs: dickens          (Text Fill)--L6--All----------------------
```

文本已居中。

在上面的例子里，逐行居中的效果看起来相当有艺术味。但有时会遇到需要让Emacs把文本块作为一个整体来对进行居中排版的情况。可以用本章前面介绍的**indent-region**命令来实现这种排版效果。为了达到视觉上的居中排版效果，一定要把文本块的缩进量安排好。

表8-3对文本居中命令进行了汇总。

表8-3：居中命令速查表

键盘操作	命令名称	动作
ESC s	**center-line**	把光标所在的文本行居中
ESC S	**center-paragraph**	把光标所在的段落居中
（无）	**center-region**	把当前定义的文本块居中

插入分页符

虽然Emacs并没有提供什么页面排版功能，但用户可以在自己的文件里插入分页符。首先，输入引用命令（“**C-q**”组合键），它的作用是告诉Emacs不管随后的字符是什么，都把它插入到文本里（而不要把那个字符解释为一个编辑命令）。然后，在文本行的第一列输入“**C-l**”字符。这个字符在屏幕上会显示为“^L”。虽然这个字符看起来好像是两个字符，但其实却是一个，删除它试试就会明白了。

“**C-l**”代表着一个名为进纸符（formfeed）的特殊字符。许多打印机（特别是行式打印机、菊花轮打印机和点阵式打印机）和不少UNIX程序（比如**more**）都会把进

纸符解释为分页符（page break）。对 **troff** 和 TEX 等排版软件来说，“**C-l**”字符是无含义的（或者更糟，可能引起问题）。对大部分激光打印机来说，“**C-l**”字符也是无含义的（但很多激光打印机都有一种能够仿真行式打印机的工作模式，该模式也许会对进纸符做出正确的解释）。

使用这种分页符的弊端是它们都是一些“硬”分页符。随着编辑工作的进行，如果某个页被加长或者缩短了，那么就必须重新设置它的分页符，这样才能让它在打印时实现正确地分页。我们认为比较好的方法是不在工作稿里添加这种分页符，等到想打印这个文件的时候，先复制它，然后再用一个简单的宏（macro）来每隔“*n*”行插入一个分页符（“*n*”代表每页的行数）。关于宏的编写方法，请参考第十章。

矩形编辑

当为移动或删除操作设定文本块的时候，文本块总是覆盖整个画面的宽度。Emacs 里有很多工作很适合用文本块来完成。可要是用文本块来编辑一个表格就不好办了吧？由于文本块会覆盖整个画面的宽度，所以不适合用来解决这类问题。Emacs 专门为此准备了另外一种用来定义文本区域的方法——矩形（rectangle）。Emacs 中的矩形是用特殊的矩形编辑命令定义和处理的长方形区域。如果需要移动或者删除垂直排列的信息块，采用矩形编辑的办法将是非常有效的。比如需要移动表格的某列数据的情况，又比如在编写程序时需要添加或者删除代码注释的情况等。

先来看一个例子。假设正在对下图中的表格进行编辑，现准备把“Working Hours”栏移动到最右边去。文本块是解决不了这个事情的，可要是掌握了矩形编辑命令，那么这样的工作可以说是轻而易举。如下所示：

初始状态：

```
Buffers Files Tools Edit Search Help
                         Flex-time Schedule

Name        Working Hours      Email Address     Home Phone

Fred        7:30-4:30          fred              (814)722-3940
Olivia      9:00-6:00          liv               (814)827-3344
Nicole      10:00-7:00         ncm               (412)734-0492
Alvin       6:00-3:00          al                (814)437-9320

--**-Emacs: flextime        (Text Fill)--L5--All----------------------
```

定义矩形区域与定义文本块的方法是一样的，它们之间的区别体现在设定文本区域之后发出的编辑命令，这些命令将告诉Emacs打算对文本块进行操作，还是打算对矩形进行操作。

定义矩形的方法是：先把光标移动到它的左上角，然后按下“**C-@**”组合键或“C-SPACE”组合键设置文本块标记；然后再把光标移动到矩形的右下角。在把光标移动到矩形的右下角之后，还要再向右移动一个字符。为什么还要移动一个字符的距离呢？记住，在定义文本块的时候，文本块标记上的字符是文本块的一部分，但光标位置上的字符却不是文本块的一部分。定义矩形时也要遵守同样的规则，只是它得在水平方向和垂直方向上都满足这一要求。现在来定义一个覆盖表格第二列的矩形。如下所示：

按下：**C-@**

```
Buffers Files Tools Edit Search Help
                        Flex-time Schedule

Name        Working Hours       Email Address     Home Phone

Fred        7:30-4:30           fred              (814)722-3940
Olivia      9:00-6:00           liv               (814)827-3344
Nicole      10:00-7:00          ncm               (412)734-0492
Alvin       6:00-3:00           al                (814)437-9320

--**-Emacs: flextime        (Text Fill)--L5--All----------------------
Mark set
```

文本块标记设置在准备移动的矩形区域的左上角。

把光标移动到这个矩形区域的右下角再往右一个空格的位置，即单词“al”中的字母“a”处。

```
Buffers Files Tools Edit Search Help
                        Flex-time Schedule

Name        Working Hours       Email Address     Home Phone

Fred        7:30-4:30           fred              (814)722-3940
Olivia      9:00-6:00           liv               (814)827-3344
Nicole      10:00-7:00          ncm               (412)734-0492
Alvin       6:00-3:00           al                (814)437-9320

--**-Emacs: flextime        (Text Fill)--L8--All----------------------
```

光标位于矩形区域右下角再往右一个空格的位置。

这样，这个矩形区域就定义好了。下一步是删除它，然后把它移动到需要的位置。以粘贴为最终目的的矩形删除命令是“**C-x r k**”（命令名是 **kill-rectangle**）。

按下：**C-x r k**

```
Buffers Files Tools Edit Search Help
                        Flex-time Schedule

Name     Email Address  Home Phone

Fred     fred           (814)722-3940
Olivia   liv            (814)827-3344
Nicole   ncm            (412)734-0492
Alvin    al             (814)437-9320

--**-Emacs: flextime        (Text Fill)--L8--All----------------------
Loaading rect . . . done
```

矩形区域被删除，它被放到一个特殊的矩形删除缓冲区（rectangle kill buffer）里。

再重复一遍：在选取矩形的时候，必须先把光标放在它的左上角，设置上文本块标记；然后再移动到矩形右下角再往右一个字符的位置。Emacs 会认为矩形就是一个规规矩矩的长方形。如有必要，它会在文本右侧添加空格，以使选取到的文本区域能够正确地围成一个矩形。

在屏幕上做任意的移动，然后用“**C-x r y**”组合键（命令名是 **yank-rectangle**）把最后一次删除的矩形重新插入到光标位置。为了把“Working Hours”栏移动到表格的最右边，需要把光标移动到“Home Phone”的后面去。如下所示：

把光标移动到“Home Phone”的后面，然后按一次 **TAB** 键：

```
Buffers Files Tools Edit Search Help
                        Flex-time Schedule

Name     Email Address  Home Phone

Fred     fred           (814)722-3940
Olivia   liv            (814)827-3344
Nicole   ncm            (412)734-0492
Alvin    al             (814)437-9320

--**-Emacs: flextime        (Text Fill)--L5--All----------------------
```

把光标移动到想重新插入矩形区域的地方。

按下：**C-x r y**

```
Buffers Files Tools Edit Search Help
                         Flex-time Schedule

Name     Email Address  Home Phone        Working Hours

Fred     fred           (814)722-3940     7:30-4:30
Olivia   liv            (814)827-3344     9:00-6:00
Nicole   ncm            (412)734-0492     10:00-7:00
Alvin    al             (814)437-9320     6:00-3:00

 --**-Emacs: flextime        (Text Fill)--L8--All----------------------
Mark set
```

Emacs 插入刚删除的矩形区域。

Emacs 按照操作精确地插入了矩形。把光标移到“Home Phone”的后面，然后再按一次 **TAB** 键的目的是为了在“Home Phone”和“Working Hours”两栏之间留出一些空间。要不然，Emacs 插入的“Working Hours”栏内容就会混杂在“Home Phone”栏的内容当中。需要提醒大家注意的是矩形没有相应的删除环（kill ring，参见第二章），但下一小节讨论的图形模式（picture mode）是个例外。只能把最近一次删除的矩形恢复回来。

矩形区域的剪切和粘贴操作需要多加练习才能掌握。一旦熟悉了这一流程，表格或者其他依赖于列的文字编辑工作就不在话下。

还有其他一些用来在屏幕上创建空白矩形的命令。举个例子，假设想把“Name”栏和“Email Address”栏之间的间隔加大两个空格。用“**C-x r o**”组合键（命令名是 **open-rectangle**）可完成这一工作。这个命令的作用是在屏幕上的任意位置插入一个空白的矩形，原有的文本将被推向右侧。首先定义一个矩形，然后再用“**C-x r o**”组合键在表格的第一栏和第二栏之间插入一些空白间隔。如下所示：

把光标移动到单词“Email”中的字母“E”上，然后按下“**C-@**”组合键

```
Buffers Files Tools Edit Search Help
                         Flex-time Schedule

Name     Email Address  Home Phone        Working Hours

Fred     fred           (814)722-3940     7:30-4:30
Olivia   liv            (814)827-3344     9:00-6:00
Nicole   ncm            (412)734-0492     10:00-7:00
Alvin    al             (814)437-9320     6:00-3:00

 --**-Emacs: flextime        (Text Fill)--L5--All----------------------
Mark set
```

Emacs 把文本块标记设置在矩形区域的左上角。

现在，需要定义准备插入到两栏之间的间隔量。移动到矩形的底部（即人名“Alvin”所在的那一行），然后再向右移动两个字符的距离。最后，按下“**C-x r o**”组合键给表格增加新的空格间隔。

把光标移动到单词“al”的后面，然后按下“**C-x r o**”组合键

```
Buffers Files Tools Edit Search Help
                    Flex-time Schedule

Name      Email Address  Home Phone       Working Hours

Fred      fred           (814)722-3940    7:30-4:30
Olivia    liv            (814)827-3344    9:00-6:00
Nicole    ncm            (412)734-0492    10:00-7:00
Alvin     al             (814)437-9320    6:00-3:00

--**-Emacs: flextime        (Text Fill)--L8--All----------------------
```

Emacs 将插入一个两个空格宽的空白矩形。它会把表格中的其他内容推向右边。

clear-rectangle 命令的作用是清除矩形中的文本，在它原来的位置留下一个空白的矩形。这就像在黑板上擦掉一列板书一样。类似于黑板板书的情况，被清除掉的文本列将永久消失，不会被保存在矩形删除缓冲区（rectangle kill buffer）里。还是用刚才的例子进行说明。在传阅了表格“Flex-time Schedule”之后，有关人员都不同意把他们的家庭电话列在表里。于是：

把光标移动到单词“Home”中的字母“H”上，然后按下“**C-@**”组合键

```
Buffers Files Tools Edit Search Help
                    Flex-time Schedule

Name      Email Address  Home Phone       Working Hours

Fred      fred           (814)722-3940    7:30-4:30
Olivia    liv            (814)827-3344    9:00-6:00
Nicole    ncm            (412)734-0492    10:00-7:00
Alvin     al             (814)437-9320    6:00-3:00

--**-Emacs: flextime        (Text Fill)--L5--All----------------------
Mark set
```

文本块标记被设置在准备清除的矩形的左上角。

把光标移动到最后一个电话号码之后的那个空格上，然后按下“**C-x r c**”组合键

```
Buffers Files Tools Edit Search Help
                         Flex-time Schedule

Name      Email Address                    Working Hours

Fred      fred                             7:30-4:30
Olivia    liv                              9:00-6:00
Nicole    ncm                              10:00-7:00
Alvin     al                               6:00-3:00

--**-Emacs: flextime        (Text Fill)--L8--All----------------------
```

clear-rectangle 命令删除了“Home Phone”栏，原先的位置上留下了一块空白。

正如大家所看到的，表格间隔还是不够完美；还需要用**delete-rectangle**命令把第二栏和第三栏之间多余的空白间隔去掉一些。类似于 Emacs 里的其他删除命令，**delete-rectangle** 命令也不会把删除的东西保存到删除环（kill ring，参见第二章）里。要想删除那些多余的空白间隔并且不保存它们，得先把光标移动到“Email Address”后面的空格处，按“**C-@**”组合键设置上文本块标记；再沿着对角线方向移动到待删除矩形的另外一端；最后，按下“**C-x r d**”组合键。

把光标移动到“Email Address”的尾部，然后按下“**C-@**”组合键

```
Buffers Files Tools Edit Search Help
                         Flex-time Schedule

Name      Email Address                    Working Hours

Fred      fred                             7:30-4:30
Olivia    liv                              9:00-6:00
Nicole    ncm                              10:00-7:00
Alvin     al                               6:00-3:00

--**-Emacs: flextime        (Text Fill)--L5--All----------------------
Mark set
```

文本块标记被设置在准备删除的矩形的左上角。

把光标移动到最后一行里“6:00”再往前几个空格的某个位置上，然后按下“**C-x r d**”组合键

```
Buffers Files Tools Edit Search Help
                          Flex-time Schedule

Name      Email Address   Working Hours

Fred      fred            7:30-4:30
Olivia    liv             9:00-6:00
Nicole    ncm             10:00-7:00
Alvin     al              :00-3:00

--**-Emacs: flextime        (Text Fill)--L8--All----------------------
```

delete-rectangle 命令删除了那个空白间隔。

如果正在做一些精彩的表格编辑工作，那么能同时保存多个矩形，当然会有很大的帮助。这样，就可以把表格的每一栏都保存为一个矩形，再专门用一个矩形来保存各栏之间的空白间隔量。如果是在下面将要介绍的图形模式里使用矩形编辑命令，就可以不受限制地保存多个矩形。

表 8-4 对矩形编辑命令进行了汇总。

表 8-4：矩形编辑命令速查表

键盘操作	命令名称	动作
C-x r k	**kill-rectangle**	删除一个矩形并把它保存起来
C-x r d	**delete-rectangle**	删除一个矩形但不把它保存起来
C-x r y	**yank-rectangle**	在光标位置插入最后一次删除的矩形
C-x r c	**clear-rectangle**	清除矩形区域的内容，并且不保存它
C-x r o	**open-rectangle**	在矩形区域里插入一个空白矩形

绘制简单的图形

无论从哪个方面看，Emacs 都算不上是一个图形软件包，可它确实提供了一些有限的绘图功能。Emacs 有一个图形模式（picture mode），它允许用键盘字符绘制简单的图形。它很适合用来在电子邮件消息之类的文字性内容里快速插入一个草图——

这一功能连大多数图形软件包都不一定具备。这个编辑模式也很适合用来绘制框图、（电子工程师用的）时序图和时间表等各种简单的图形。

千万不要小看这个简单的组件！我们见过许多用**troff**或TEX精心排版的文件，里面总是有一两个用星号和短划线字符绘制出来的草图。当然可以用图形处理软件创建出更好看的图形来；但在现实世界里，**pic**或者TEX系列的图形处理软件用起来既费时间又痛苦。用星号和短划线字符绘制出来的草图虽然不那么美观，但简便性却是谁也比不了的。

图形模式相当于把被编辑区域变成一块按行、列划分的绘图板。在图形模式里，可以用键盘字符创建出简单的图形（比如图8-2中的那个），并且不让它们被自动换行模式（auto-fill mode）的字环绕功能“重新排版”。

进入图形模式的命令是“**ESC x edit-picture**”。状态行上将出现“Picture”字样，跟在它后面的是默认绘制方向（我们马上就要讲到它们）。按下“**C-c C-c**”组合键将退出图形模式，并返回到进入此编辑模式之前所在的一种主编辑模式里。

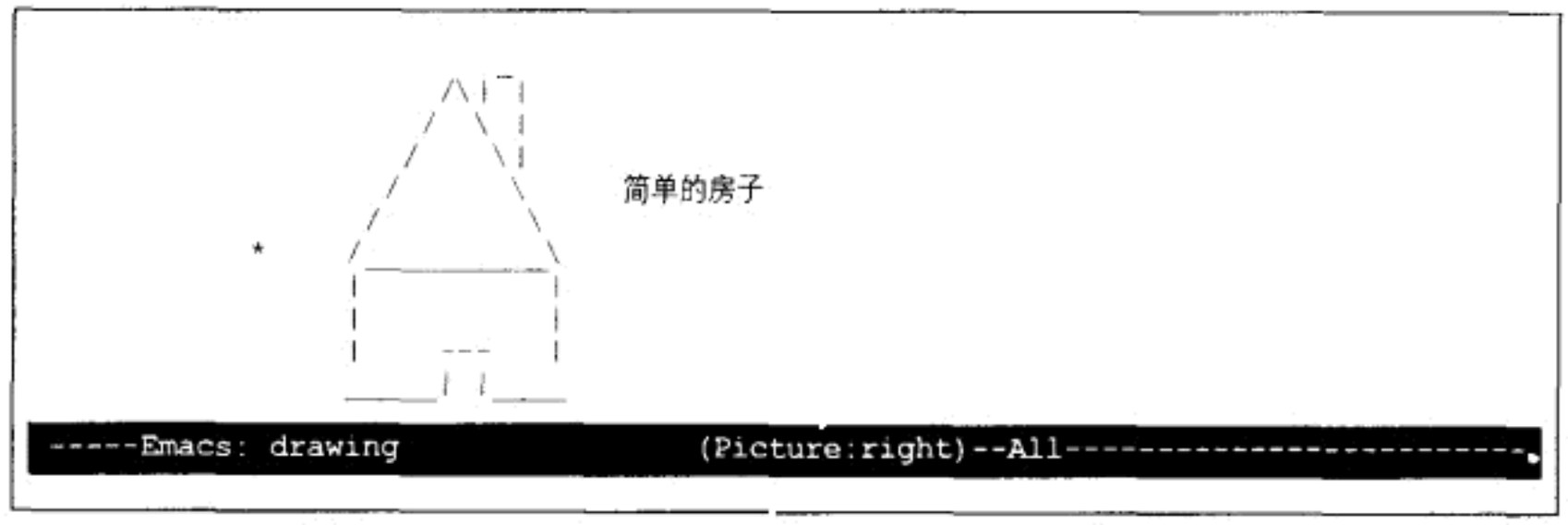


图 8-2：在图形模式中绘制图形

在图形模式中绘制图形

在图形模式里，可以沿8个方向中的任何一个用任何一种字符“画图”。不过，虽然可以沿8个方向绘图，但每次只有一个方向是可用的，这个方向就叫做“默认绘制方向（default direction）”。刚进入图形模式的时候，默认绘制方向是“右”。也就是说，如果连续按4次短划线键，就会向右画出一条“----”这样的直线。默认绘制方向会显示在状态行上，比如下面这样：

```
(Picture:right)
```

通过输入可以改变默认绘制方向的特殊命令，就可以沿另外7个方向画图。比如，“**C-c **”命令将把默认绘制方向设置为“右下”（译注1）；状态行上将显示“(Picture:se)”字样。如果沿这个方向连按4次短划线键，它们将排列为台阶的样子，如下所示：

```
-
 -
  -
   -
```

图8-3给出了用来在图形模式里设置各种默认绘制方向的命令。

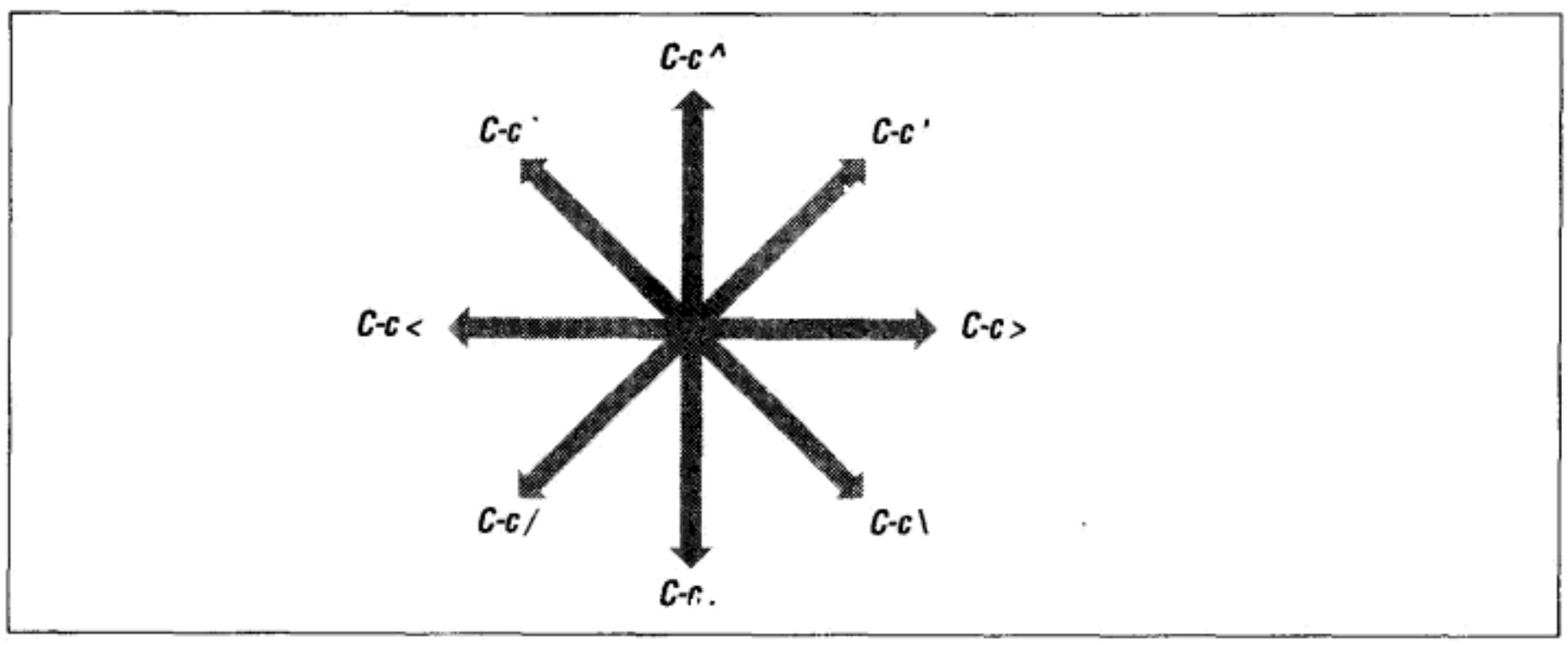


图8-3：图形模式中的方向移动命令

要想记住在Emacs图形模式里设置默认绘制方向的命令并不困难，比如“**C-c ^**”命令中的“^”指向上方，而“**C-c `**”命令中的“`”指向左上方等等。如果你想出了一个记忆“**C-c .**”命令的好方法，别忘了告诉我们！也许你会把它记为“往下点儿”。

设置好默认绘制方向之后，重复按下任何键都会沿该方向画出一条字符线。可以在草稿编辑缓冲区（即“*scratch*”编辑缓冲区）里，用上图中的命令改变一下默认绘制方向试试。我们先一起画个长方形试试。

译者1：southeast，东南。外国人习惯用东、南、西、北及其组合来表示方向，而中国人习惯用上、下、左、右及其组合来表示方向；两者的对应关系是上北、下南、左西、右东。书中将采用中国人习惯的说法。

输入：**ESC x edit-picture**

```
Buffers Files Tools Edit Search Help

-----Emacs:*scratch*        (Picture:right)--L1--All-----------------
Type C-c C-c in this buffer to return it to Lisp Interaction mode.
```

把编辑缓冲区放到图形模式下，默认绘制方向是“右”。

输入：**TAB ESC 20 -**

```
Buffers Files Tools Edit Search Help
        -------------------

-----Emacs:*scratch*        (Picture:right)--L1--All-----------------
ESC 2 0 -
```

Emacs向右画出一条直线。接下来，把默认绘制方向改为“下”，再用“|”字符画出长方形的右边线。

输入：**C-c . ESC 5 |**

```
Buffers Files Tools Edit Search Help
        -------------------|
                           |
                           |
                           |
                           |
--**-Emacs:*scratch*        (Picture:down)--L6--All-----------------
ESC 5 |
```

Emacs向下画出一条直线。接下来，把默认绘制方向改为“左”，再画出长方形的底边线。

输入：**C-c < ESC 20 -**

```
Buffers Files Tools Edit Search Help
        ------------------
        |                |
        |                |
        |                |
```

图形模式中的光标移动操作

图形模式对基本的光标移动操作做了一些细微却又非常重要的改变。这些改变用一句话就能概括：图形模式里的“方向”概念说的是二维空间里的方向，而不是只相对于文件首尾的一维概念。比如，在其他编辑模式里，“**C-f**”组合键的作用大都是朝文件尾方向以每次一个字符的方式移动光标，连续按下“**C-f**”组合键等于是让光标做横越屏幕的移动；而到达文本行的末尾时，它将跳到下一行的第一个字符处，最终将到达文件的结尾。但在图形模式里，“**C-f**”组合键的含义是“向右移动”。当在图形模式里移动到行尾时，“**C-f**”组合键不会回绕到下一行去；它将继续向当前文本行添加字符。也就是说，一旦越过行尾，继续按“**C-f**”组合键就等同于按空格键。

“**C-b**”组合键也有类似的改变。在其他编辑模式里，“**C-b**”意味着向回移动到文件的最开始。但在图形模式里，它却意味着向回移动一位；而一旦到达行首，它就将停在那里不再继续向前移动。

“**C-p**”和“**C-n**”组合键则分别变成绝对的“向上移动”和“向下移动”命令。我们实际操作一回试试：先移动到某一行的末尾，再按下“**C-p**”组合键。在其他的编辑模式里，光标将上移一行，但仍然停留在行尾。如果上一个文本行比当前文本行短，光标在上移的同时也会相应地向左移动。但在图形模式里，“**C-p**”和“**C-n**”组合键引起的光标移动将是绝对的。它们将永远沿直线上移或者下移。

“**C-f**”、“**C-b**”、“**C-p**”和“**C-n**”这4个组合键能移动到想去的任何位置。键盘上的方向键通常也能够工作；但还需要掌握沿默认绘制方向进行移动的光标移动命令——它们将使侧向移动更迅速。“**C-c C-f**”组合键的作用是沿默认绘制方向前进（这里所说的“前进”其方向可能是上、下、左、右及各种45°方向），而“**C-c C-b**”组合键的作用是沿默认绘制方向后退。（相对于默认绘制方向的向“上”及向“下”移动没有定义。）

举个例子。假设画了一个如图8-2所示的小房子，现在想沿左房顶向下移动光标。于是，先用“**C-c /**”组合键把默认绘制方向设置为“左下”。如果光标在左房顶最高处的“瓦片”上，按下“**C-c C-f**”组合键将沿左房顶向下移动一行，而按下“**C-f**”组合键将移动到右房顶最高处的“瓦片”上，如图8-4所示。

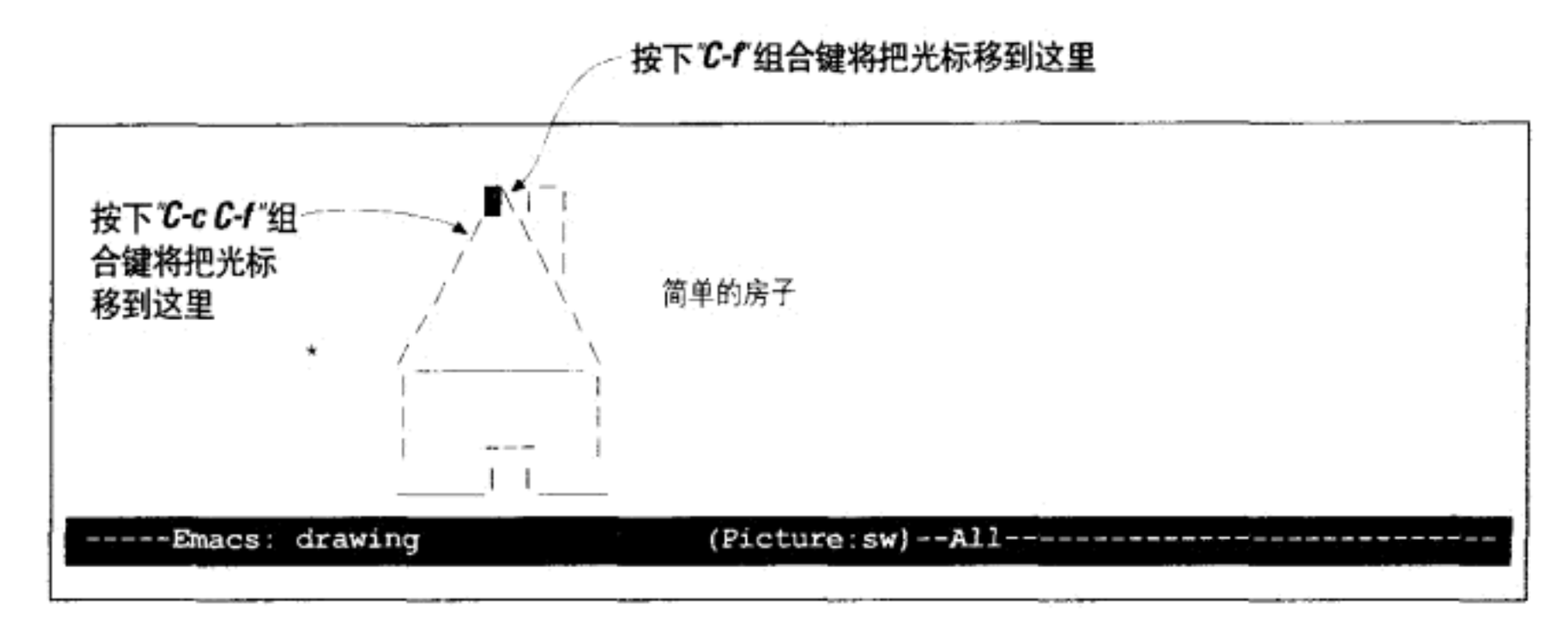


图 8-4: 图形模式中的默认方向

插入文本行

如果在图形模式里多呆一会儿，还将发现更让人吃惊的事情：在图形模式里按回车键会移动到下一行的开始，但并不会插入一个空白行，这是因为Emacs会假定不想改变各行之间的相对关系。如果真的想插入一个新行，就需要使用"**C-o**"组合键；当前行的下面就将出现一个新的空白行，但光标却不移动。请看下图，光标的初始位置是在第一行中的"0"字符处。如果想在这两行之间插入一个新行，就需要按下"**C-o**"组合键。如下所示：

初始状态：

```
Buffers Files Tools Edit Search Help
abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ

--**-Emacs:*scratch*      (Picture:right)--L1--All----------
```

初始状态文本；光标位于第一行的字符"0"上。

按下：**C-o**

```
Buffers Files Tools Edit Search Help
abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ

--**-Emacs:*scratch*      (Picture:right)--L1--All----------
```

"**C-o**"命令插入一个新行，但没有移动光标。

在图形模式里，想用一个真正的“回车符”把某行从半中间断开还真有点难度。一种可行的方法是：先移动到文本行半中间的某个点上，然后用“**C-k**”组合键删除其后半截；再用“**C-o**”组合键插入一个空白行，按回车键移动到空白行的开头，最后再用“**C-y**”组合键把上一行刚被删除的后半截恢复在此空白行上。另外一种办法是先使用断行命令（“**ESC C-o**”组合键），再把新行前半截的空格都删除。

删除操作也得像这样迂回进行。“**C-c C-d**”组合键是图形模式下的字符删除命令，它的作用是删除光标位置下的字符，并把该行上的其余文本向左移。单独使用“**C-d**”组合键将删除光标位置下的字符，并把它替换为一个空格。而**DEL**键将删除光标左侧的字符，并把它替换为一个空格。

表8-5对图形模式命令和它们在文本模式下的“正常”行为进行了对比。

表8-5：图形模式与文本模式的对比

键盘操作	文本模式	图形模式	图形模式下的等价命令
RETURN	插入一个空白行	把光标移动到下一行的开始	“**C-o**”组合键插入一个空白行
C-d	删除字符并把其余文本向左移	把字符替换为一个空格，不移动其余文本	“**C-c C-d**”组合键相当于文本模式下的“**C-d**”组合键
SPACE	把文本向右移，插入一个空格	把光标向右移，沿途经过的所有字符都将被替换为空格	无；请回到文本模式去插入空格[a]
C-k	删除当前行上的文本；按两次“**C-k**”组合键将删除这一行	删除当前行上的文本，但不删除这一行	删除某行需回到文本模式进行操作或者使用**delete-rectangle**命令
TAB	插入制表符并把后续文本向右移	向右移动光标，但不影响沿途经过的文本	要想插入与一个制表位等价的空格，请回到文本模式进行操作
C-n	移动到下一行	向下移动，但光标的列坐标保持不变	（无）
C-p	移动到上一行	向上移动，但光标的列坐标保持不变	（无）

表 8-5: 图形模式与文本模式的对比（续）

键盘操作	文本模式	图形模式	图形模式下的等价命令
C-f	向文件尾方向移动一个字符	向右移动一个字符	（无）
C-b	向文件头方向移动一个字符	向左移动一个字符；到达行首后就不再移动	（无）

a. 如果想插入的是一个空白块，就得使用 **open-rectangle** 之类的命令。具体用法请参考本章前面对这一命令的介绍。另外，如果想在行尾插入空格，就得使用“**C-f**”之类的组合键。

有些工作还是临时切换回自己习惯使用的编辑模式里进行操作比较方便一些。“**C-c C-c**”组合键能回到进入图形模式前所在的编辑模式。在进行完必要的编辑之后，输入“**ESC x edit-picture**”命令可以重新进入图形模式。

如果想对绘制的东西进行平移，最好的方法是在图形模式里使用矩形编辑命令进行操作，这也正是下一小节将要讨论的内容。

在图形模式中使用矩形编辑命令

矩形编辑命令和图形模式是一对好搭档。用矩形编辑命令剪切和移动图形是非常方便的。本章前面介绍的矩形编辑命令都可以用在图形模式里，图形模式还有几个特有的矩形编辑命令。

在图形模式里能够保存的矩形可就不止一个了，可以同时把多个矩形区域分别保存到不同的“寄存区（register）”里。寄存区是 Emacs 开辟的存储空间，当前 Emacs 会话中创建的寄存区将一直呆在内存里，直到这次会话结束。这种能够保存多个矩形区域的能力对复杂的编辑工作（比如重新安排一个5栏表格中各栏的排列顺序时）特别有帮助。如果想清除一个矩形区域，并把它保存到一个寄存区里，请输入“**C-c C-w** *r*”命令。命令中的“*r*”是用来保存该矩形区域的寄存区的名字，它必须是一个单字符，比如“1”。“*r*”的值可以取任意字符，数字、字母、大写字母和小写字母都可。也就是说，键盘上有多少个独一无二的字符，就能保存多少个寄存区！（不过，最好不要用键盘上的功能键来充当寄存区的名字，因为功能键往往代

表着一个字符序列，而这会让 Emacs 不知所措。）这个命令会在清除的矩形区域处留下一块空白。如果想删除一个矩形区域而不留下一块空白，请在这个命令的前面加上一个“**C-u**”，即使用“**C-u C-c C-w r**”命令。

要想插入一个保存起来的矩形区域，需要使用“**C-c C-x** *r*”命令。命令中的“*r*”是用来保存该矩形区域的寄存区的单字符名字。

另外一个图形模式专用的矩形编辑命令是“**C-c C-k**”（命令名是 **picture-clear-rectangle**）。用“**C-c C-k**”清除一个矩形区域，等于把矩形区域中的文本都转换为空格，在它的位置上留下一块空白。“**C-c C-y**”可以把用“**C-c C-k**”清除为空白的那个矩形区域原来的内容恢复回来。“**C-c C-k**”每次只能保存一个矩形区域的内容，大家可能还记得，“**C-c C-y**”也只能把最近一次删除的矩形区域的内容粘贴回来。

表 8-6 对图形模式里的编辑命令进行了汇总。

表 8-6: 图形模式编辑命令速查表

键盘操作	命令名称	动作
（无）	**edit-picture**	进入图形模式
C-c C-c	**picture-mode-exit**	退出图形模式，返回此前的编辑模式
C-c ^	**picture-movement-up**	把默认绘制方向设置为上
C-c .	**picture-movement-down**	把默认绘制方向设置为下
C-c >	**picture-movement-right**	把默认绘制方向设置为右
C-c <	**picture-movement-left**	把默认绘制方向设置为左
C-c `	**picture-movement-nw**	把默认绘制方向设置为左上
C-c '	**picture-movement-ne**	把默认绘制方向设置为右上
C-c /	**picture-movement-sw**	把默认绘制方向设置为左下
**C-c **	**picture-movement-se**	把默认绘制方向设置为右下
C-c C-f	**picture-motion**	沿默认绘制方向向前移动光标
C-c C-b	**picture-motion-reverse**	沿默认绘制方向向后移动光标

表 8-6: 图形模式编辑命令速查表（续）

键盘操作	命令名称	动作
C-f	**picture-forward-column**	把光标向右移动一个字符
C-b	**picture-backward-column**	把光标向左移动一个字符
C-n	**picture-move-down**	把光标向下移动一个字符
C-p	**picture-move-up**	把光标向上移动一个字符
C-d	**picture-clear-column**	把光标下的字符转换为一个空格，不把其右侧的文本向左移
C-c C-d	**delete-char**	删除光标下的字符，把其右侧的文本向左移
C-k	**picture-clear-line**	删除当前行上的文本，但即使连按两次也不会删除这一行
C-o	**picture-open-line**	插入一个空白行
C-c C-w r	**picture-clear-rectangle-to-register**	清除矩形区域的内容并把它保存到寄存区 *r* 里
C-u C-c C-w r	**picture-clear-rectangle-to-register**	删除矩形区域并把它保存到寄存区 *r* 里
C-c C-x r	**picture-yank-rectangle-from-register**	把保存在寄存区 *r* 里的矩形区域内容粘贴到当前位置

Emacs 的大纲模式

在编写书籍、大段的备忘录或者技术报告等长篇大论的时候，要想随时掌握其通篇的布局结构往往是很困难的。如果不需要调整段落次序就能把大纲充实为文章，那当然很理想；但实际情况往往并不会这么顺利，甚至可能会相当麻烦。内容和标题经常会交织在一起，很难既看得见树木，又看得见森林。

Emacs 的大纲模式（outline mode）为这一问题提供了一种内置的解决方案。这个编辑模式允许有选择地展开和隐藏文章的段落内容——以它与文档结构的关系为依据。比如，可以把文档的内容都隐藏起来而只留下文章的各级标题，以便对文档的整体布局有一个了解。在查看文章各级标题的时候，可以把注意力集中在文章的

布局结构上，而不必关心遣词造句的问题；在解决布局结构方面的问题后，再让文章的内容重新显示出来。

如果只是写一份文字量很少的大纲，大纲模式的优势并不明显；它更适用于编写包含多级标题（或者大段程序）的文档。文档越长，要想迅速掌握它的整体布局也就越困难；而越是在这种情况下，大纲模式能够有选择地隐藏和显示文章内容的功能对我们就更有帮助。

使用大纲模式的好处确实不少，但有一点需要大家特别注意：如果你的经验不够老到，大纲模式会从很多方面给你带来意想不到的问题，把文档弄得一团糟。最好的方法是用大纲模式对文档的一份副本进行编排（大家马上就会看到这很容易做到），而不是对原始文件进行编辑。这种使用副本进行工作的方法能够让你享受到大纲模式带来的好处，而原始文件却不会有任何风险。

大纲模式要求在大纲或者文档里遵守一些特殊的约定。在下面这个对照表里，我们把传统意义上的大纲列在左边，把符合大纲模式要求的同一份大纲列在了右边：

传统意义上的大纲	大纲模式
All About the Universe	All About the Universe
I. Preface	*Preface
A. Scope of book	**Scope of book
This book is all-inclusive	This book is all-inclusive
B. Intended audience	**Intended audience
Universe dwellers	Universe dwellers
II. Chapter 1	*Chapter 1
A. Universe basics	**Universe basics

传统大纲使用了一种层次化的布局结构，各级标题用罗马数字、大写字母、数字和小写字母等进行区分。而大纲模式里的各级标题在默认设置情况下是靠星号（*）来区分的，一颗星表示这是一个一级标题，两颗星表示这是一个二级标题，依次类推。不以星号打头的文本行（比如上表中的“This book is all-inclusive”）被称为“正文（body）”。Emacs要求星号必须出现在第一列里（可以使用传统大纲的缩进安排），但前提是星号必须从第一列开始出现。

上面给出的大纲示例里只有两个正文行。但既然是写书，正文行肯定会越来越多。就上表里的大纲示例而言，“This book is all-inclusive”这句话将被替换为一大段前言，而另一个正文行将被充实为这本书的第一章内容。如果使用得当，拟订大纲和书写正文这两种不同的工作，在大纲模式里并不会有明显的区别。随着大纲的逐步充实和细化，它也就逐渐演变为一篇文章（注 2）。

进入大纲模式

要想进入大纲模式，请输入“**ESC x outline-mode RETURN**”命令。状态行上将出现“Outline”字样。（大纲模式也可以作为一种副编辑模式使用，我们将在后面的内容里讨论这个问题。）

进入大纲模式后，可以通过几个特殊的命令，从大纲的某个部分快速移动到另外某个部分。比如，按“**C-c C-n**”组合键将移动到下一个标题或子标题；而“**C-c C-p**”组合键将移动到上一个标题。“**C-c C-f**”组合键将移动到同级的下一个标题——可以用这个命令从某个一级标题移动到大纲里的另外一个一级标题，或者从某个二级标题移动到另外一个二级标题（这两个二级标题必须在同一个一级标题下）。“**C-c C-b**”组合键则向回移动到同级的上一个标题。如果想从某个二级标题移动到它的一级标题，即在大纲的层次结构里上移一级，就需要按下“**C-c C-u**”组合键。（如果已经在某个一级标题上，那么“**C-c C-u**”将蜂鸣，因为此时已经没有上级标题可去了。）图 8-5 给出了这些光标移动命令在大纲示例上的移动效果。

这些命令使文章布局方面的许多问题都迎刃而解。如果经常遇到诸如“我知道自己正在写的是论据，可我记不起上一级的论点是什么”之类的事情，那么按下“**C-c C-u**”组合键就能返回到大纲的上一个层次。如果想了解该小节里的论据都是如何组织和安排的，可以用“**C-c C-b**”和“**C-c C-f**”组合键查看其前后的其他小标题。

注 2：　当然，应该在文章完成之后把那些星号去掉。如果使用的是 **troff** 或 TEX 之类的排版软件，就可以用一个查询－替换操作把星号风格的标题修改为与宏定义包定义风格相对应的标题。如果只是想在文章写好之后去掉文件里的星号，用一个正则表达式（将在第十三章介绍）替换就能达到目的。另外，如果你胸有成竹，可以先对大纲模式进行定制，让它从一开始就按文档排版软件所能识别的标题风格进行工作，如果选择了这条路，就可以把拟订大纲和书写正文两种功能完美地集成在一起。

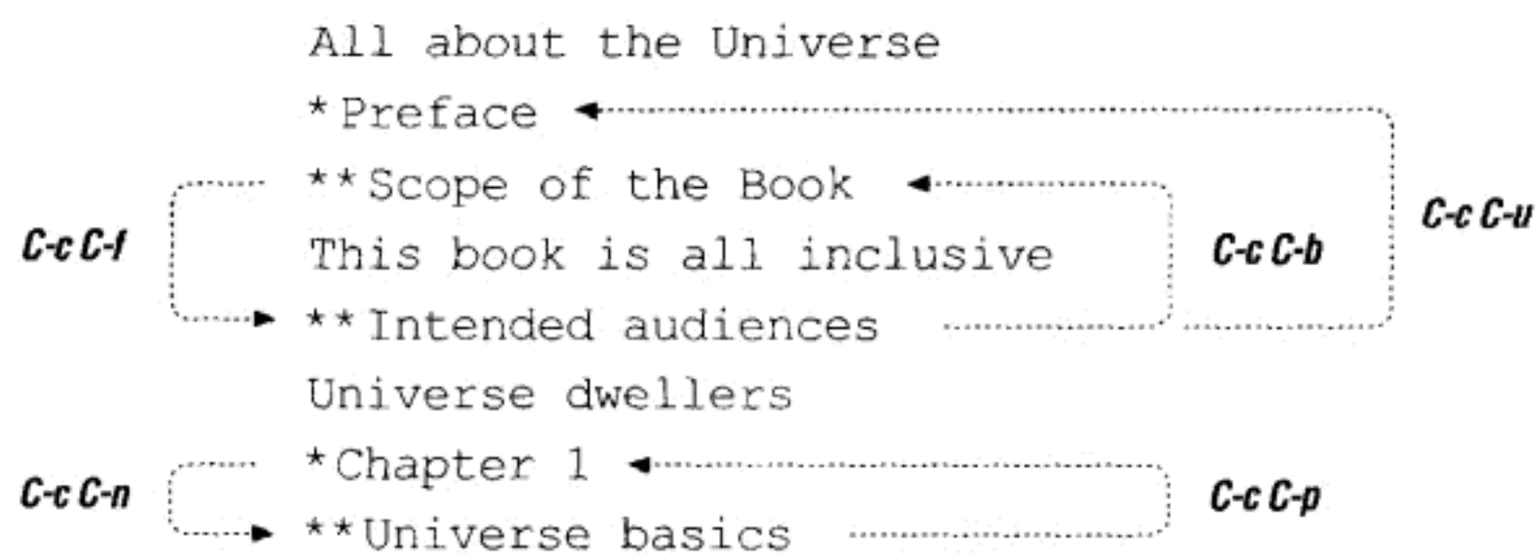


图 8-5: 在大纲模式里移动

文本的隐藏和显示

大纲模式最重要的功能之一是它能够有选择地隐藏和显示文本的不同部分。用大纲模式查看一份长文档的布局安排是它最出彩的功能 —— 把标题以外的所有东西都隐藏起来之后，就可以清楚地看到整个文档的布局结构，各小标题是否承上启下、它们是否需要重新安排等都一目了然。

hide-body 命令（“**C-c C-t**” 组合键，隐藏正文）听起来有点像侦探小说里“隐藏证据”的味道，它的作用是把除各级标题（即以星号打头的文本行）以外的正文（或文本）行都隐藏起来，不让它们显示在屏幕上。在隐藏正文的时候，Emacs 会给相应的标题行加上一个省略号(...)。这个省略号的作用是表明这里隐藏着一些文本。编辑缓冲区本身并没有被修改 —— 如果注意看状态行的左边，就会发现表明该编辑缓冲区已被修改过的星号并没有出现。如果在有文本被隐藏的时候对文件进行了存盘退出操作，Emacs会把显示在屏幕画面上的东西和被隐藏了的文本都保存起来；而下次读入这个文件的时候，Emacs 又会把被隐藏的文本都显示在屏幕画面里。

使用 **hide-body** 命令可以迅速查看到一份长文档的布局结构。可以先给文档复制出一份副本，然后在副本里用一个查找－替换操作给所有的一级标题加上一颗星号（*）标记，给所有的二级标题加上两颗星号标记，依次类推（注 3）。最后，按下

注 3: 请注意，首先进行最低一级标题的替换。如果有三级标题，要先给三级标题加上 *** 标记，然后是二级标题，最后是一级标题。要注意一级标题，文件中也许有不是标题的星号，如果使用正则表达式查找替换仅仅出现在行首的星号，就不会有什么问题了。

"**C-c C-t**"组合键查看不带文本内容的各级标题，请看下面的例子。我们将从刚才那个简单的大纲开始，先隐藏其正文，如下所示：

按下：**C-c C-t**

```
All About the Universe
*Preface
**Scope of book...
**Intended audience...
*Chapter 1
**Universe basics
```

正文被隐藏起来，原来的两个正文行现在用省略号表示。

要想查看所有被隐藏起来的文本 —— 不管它们是标题还是正文，请按下"**C-c C-a**"组合键（命令名是**show-all**）。**hide-body**和**show-all**这两个命令是惟一以整个大纲为操作对象的命令，其他命令（这些命令可就多了）则都是以光标所在的大纲部分为操作对象。用不着一口气把全部的命令都学会，先掌握几个最基本的，其余的可以到需要使用它们时再学。在大纲模式里做编辑工作的时候，使用**hide-body**和**show-all**命令的时间占百分之九十以上。

可以用**hide-subtree**命令［"**C-c C-d**"（注4）］组合键把某个标题的下级小标题和正文都隐藏起来。光标可以在这个标题上，也可以在其相关文本中的任何位置。接着以刚才的大纲为例子，准备把"Preface"分枝隐藏起来。如下所示：

按下：**C-c C-d**

```
All About the Universe
*Preface ...
*Chapter 1
**Universe basics
```

省略号表示有被隐藏起来的文本。

注意屏显画面的变化。发出**hide-body**命令的时候，原先的两个正文行不见了，取而代之的是两个省略号。发出**hide-subtree**命令的时候，大纲的某个分枝（包括其中的下级小标题）不见了，取而代之的是一个省略号。

注4：　如果使用的Emacs版本早于19.25，这个命令默认的按键绑定是"**C-c C-h**"。

“**C-c C-s**”组合键（命令名是**show-subtree**）的作用正好与**hide-subtree**相反。如果光标在某个标题上并且该标题下有被隐藏起来的文本，“**C-c C-s**”组合键将把所有被隐藏起来的文本都显示出来，不管它们是正文还是下级小标题。

“**ESC x hide-entry RETURN**”命令的作用是，把紧跟在某个给定标题下的文本隐藏起来，别的下级小标题及其文本都不受影响。要想显示用**hide-entry**命令隐藏起来的文本，请发出“**ESC x show-entry RETURN**”命令。

文本被隐藏时的编辑

知道如何隐藏和显示文本之后，我们再来看看被隐藏文本的某些特点。有时候，需要对含有隐藏文本的文档进行编辑。这是一种调整文档布局结构的重要手段，但大家必须高度重视这类编辑工作可能带来的隐患。假设已经用大纲模式把全体正文都隐藏起来了，只有各级标题还显示在屏幕画面里。换句话说，此时看到的是文档的一份真正的“大纲”。如果移动了一个标题而这个标题带有隐藏着的小标题和相关文本，那么所有被隐藏起来的东西也会随着可见文本的移动而移动。稍后，当再次“显示”全篇文档的时候，刚才隐藏的文本将出现在新位置上——就在刚才移动的那个标题之下。类似地，如果删除了一个标题，它下面隐藏的文本也将被删除掉。

这个功能使段落的移动工作很方便。但有些事情大家必须注意：如果删除了某个标题后面的省略号，Emacs就会删除相关的被隐藏文本。为了避免误操作，Emacs会限制进行这样的操作——它不允许用**DEL**键来删除省略号，也不允许用“**C-b**”之类普通的光标移动命令把光标移动到省略号上。如果想用“**C-d**”命令删除省略号，那么Emacs会把隐藏文本重新显示到屏幕上，而不是一声不吭地删除它们。但如果就是想这样做或者因为运气不好，在编辑过程中因删除省略号（比如，使用了“**C-k**”命令）而丢失文本的事情还是会发生的。虽然还可以用“**C-y**”命令把省略号恢复回来，但当重新显示文本的时候，有些东西可能已经丢失了。重新输入省略号并不能挽回丢失的信息，只能从自动保存（auto-save）文件（文件名是“*#filename#*”）或备份文件（文件名是“*filename~*”）里想法恢复它们。

对隐藏文本进行的编辑操作还有一个规定：如果想把隐藏着的文本移动到另外一个编辑缓冲区里，这两个编辑缓冲区就必须都在大纲模式下。如果试图把大纲里的隐藏文本移动到另外一个不在大纲模式下的编辑缓冲区里，就会看到来自大纲的文本

段落之间会夹杂着一些“^M”字符，有些文本甚至会丢失。如果把两个编辑缓冲区都放到大纲模式下，就能毫无问题地在这两个窗口之间移动文本。

最后，在文档里有文本被隐藏起来的同时，当然还可以添加新的标题和文本。但一定要小心，千万不能在省略号上进行输入，因为在省略号上的输入内容将会被插入到隐藏文本里。（既使能够预见到会发生什么样的事情，这种做法也不值得。）

使用大纲副模式

大纲模式也可以作为一个副编辑模式（minor mode）来使用。也就是说，完全可以在自己喜欢的主编辑模式（major mode）里使用大纲。以副编辑模式启动大纲模式的命令是“**ESC x outline-minor-mode RETURN**”，状态行上将出现“`Outl`”字样。从某些方面来说，这个编辑模式不是那么方便，大多数大纲命令前面的前缀将不再是一个简单的“**C-c**”。在大纲副模式下，必须给大纲命令都加上“**C-c C-o**”前缀，这样才能保证它们不会与主编辑模式里正常的“**C-c**”命令发生冲突。因此，如果想移动到下一个标题位置上去，就必须按下“**C-c C-o C-n**”组合键（相当于大纲模式里的“**C-c C-n**”命令）。

请注意，混合使用大纲主模式和大纲副模式不仅多余，而且有可能导致难以预料的后果。在某个编辑缓冲区进入大纲模式或者大纲副模式之后，就不要再画蛇添足了。在主模式里打开同功能的副模式（不仅限于大纲模式）反而会让 Emacs 不知所措。应该宁肯先保存文件，退出编辑缓冲区，然后再重新打开这个文件，也不应该让一个编辑缓冲区在大纲主模式和大纲副模式之间来回切换。

对大纲模式进行定制

现在，再来看看大纲模式的工作情况。（这部分内容比较晦涩难懂，如果你是一位 Emacs 新手，可以跳到下一小节去。）大纲模式由一个名为**outline-regexp**的变量控制，这个变量的值是一个正则表达式。这个正则表达式的作用是匹配任何一个标题行的开头部分。匹配到的字符串长度决定标题的级别。比如，变量 **outline-regexp** 的默认值是“`*+`”（它实际上会更复杂一些，但两者相差不多）。这个正则表达式匹配的是一个或者多个星号。大纲模式认为一颗星代表着一个一级标题，因为它匹配

到一个字符；两颗星代表着一个二级标题，因为它匹配到两个字符，依次类推。实际匹配到多少个字符并不重要，但更长的匹配永远代表着更低级别的标题。

如果各位足够聪明，就可以利用这个正则表达式让大纲模式适应自己的实际工作情况。请看下面这个正则表达式：

```
\\(\\.l1\\)\\|\\(\\.le2\\)\\|\\(\\.lev3\\)\\|\\(\\.level4\\)
```

够乱的吧？这个正则表达式本可以写得更紧凑些，不过那样就更难懂了。它使用了将在第十三章介绍的几个知识点。就这个正则表达式而言，它将匹配以下格式的标题：

.l1	一级标题
.le2	二级标题
.lev3	三级标题
.level4	四级标题

如果大家对 **troff** 比较熟悉，就会发现上面这些标题的格式与 **troff** 中的情况比较接近。现在，只需把上面这些标题转换为熟悉的 **troff** 标题宏（注 5），再把下面这条语句添加到“*.emacs*”文件里：

```
(setq outline-regexp " the-ugly-thing-you-just-saw ")
```

就可以用大纲模式来处理正规的 **troff** 文本。TEX用户也能构造出类似的东西。表 8-7 对大纲模式下的常用命令进行了汇总。

注 5：这个问题与本书内容无关，但完全可以这样做。它需要至少掌握一个 **troff** 小技巧：**troff** 软件里的宏名字只能有两个字符，但当它遇到类似于“.levle4”这样的东西时，它会把这个东西看做是对“.le”宏的一次调用，而“vel4”则是这次宏调用的第一个参数。也就是说，写一个名为“.le”的宏并让它测试其第一个参数是否是“2”、“v3”、“vel4”就可以达到目的。再顺便说一句，troff 的新版本（特别是自由软件基金会的 **groff**）已经允许使用两个以上的字符作为宏的名字。

表 8-7：大纲模式命令速查表

键盘操作	命令名称	动作
C-c C-n	**outline-next-visible-heading**	移动到下一个标题
C-c C-p	**outline-previous-visible-heading**	移动到上一个标题
C-c C-f	**outline-forward-same-level**	移动到下一个同级的标题
C-c C-b	**outline-backward-same-level**	移动到上一个同级的标题
C-c C-u	**outline-up-heading**	移动到上一级别的标题层次
C-c C-t	**hide-body**	隐藏全体正文行
C-c C-d	**hide-subtree**	隐藏某标题的全体下级小标题及其相关正文
（无）	**hide-entry**	隐藏某标题的正文部分（不包括它的下级小标题和那些小标题的正文部分）
C-c C-l	**hide-leaves**	隐藏某标题的正文部分，同时隐藏它全体下级小标题的正文部分（只留下标题）
C-c C-a	**show-all**	显示被隐藏起来的所有东西
C-c C-s	**show-subtree**	显示某给定标题的下级小标题和它的相关文本
（无）	**show-entry**	显示某给定标题的相关文本（不包括它的下级小标题及它们的正文部分）
C-c C-k	**show-branches**	显示某标题的正文及其全体下级小标题的全部正文
C-c C-i	**show-children**	显示某标题的下一级小标题（不包括正文文本）

第九章

用Emacs设置排版标记

本章内容：

- 设置troff和nroff排版标记
- 设置TEX和LATEX排版标记
- 编写HTML文档
- Emacs的Html-helper模式

上一章对Emacs能够实现的几种基本排版操作进行了介绍。要想用Emacs实现更专业的排版，就必须给文本加上一些特殊的标记——排版代码，然后用一个文本处理软件再对它进行处理。

nroff、**troff**、TEX和LATEX等文本处理软件会根据文本文件里的排版标记对文本进行处理，然后在激光打印机上产生一份打印输出。这本书就是用自由软件基金会提供的一种名为**groff**的工具软件（它是**troff**家族中的一员）生成的。

生成高质量的打印文稿是标记语言的用途之一，而WWW又为标记语言提供了另外一个广阔天地。程序员用HTML（Hypertext Markup Language，超文本标记语言）给ASCII文本加上排版标记，然后把文件放到WWW服务器上。世界各地的用户们再用一种名为浏览器的工具软件去浏览这些文件的内容。浏览器以加有排版标记的文件为输入，以专业效果（虽然有一些高低之分）的格式把它们显示在用户的显示器上，文件的显示效果取决于浏览器的能力。常见的UNIX浏览器包括Mosaic、Netscape Navigator、Lynx以及Emacs下的W3等等。

为这些种类繁多的文本和超文本处理系统创建标记文件并不困难，有标准配置的Emacs就足以完成这项工作了。这一章介绍的几个编辑模式能大大简化给文件添加排版标记这项相当琐碎和麻烦的工作，很值得学习。

有些字处理软件和其他工具把排版和编辑两种功能集成在一起。这类软件通常被称为“所见即所得”（WYSIWYG，英文“what you see is what you get”的缩写）工具。但这类工具的文件格式往往不互相兼容。如果想把文件传递到另外一个系统上去，得先设法把它的排版标记去掉。Emacs是一种ASCII文本编辑器，它不是一个“所见即所得”工具。

在这一章里，我们将探讨以下几种排版工具：

- **troff**是UNIX世界传统的文本排版程序。**nroff**的功能与**troff**相当，主要用来对字符终端的输出进行排版，**man**命令用的就是它。**groff**来自自由软件基金会，与**troff**和**nroff**的功能相当。
- TEX这个排版软件的作者是Donald Knuth，主要用来排版图书，它是我们将要介绍的文本处理软件中最精巧的一个。LATEX是一套由Leslie Lamport开发出来的TEX命令。TEX和LATEX可以从美国数学学会（American Mathematical Society）获得，但需要支付发行费用。
- Html-helper是Emacs中一种专门用来给WWW网上的文件添加排版标记的主编辑模式。编写的HTML文档可以用Netscape或者Emacs的W3等浏览器来查看，关于W3的使用方法请参考第七章里的介绍。

Emacs有各种能够帮助在文本里插入排版命令［或者叫“标记（markup）”］的主编辑模式。虽然Emacs提供的帮助有多少之分，但使用专为文本排版工具而设计的编辑模式肯定会使工作更加顺畅。

注释

这一章所介绍的编辑模式与将在第十二章中介绍的C模式（C mode）和FORTRAN模式（FORTRAN mode）等程序设计语言编辑模式有一个共同的特点：即都允许你在文档里使用注释，而且都使用同一个键盘命令“**ESC ;**”（命令名是**indent-for-comment**）插入正确的注释语法。表9-1列出了本章各种排版工具的注释语法，如下所示。

表 9-1：各种标记模式下的注释

如果在下列模式里敲入 "ESC ;"	Emacs 将插入
nroff 模式	.\"
TEX 模式	%
LATEX 模式	%
html-helper 模式	<!-- -->
	<!-- -->

设置 troff 和 nroff 排版标记

除了多出一些操作上的方便以外，**nroff**模式与文本模式毫无二致。就将要探讨的问题来看，**nroff**与它的近亲**troff**和**groff**之间没有任何差别；在这一节里介绍的命令对整个 **troff** 家族都适用。

要想进入 nroff 模式，请输入“**ESC x nroff-mode RETURN**”命令。状态行上将出现“`Nroff`”字样。nroff 模式提供以下 4 项功能：

- 它修改了 **paragraph-separate** 变量的正则表达式，重新定义了“段落”的含义。修改后的设置情况将保证 **fill-paragraph** 命令（“**ESC q**”组合键）能够正确地工作。
- 它提供了一些用来在 **troff** 文档里漫游的特殊命令。这些命令能够以文档中的 **troff** 命令为跳跃点做前后移动。
- 一个特殊的“nroff 配对模式”（electric nroff mode，一个与 nroff 模式相关联的副模式）能自动提供某些排版命令的收尾标记，确保它们能配对出现。
- 它提供了一些用来在文档中插入注释的操作命令。

段落的分隔与段落重排

我们将依次对上述 4 项功能进行介绍。先来看看“段落”的重新定义问题。如果经常做大量的文本编辑工作并用惯了“**ESC q**”组合键（命令名是 **fill-paragraph**），就一定要提防 **troff** 文档中的“段落”陷阱。Emacs 把空白行视为段落的结束标志，但**troff**文档却往往通篇没有一个空白行。因此，Emacs 将会因把整个文件当做一个

段落来进行重排而弄出大麻烦。[幸亏Emacs 19里的**undo**命令（“**C-x u**”组合键）的功能很强，足以恢复这种混乱的局面。]

troff排版命令（通常被称为“请求（request）”或者“宏（macro）”）必须从左页边开始输入，而且必须以一个句点（.）或单引号（′）打头。不管**fill-paragraph**命令干了些什么，它也不能改动这些行。因此，除了普通的段落定义（以空白行分隔的文字块）之外，nroff模式还将把以一个句点，或一个单引号打头的文本行定义为一个段落。举个例子，请看下图中的文本：

初始状态：

```
Buffers Files Tools Edit Search Help
.H1 "Various Ways of Questioning about the Thing"
.H2 "Philosophical and Scientific Questioning"
.PP
From the range of the basic questions
of metaphysics we shall
here ask this
.I
one
.R
question:What is a thing?
.PP
-----Emacs:heidegger          (Nroff Fill)--L5--All----------------------
```

这是一些为**troff**准备的文本。要想对第一段（“From the range”）进行重排，请把光标移动到这个段落的中间并按下“**ESC q**”组合键。如果是在其他编辑模式里，这个命令会把整个文件重排为一个很长的段落。但在nroff模式里，“**ESC q**”组合键将只对两个宏定义标记之间的一小段文本进行重排。

按下：**ESC q**

```
Buffers Files Tools Edit Search Help
.H1 "Various Ways of Questioning about the Thing"
.H2 "Philosophical and Scientific Questioning"
.PP
From the range of the basic questions of metaphysics we shall here ask
this
.I
one
.R
question:What is a thing?
.PP
-----Emacs:heidegger          (Nroff Fill)--L4--All----------------------
```

Emacs对文本进行段落重排，只有夹在两个宏定义标记中间的那个段落发生了变化。

如上所示，Emacs没有重排整个文档，甚至没有重排整个段落，它只对夹在“.PP”宏定义标记和“.I”宏定义标记之间的那一小部分文本进行了重排。

在文档中移动

如果工作文件里散布着很多**troff**排版命令，想在其中从某个地方移动到其他地方有时候会很让人着急，必须跳过文件里的宏定义标记，可它们好像总拦在路上。nroff模式提供了一种简便的方法，使得能够跳过两个文本行之间的一切宏定义标记从上一行直接移动到下一行。“**ESC n**”组合键将把光标移动到下一个文本行，“**ESC p**”组合键将把光标移动到上一个文本行。如图9-1所示。

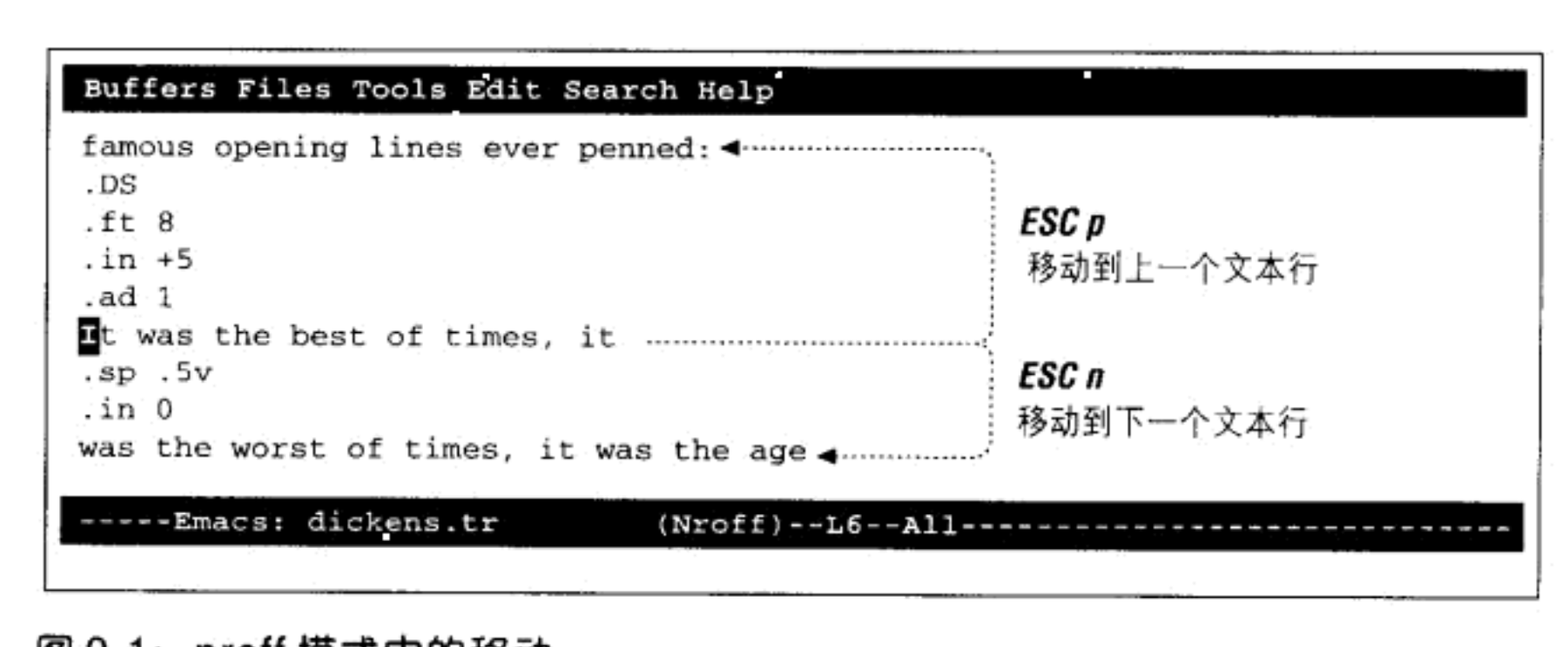


图9-1：nroff模式中的移动

排版标记要单独占用一个文本行，所以标记文件看上去好像加长了很多。nroff模式专门提供了一个命令来统计“真正”的文本行有多少。“**ESC ?**”命令就是用来统计某个给定的文本块里（不包括排版标记占用的那些行）到底有多少行文本的。如果想知道文件里总共有多少个文本行，可以先用“**C-x h**”组合键把整个编辑缓冲区选取为一个文本块，再用“**ESC ?**”命令统计出其中的文本行个数。（用来创建各种标题的标记语句将包括到文本行的统计当中，因为它们将以文本行的面目出现在打印稿里。）

分节符的定义和使用

重新定义Emacs的**page-delimiter**变量，以便能够对**troff**的分节符进行查找费不了多大的事，但对工作却会有很大的帮助。这个变量是一个正则表达式，它对页面的

分隔符进行了定义；诸如“**C-x]**”（命令名是**forward-page**）之类的命令就是靠**page-delimiter** 变量来判断下一页是从什么地方开始的。在默认的情况下，变量 **page-delimiter** 查找的是换页符（即“**C-l**”字符），但可以重新设置这个变量，让它查找 **troff** 的分节符。

举个例子。假设正在使用ms宏定义包。用ms宏定义包设置的文档会使用宏定义标记“**.NH**”来开始一个新的节（section）。于是，把 **page-delimiter** 变量设置为 "**^ \\ .NH**" 就等于是告诉 Emacs 出现在某行开头的“**.NH**”标记表示将从这里开始一个新的打印页。做了这样的设置之后，就可以用翻页命令在**troff**文档的节之间移动了。

要想对 **page-delimiter** 变量做这样的设置，请输入“**ESC x set-variable RETURN page-delimiter RETURN "^ \\ .NH" RETURN**”命令。如果想让这个设置固定下来，就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq nroff-mode-hook '(lambda () (setq page-delimiter "^\\.NH")))
```

如果使用的宏定义包不是ms，就得自己想法写出**page-delimiter**变量用的正则表达式。对正则表达式的详细介绍请参考第十三章。

注意：**page-delimiter** 变量并不是一项 nroff 模式独有的功能。不管编辑哪类文档，都可以使用类似的技巧。

宏定义标记的配对出现

nroff 配对模式（Electric nroff mode）又多提供了一项功能。**troff** 的宏定义标记经常是配对出现的，一个起始标记对应着一个配对的收尾标记。比如“.TS”表示某个表格从此处开始，而“.TE”则表示这个表格到此处结束。在nroff配对模式里，Emacs 会自动插入一组必须配对出现的宏定义标记中的收尾标记，只需输入其中的起始标记即可。

要想进入nroff配对模式，首先要进入nroff模式，然后输入“**ESC x electric-nroff-mode**”命令。事实上，nroff 配对模式只是一个给“正宗的”nroff 模式增加了新功能的副模式，本身不是一个独立的主模式。因此，不能直接进入 nroff 配对模式。

进入nroff配对模式之后，再遇到需要输入一组必须配对出现的宏定义标记时，只需先输入其中的起始标记再按下“**C-j**”组合键（或**LINEFEED**）即可。Emacs会自动插入其收尾标记，并且会在它们之间插入一个空白行；然后，Emacs将把光标放在这两个宏定义标记之间的空白行上。不用移动光标就可以继续输入文本——这种安排特别适合那些喜欢一边写作一边设置排版标记的人们使用。下面的画面给出了用nroff配对模式输入一组需要配对出现的宏定义标记“.DS”和“.DE”的情况：

输入：**.DS**

```
Buffers Files Tools Edit Search Help
.DS

--**-Emacs:nroff          (Nroff Fill Electric)--L2--All----------
```

输入一组必须配对出现的两个命令中的起始命令。

按下：**C-j**

```
Buffers Files Tools Edit Search Help
.DS

.DE

--**-Emacs:nroff          (Nroff Fill Electric)--L2--All----------
```

Emacs插入与之配对的结束命令，并把光标放在它们的中间。

nroff配对模式可以自动补足由-ms、-me和-mm等宏定义包所定义的各组配对标记，一些通常配对出现的**troff**指令也包括在内。表9-2列出了nroff配对模式能够识别和自动补足的各组必须配对出现的**troff**宏定义标记。

表9-2：nroff配对模式支持的配对命令组

起始标记（由用户输入）	收尾标记（由Emacs自动插入）	使用情况
.(b	.)b	me宏定义包
.(l	.)l	me宏定义包
.(c	.)c	me宏定义包

表 9-2: nroff 配对模式支持的配对命令组（续）

起始标记（由用户输入）	收尾标记（由 Emacs 自动插入）	使用情况
.(x	.)x	me 宏定义包
.(z	.)z	me 宏定义包
.(d	.)d	me 宏定义包
.(q	.)q	me 宏定义包
.(f	.)f	me 宏定义包
.LG	.NL	ms 宏定义包
.LD	.DE	ms 宏定义包
.CD	.DE	ms 宏定义包
.BD	.DE	ms 宏定义包
.DS	.DE	ms 宏定义包
.DF	.DE	ms 宏定义包
.FS	.FE	ms, mm 宏定义包
.KS	.KE	ms 宏定义包
.KF	.KE	ms 宏定义包
.LB	.LE	mm 宏定义包
.AL	.LE	mm 宏定义包
.BL	.LE	mm 宏定义包
.DL	.LE	mm 宏定义包
.ML	.LE	mm 宏定义包
.RL	.LE	mm 宏定义包
.VL	.LE	mm 宏定义包
.RS	.RE	ms 宏定义包
.TS	.TE	tbl 预处理器
.EQ	.EN	eqn 预处理器
.PS	.PE	pic 预处理器
.BS	.BE	mm 宏定义包
.na	.ad b	**troff** 指令
.nf	.fi	**troff** 指令
.de	..	**troff** 指令

PDG

nroff 模式中的注释

下面再来看看nroff模式里的注释是如何工作的。如果出现在文本行行首的3个字符是“.\"”，那么这一行就将被整个看做是一句注释（注释不会出现在打印稿里）。如果在文本行的半中间出现了“\"”这两个字符，则它们后面的内容就会被看做是注释。就我们的使用经验看，nroff模式中的注释似乎有些小问题，但大家最好还是应该练习一下这项功能的使用——别看它有毛病，你可能会发现它很有用呢。

假设已经启用了自动换行模式（auto-fill mode，这是一个副模式）。然后在输入过程中，决定在文本行上插入一句注释。Emacs监测到输入了一个注释标记（\"）。当输入的东西超出了右页边时，Emacs将自动开始一个新行；但它会认为在新行上输入的东西还是注释。为了让工作更顺畅，Emacs将把注释标记“\"”自动插入到新行的行首。Emacs只会在新行的行首自动插入注释标记。如果不在自动换行模式里，或者是按下了回车键，那么它就不会做任何特殊的事情。

遗憾的是，Emacs漏掉了那个打头的英文句号。这将导致**troff**在打印输出里额外留出一个空白行。除了回过头去动手插入那个英文句号外，我们不知道还有什么其他明显的办法能够纠正这一问题。也许定义一个宏（让它使用一个正则表达式查找与替换操作）来自动纠正插入在行首处的注释标记会更简单一些。（我们把它留给大家作为一个练习，详细情况请参考第十章。）

命令“**ESC ;**”（命令名是**indent-for-comment**）能够在当前行的末尾自动插入一个注释标记（\"）。如果这一行上没有别的东西，那么**indent-for-comment**命令的行为将非常奇怪：第一次按下“**ESC ;**”组合键的时候，它会在行首插入“\"”这两个字符；第二次按下“**ESC ;**”组合键的时候，它会把刚才的注释标记替换为“. \"”（注意：在句点和反斜线之间有一个空格）；第三次按下“**ESC ;**”组合键的时候，它才会变为“.\"”。也就是说，如果想在一个空白行上设置一个传统的nroff注释标记，就必须在该空白行上连按三次“**ESC ;**”组合键。如果不喜欢反复多次地按这个组合键，可以把它绑定为一个一次就能解决问题的组合键。

把 nroff 模式设置为默认的主编辑模式

如果需要频繁使用**troff**，那么把nroff模式设置为默认的主编辑模式可能更方便一些。nroff模式完全可以看作额外多出几条命令的文本模式，它几乎能胜任各项工

作。如果想让nroff模式成为默认的主编辑模式，请把下面这条语句添加到“*.emacs*”文件的最开始：

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

如果“*.emacs*”文件里有把默认的主编辑模式设置为其他模式（比如文本模式）的语句，记得要删除它。为了让这条新语句生效，需要保存这个文件并重新进入Emacs。表 9-3 对 nroff 模式里的各种命令进行了汇总。

表 9-3：nroff 模式命令速查表

键盘操作	命令名称	功能
（无）	**nroff-mode**	进入 nroff 模式
ESC n	**forward-text-line**	把光标移动到下一个文本行
ESC p	**backward-text-line**	把光标移动到上一个文本行
ESC ?	**count-text-lines**	统计文本块中的文本行数
（无）	**electric-nroff-mode**	进入一个副编辑模式。在这个模式里，输入必须配对出现的 **nroff** 命令组中的第一个，按“**C-j**”组合键，Emacs 就会自动插入该配对命令组中的第二个命令
C-j	**electric-nroff-newline**	只能用在nroff配对模式里，自动插入一组必须配对出现的 **troff** 宏定义标记中的第二个
ESC ;	**indent-for-comment**	在文本里插入一个注释标记

设置 TEX和 LATEX排版标记

GNU Emacs 对TEX文件的标记支持功能，比它对**troff**文件的标记支持功能更全面。TEX模式和LATEX模式都是主编辑模式，只要启用了它们，各种专用操作就会生效，帮助用户在Emacs里对这些文件进行标记和排版处理。TEX模式的许多功能也能用在LATEX模式里。LATEX用户应该在学完这一小节之后，再去学习在LATEX模式里增加的其他功能。

进入TEX模式的方法是输入“**ESC x tex-mode RETURN**”命令。Emacs将检查文件是否像是一个TEX或LATEX文件，并进入正确的编辑模式里。如果想特别指定TEX

模式，请输入“**ESC x plain-tex-mode RETURN**”命令。类似地，如果想启动LaTeX模式，请输入“**ESC x latex-mode RETURN**”命令。

括号的配对出现

TeX命令的常见格式是“\keyword{text}”。TeX模式不会特别注意所用的关键字是否“正确”，这是因为TeX本身是可扩展的，允许用户定义自己的关键字。不过，它确实提供了一种用来避免出现常见TeX错误的功能，如花括号（{ }）和美圆符号（$$）不配对。

在TeX里，花括号（{ }）和美圆符号（$$）永远要配对出现；Emacs会检查每个左花括号和美圆符号是否都有对应的配对符号。输入一个右花括号或美圆符号时，光标将迅速移动到与之对应的配对符号处（前提是配对符号也在屏幕上。如果配对符号不在屏幕上，它就会把左、右括号之间的上下文显示在辅助输入区里），然后再回到原处。这种可视性的检查工作对光标在文件中的位置没有影响。如果在光标移向左括号的同时继续输入，那么输入的文本将出现在正确的位置上，它不会把字符输入到错误的地方（注1）。

花括号经常会多个嵌套在一起，所以光标可能不会移动到距离最近的左括号处，而是移动到与这个右括号相对应的那个左括号处。举个例子，请看下面这个嵌套着3重花括号的文本行：

```
\keyword{this is some  \boo{funny text with  \bar{many} nested} braces}
         1                 2                     3    3        2        1
```

即使找不到配对的括号，Emacs也不会报警；而且，Emacs也无法肯定它找到的括号是否正是那个“正确”的配对括号。用户必须用自己的眼睛去观察，这样才能确保Emacs找到了另一半括号，而且它找到的那个括号是正确的。Emacs也不会在整个编辑缓冲区的范围里去搜索配对括号——这样做太花时间了。它只向回搜索一

注1：“向回移动到配对符号的起始元素处”是Emacs在遇到圆括号和方括号时的标准动作（这一问题的详细探讨请参考第十二章）。尽管圆括号和方括号与TeX的排版工作没有什么瓜葛，Emacs也同样会在这个编辑模式里检查圆括号和方括号的配对情况。因为Emacs自己的出错信息通常是笼统地说“括号不配对”，所以在此后的讨论中，我们也将经常会用“括号”一词来笼统地表示花括号、美圆符号和圆括号。

个有限的距离，如果找不到，就放弃搜索。因此，如果配对括号远在文件的其他部位，在输入右括号时可能就看不到左括号。

Emacs 能够帮助生成配对的括号。命令“**C-c {**”将在文本里插入一对左、右括号，然后把指示打字位置的输入光标放到两个括号的中间。如下所示：

按下：**C-c {**

```
Buffers Files Tools Edit Search TeX Help
Blah, blah, blah blah{}

--**-Emacs: texblahs          (TeX Fill)--L1--All----------
```

“**C-c {**”组合键插入一对花括号。

按下“**C-c }**”组合键会把光标移过右括号。它总是能根据当前位置找到正确的右括号。请看下面这个例子。假设光标在单词“*funny*”上时按下了“**C-c }**”组合键，光标将移动到对应的右括号（即单词“*nested*”后面的那个右括号）的后面。如下所示：

按下：**C-c }**

```
Buffers Files Tools Edit Search TeX Help
\keyword{this is some \boo{funny
text with \bar{lots of} nested}braces}

--**-Emacs: funnies           (TeX Fill)--L2--All----------
C-c }
```

“**C-c }**”组合键跳出花括号的当前层次。

如果光标原来在单词“*boo*”上或者更靠前的地方，“**C-c }**”组合键将把光标移动到第二行的最后一个括号的后面。如果 Emacs 没有找到配对的右括号，就将显示一条“unbalanced parentheses（括号不配对）”的出错信息。如果光标所在的段落并没有被括号包围起来而你却按下了“**C-c }**”组合键，就也会看到这条出错信息——这多少有点容易引起误会。

要想检查是否有不配对的花括号和美圆符号，请输入“**ESC x validate-tex-buffer RETURN**”命令。这个命令会在整个编辑缓冲区的范围内检查是否有尚未配对的圆

括号、花括号、美圆符号或者其他类似的符号。如果找到这类错误，Emacs将打开一个“*Occur*”编辑缓冲区，其顶部显示“Mismatches:(不配对)”字样，然后是Emacs找到的出错文本行的清单。可以利用“**ESC x goto-line**”命令迅速到达出现错误的每一行。

有时候，编辑缓冲区里一个比较靠前的不配对括号会因连锁反应，而造成文件里出现大量“错误”的局面。如果认为某个改正也会因连锁反应，而把剩余的大部分错误都“改正”过来，可以随时再执行一次 **validate-tex-buffer** 命令。

逐个改正错误的时候，刚才介绍的“**C-c }**”组合键可以帮助查明与某给定左括号正确配对的右括号到底在什么地方，只需你把光标放在左括号的后面再按下“**C-c }**”组合键即可。

引号和段落分隔

TEX模式还有一项专门对引号和段落分隔符进行处理的功能。在TEX或LATEX模式里输入双引号字符（"）将使Emacs插入左双引号（``）和右双引号（''）。这些字符其实是两个用来模拟左双引号和右双引号的单引号；左、右双引号不是ASCII字符集里的标准字符。如果确实需要输入一个双引号（"），可以在双引号字符前加上一个字符引用命令，即按下“**C-q "**”组合键（命令名是 **tex-insert-quote**）。

按“**C-j**”（或者 **LINEFEED**）组合键可以正确插入TEX和LATEX的段落分隔符——两个 **RETURN** 字符。它同时检查前一段落里是否存在有不配对的括号。比如，如果在按下“**C-j**”组合键的时候，前一个段落里还有个不配对的左括号——即少了一个右括号，那么Emacs会报告:

```
Paragraph being closed appears to contain a mismatch
```

这种警告可以让用户一边写作一边对段落进行检查，用不着等到最后再对整个编辑缓冲区做全面检查，这有助于做到有错即纠。

注释

在TEX模式里，按下“**ESC ;**”组合键将在当前行的末尾插入一个注释标记（一个百分号“%”）。

文本的排版和打印

在不退出Emacs的前提下，不仅能够给文件添加TEX排版标记，还可以对它们实际进行处理并查改设置错的排版标记。用来对整个编辑缓冲区进行排版处理的键盘动作是按下“**C-c C-b**”组合键（命令名是**tex-buffer**）。原本应该显示到屏幕上的信息将被引导到一个名为“`*tex shell*`”的编辑缓冲区，Emacs会把这个编辑缓冲区显示在屏幕上。如果没有在屏幕上看到这个编辑缓冲区，请按下“**C-c C-l**”组合键（命令名是**tex-recenter-output-buffer**），这将使它自动显示在屏幕上。

对TEX编辑缓冲区的部分内容进行排版处理需要一个特殊的技巧。TEX文件有一个专门用来完成各种初始化设置工作的文件头，随后才是包含文本和排版标记的正文部分。因此，为了对编辑缓冲区的部分内容进行排版处理，Emacs先得找到文件头，然后先把它发送给TEX排版软件，再把选取的文本块（即准备对之进行排版处理的那部分文件内容）发送过去。为了让Emacs能够找到文件头，需要把文件头放在下面这两个字符串之间：

```
%**start of header
%**end of header
```

上面两个字符串都是以百分号（%）打头的，而这个符号在TEX里是注释语句的语法，所以这两行语句不会影响文件的排版处理工作。

请看下面这个例子。对一个典型的TEX文件来说，文件头将对文件里用到的各种关键字（比如字体设置等）进行定义，随后才是加有排版标记的文本。在插入上面这两个字符串之后，整个文档的布局应该是下面这样的：

```
%**start of header
TeX header stuff
%**end of header
Regular text
The region you want to print
More regular text
```

对TEX文件来说，这两个字符串会告诉Emacs文件头在什么地方。在打算打印的文本块的一端设置上文本块标记，再把光标移动到文本块的另一端，然后按下“**C-c C-r**”组合键（命令名是**tex-region**）。如果在排版工作完成之前又想取消它，请按下“**C-c C-k**”组合键（命令名是**tex-kill-job**）。

用这些排版命令处理 TEX（或 LATEX）文件会生成一个“*.dvi*”文件，这是一种独立于具体打印设备的过渡性文件；如果想把它打印出来，还需要把它再进一步转换为打印机的执行命令。要想把“*.dvi*”文件打印出来，需要使用“**C-c C-p**”组合键（命令名是 **tex-print**）；它将完成对“*.dvi*”文件的转换并把它送到默认打印机去。“**C-c C-q**”（命令名是 **tex-show-print-queue**）组合键可以把打印队列显示出来，这样用户就能知道自己该在什么时候去打印机那里查看排版后的打印输出稿了。

有几个重要的变量会告诉 Emacs 如何来打印一份 TEX 文件。如果“**C-c C-p**”或者“**C-c C-q**”组合键工作不正常，就得检查它们的设置情况；如果这些命令根本就不工作，那么系统上的 TEX 软件的配置可能不合标准，或者是打印命令、打印队列命令与普通情况稍有差异。变量 **tex-command** 负责确定用来运行 TEX 软件的命令，它的默认值在 TEX 模式里是“*tex*”，在 LATEX 模式里是“*latex*”。变量 **tex-dvi-print-command** 负责确定用来打印“*.dvi*”文件的命令；它的默认值是“`lpr -d`”。再来看看打印队列，用来显示打印队列的命令是由变量 **tex-show-queue** 控制的。在默认的情况下，**tex-show-queue** 变量被设置为“lpq”。这些默认设置都是比较合理的，但 TEX 的配置情况变化非常大。因此，如果各位的 TEX 模式需要进行一些定制才能正确工作，请不必感到惊讶。

LATEX 模式与 TEX 模式的差异

LATEX 是 TEX 的一种简化版，推出 LATEX 的目的是想用 TEX 软件模仿商用软件 Scribe。对 TEX 模式的介绍全部适用于 LATEX 模式，所以在学习这一小节之前，应该先看看介绍 TEX 模式的上一节内容。LATEX 模式多了一项功能：可以用“**C-c C-e**”组合键插入一组配对命令中的第二个命令。请看下图。

输入：**\begin{document} RETURN C-c C-e**

```
Buffers Files Tools Edit Search TeX Help
\begin{document}
\end{document}
--**-Emacs: *scratch*        (LaTeX Fill)--L3--All----------
```

如果输入 LATEX 一组配对排版命令中的第一个命令后再按下“**C-c C-e**”组合键，Emacs 将自动插入这组配对命令中的第二个命令。

"**C-c C-e**" 组合键适用于任何关键字。因为 LaTeX（类似于 TeX）是可定制的，所以Emacs无法判断使用的关键字是不是一个有效的LaTeX关键字，甚至无法判断是不是对它进行过定义。举个例子，如果输入的是"**\begin{eating} C-c C-e**"，Emacs就会插入"**\end{eating}**"。使用的关键字是否合法有效责任全在于用户自己。

如果想打印输出某个LaTeX编辑缓冲区中的部分内容，则不必给文件头加上任何特殊的标记。LaTeX文件头格式相当统一，所以Emacs不需要任何帮助就能找出文件头的起始和结束位置。

表 9-4 对 TeX和 LaTeX命令进行了汇总。

表 9-4：TeX和 LaTeX命令速查表

键盘操作	命令名称	功能
（无）	**tex-mode**	根据文件内容进入 TeX模式或 LaTeX模式
（无）	**plain-tex-mode**	进入 TeX模式
（无）	**latex-mode**	进入 LaTeX模式
C-j	**tex-terminate-paragraph**	插入两个硬回车（标准的段落结束标记）并检查段落的语法
C-c {	**tex-insert-braces**	插入两个花括号，然后把光标放在它们中间
C-c }	**up-list**	如果光标在两个花括号中间，就把光标移到紧跟右括号的后面
（无） *TeX → Validate Buffer*	**validate-tex-buffer**	检查编辑缓冲区有无语法错误
C-c C-b *TeX → TeX Buffer*	**tex-buffer**	对编辑缓冲区进行 TeX或 LaTeX排版处理
C-c C-l *TeX → TeX Recenter*	**tex-recenter-output-buffer**	把排版信息画面显示在屏幕上，（至少）会显示最后一条出错信息
C-c C-r *TeX → TeX Region*	**tex-region**	对文本块进行 TeX或 LaTeX排版处理
C-c C-k *TeX → TeX Kill*	**tex-kill-job**	中断 TeX或 LaTeX的排版处理工作

表 9-4：TEX和 LATEX命令速查表（续）

键盘操作	命令名称	功能
C-c C-p *TEX → TEX Print*	**tex-print**	打印 TEX 或 LATEX的排版输出
C-c C-q *TEX → Show Print Queue*	**tex-show-print-queue**	查看打印队列
C-c C-e	**tex-close-latex-block**	只能用在LATEX模式里，自动插入某个命令组中的第二个命令

编写 HTML 文档

随着WWW的日益流行，许多人正在用HTML语言给文档加上Web格式的排版标记。HTML文档的内容都是些简单的ASCII文本，但里面加有各种用来定义文本显示特性的标签（tag）。HTML文档写起来并不困难，Emacs自己就已经足够写HTML标签和文本用的了。HTML标签通常采用下面这样的格式：

```
<tagname>text being tagged</tagname>
```

在Emacs里编写HTML还有几个增值模式可供使用，包括html模式、html-helper模式、html菜单编辑模式，以及包括SGML模式和psgml模式等在内的各种SGML工具等。（HTML是SGML的一个具体实现。SGML是Standard Generalized Markup Language（标准化通用标记语言）的缩写。）我们将从这么多的工具里挑出html-helper模式介绍给大家。

Html-helper模式允许输入任何标签，不管它属于HTML 2.0、HTML 3.0还是由某第三方创造出来的一个扩展。但也正是由于它这种兼收并蓄的特性，html-helper模式不适合用来检查HTML文档的语法错误。检查HTML文档最好的方法是用几种流行的浏览器来浏览它们。

我们并不打算在以下各个小节里教大家如何去编写HTML。（关于编写HTML的进一步资料，请参考《HTML: The Definitive Guide》一书，它的作者是Chuck Musciano和Bill Kennedy，由O'Reilly & Associates出版公司出版。）我们的目的是把使用html-helper模式编写HTML文档的基本方法介绍给大家。

Emacs 的 Html-helper 模式

Html-helper模式是由Nelson Minar开发的，它能以非常灵活的方式编写HTML；根据不同的经验和个人爱好，在启用或者禁用各种帮助功能方面，每个用户有充分的选择余地。它提供了大量用来输入HTML配对标签的键盘命令；而且，如果给它们加上修饰（办法是先按下“**C-u**”组合键），它们就能自动把标签加到文本块的两端而不是把它们放到光标位置上。作为键盘命令的一种替代品，html-helper模式还支持自动补足功能。只需输入标签的头几个字符，再按下“**ESC TAB**”组合键（命令名是**tempo-complete-tag**）就能把它输入到文档里。如果可用的选择不止一个，那么Emacs会把各种可能的补足情况列在一个窗口里以供挑选。

Html-helper模式不包括在Emacs的默认配置里，但可以从因特网下载，它的URL是*ftp://ftp.reed.edu/pub/src/html-helper-mode.tar.gz*。（如果你对下面的内容看不太懂，请向你的系统管理员求助，注2。）找到这个tar文件后，先把它下载到某个目录里，例如“*~/elisp*”，然后进入那个目录并输入以下命令：

```
% gunzip html-helper-mode.tar.gz
% tar xvf html-helper-mode.tar
```

系统将对tar文件进行解包操作。这个tar文件包含有以下几个组件：

- *html-helper-mode.el* —— 对应于html-helper模式的LISP文件
- *tempo.el* —— 允许插入各种HTML文档模板的Lisp文件
- *key bindings.html* —— 键盘命令清单
- *changelog.html* —— 关于升级和改进方面的日志记录
- *configuring.html* —— 关于配置html-helper模式的指示
- *differences.html* —— 介绍HTML模式和html-helper模式之间的差异
- *documentation.html* —— 关于html-helper模式的文档

注2：如果打算安装与Emacs一起工作的LISP软件包，请系统管理员帮忙通常不是个坏主意。他也许会把那个软件包安装到系统上的某个中央地点，这可以让更多的人来访问和使用它。

启动 Html-helper 模式

要想启动 html-helper 模式，请输入“**ESC x load-file RETURN**”，Emacs 会询问想加载哪个文件，输入*html-helper-mode.el*或*html-helper-mode.elc*文件的完整路径名作为回答，然后按回车键。接着，输入“**ESC x html-helper-mode**”命令；状态行上将出现“HTML helper”字样。

把html-helper模式设置为Emacs启动工作的一部分也很简单。把下面这些语句添加到“*.emacs*”文件里即可：

```
(setq load-path (cons "PUT_THE_PATH HERE" load-path))
(autoload 'html-helper-mode "html-helper-mode" "Yay HTML" t)
(setq auto-mode-alist (cons '("  html$" . html-helper-mode) auto-mode-alist))
```

请把存放*html-helper-mode.el*文件的子目录的完整路径，放到第一条语句中的引号里。第二条语句的作用是让Emacs在启动Emacs时，自动加载html-helper模式。第三条语句告诉Emacs：只要是准备编辑一个以“*.html*”为后缀名的文件，它就自动进入 html-helper 模式。

特殊字符的输入

在HTML里，尖角括号（“<”和“>”）和“&”都是特殊字符。html-helper模式允许直接输入特殊符号，但如果想在HTML文档里显示“<”、“>”或“&”字符，就必须输入与之对应的转义代码才行。在输入字符之前，先按下“**C-c**”组合键就可以输入与该字符对应的转义代码。比如，按下“**C-c <**”组合键将输入对应于小于号的转义代码（&lt)。

HTML 文档框架

Html-helper 模式为HTML 初学者准备了一个很有用的功能，即能选择让 Emacs 在创建一个新HTML 文件时自动插入一个文档框架。（有经验的老手也许会因已经自行开发出一套模板文件而用不着这项功能了。）如果想让html-helper模式自动插入一个文档框架，请把下面这条语句添加到“*.emacs*”文件里：

```
(setq html-helper-build-new-buffer t)
```

保存“*.emacs*”文件并重新启动Emacs。然后来创建一个新的HTML文档。

输入：**C-x C-f new.html**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>

<hr>
<address></address>
<!--hhmts start -->
<!--hhmts end -->
</body> </html>

--**-Emacs: new.html          (HTML helper)--L1--All----------------
Loading html-helper-mode...done
```

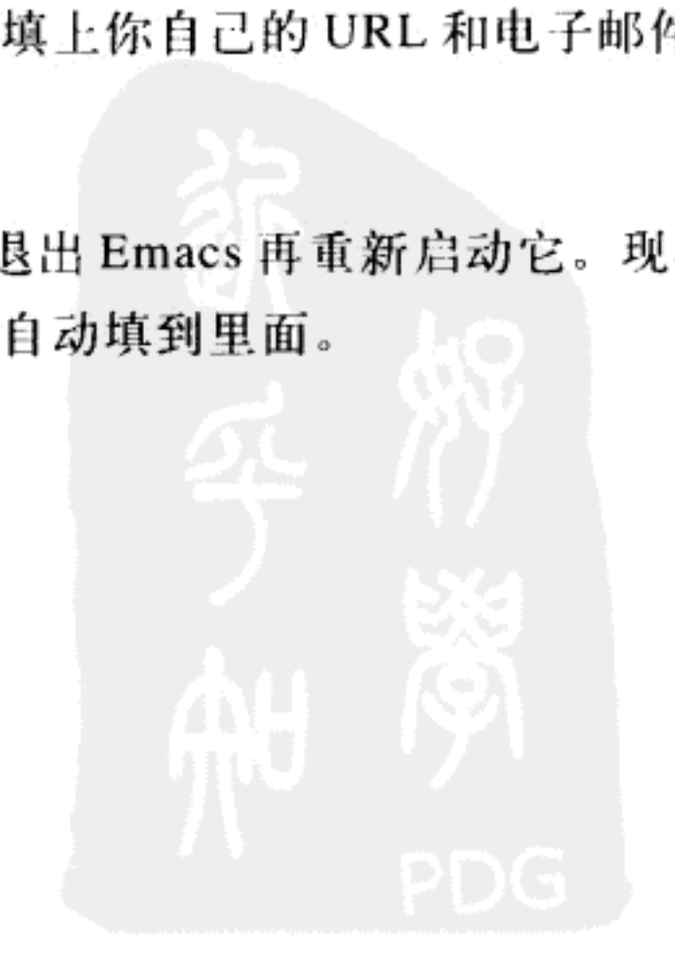
Html-helper模式插入一个文档框架，编写一个HTML文档所需要的基本元素都已经准备齐全。

HTML的基本元素都包含在这个框架里。整个文档被括在<html> </html>标签的中间。随后是标题部分和正文部分。再往后的那个<hr>标签告诉浏览器在此插入一条水平线，这条线路通常被称为水平标尺。<address>标签给文档的作者留出一块用来填写其因特网地址的地方。如果把下面这条语句添加到“*.emacs*”文件里，HTML helper模式就能把这个地址自动添加到今后创建的每一个HTML文档里：

```
(setq html-helper-address-string
  "<a href= \"http://www.merryold.co.uk/~chuck/\">Charles Dickens &lt;
                                        chuck@merryold.co.uk&gt;</a>")
```

当然，如果你不是Charles Dickens，就得在这里填上你自己的URL和电子邮件地址。

保存“*.emacs*”文件，但不保存此编辑缓冲区；退出Emacs再重新启动它。现在，当打开*new.html*文件的时候，Emacs就会把地址自动填到里面。



输入：**C-x C-f new.html**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>

<hr>
<address><a href="http://www.merryold.co.uk/~chuck/">Charles Dickens
&lt;chuck@merryold.co.uk&gt;</a></address>
<!-- hhmts start -->
<!-- hhmts end -->
</body> </html>

--**-Emacs: new.html          (HTML helper)--L1--All--------------------
```

Emacs 插入的 HTML 文档框架，里面已经填上了地址。

开始写作。在文档框架里放上一个文档标题（title）和一个一级标题（header）（它们经常是一模一样的），然后开始输入文字内容。在输入一段文字之后，按“**ESC RETURN**”插入一个HTML段落分隔符 —— 这是很少几个不需要配对的标签之一，因此只需输入一个就行。比如，先把狄更斯小说《双城记》（A Tale of Two Cities）的第一段输入进去，然后按下“**ESC RETURN**”组合键。请看下图。

先输入狄更斯小说《双城记》第一段，然后按下“**ESC RETURN**”组合键。

```
Buffers Files Tools Edit Search Help
<html> <head> <title>A Tale of Two Cities</title> </head> <body> <h1>A
Tale of Two Cities</h1> It was the best of times, it was the worst of
times, it was the age of wisdom,it was the age of foolishness, it was
the epoch of belief, it was the epoch of incredulity, it was the season
of Light, it was the season of Darkness, it was the spring of hope, it
was the winter of despair, we had everything before us, we had nothing
before us, we were all going direct to Heaven, we were all going direct
the other way—in short, the period was so far like the present period,
that some of its noisiest authorities insisted on its being received,
for good or evil, in the superlative degree of comparison only.
<p> <hr>
--**-Emacs: dickens.html      (HTML helper)--L16--Top-------------------
```

Emacs 插入一个段落分隔符“<p>”。

在文本块的首尾配对设置 HTML 标记

在编写HTML的时候，对现有文件进行编码往往要多于真正的写作。如果在输入任何一个标签之前按下了“**C-u**”组合键，那么 Emacs 会把标签自动放到一个文本块的两端，而不是把它们插入到光标的当前位置。

请看下面的例子。我们准备给 Strunk 和 White 两人合著的《Elements of Style》书中的某些文字加上一些标记 —— 这些文字输入到一个文件里（注 3）。打开一个名为 *strunk.html* 的新文件，Emacs 会自动插入一个文档框架。把光标移动到一级标题的标签下方，准备插入未经标记排版的 ASCII 文本文件 *strunk*。

输入：**C-x C-f strunk.html**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>

<hr>
<address><a href="http://www.strunk.com/~white/">E.B. White
&lt;white@strunk.com&gt;
</a></address>
<!-- hhmts start -->
<!-- hhmts end -->
</body> </html>
--**-Emacs:strunk.html        (HTML helper)--L1--All--------------------
```

Emacs 自动插入一个 HTML 文档框架。

把光标移到配对出现 <h1> 标签组的后面，输入：**C-x C-i strunk**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>
Elementary Rules of Usage

1. Form the possessive singular of nouns by adding 's.

Follow this rule whatever the final consonant. Thus write,

Charles's friend
Burns's poem
the witch's malice
--**-Emacs: strunk.html        (HTML helper)--L7--Top--------------------
```

Emacs 插入 ASCII 文本文件 *strunk*，我们将给它添加一些 HTML 标签。

注 3：当然，我们只是用这段文字来练习编写 Web 页面，而不是要把它张贴到因特网上，因为它受版权保护。

在这个例子里，我们打算让文档标题（title）和一级标题（header）不一样。

把光标移动到<title>和</title>之间，然后输入：**The Element of Style, Chapter 1**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title>The Elements of Style, Chapter 1 </title>
</head>

<body>
<h1></h1>
Elementary Rules of Usage

1. Form the possessive singular of nouns by adding 's.

Follow this rule whatever the final consonant. Thus write,

Charles's friend
Burns's poem
the witch's malice
--**-Emacs: strunk.html        (HTML helper)--L2--Top----------------------
```

文档标题输入完毕。

把“Elementary Rules of Usage”选取为一个文本块，按“**C-w**”组合键把它剪切下来，然后用“**C-y**”组合键把它插到标签<h1>和</h1>的中间。

```
Buffers Files Tools Edit Search Help
<html> <head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

1. Form the possessive singular of nouns by adding 's.

Follow this rule whatever the final consonant. Thus write,

Charles's friend
Burns's poem
the witch's malice
--**-Emacs: strunk.html        (HTML helper)--L6--Top----------------------
```

一级正文标题输入完毕。

接下来，我们把逻辑样式用到第一个句子上。

把第一个句子选取为一个文本块，然后按下“**C-u C-c C-s s**”组合键（命令名是 **tempo-template-html-strong**）。

```
Buffers Files Tools Edit Search Help
<html><head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

<strong>1. Form the possessive singular of nouns by adding 's.</strong>

Follow this rule whatever the final consonant. Thus write,

Charles's friend
Burns's poem
the witch's malice
--**-Emacs: .strunk.html      (HTML helper)--L8--Top--------------------
```

Emacs 把 和 标签分别插到文本块的两端。

接下来，按一次回车键，然后用“**ESC RETURN**”组合键插入一个段落分隔符。再移动到随后的第二段文字的后面，并再次按下“**ESC RETURN**”组合键。

输入：**C-n ESC RETURN C-d C-n ESC RETURN**

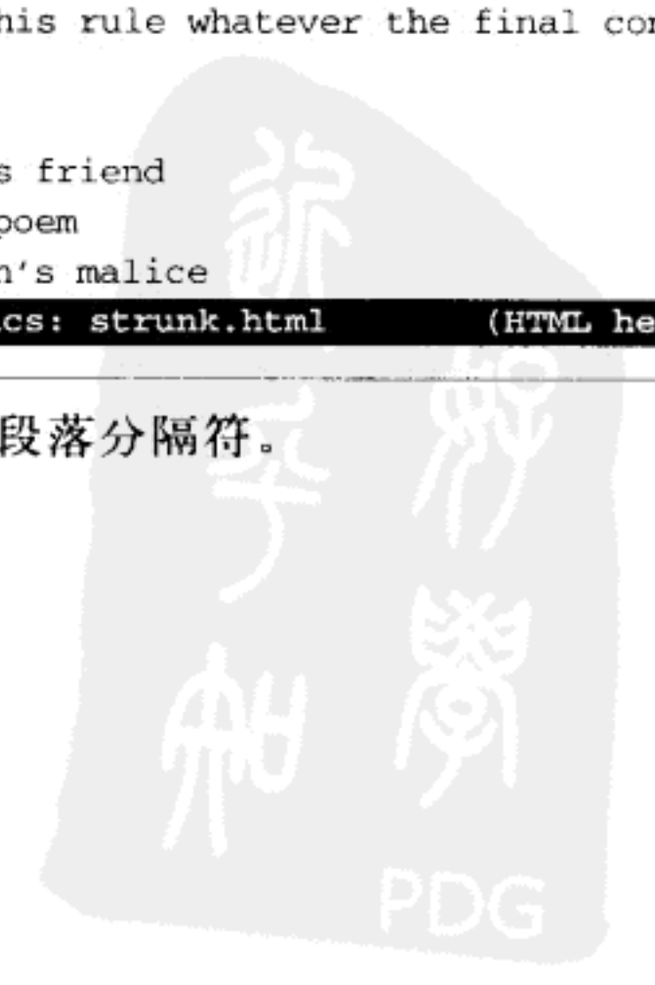
```
Buffers Files Tools Edit Search Help
<html><head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

<strong>1. Form the possessive singular of nouns by adding 's.</strong>
<p>
Follow this rule whatever the final consonant. Thus write,
<p>

Charles's friend
Burns's poem
the witch's malice
--**-Emacs: strunk.html      (HTML helper)--L12--Top-------------------
```

Emacs 将插入两个段落分隔符。



我们接下来将对从“Charles's friend”开始的几行文本进行标记。我们准备把它们安排在一个项目符号列表（bulleted list）里。

把示范文字选取为一个文本块，然后输入：**C-u C-c C-l u**

```
Buffers Files Tools Edit Search Help
<html><head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

<strong>1. Form the possessive singular of nouns by adding 's.</strong>
<p>
Follow this rule whatever the final consonant. Thus write,
<p>
<ul>
  <li>Charles's friend
Burns's poem
the witch's malice

</ul>
--**-Emacs: strunk.html        (HTML helper)--L17--Top-------------------
```

Emacs将插入项目符号列表（即无序列表，unordered list）的起始标签和结束标签。

对列表的处理差不多已经完成了，但我们还想再对它做一些细致的调整。把光标移动到单词“Burns”的字母“B”上，按下“**C-c C-l i**”组合键（命令名是**html-helper-smart-insert-item**，之所以称它为“smart insert（智能插入）”是因为<li>标签能够用来对项目符号列表、编号列表以及菜单等进行标记）。Emacs插入<li>标签，再对“the witch's malice”做同样的设置。注意html-helper模式会自动把列表缩进一点点。如果把列表嵌套在另外一个列表里，它的缩进量还会更大一些。因为空白行和缩进对HTML文档的显示效果并没有什么影响，所以这里的缩进只是为了提高HTML文件的可读性而已。如果想禁用缩进功能，请把下面这条语句添加到“*.emacs*”文件里：

```
(setq html-helper-never-indent t)
```

如果还想继续对这个文件进行标记，可以用同样的方法给其余文字都加上HTML标签。在插入HTML标签的时候，既可以使用各种键盘命令，也可以使用自动补足功能。我们下面就来说说自动补足功能。

Html-helper 模式下的自动补足功能

Html-helper模式里的键盘命令实在是太多了；根本不可能把它们都记下来。好在我们还有另外一种选择 —— 输入前几个字符之后按下“**ESC TAB**”组合键，Emacs或者自动补足HTML标签，或者显示一个窗口把各种可用的补足形式都列在其中以供选择（如果鼠标可以用，那么就可以用鼠标中键挑选一个补足形式）。自动补足功能在 Emacs 19.23 及其以后的版本里都能工作。

假设还在strunk.html文件的文本上工作。已经完成了下一个段落的标记，现在想再设置一个项目符号列表。如下所示：

输入：**<ulESC TAB**

```
Buffers Files Tools Edit Search Help
</ul>
Exceptions are the possessives of ancient proper names in -es and -is,
the possessive of Jesus', and such forms as for conscience' sake, for
righteousness' sake. But such forms as Moses' laws, Isis' temple are
commonly replaced by
<p>

<ul>
  <li>
</ul>the law of Moses
the temple of Isis
--**-Emacs: strunk.html        (HTML helper)--L10--65%--------------------
```

Emacs 插入了列表的首、尾标签和一个列表数据项标签。

在上面的例子里，必须把列表中的数据项剪粘到适当的位置。自动补足功能无法对文本块进行标记，但很适用于先输入标签的一部分，再由 Emacs 补足它的情况。

如果你选择亲自动手输入HTML标签，就将发现Emacs会以在大多数其他编辑模式里对圆括号和方括号进行配对检查的类似方法，对html-helper模式里对尖角括号进行配对检查。当你输入右尖角括号之后，Emacs将把光标移动到与之对应的左尖角括号处，然后再回到那个右尖角括号，使你能够对尖角括号的配对情况进行视觉上的检查。

启用输入提示功能

有些HTML标签需要输入信息域中的内容。比如在输入超链接的时候，就必须输入

该链接的URL地址和用来让用户选取该超链接的文本。如果用按下“**C-c C-a l**”（小写的字母“L”）组合键的方法输入一个链接，那么Html-helper模式将自动插入如下所示的标签：

```
(a href="▌> </a>
```

并把光标放到第二个双引号上，于是可以直接输入URL（网址）。但要是启用了提示功能，html-helper模式还可以让这个标记操作更简便。启用提示功能的办法是把下面这条语句添加到“*.emacs*”文件里：

```
(setq tempo-interactive t)
```

请看下面的例子。假设打算在strunk.html文档里放上一个指向某个介绍Robert Burns生平事迹的Web主页的（假想的）超链接，我们来看看输入提示功能是如何工作的。

把“Burns's”选取为一个文本块，然后按下“**C-u C-c C-a l**”（小写的字母“L”）组合键：

```
Buffers Files Tools Edit Search Help
<html> <head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

<strong>1. Form the possessive singular of nouns by adding 's.</strong>
<p>
Follow this rule whatever the final consonant. Thus write,
<p>
<ul>
  <li>Charles's friend
  <li><a href="Burns's poem
  <li>the witch's malice
</ul>
--**-Emacs: strunk.html        (HTML helper)--L14--Top--------------------
URL: ▌
```

Emacs提示输入超链接的URL（网址）。

输入：**http:// www.uofillyria.edu/poets/RobertBurns.html RETURN**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title>The Elements of Style, Chapter 1</title>
</head>

<body>
<h1>Elementary Rules of Usage</h1>

<strong>1. Form the possessive singular of nouns by adding 's.</strong>
<p>
Follow this rule whatever the final consonant. Thus write,
<p>
<ul>
  <li> Charles's friend
  <li> <ahref="http://www.uofillyria.edu/poets/
       RobertBurns.html">Burns </a>\s poem
  <li> the witch's malice
</ul>
--**-Emacs: strunk.html        (HTML helper)--L14--Top-------------------
```

Emacs 把这个超链接插入到文档里。

如果还没有输入超链接本身的文字内容，Emacs也同样会提示输入它。（因为我们已经把超链接本身的文字内容选取为一个文本块，又给标记命令加上了“**C-u**”前缀，所以Emacs会把超链接插入到文本的两端。掌握给现有文本增加超链接的办法是很有必要的，因为大多数作者都是用“先写作、后加超链接”的办法逐步对文档进行“超链化”。）

如果在启用输入提示功能的情况下输入列表，Html-helper模式将提示输入列表中的数据项。下面，我们一起用编号列表标记命令（“**C-c C-l o**”组合键）来给 Emacs 列一个优点清单，这次使用的文件叫做 *emacs.html*。如下所示：

输入：**C-x C-f emacs.html**，然后移过 <h1> </h1>，再按下“**C-c C-l o**”：

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>
<ol>
  <li>
</ol>
--**-Emacs: emacs.html        (HTML helper)--L8--All---------------------
Item:
```

Emacs 提示输入一个列表数据项。

输入：**Emacs is extensible RETURN**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>
<ol>
  <li> Emacs is extensible
</ol>
--**-Emacs: emacs.html        (HTML helper)--L8--All----------------------
```

Emacs 插入列表的第一个数据项。

输入提示功能到底是有用还是瞎操心全在于个人。对于HTML新手，输入提示功能可以帮助做到把每个标签的必要信息都输入齐全。如果不喜欢它，那么只要把添加到“*.emacs*”文件里的那条语句删掉即可。

时间戳的自动刷新

时间戳（timestamp）的作用是告诉你以及浏览你主页的人们，所看到的信息是在什么时候做的最后一次更新。这需要每编辑一次文档就在其中放上一个时间戳，但可能会忘记这件事。不过，如果把下面这条语句添加到“*.emacs*”文件里，那么html-helper 模式就会在编辑和保存某个 HTML 文件时自动插入和刷新时间戳：

```
(setq html-helper-do-write-file-hooks t)
```

以后每次保存文件时都能观察到时间戳的变化。现在就可以按下“**C-x C-s**”组合键试试，时间戳将改变为当前的日期和时刻。退出Emacs时不会看到时间戳所发生的变化，但它确实会发生变化。如果只是在浏览这个文件，或者编辑之后没存盘，Emacs 也不会刷新时间戳。如果刚开始编辑一个新的页面，则它的时间戳部分将是空白的。如下所示：

输入：**C-x C-f ralph.html**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title></title>
</head>

<body>
<h1></h1>

<hr>
<address><a href="http://www.homestyle.com/ralph/">Ralph Radisson
&lt;ralph@homestyle.com&gt;</a></address>
<!- hhmts start ->
<!- hhmts end ->
</body> </html>

--**-Emacs: ralph.html        (HTML helper)--L1--All----------------------
```

Emacs 插入一个 HTML 文档框架，里面带着一个空白的时间戳。

在这个文件里写了一些东西并对它进行了存盘之后，html-helper模式就将自动对时间戳进行刷新。

按下：**C-x C-s**

```
Buffers Files Tools Edit Search Help
<html> <head>
<title>Ralph's Homestyle Home Page</title>
</head>
<body>
<h1>Ralph's Homestyle Home Page</h1>

<hr>

Hello and welcome to my Homestyle Home Page, where you can learn
the secrets of my famous Homestyle Hamburger Hash.
<address><a href="http://www.homestyle.com/ralph/">Ralph Radisson
&lt;ralph@homestyle.com&gt;</a></address>
<!- hhmts start ->
Last modified:Fri Dec 2 10:55:03 1996
<!- hhmts end ->
</body></html>

--**-Emacs: ralph.html        (HTML helper)--L1--All----------------------
```

Emacs 会在每次文件存盘时自动刷新这个时间戳。

表 9-5 列出了 html-helper 模式里的各种键盘命令。有些键盘命令是为表单（form）或 HTML 3.0 中的某些高级功能而准备的。有些标签在一般情况下会出现在不同的行上（比如某些用来标记列表的标签）；但在下面这份表格里，它们都显示在同一行上。

表 9-5: Html-helper 模式命令速查表

键盘操作	命令名称	动作
C-u	**universal-argument**	当加在其他标记命令的前面时，把标签插入到文本块的两端
C-c >	**tempo-template-html-greater-than**	插入大于号的转义代码“>”
C-c <	**tempo-template-html-less-than**	插入小于号的转义代码“<”
ESC RETURN	**tempo-template-html-paragraph**	插入段落分隔符“<p>”
C-c C-s s	**tempo-template-html-strong**	插入 标签
C-c C-s e	**tempo-template-html-emphasized**	插入 标签
C-c C-s b	**tempo-template-html-blockquote**	插入<blockquote> </blockquote>标签
C-c C-s p	**tempo-template-html-preformatted**	插入 <pre> </pre>标签
C-c C-l u	**tempo-template-html-unordered-list**	插入 <ul> <li> </ul>标签
C-c C-l o	**tempo-template-html-ordered-list**	插入 <ol> <li> </ol>标签
C-c C-l t	**tempo-template-html-definition-item**	插入 <dt> <dd>标签
C-c C-l l	**tempo-template-html-item**	插入 <li>标签
C-c C-l d	**tempo-template-html-definition-list**	插入 <dl> <dt> <dd> </dl>标签
C-c C-l m	**tempo-template-html-menu**	插入 <menu> <li> </menu>标签
C-c C-l r	**tempo-template-html-directory**	插入 <dir> <li> </dir>标签
C-c C-l i	**html-helper-smart-insert-item**	插入 <li>标签
ESC TAB	**tempo-complete-tag**	补足当前标签
C-c RETURN	**tempo-template-html-break**	插入一个换行符“ ”
C-c -	**tempo-template-html-horizontal-rule**	插入一个水平标尺“<hr>”

表 9-5: Html-helper 模式命令速查表（续）

键盘操作	命令名称	动作
C-c SPC	**tempo-template-html-nonbreaking-space**	插入一个空格字符的转义代码“ ”并设定不得在此处断开文本行
C-c &	**tempo-template-html-ampersand**	插入一个“&”字符的转义代码“&”
ESC C-t	**html-helper-insert-timestamp-delimiter-at-point**	插入时间戳分隔符
C-c C-b t	**tempo-template-html-title**	插入 <title> </title>标签
C-c C-a l	**tempo-template-html-anchor**	插入 <a href ="" > </a> 标签
C-c C-a n	**tempo-template-html-target-anchor**	插入 <a name =""> </a> 标签
C-c C-t 6	**tempo-template-html-header-6**	插入 <h6> </h6>标签
C-c C-t 5	**tempo-template-html-header-5**	插入 <h5> </h5>标签
C-c C-t 4	**tempo-template-html-header-4**	插入 <h4> </h4>标签
C-c C-t 3	**tempo-template-html-header-3**	插入 <h3> </h3>标签
C-c C-t 2	**tempo-template-html-header-2**	插入 <h2> </h2>标签
C-c C-t 1	**tempo-template-html-header-1**	插入 <h1> </h1>标签
C-c TAB e	**tempo-template-html-align-alt-image**	插入 <img align =""src =""alt =""> 标签
C-c TAB i	**tempo-template-html-image**	插入 <img src =""> 标签
C-c TAB t	**tempo-template-html-alt-image**	插入 <img alt =""src =""> 标签
C-c TAB a	**tempo-template-html-align-image**	插入 <img align =""src =""> 标签
C-c C-f p	**tempo-template-html-input-textarea**	插入 <textarea name =""rows =""cols="" > </textarea> 标签
C-c C-f x	**tempo-template-html-input-reset**	插入 <input type ="RESET" value =""> 标签
C-c C-f b	**tempo-template-html-input-submit**	插入 <input type ="SUBMIT" value =""> 标签

表 9-5: Html-helper 模式命令速查表（续）

键盘操作	命令名称	动作
C-c C-f a	**tempo-template-html-input-audio**	插入 <input type ="AUDIO" name = ""> 标签
C-c C-f s	**tempo-template-html-input-scribble**	插入<input type ="SCRIBBLE" name ="" size =""> 标签
C-c C-f g	**tempo-template-html-input-image**	插入 <input type ="IMAGE" name = "" src ="" > 标签
C-c C-f r	**tempo-template-html-input-radio**	插入 <input type ="RADIO" name = ""> 标签
C-c C-f c	**tempo-template-html-input-select**	插入 <select name ="" > 标签
C-c C-f u	**tempo-template-html-input-url**	插入 <input type ="URL" name ="" size ="" > 标签
C-c C-f d	**tempo-template-html-input-date**	插入<input type ="DATE" name = "" size = "" > 标签
C-c C-f .	**tempo-template-html-input-float**	插入 <input type ="FLOAT" name = "" size ="" > 标签
C-c C-f i	**tempo-template-html-input-int**	插入 <input type ="INT" name ="" size ="" > 标签
C-c C-f t	**tempo-template-html-input-text**	插入 <input name ="" size =""> 标签
C-c C-f f	**tempo-template-html-form**	插入 <form action =""> </form> 标签
C-c C-n m	**tempo-template-html-margin**	插入 <margin> </margin> 标签
C-c C-n f	**tempo-template-html-footnote**	插入 <footnote> </footnote> 标签
C-c C-n n	**tempo-template-html-note**	插入 <note role = "" > </note> 标签
C-c C-n a	**tempo-template-html-abstract**	插入 <abstract> </abstract> 标签
C-c C-p r	**tempo-template-html-render**	插入<render tag = "" style = ""> 标签
C-c C-p -	**tempo-template-html-subscript**	插入 标签
C-c C-p ^	**tempo-template-html-superscript**	插入 标签
C-c C-p x	**tempo-template-html-strikethru**	插入 <s> </s> 标签
C-c C-p f	**tempo-template-html-fixed**	插入 <tt> </tt> 标签

表 9-5: Html-helper 模式命令速查表（续）

键盘操作	命令名称	动作
C-c C-p u	**tempo-template-html-underline**	插入 <u> </u> 标签
C-c C-p i	**tempo-template-html-italic**	插入 标签
C-c C-p b	**tempo-template-html-bold**	插入 标签
C-c C-s a	**tempo-template-html-address**	插入 <address> </address> 标签
C-c C-s l	**tempo-template-html-lit**	插入 <lit> </lit> 标签
C-c C-s g	**tempo-template-html-arg**	插入 <arg> </arg> 标签
C-c C-s m	**tempo-template-html-cmd**	插入 <cmd> </cmd> 标签
C-c C-s .	**tempo-template-html-abbrev**	插入 <abbrev> </abbrev> 标签
C-c C-s y	**tempo-template-html-acronym**	插入 <acronym> </acronym> 标签
C-c C-s n	**tempo-template-html-person**	插入 <person> </person> 标签
C-c C-s q	**tempo-template-html-quote**	插入 <q> </q> 标签
C-c C-s d	**tempo-template-html-definition**	插入 <dfn> </dfn> 标签
C-c C-s v	**tempo-template-html-variable**	插入 <var> </var> 标签
C-c C-s k	**tempo-template-html-keyboard**	插入 <kbd> </kbd> 标签
C-c C-s r	**tempo-template-html-citation**	插入 <cite> </cite> 标签
C-c C-s x	**tempo-template-html-sample**	插入 <samp> </samp> 标签
C-c C-s c	**tempo-template-html-code**	插入 <code> </code> 标签
C-c C-b b	**tempo-template-html-base**	插入 <base href = "" > 标签
C-c C-b l	**tempo-template-html-link**	插入 <link href = "" > 标签
C-c C-b n	**tempo-template-html-nextid**	插入 <nextid> 标签
C-c C-b i	**tempo-template-html-isindex**	插入 <isindex> 标签

这个表格就已经够长的了（有那么多的键盘命令），可每天仍有许多的新 HTML 标签创造出来。插入新标签的方法之一是创造自己的命名宏（named macro）。第十章对如何编写宏命令以进一步简化操作方面的问题进行了探讨。

第十章

Emacs 中的宏

本章内容:

- 什么是宏
- 定义宏
- 向现有的宏里增加编辑命令
- 命名并保存宏
- 执行一个已命名的宏
- 建立复杂的宏
- LISP 函数——宏的补充

什么是宏

什么是宏（macro）？在Emacs里，“宏”是一组被录制下来的按键动作，可以反复多次地使用。有了宏，就可以从重复性的工作中解放出来。比如，假设想删除某个表格的第三栏。普通的方法是：先移动到第一行，再移动到第三栏，删除它；然后移动到第二行，重复执行完全相同的一组命令；如此周而复始直到全部完成。即使手不累，可心早就烦了。Emacs允许对表格第一行进行处理时所使用的按键动作录制下来，然后重复“播放”这些动作直到整个工作完成（注1）。

在Emacs里使用的任何命令或者动作，从输入文本到编辑再到切换编辑缓冲区，都可以在一个宏里完成。要想把宏用足用好，其中的关键是能否准确地判断出自己在什么场合会做重复性的工作。也就是说，在什么情况下会连续多次以同一顺序重复按下同一组按键。一旦学会判断什么是重复性工作，就能对应该在什么时候使用宏有一个良好的感觉。接下来，还需要有另外一种天赋，那就是在识别出一组“几乎完全相同”的按键动作之后，如何使它们到达“完全相同”的地步，即发掘出这样一组按键动作：如果重复使用它们，它们就会完全按照意愿达到目标。这两种技巧学习起来并不是特别困难，只要稍加练习，就能够随时享受宏带来的方便。

注1：当然，正如第八章所描述的，通过把表格的第三栏标记为矩形，也可以删除第三栏。但让我们感到欣慰的是：当你发现自己正在做一些重复性工作时，宏是一个“记忆”工具。

这听起来好像是一种“懒人的”编程思路，而事实也确是如此：宏能够用一种简单的方法来完成复杂的工作，而你则既不需要学习LISP程序设计语言，又不需要额外学习特殊的定制技巧。如果你设计的宏正好适用于必须频繁接触的某些工作，甚至可以把它们保存起来，然后在需要它们的时候再随时加载。这样，你可以创造出一系列非常方便的宏，让它们成为编辑文件时的专用命令。你可以不受Emacs提供的命令的限制创造出自己的东西来！

在 Emacs 里的工作情况决定了使用宏的目的。我们主要用宏：

- 给文本设置排版标记。
- 把某个编辑缓冲区里的标题作为一份大纲，复制到另外一个编辑缓冲区里。
- 做一些查询－替换无法应付的查找加替换类型的操作。
- 建立索引目录。
- 重新对从另外一个应用软件里导入进来的文件进行排版。
- 把表格从一种格式套用为另外一种格式。
- 用一个命令来完成某个程序的编译、运行和测试。

只要学会几个基本的宏操作命令，就能在想出它更多的用途。

定义宏

要想开始一个宏的定义工作，请按下“**C-x (**”组合键。状态行上将出现“`Def`”字样，表示现在是在宏定义模式（macro definition）里。在这个模式里，Emacs将把所有的按键动作记录下来（不管它们是命令还是文字性的文本），从而可以在今后反复地使用。要想结束一个宏的录制工作，请按下“**C-x)**”组合键，离开宏定义模式。而Emacs也将停止记录按键动作。如果在宏的录制过程中出现了错误，或者如果按下了“**C-g**”组合键，Emacs也会停止录宏的录制。

录制宏的时候，Emacs会在记录按键动作的同时执行它们，这就是说，录制宏时输入的任何内容，都将被看做是一个正常的命令并被执行。定义宏时的操作与平时的编辑工作完全一样，以便看清录制中的宏是不是符合自己心愿；如果发现正在录制

的宏与想让它完成的工作不太一致，则可以随时取消它（按下“**C-g**”组合键）。令人遗憾的是，要想在Emacs里对宏进行编辑和修改是非常困难的。如果真的出了错，那么惟一现实的选择是取消这个宏，然后再重新开始另外录制一个（注2）。

要想执行已经录制好的宏，需要按下“**C-x e**”组合键（命令名是**call-last-kbd-macro**）；Emacs将把刚才录制下来的按键动作精确地重演一遍。在任一时刻只能有一个活动的宏。如果又定义了另外一个宏，那么新定义的宏就将成为活动的宏，而早先定义的那个将被它覆盖。

示例

请看下面定义和执行宏的演示示例。我们想在打印稿里使用双倍行距，就标准的ASCII文件而言，要想打印一份这样的文件在UNIX下是没有现成的办法可用的。下面这个宏能够把编辑缓冲区里的文本全部设置为双倍行距。

键盘操作	动作
C-x (	开始录制一个宏；状态行上将出现“Def”字样
C-e	移动到当前行的行尾
C-f	移动到下一行的第一个字符
RETURN	插入一个空白行
C-f	向右移动一个字符（到达下一行的行首）
C-x)	结束宏的录制工作

大家可以找个文件副本来试试这个宏。

注2：这种说法并不很正确。如果能用undo命令撤销刚才犯的错误，那么完全可以删除并重新输入几个字符，或者移动一下光标，然后再继续录制宏——就像什么事情也没有发生过一样。宏执行起来非常快，所以几个额外的命令不会对它构成损害。但这样的宏往往会存在着一些潜在的问题。记住：只有能够反复使用的宏才是真正有价值的。

按上表给出的按键动作定义一个宏。

```
Buffers Files Tools Edit Search Help
It was the best of times, it was the worst of times, it was the age of

wisdom, it was the age of foolishness, it was the epoch of belief,
it was the epoch of incredulity, it was the season of Light, it was the
season of Darkness, it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we were
all going direct to Heaven, we were all going direct the other way-in
short, the period was so far like the present period, that some of its
noisiest authorities insisted on its being received, for good or evil,
in the superlative degree of comparison only. It was the best of times,
it was the worst of times, it
--**-Emacs: dickens          (Text Fill)--L3--All-----------------------
```

在录制这个宏的过程中，对第 1 行进行了双倍行距处理，结束时光标留在了第 2 行上。

让我们“勇敢地”假设这个宏没有毛病，接下来让它重复执行 4 次 —— 给用来执行一个宏的命令加上“**C-u**”前缀就能做到这一点。当然，在实际工作中，在如此大胆地放手操作之前至少应该先试一次。

输入：**C-u C-x e**

```
Buffers Files Tools Edit Search Help
It was the best of times, it was the worst of times, it was the age of

wisdom, it was the age of foolishness, it was the epoch of belief, it

was the epoch of incredulity, it was the season of Light, it was the

season of Darkness, it was the spring of hope, it was the winter of

despair, we had everything before us, we had nothing before us, we

were all going direct to Heaven, we were all going direct the
other way-in short, the period was so far like the present period, that
some of its noisiest authorities insisted on its being received, for good
or evil, in the superlative degree of comparison only.
--**-Emacs: dickens          (Text Fill)--L11--All----------------------
C-u C-x e
```

现在对前 5 行完成了双倍行距的设置工作：1 个是在录制宏时完成的，其余 4 个是通过执行那个宏而完成的。

宏的工作情况很理想；于是我们可以自信地把编辑缓冲区里剩余的文本行，都设置

为双倍行距：先输入几个“**C-u**”，然后输入“**C-x e**”（注3）。当然，如果要更精确一些，可以先用“**ESC x count-lines-region RETURN**”命令找出准备设置双倍行距的区域里到底有多少行文本。比如，如果只想给当前段落设置双倍行距，而**count-lines-region**命令告知该段落只剩下6行还需要设置，那么只要给出“**ESC 6 C-x e**”命令即可。

下面是一些容易犯的错误，请大家注意避免：

- 在结束宏的录制工作时别忘了按下“**C-x)**”组合键。如果在宏的录制工作结束之前就试图执行之，那么Emacs会报警，而已经录制的击键动作也都将丢失。
- “**C-g**”组合键将终止宏编辑命令的录制工作，并且会使Emacs忘记已经录制的击键动作。
- 任何错误都将自动终止一个宏的执行。如果Emacs蜂鸣报警，就得重新开始。
- Emacs将“毫无理智”地严格按录制下来的击键动作来执行。因此，一定要避免作诸如“我在执行这个宏的时候肯定是在行首（或者行尾）”之类的假设。

如果执行了一个宏却做错了事，可以用“**C-x u**”组合键（正常的undo命令）来撤销。Emacs会知道此时的“撤销上一条命令”操作的真正含义是“撤销整个宏”而不是“撤销宏里的最后一条命令”。

技巧：如何创建良好的宏

录制和重复使用击键动作的技巧并不难掌握。但新手往往会犯一些典型的错误：录制了一个宏，使用它，然后发现它的执行情况和预想并不是百分之百地吻合。其实只要稍加注意，就可以让自己的宏更有用，错误更少。

所谓良好的宏是那些能够在任何需要场合都能正确工作的宏。因此，应该在宏里使用绝对型命令而非相对型命令。比如，如果正在写一个用来把某排版字符串放到光标所在单词两端的宏，那么肯定希望这个宏不管单词本身有多长都应该能够工作。

注3：为什么要先输入几个“**C-u**”呢？因为宏会在其执行过程中遇到错误时报警并停下来。而在这个例子里，如果光标已经到达文件尾，那么再向前移动一个字符就会产生一个“错误”。因此，哪怕设置双倍行距的那个文件只有25行，用“**C-u C-u C-u C-u C-x e**”命令来重复执行这个宏256次也是不会有问题的。

因此，应该使用像“**ESC f**”（命令名是 **forward-word**）之类的绝对型命令，而不是几个每次右移一个字符的“**C-f**”组合键。类似地，如果想移动到行首或者行尾，就应该使用“**C-e**”和“**C-a**”之类的命令，而不使用让光标做前后相对移动的命令。

宏通常都是以一个查找命令开始的，它的作用是进入要在文件里开始执行这个宏的地方。对于这种情况，输入查找参数（比如“**C-s** *searchstring*”）要比简单地使用“重复上一次查找”命令（如“**C-s C-s**”）效果更好。这是因为，在录制宏时，使用的查找字符串与打算执行那个宏时，所需要使用的查找字符串极有可能是不同的，而“**C-s C-s**”只记得住上一次查找操作所使用的查找字符串。

有时候，给宏额外增加几个并非严格必要的命令（通常是“**C-a**”、“**C-e**”）是很有好处的，它们的作用是确保操作发生在文本行上的正确位置。宏在录制时做的假设越少，它在执行时取得的效果往往也就越好。因此，如果一组命令只有在行尾时才能正确执行，那么宏就应该以“**C-e**”命令开始，哪怕已经“知道”“只有在行尾时才会发出这个宏编辑命令”。

除了上面这些条条框框，我们还希望大家记住这样一件事：既然打算定义一个宏，那么肯定是想反复多次地执行它。也就是说，几乎百分之百地会用到“**C-u C-x e**”（连续执行 4 次）、“**C-u C-u C-x e**”（连续执行 16 次），甚至“**C-u C-u C-u C-u C-u C-x e**”（连续执行 1024 次）等命令。只要稍有预见性，就能创建出可以连续执行无数次而不会出现问题的宏。

一般说来，好的宏是由 3 个部分组成的：

1. 找到准备开始工作的地方（通常利用查找操作来实现）。
2. 对文本进行预定的编辑处理。
3. 做好下次循环的准备工作。

那么，宏是如何做好下次循环的准备工作的呢？请看下面这个例子。假设写一个用来删除某表格第三栏的宏。在删除了那一栏之后，这个宏应该位于下一行的行首（或者应该处于的其他地方）。这样，如果需要重复使用这个宏，就不用再去调整位置。

再来看一个稍微复杂点的例子。如果在某个宏的开头使用了一个查找操作，就必须在这个宏结束时把光标移动到本次查找点的后面。否则，宏将在文件的这个查找点

处陷入死循环，永远也到不了下一个查找点。作为一条基本规则，如果宏是用来对一个文本行进行操作的，那么它就应该以移动到下一行的行首作为结束。记住：目标是设计出一套能够连续地重复执行多次的击键动作，而整个过程应该是不会中断的。

体面地结束是设计宏编辑命令时应该努力追求的另外一个目标，而这往往又与选择录制的击键动作有关。比如，有几个Emacs命令（比如“**C-n**”）即使到达了编辑缓冲区的末尾也不会停止执行和报警，它会在文件尾添加新行。如果宏里有一个放错了位置的“**C-n**”而且还让它执行了好几百次，那么它即使到了应该停止的时候也不会停止。因此，如果想移动到下一个文本行的行首，就应该使用“**C-e C-f**”而不是“**C-a C-n**”。这两者的执行效果是一样的，但前者不会让文件无端地增加出空白行来。

示例：一个更复杂的宏

有时候，需要把文件里出现某个特定单词（或短语、句子等）的地方都找出来。下面这个宏能够把编辑缓冲区里所有包含单词“Emacs”的句子都找出来，并把它们放到另外一个编辑缓冲区里。如果想试试这个宏，需要先把一些关于Emacs的文字输入到某个编辑缓冲区里（如果手里没有现成的材料，可以照抄下面示例画面里的内容）。

键盘操作	动作
C-x (	开始录制一个宏；状态行上将出现“`Def`”字样
C-s emacs	查找单词“Emacs”
RETURN	如果查找成功，停止查找操作；如果查找不成功，它将报警并停止这个宏的执行
ESC a	移动到句首[a]
C-@	设置文本块标记
ESC e	移动到句尾
ESC w	把句子复制到删除环里
C-x b emacsrefs RETURN	移动到名为“`emacsrefs`”的编辑缓冲区
C-y	插入这个句子
RETURN	为下一个句子准备一个新行

键盘操作	动作
C-x b RETURN	返回刚才的编辑缓冲区
C-x)	结束宏的录制工作；状态行上的“Def”字样将消失

a. “**ESC a**”命令关于“句子”的概念由变量**sentence-end**控制，这个变量是一个相当复杂的正则表达式。在默认的情况下，句子将以句号（.）、问号（?）或惊叹号（!）结束；它们后面还可以再跟有引号或括号（包括方括号和花括号）；最后是两个以上的空格或一个换行符。

现在，假设已经录制好这个宏，可以用“**C-x e**”来执行它了。下面这个画面就是运行这个宏时所发生的情况。

按下：**C-u C-x e**

```
Buffers Files Tools Edit Search Help
                    Some Sentences About Editors

TECO is an archaic editor. Emacs is a modeless editor. vi is a modal
editor. EDT is a proprietary editor. GNU Emacs was written by
Stallman. Unipress Emacs was written by Gosling.
--**-Emacs: emacstext        (Text Fill)--L5--All----------------------
Emacs is a modeless editor.
GNU Emacs was written by
Stallman.
Unipress Emacs was written by Gosling.
--**-Emacs: emacsrefs        (Text Fill)--L5--All----------------------
```

通过执行这个宏，我们创建了一个名为“emacsrefs”的编辑缓冲区并把“emacstext”编辑缓冲区里出现单词“Emacs”的句子都复制到它里面来了。

完成操作时有可能看不到“emacsrefs”编辑缓冲区，但这不是故障：只要输入“**C-x b emacsrefs**”命令就能让这个新编辑缓冲区出现在屏幕上。

类似于上面这个例子中的情况，在定义一个宏时，可以在任意数量的编辑缓冲区之间来回切换。宏不必局限于一个编辑缓冲区。跨越多个编辑缓冲区的宏比较难于调试，涉及到的编辑缓冲区越多，要想记住光标和文本块标记的位置也就越难。另外，用户对正在访问哪一个编辑缓冲区这一问题的判断也更容易出现错误。因此，对于这种情况，明确地给出编辑缓冲区的名字（就像我们在例子里所做的那样）就将是一个比较好的做法。不过，一旦大家习惯了使用多个编辑缓冲区进行编辑，就肯定会对自己能够轻松完成的工作之多感到惊讶。

窗口在某些宏里也很有用，同样必须给它们以足够的重视。比较好的做法是在屏幕上只有一个窗口的时候开始执行宏编辑命令，让宏来打开其他的窗口，到最后再（用“**C-x 1**”组合键）把窗口关闭得只剩下一个。如果在录制宏的时候屏幕上有2个窗口，可在执行这个宏的时候屏幕上却有4个窗口，那么其后果将完全无法预料！一般来说，用“**C-x b** *buffername*”命令移动到一个有名字的编辑缓冲区要比用“**C-x o**”组合键移动到另一个窗口更值得信赖（后者会因过于含糊而不适用于一般情况）。这是因为所谓“另一个窗口”有可能是任何东西，比如“`*Help*`”编辑缓冲区、“`*Completion*`”编辑缓冲区、“`*shell*`”编辑缓冲区等等。移动到一个有名字的编辑缓冲区就精确得多，不管目标编辑缓冲区是如何（或者是否）显示在屏幕上的，它永远会进入正确的地方。

向现有的宏里增加编辑命令

虽然不能对宏进行编辑，但可以用“**C-u C-x (**”组合键在宏的尾部添加一些编辑命令。这个命令先执行完已经录制好的宏，然后会等待添加更多的击键动作。添加完击键动作之后，按下“**C-x)**”组合键将结束宏的录制工作。

这个命令特别适合用来补充不太完整的宏。如果宏编辑命令不太完整，解决方法之一是在宏执行完毕之后再把剩下的命令亲手输进去，另一种方法则是把必要的额外击键动作追加到这个宏的末尾。如果发现录制好的宏没有做好下次循环执行的准备工作，那么还是把必要的额外击键动作追加到这个宏的末尾比较好。在前面那个把包含单词“Emacs”的句子，复制到另一个编辑缓冲区里的例子里，我们最初录制的宏里就没有那个用来移动到第一个编辑缓冲区去的命令，它是后来才补充进去的。

示例

我们来看看下面这个示例。假设只想把文档各个段落里第一次出现单词“Emacs”的句子收集到一起，而这个文档恰好是在**troff**软件（它是UNIX内置的排版系统）里写成的。用**troff**写成的文档其各个节都是以“*.NH*”标记开始的，正好可以利用这一点识别出它们。我们来修改刚才的宏，把这个功能增加进去。如下所示：

键盘操作	动作
C-u C-x (	开始录制一个宏；状态行上将出现“Def”字样
C-s .NH	已经找到单词“*Emacs*”并返回原来的编辑缓冲区。接下来查找下一节的开始
RETURN	如果查找成功，则停止查找操作
C-x)	结束宏的录制工作；状态行上的“Def”字样将消失

命名并保存宏

在很多时候，一个宏只是某个一次性问题的临时解决方法。它虽然能够减少工作量，但到不了保存起来以供今后再次使用的程度。可有时候，录制的宏能够完成需要经常做的事情，于是想把这些宏保存起来。比如，假设你是一位作家，经常需要把某个单词或者某个短语设置为某种字体，比如粗体等。虽然具体需要使用什么样的命令，取决于使用的是哪一种文本排版软件包，但录制一个宏来把单词设置为粗体肯定能简化操作。可是，不管录制宏的工作有多简单，每次录制这个宏时还是得输入一大堆的东西，你迟早会厌烦这种重复性的劳动。如果文字处理量确实很大，那么就可能需要在多次编辑会话中频繁地用到这个宏；于是，你既不想让它在你退出Emacs的时候消失，也不想让它在你录制另外一个宏的时候被覆盖。

说的再明确点，你想要的是一组能够随时用来把单词、句子和段落等设置成粗体字或斜体字的“永久性的”宏。如果你是一位程序员，那么需要的就是一些能够帮助对程序中的注释进行正确排版的宏（Emacs已经在这方面为我们准备了一些内置的编辑功能，请参考第十二章）。

在这一小节里，我们将介绍怎样才能把宏保存起来，以便它们能够用在其他的编辑会话过程中。要想把一个宏保存起来，请按以下步骤进行操作：

1. 录制这个宏（如果已经录制好，就不用再做一次了）。
2. 输入“**ESC x name-last-kbd-macro**”并按下回车键。然后给宏起一个名字，再按下回车键。最好给它起一个不那么有“Emacs味”的名字以避免让Emacs把宏和它自己的某个命令弄混。在进行了这样的操作之后，Emacs就会记住这个宏，并且在以后的编辑会话里都不会忘记它。如果想使用这个宏，输入命令“**ESC x** *name*”（其中的“*name*”就是这个宏的名字）即可。

3. 如果想把这个宏定义永久地保存起来，还必须再进一步：必须把这个宏定义插入到一个文件里保存起来。用“**C-x C-f** *filename* **RETURN**”命令打开那个准备把宏定义插入到其中的文件，然后用“**ESC >**”组合键移动到它的文件尾。如果想让这个宏在每次进入Emacs的时候都可用，就需要打开“*.emacs*”文件；如果不是这样，则可以给那个文件起一个与宏相一致的名字，或者就用一个名为“*macros*”的文件把所有的宏定义都保存在里面。

4. 输入“**ESC x insert-kbd-macro RETURN** *macroname* **RETURN**”命令。Emacs将把代表这个宏的LISP代码插入到这个文件里。能不能看懂这些代码并没有多大关系。

5. 按下“**C-x C-s**”组合键以保存那个文件。

现在，宏就永久性地保存在那个文件里了。

如果各位胆子够大，还可以亲自编辑那个文件以修改宏定义。但我们并不鼓励这样做。Emacs保存起来的宏定义是击键动作的一个脚本，想编辑它是不那么容易的，稍有不慎，就会把事情搞成一团糟。

执行一个已命名的宏

给宏起了名字之后，就可以用“**ESC x** *macroname* **RETURN**”命令来执行它。还可以用“**ESC** *n* **ESC x** *macroname* **RETURN**”命令来重复执行这个宏“*n*”次。可以同时使用任意个有名字的宏而不仅仅局限为一个 —— 就像键盘宏命令（keyboard macro）的情况一样。

如果把宏保存在“*.emacs*”文件里，要想执行它们就很容易：因为只要运行Emacs，那些宏定义就会被加载进来。只需输入“**ESC x** *macroname* **RETURN**”命令即可。

如果把宏保存在其他的文件里，它们就不会被自动加载进来。在执行某个宏编辑命令之前，必须用“**ESC x load-file**”命令让Emacs读入宏定义文件。比如，假设定义了一个名为“*bold-word*”的宏，并且已经把它保存在子目录“*~/emacs*”（主目录中的*emacs*子目录）里的文件“*nroff.macs*”里。如果想使用这个宏，就先得执行“**ESC x load-file ~/emacs/nroff.macs RETURN**”命令。如果想让这个文件被自动加载进来，就必须把下面这条语句添加到“*.emacs*”文件里：

```
(load-file '~/emacs/nroff.macs')
```

不管宏定义文件是由用户本人手动加载的还是通过“*.emacs*”文件自动加载的，只要加载了这个文件，就随时都能发出“**ESC x bold-word RETURN**”命令。

示例

下面是个很有用的小技巧。假设已经录制并保存了一个名为“*make-double-spaced-text*”的宏。不过，要是每次用到这个宏的时候都得输入“**ESC x make-double-spaced-text RETURN**”就未免有点浪费时间了；而且，说实话，这还不如手动按回车键简单。那像这样的问题又该如何解决呢？可以定义一个键盘宏命令（keyboard macro）来执行永久性宏编辑命令。

键盘操作	动作
C-x (	开始键盘宏命令的定义
ESC x make-double-spaced-text RETURN	执行保存起来的宏
C-x)	结束键盘宏命令的定义

现在，只需按下“**C-x e**”组合键，就能执行 **make-double-spaced-text** 宏。

再引申一步，还可以把宏编辑命令绑定到某个按键上。对这个问题的探讨请参考第十一章。我们先在这儿举一个例子，如果想把宏“**make-double-spaced-text**”绑定到“**C-c C-d**”组合键上，需要使用“**ESC x local-set-key C-c C-d make-double-spaced-text RETURN**”命令。（这种把宏绑定到某个按键上去的方法只对本次Emacs会话有效。如果想了解怎样才能让按键绑定成为永久性的，请参考第十一章内容。）

建立复杂的宏

我们已经把录制、执行和保存键盘宏命令的基本操作都介绍给大家了。下面再介绍几个Emacs允许用在宏编辑命令里的高级操作：暂停宏的执行以等待键盘输入和在宏里插入一个查询。

暂停宏的执行以等待键盘输入

有时候，暂停一个宏的执行以便通过键盘输入一些东西是很有必要的。比如，如果你有很多备忘录要写，就可以录制这样一个宏：它负责给出一个备忘录模板，而且会在适当时刻暂停执行以便填写有关的内容(比如日期和收信人等)。你可以用在宏里插入一个递归编辑的方法来完成这项工作(或者其他类似的工作)。这个“递归编辑”就好像是在说“先暂停一下，让我输入点东西，等我弄好了再继续执行宏编辑命令”。

在对宏进行定义的过程中，如果在某个地方按下了“**C-u C-x q**”组合键，就表示想在那里插入一个递归编辑；Emacs也就会从那个地方开始进入一个递归编辑(状态行上将出现一对方括号，用户可以根据这一点来判断自己是否在递归编辑里)。在递归编辑期间输入的任何东西都不会被录制到这个宏里。可以输入任何东西，但结束时必须用“**ESC C-c**”组合键退出递归编辑。按下“**ESC C-c**”组合键时，状态行上的方括号将消失；而一旦画面上不再有方括号，就表示已经离开递归编辑——此后输入的任何东西又将被录制到宏里。可以根据需要在宏里插入任意多次暂停。

示例

请看下面这个示例。这个宏将在屏幕上显示一个备忘录模板，并且会用递归编辑等待输入日期、收信人姓名和发信人姓名。

键盘操作	动作
C-x (	开始录制一个宏；状态行上将出现“Def”字样
Memorandum	把“Memorandum”显示到屏幕上
ESC s	使单词“Memorandum”居中
RETURN RETURN	把光标下移两行
Date: TAB	把“Date:”显示到屏幕上
C-u C-x q	进入一个递归编辑，在此期间中的击键动作不会被录制为宏的一部分
ESC C-c	退出递归编辑
RETURN RETURN	把光标下移两行
To: TAB	把“To:”显示到屏幕上
C-u C-x q	进入一个递归编辑
ESC C-c	退出递归编辑

键盘操作	动作
RETURN RETURN	把光标下移两行
From: TAB	把“From:”显示到屏幕上
C-u C-x q	进入一个递归编辑
ESC C-c	退出递归编辑
RETURN RETURN	把光标下移两行
ESC 78 -	用连字符画一条横穿屏幕的直线
RETURN	插入一个空白行
C-x)	结束宏的录制工作；状态行上的“Def”字样将消失

以下几个画面演示了这个宏的执行情况。

按下：**C-x e**

```
Buffers Files Tools Edit Search Help
                    Memorandum

Date:  ▮

--**-Emacs: memo          [(Text Fill)]--L3--All----------------
```

宏暂停执行以等待输入日期。

输入：**10/31/97 ESC C-c**

```
Buffers Files Tools Edit Search Help
                    Memorandum

Date:  10/31/97

To:  ▮

--**-Emacs: memo          [(Text Fill)]--L5--All----------------
```

宏暂停执行以等待输入收信人姓名。

输入：**Fred ESC C-c**

```
Buffers Files Tools Edit Search Help
                              Memorandum

Date:   10/31/97

To:     Fred

From:   ▮

--**-Emacs: memo              [(Text Fill)]--L7--All--------------------
```

宏暂停执行以等待输入发信人姓名。

输入：**Maurice ESC C-c**

```
Buffers Files Tools Edit Search Help
                              Memorandum

Date:   10/31/97

To:     Fred

From:   Maurice

------------------------------------------------------------------------
▮

--**-Emacs: memo              (Text Fill)--L10--All----------------------
```

宏用短划线字符画出一条横贯画面的直线，以隔开备忘录的标题部分和内容部分。

由宏负责的工作到这里就结束了；接下来将由你来输入备忘录的内容部分。当然，你也可以回头去编辑自己刚才填写的任何一组文字。

在宏里插入一个查询

对一个宏来说，如果用它来完成的工作越复杂，它的用途就越单一，它的适用面也就越窄。虽然宏能够用来做很多的事情，但它们毕竟不是程序——不能在宏里使用 **if** 语句、循环语句或者只有程序才具备的其他东西。说的再明确点，宏不能从用户那里接受输入，并根据这个输入采取必要的行动。如果各位需要的是这方面的功能，请去阅读第十三章。

但我们还是有办法让宏从用户那里获得输入数据的，虽然这个办法有一定的局限性。宏可以在运行时向用户提出查询；它的工作情况与查询－替换操作很相像。如果想

录制一个这样的宏，就必须在这个宏的录制过程到达想让它暂停执行的地方时按下“**C-x q**”组合键。不会立即发生什么事情，可以像平时那样继续录制这个宏直到它完成。

可等到执行这个宏的时候，就会发生一些有趣的事情。当宏执行到按下“**C-x q**”组合键的地方时，Emacs 会在辅助输入区里给出一个如下所示的查询：

```
Proceed with macro? (y, n, RET, C-l, C-r)
```

下面是对这个查询的回答，它与我们在第三章里介绍的查询－替换操作的情况很相似：

- 回答“**y**”表示继续执行这个宏，执行完毕则进入下一次循环——如果有下一次循环。
- 回答“**n**”表示停止执行这个宏，但进入下一次循环（如果有）。
- 按回车键表示停止执行这个宏并取消尚未执行的循环。
- 按“**C-r**”组合键将开始一次递归编辑，可以做任何编辑或移动；退出这次递归编辑时，宏将从刚才被打断的位置开始准备继续执行。按“**ESC C-c**”组合键退出递归编辑；Emacs 会询问是否想继续执行这个宏，输入“**y**”表示“是”，输入“**n**”或回车键表示“否”。
- 按“**C-l**”组合键将把光标所在的文本行放到画面的中央（以便人们能够清楚地看到它前后的上下文）。类似于按下“**C-r**”组合键的情况，Emacs 也会询问是否想继续执行这个宏，输入“**y**”、“**n**”或回车键作为回答。
- 按“**C-g**”组合键（这个组合键没有被列为一个操作选项）取消这个查询和整个宏；它的作用类似于按下回车键。

示例

假设已经录制了一个用来把某程序的注释复制到另外一个编辑缓冲区里的宏。程序中的注释都被上、下两行星号包围着，所以这个宏将以查找一个星号字符串的查找命令开始。但并非所有的星号字符串后面都跟有注释，也并非所有的注释都值得复制。因此，在查找命令的后面加上一个查询，以便能够有选择地决定被找到的注释有哪些是值得复制的。

下面是这个宏的录制过程。

键盘操作	动作
C-x (	开始录制一个宏；状态行上将出现“Def”字样
C-s ******	查找一个星号字符串
RETURN	如果查找成功，退出查找操作
C-x q	在宏里插入一个查询；Emacs 会在宏执行到这个位置时询问是否想继续前进
C-e	移动到行尾
C-f	移动到下一行的行首
C-@	设置文本块标记
C-s *** RETURN**	查找配对的星号字符串
C-p	上移一行
C-e	移动到行尾
ESC w	把注释复制到删除环里
C-e C-f C-e C-f	把光标移动到这段注释的紧后面
C-x b comments	移动到名为“comments”的编辑缓冲区
C-y	把注释插入到编辑缓冲区里
RETURN	移动到下一行
C-x b	返回原来的编辑缓冲区
C-x)	结束宏的录制工作；状态行上的“Def”字样将消失

LISP 函数 —— 宏的补充

宏是简化重复性工作的一种重要工具。它们允许各位用自己录制的命令来完成复杂的任务，而不需要去学习任何还不知道的内容，只要知道如何用基本的Emacs命令移动光标和编辑文本就足够了。即使是一位Emacs的初学者，也能毫无困难地使用宏。

Emacs的灵活性几乎是无限的，但宏并不能解决所有的问题。在许多情况下，除了编写一个LISP函数来完成预期的工作之外没有任何其他的选择。如果你了解LISP或者愿意去学习这种程序设计语言，就可以自行编写LISP函数来解决那些比键盘宏

命令所能对付的情况更复杂的问题。我们将在第十三章介绍编写LISP函数的基本方法。

表 10-1 对这一章里介绍的与宏有关的的编辑命令进行了汇总。

表 10-1：与宏有关的 Emacs 命令

键盘操作	命令名称	动作
C-x (	**start-kbd-macro**	开始录制一个宏
C-x)	**end-kbd-macro**	结束宏的录制工作
C-x e	**call-last-kbd-macro**	执行最后一次录制的宏
ESC *n* **C-x e**	**digit-argument** 和 **call-last-kbd-macro**	把最后一次录制的宏执行 *n* 次
C-u C-x (	**universal-argument; start-kbd-macro**	执行最后一次录制的宏，然后等待追加新的按键动作
（无）	**name-last-kbd-macro**	给刚才录制的宏起个名字（做好保存这个宏的准备）
（无）	**insert-last-keyboard-macro**	把起了名字的宏插入到一个文件里
（无）	**load-file**	加载内容为宏命令定义的文件（用来加载保存起来的宏文件）
（无）	宏名	执行保存起来的某个键盘宏命令
C-x q	**kbd-macro-query**	在宏定义里插入一个查询
C-u C-x q	（无）	在宏定义里插入一个递归编辑
ESC C-c	**exit-recursive-edit**	退出递归编辑

第十一章

对 Emacs 进行定制

本章内容:

- 键盘的定制
- 终端支持
- Emacs 变量
- Emacs 的 LISP 程序包
- 自动模式的定制

Emacs 几乎能按照所能想像到的任何方式进行定制。屏幕上显示的任何东西、任何命令、任何按键动作和任意一条消息等等几乎都能被定制。大家可能已经猜到了，几乎所有的定制工作都需要通过 Emacs 的启动文件“*.emacs*”来完成。

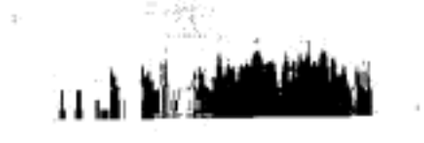
有些定制工作需要对 Emacs 中的 LISP 程序设计有一定的了解（请参考第十三章）；另外一些定制工作则相当简单，用不着那么大的学问。在这一章里，我们将向大家介绍一些简单而又不需要程序设计知识的定制工作。就目前情况而言，大家只要知道每一个 Emacs 命令都对应一个 LISP 函数就足够了。LISP 函数的基本格式如下所示:

```
(function-name arguments)
```

比如，如果想把光标往前移动一个单词，就需要按下“**ESC f**”组合键。而这个动作其实是通过调用下面这个 LISP 函数来实现的:

```
(forward-word 1)
```

这一章会介绍很多与Emacs的定制和它的高级用法有关的问题。我们将从读者最想知道，或者说最需要知道的两个与Emacs密切相关的事物（即键盘和终端）开始讲起。大家将学习怎样对键盘命令进行定制，怎样设置和使用键盘上的特殊键，怎样对终端进行设置才能让它正确地与Emacs配合工作。此外，本章还对Emacs变量和给Emacs添加额外功能的程序包进行了介绍，大家可以利用这些东西去改变和拓展



Emacs的操作行为。详细的变量清单和程序包清单收录在本书的附录三和附录四里。更高级的定制技巧请参考第十三章。

在我们开始探讨这些话题之前，有两件关于“*.emacs*”文件的重要事项要告诉给大家。首先，如果向“*.emacs*”文件插入代码时出了错，那么插入的东西就可能会造成 Emacs 不工作或者工作不正常。如果遇到这样的问题，有个简单的方法可以让 Emacs 在不运行“*.emacs*”文件的情况下启动：用命令行选项“**-q**”启动 Emacs，Emacs将不运行“*.emacs*”文件。然后可以检查这个文件，看它在什么地方出了错。

其次是在定制Emacs环境方面我们所能给大家的最重要的一句忠告：多与其他用户进行交流。在终端配置问题上陷入困境或者在设置某个特殊编辑模式遇到麻烦的时候，其他用户可能早就遇到过并解决了这类问题。向他人求助决不是“不诚实”或者“懒惰”，事实上，Emacs的缔造者是鼓励这样做的，他们更愿意看到他们的软件被大家所共享而不是独藏。Emacs 甚至提供了一种测试其他用户的“*.emacs*”文件的简单方法：用选项“**-u** *username*”启动 Emacs，则该 *username* 的“*.emacs*”文件将取代原先的“*.emacs*”文件运行。

键盘的定制

也许Emacs用户最想定制的东西就是运行Emacs编辑命令的按键动作了。我们说过，按键动作是通过“按键绑定”与命令关联起来的。

事实上，Emacs 里的每个按键动作都会运行一个命令。对应于可打印字符（字母、数字、标点符号和空格）的按键所运行的命令是**self-insert-command**，其作用是把被按下的键插入到当前编辑缓冲区的光标位置上。（可以给 Emacs 新手开个愚人节玩笑 —— 把他们的可打印字符的按键绑定改掉。）

对大多数操作而言，默认的按键绑定集是很适当的；但在很多情况下，可能需要添加或者改变按键绑定。Emacs 有好几百条命令，其中只有一部分带按键绑定。正如大家已经知道的，输入“**ESC x** *command-name* **RETURN**”就能输入那些不带按键绑定的命令。

如果经常要用到一个不带绑定的命令，就可以考虑把它绑定到一个按键组合上以方

便工作。有的人也许想把键盘上的特殊键（比如箭头键、数字小键盘或者功能键等）与经常使用的命令关联起来。如果你使用的是一个运行X窗口系统的图形终端，那么可能还需要对鼠标动作和菜单选项进行定义。这一章将介绍一些按键绑定和修改按键绑定的例子，鼠标和菜单事件的绑定将留做第十四章的内容。

另一个现在就必须弄清楚的重要概念是“键位映射图”（keymap），它是多个按键绑定构成的一个集合。Emacs最基本的默认按键绑定保存在一个名为“**global-map**”的全局键位映射图里。与全局键位映射图相对应的是“局部（local）键位映射图”，每个编辑缓冲区都有一个自己的局部键位映射图。局部键位映射图的作用是实现某具体编辑模式（比如C模式、文本模式、shell模式等）里的编辑命令，不同的编辑模式有它们各自的键位映射图；进入某个编辑模式时，它所对应的键位映射图就会被安装为局部键位映射图。按下某个按键的时候，Emacs会先查看当前编辑缓冲区的局部键位映射图（如果有）里有没有对它进行定义。如果Emacs没有在局部键位映射图里找到与按键对应的定义项，就会查看**global-map**键位映射图。如果找到了与按键对应的定义项，那么按键的关联命令就会被执行。

那么、绑定到组合键上的命令，比如“**ESC d**”（命令名是**kill-word**），是怎样被查到和执行的呢？情况是这样的：“**C-x**”、“**ESC**”和“**C-c**”等按键其实都被绑定到一些特殊的内部函数上，而这些函数会让Emacs等到另一个按键被按下后再去另外一个键位映射图里查找该按键的绑定定义；如果后面的按键在一秒钟后还没有被按下，那么会在辅助输入区里显示“C-x-”。供“**C-x**”和“**ESC**”使用的键位映射图分别叫做**ctl-x-map**和**esc-map**；而“**C-c**”组合键则是为关联于某具体编辑模式（比如C模式和shell模式等）的局部键位映射图保留的。

举个例子，按下“**ESC d**”组合键的时候，Emacs将先到该编辑缓冲区的局部键位映射图里去查找，假设它没有在那里找到与之对应的定义项。于是，Emacs会再到**global-map**里去查找。Emacs在这里找到一个对应于“**ESC**”的定义项，它定义了一个特殊的函数（叫做“**ESC prefix**”）；这个函数将等待下一个按键的到来，然后根据**esc-map**键位映射图来决定要执行哪一个命令。现在，按下“**d**”的时候，“**ESC prefix**”函数将在esc-map键位映射图里查找有无对应于字母“**d**”的定义项，它找到了**kill-word**命令，于是就执行这个命令。

通过在键位映射图里增加（或者改写现有的）定义项的方法，能创造出自己的按键

绑定。有 3 个函数可以用来完成这一操作，它们是 **define-key**、**global-set-key** 和 **local-set-key**。这 3 个函数的格式如下所示：

```
(define-key  keymap  "keystroke"  'command-name)
(global-set-key "keystroke"  'command-name)
(local-set-key "keystroke"  'command-name)
```

注意，“*keystroke*”要用双引号引起来，“*command-name*”的前面要用单引号。这是 LISP 语言的语法所要求的，详细情况请参考第十三章。“keystroke”是一个或者多个字符，可打印字符或特殊字符都可以。如果是特殊字符，就必须使用表 11-1 给出的表达记号。

表 11-1：特殊字符的表达记号

特殊字符表达记号	定义情况
\C-x	**C-x**（其中“x”是任意字符）
\C-[或 **\e**	**ESC**；转义符
\C-j 或 **\n**	**LINEFEED**；换行符
\C-m 或 **\r**	**RETURN**；回车符
\C-i 或 **\t**	**TAB**；制表符

于是，字符串“**abc\C-a\ndef**”就等于是把“**abc**”、“**C-a**”、换行符 **LINEFEED** 和“**def**”合并起来而得到的一个字符串。注意，控制字符是不区分大小写的。换句话说，“**\C-A**”和“**\C-a**”是一样的。

函数 **define-key** 是最通用的，它可以用来绑定任何一个键位映射图里的按键。**global-set-key** 只能绑定全局键位映射图里的按键，因为 **global-map** 只有一个；“**(global-set-key . . .)**”和“**(define-key global-map . . .)**”的作用完全一样。**local-set-key**的作用是绑定当前编辑缓冲区局部键位映射图里的按键，因此它只能用来在某次 Emacs 会话期间临时性地定义按键绑定。

下面是一个简单的键盘定制示例。假设你正用某种程序设计语言写程序代码。对程序进行编译时，编译器给出的出错信息带有出错语句的行号，想快速移动到源代码文件的那一行去改正错误（注 1）。这需要使用 **goto-line** 命令，但这个命令在默认的

注 1：　对付这种情况其实有一个更好的办法，我们将在下一章对此做详细讨论。

情况下没有被绑定到任何按键上。现在，假设想把它绑定到“**C-x l**”组合键上。于是，需要添加到“*.emacs*”文件里的语句应该是:

```
(define-key ctl-x-map "l" 'goto-line)
```

这就把**ctl-x-map**键位映射图里的“l”字母格与**goto-line**命令绑定在一起。也可以用下面两条语句中的任何一条来达到同样的目的:

```
(define-key global-map "\C-xl" 'goto-line)
(global-set-key "\C-xl" 'goto-line)
```

这两条命令能够达到同样的目的，而且使用起来可能更方便，因为很可能记不住对应于“**C-x**”的键位映射图叫做**ctl-x-map**。

再看一个例子。我们在第二章里把“**C-x ?**”组合键绑定为**help-command**命令，把“**C-h**”组合键绑定为**backward-char**命令。这两个按键绑定的定义情况如下所示:

```
(global-set-key "\C-x?" 'help-command)
(global-set-key "\C-h" 'backward-char)
```

这两个定义也可以写成下面这样:

```
(define-key ctl-x-map "?" 'help-command)
(define-key global-map "\C-h" 'backward-char)
```

把一个按键绑定插入到“*.emacs*”文件里之后，需要“运行”（或者叫“求值”）这个文件以便改动能够生效。用来完成这一工作的命令是“**ESC x eval-current-buffer RETURN**”。更好的方法是按下“**C-x C-e**”组合键，它（我们将在下一章对它做进一步的讨论）的作用是只执行光标所在的那一行上的LISP代码。如果没有进行上述操作，那么所做的改变就只有等到下次启动Emacs时才会生效。

特殊键

更复杂的键盘定制任务是把命令绑定到终端上的特殊键上，比如方向键、数字小键盘或者功能键等。这个级别的定制需要多做一些准备工作。如果各位喜欢使用特殊键，那么多花些功夫还是值得的。

不论什么型号的终端，它上面的特殊键都会产生一组“字符代码”，即由多个字符组

成的一个序列，这些序列通常由一个特殊的字符（比如“ESC”字符）打头。因此，需要做的第一件事是找出终端上的特殊键都会产生哪些字符代码。比如，许多终端都遵守 ANSI 标准，即当数字小键盘切换到应用模式（application mode，即 PC 键盘上 NumLock 指示灯不亮时的状态）时，它们能够产生复杂的代码序列；而切换到普通的数字模式（即 PC 键盘上 NumLock 指示灯亮时的状态）时，数字小键盘上的按键输入的将是数字和标点符号（英文句号、逗号和短划线）。在应用模式下，数字小键盘所产生的字符代码全都以“**ESC O**”打头，后面跟着一些其他的字符，比如，按键“7”所对应的字符代码将是“**ESC O w**”。

终端上的特殊键所产生的字符代码，可以在该终端的使用手册里查到。如果找不到有关资料，可以用 **quoted-insert** 命令把那些字符代码查出来。具体操作是：先切换到一个空白的编辑缓冲区里——比如“`*scratch*`”编辑缓冲区，然后先按下“**C-q**”组合键（注 2），再依次按下各特殊键。控制字符的前面会有一个“^”字符，比如“**^X**”代表的就是“**C-x**”（注 3）。有些特殊字符也会被显示为控制字符，比如下面这几个：

ESC 转义符　　^[
RETURN 回车符　　^M
LINEFEED 换行符　　^J

TAB 制表符将以制表位（即一个或者多个空格）的面目出现。所以，数字小键盘上的按键“7”所对应的字符代码将是“**^[Ow**”。

但这种利用“**C-q**”组合键的技巧有很大的局限性，它只能对单个字符进行处理。因此，如果特殊键会产生多于一个字符的序列，则从第二个字符开始，以后的字符都必须是普通的（可打印的）；否则，Emacs 就将把它们解释为编辑命令。但根据有关标准，任何键盘上的特殊键所产生的字符代码其构成都是一个特殊字符（比如转义符“ESC”）加几个可打印字符，所以从原则上讲应该是安全的。

某些终端上的特殊键不适合做 Emacs 定制，因为它们产生的字符代码与某些基本的 Emacs 命令一模一样。比如，Wyse WY-50 终端上的方向键所产生的字符代码就分

注 2：　或者“**ESC q**”组合键——如果遇到前面介绍过的流控制问题。

注 3：　注意：控制字符虽然被显示为两个字符，可它们实际占用的只是一个字符的位置。如果在光标位于某个控制字符上时按下“**C-f**”组合键，则光标将向右移两个字符。

别是“**C-k**”（上）、“**C-j**”（下）、“**C-h**”（左）和“**C-l**”（右）；完全没有必要去重新绑定这些按键！有些终端虽然有数字小键盘，可它却没有应用模式。换句话说，它们的数字小键盘只能产生数字和标点符号。很明显，这样的数字小键盘也不能用于Emacs定制。这里有一条基本规则：只有在特殊键所产生的字符代码是以一个特殊字符（比如转义符“ESC”）开头，并且至少有3个字符长时，才能在Emacs里对它们进行定制。

如果终端上的特殊键符合这个要求，那么可以在“*.emacs*”文件里写一些LISP代码把这些特殊键和挑选出来的命令绑定在一起。这些LISP代码必须先创建一个与特殊按键的常用前缀相对应的键位映射图，然后再把各个特殊键绑定到某个命令上。请看下面这段LISP代码，它可以用来绑定ANSI兼容终端（比如DEC VT100或VT200型终端）的方向键、功能键和数字小键盘上的按键：

```
(setq term-file-prefix nil)
(send-string-to-terminal '\e=')

(setq SS3-map (make-keymap))   ;set up the keymap for the keypad
(define-key global-map "\eO" SS3-map)

(define-key SS3-map "A" 'previous-line)          ; up arrow
(define-key SS3-map "B" 'next-line)              ; down-arrow
(define-key SS3-map "C" 'forward-char)           ; right-arrow
(define-key SS3-map "D" 'backward-char)          ; left-arrow

(define-key SS3-map "M" 'isearch-forward)        ; Enter

(define-key SS3-map "P" 'other-window)           ; PF1
(define-key SS3-map "Q" 'query-replace)          ; PF2
(define-key SS3-map "R" 'replace-string)         ; PF3
(define-key SS3-map "S" 'suspend-emacs)          ; PF4
(define-key SS3-map "l" 'delete-char)            ; ,

(define-key SS3-map "n" 'delete-next-word)       ; .
(define-key SS3-map "p" 'isearch-backward)       ; 0
(define-key SS3-map "q" 'beginning-of-line)      ; 1
(define-key SS3-map "r" 'find-file)              ; 2
(define-key SS3-map "s" 'end-of-line)            ; 3
(define-key SS3-map "t" 'backward-word)          ; 4
(define-key SS3-map "v" 'forward-word)           ; 6
(define-key SS3-map "w" 'beginning-of-buffer)    ; 7
(define-key SS3-map "x" 'save-buffer)            ; 8
(define-key SS3-map "y" 'end-of-buffer)          ; 9
```

在上面的代码里，分号后面的文字是一些注释，Emacs不会执行它们。

这段代码的头两行需要做些解释。Emacs有一些内建的LISP代码能够对某些型号的终端（尤其是 DEC VT100 和 VT200 系列以及各种工作站）上的特殊键进行绑定，但对其他型号的终端就无能为力了。上面这段代码的第一行语句将消除Emacs内置LISP代码（如果有的话）的设置情况，从而确保这段代码的其他语句对任何型号的终端都能正确工作。

第二行语句是专门为VT100 / VT200系列终端而准备的，这类终端的数字小键盘在应用模式和数字模式下都能使用。在数字模式下，数字小键盘上的按键只是数字、短划线、逗号和英文句号的“副本”。很明显，这个模式在我们的示例里是没有用的；所以我们需要强制终端进入应用模式。向终端发送“ESC =”序列就能达到这一目的，这就是第二行中 **send-string-to-terminal** 命令的由来。注意：如果终端不是 *VT100*、*VT200* 或兼容型号，就不要使用这一行。

再看接下来的两行语句。它们一个用来创建键位映射图，一个用来把新键位映射图挂到全局键位映射图上，好让新键位映射图发挥作用。第一条语句创建了一个名为 **SS3-map** 的键位映射图；可以随便给自己的键位映射图起名字，我们之所以会给这个键位映射图起一个这样的名字是因为DEC终端的文档把“**ESC O**”前缀叫做“SS3”。第二条语句把这个键位映射图绑定到全局键位映射图的“**ESC O**”序列上（注 4）。请注意：不仅仅是编辑命令，键位映射图也可以被（递归地）绑定到键位映射图里的字母格里。于是，只要“**ESC O**”被键入，即只要某个能产生一个以“**ESC O**”打头的字符序列的按键被按下，这个LISP命令就会让global-map键位映射图把控制权传递给 **SS3-map**。

上面这段代码中的其他LISP命令把各方向键和数字小键盘上的按键分别绑定到相应的Emacs编辑命令上。4 个方向键被分别绑定到相应的光标移动命令上，而数字小键盘上其他按键的绑定情况如下图所示：

注 4：它实际上是把 **SS3-map** 绑定到 **esc-map** 的“O”字母格里；有关说明请参考本章开头部分对键位映射图的解释。

PDG

<table>
<tr><td>F1
移动到
其他窗口</td><td>F2
查询－
替换</td><td>F3
替换
字符串</td><td rowspan="3">F4
挂起
Emacs</td></tr>
<tr><td>7
移动到编辑
缓冲区的开始</td><td>8
保存编辑
缓冲区</td><td>9
移动到编辑
缓冲区的末尾</td></tr>
<tr><td>4
后移一个
单词</td><td>5</td><td>6
前移一个
单词</td></tr>
<tr><td>1
移动到
行首</td><td>2
查找文件</td><td>3
移动到
行尾</td><td rowspan="2">ENTER

向文件头方向
开始递增
查找工作</td></tr>
<tr><td colspan="2">0
向文件尾方向
开始递增查找工作</td><td>.
删除一个
单词</td></tr>
</table>

注意：按键“5”和“-”（短划线）没有被绑定。作为一个练习，想想需要写出什么样的LISP代码才能把这两个按键绑定到自己最常用的两个Emacs编辑命令上。

终端支持

显示设备的种类有很多，比如终端、PC个人电脑和图形工作站等等。只要不是“最笨的”显示设备，Emacs就都能在其上运行；但它能否顺利工作则要取决于几个UNIX因素。

第一个因素叫做“环境变量”。登录到一个UNIX系统上的时候，这些变量中已经有几个是设置好的。环境变量的值（通常是一些字符串）由UNIX操作系统的shell负责管理，它们的作用是作为计算环境的各项特征的指示，shell会把它们传递到发出的每一个命令上。

为了便于区分，环境变量的名字全部采用大写字母。典型的环境变量包括**PATH**和**SHELL**，前者告诉UNIX要在哪些目录里查找命令，后者的值是当前使用的shell的名字。我们关心的环境变量是**TERM**，它的作用是告诉UNIX操作系统和有关程序（比如Emacs）正在使用哪一种终端。

TERM一般在你登录时设置，具体负责这项工作的或者是某个系统全局性的初始化文件，或者是你自己的“*.profile*”文件（C shell用户使用的是“*.login*”文件）。在

某些系统上，甚至需要在登录时给出终端类型。要想查看**TERM**的值，请在UNIX提示符处输入“**echo $TERM**”命令；美圆符号“$”表示想查看的是环境变量的值。如果想对**TERM**进行设置，请输入“**TERM=termname**”（适用于Bourne、Korn和bash等几种shell。注意等号两端不能有空格）或“**setenv TERM termname**”（适用于C shell）命令。如果不知道自己的终端类型，可以询问系统管理员。如果他也不知道，请继续往下看，你很快就会知道怎样才能把它查出来。

与终端有关的另一个UNIX因素是目录*/etc*里的**termcap**文件，这个文件对终端的物理指标进行了定义，比如字符的宽度和高度、终端本身用来完成卷屏、反显和光标移动等操作的硬件命令等。

从AT&T System V演化而来的UNIX版本使用的是**terminfo**而不是**termcap**文件；请询问系统管理员你所使用的UNIX属于哪一个系列。**terminfo**使用了一个比单个文件更复杂的数据库来描述终端的物理指标，但保留了许多与**termcap**文件相接近的特点（基本上做到了向上兼容）。

termcap数据项写得很精炼，非专业人员一般不容易看懂；而**terminfo**数据项是经编译得到的机读格式。不过，两种情况所支持的终端的名字还都比较容易辨认。

如果所用Emacs版本使用的是**termcap**，那么就能用UNIX的**grep**工具来搜索*/etc/termcap*文件。先根据它的制造商和型号猜测一个终端类型，然后输入命令：

```
% grep -i guess /etc/termcap
```

“**-i**”选项的作用是不让**grep**区分大小写。假如终端的型号是DEC VT100，就输入“**grep vt100 /etc/termcap**”命令。如果成功，**grep**将显示一些下面这样的输出行：

```
d0|vt100|vt100-am|dec vt100:\
```

用垂直线字符隔开的字符串都可以用做环境变量**TERM**的取值。这些字符串是同一种终端的不同名称，**termcap**文件里的大多数条目都是这样的。

如果所用Emacs版本使用的是**terminfo**，就可以通过**terminfo**数据库来检索终端类型。这个数据库组织结构为目录*/usr/lib/terminfo*下的一个目录树。这个目录树由很多（下级）目录构成，这些下级目录的名字都只有一个字符，或者是一个数字，或者是字母表里的一个字母。终端类型按其名称的第一个字符（数字或字母）被分别

归类到相应的下级目录里，下级目录里的每个文件代表着一个终端类型。比如，文件*/usr/lib/terminfo/v/vt100*里保存的就是VT100的数据项。这些文件都是独立的终端描述文件。

有两种方法可以查出终端的**terminfo**名称。第一种方法是先根据所猜的终端名称的第一个字母用**cd**命令进入*/usr/lib/terminfo*子目录里相应的下级子目录里，再用**ls**命令列出其中的终端描述文件并把最有“嫌疑”的那个文件挑出来。用这个文件名作为终端类型，把环境变量**TERM**设置为它的名字。

检索**terminfo**数据库的另一个方法是使用UNIX操作系统的**find**命令。输入:

```
% find /usr/lib/terminfo -name "guess" -print
```

注意，“guess”必须放在双引号里，并且允许包含匹配文件用的通配符，比如“*”和“?”。如果检索成功，**find**会把它找到的终端描述文件的名字列出来；再从中挑选一个作为环境变量**TERM**的值即可（不用加上目录名作为前缀）。

termcap文件（和**terminfo**数据库）的内容随系统的不同而千差万别。在UNIX作为商业产品的早期，某UNIX系统的**termcap**文件不支持各种流行终端并不是件稀奇的事。随着UNIX操作系统的发展，**termcap**文件也在成长；如今，一个**termcap**文件往往能支持好几百种终端类型。但尽管如此，有的终端还是有可能没有被包括在**termcap**或**terminfo**里；如果终端是由某个独立制造商（比如WYSE）生产的，那么就更有可能如此。不过，这类终端通常是主流终端的克隆产品（仿制VT100的情况最常见）或者可以按主流终端进行配置。可以让自己的终端仿真某个主流的产品，然后再把环境变量**TERM**设置为相应的终端类型。一般说来，大多数**termcap**文件和**terminfo**数据库都完全不支持终端的情况是极为罕见的（注5）。

许多功能——比如字符的插入功能和屏幕的部分卷屏功能——大多数现代终端用硬件就能实现，这就大大减轻了Emacs的负担。如果某项功能可以由终端的硬件来完成，Emacs将坐享其成；如果某项功能不能由终端的硬件来完成，那么Emacs能用软件仿真出同样的效果。Emacs甚至知道如何在缺少某些基本功能，比如反显功

注5: 对**termcap**文件和**terminfo**数据库的详细讨论请参考《termcap and terminfo》一书，它的作者是John Strang、Linda Mui和Tim O'Reilly，由O'Reilly & Associates出版公司出版。

能（Emacs的状态行要用到这一功能）的情况下“委曲求全”。只有一项功能是终端所必须具备的——它必须能显示光标。如果启动Emacs时看到一条消息告知终端功能不足，就表明或者是**TERM**环境变量设置不正确，或者是所用终端太“笨”；这第二种可能性如今已经是非常罕见了。

关于终端我们再多说几句。你可能会发现**termcap**或**terminfo**里有好几项看起来都适用于你的终端。对于这种情况（假设使用的是**termcap**），排在第一个的那一项最有可能是正确的，或者至少是一个合理的默认值；**termcap**文件通常就是按这个原则设计出来的。

有的终端在模仿某知名品牌（比如VT100）的同时可能还具备某些额外的功能。对于这种情况，如果指定的是它实际的终端类型，而不是让它“自贬身份”去仿真最低档的主流品牌，那么Emacs也许能运行得更有效率。为了在**termcap**或**terminfo**里找到一个能够利用那些额外功能的设置项而多花些功夫是非常值得的。

流控制问题的解决办法

流控制（flow-control）问题指的是一种与终端支持有关的设备冲突现象，不少终端（虽然如今已不多见）就是因这个问题的影响才不能与Emacs很好地配合工作的。这个问题是这样的：为了预防缓冲区溢出，有些操作系统或数据通信设备（终端、调制解调器、网络等等）把“**C-s**”和“**C-q**”字符用做终端输入数据的“停止”和“前进”信号。

如今，使用其他流控制手段（比如额外的硬件）的数据通信系统越来越多，但仍有一些还在使用“**C-s / C-q**”方法。用户和作为最终目标的计算机之间的通信环节越多（比如一个终端先连接到一个调制解调器、再连接到另外一个调制解调器、再连接到一个局域网，再连接到一台计算机，再远程登录到另外一台计算机……的情况），出现这种问题的概率也就越大。

一般说来，用户和计算机之间的硬件设备越多，“**C-s**”和“**C-q**”被用做流控制字符的可能性也就越大；而这将使Emacs无法在编辑命令里使用这两个字符。有个简单的方法可以查明这种情况是否会影响自己：启动Emacs并按下“**C-s**”组合键。如果能够在状态行上看见“`Isearch:`”字样，就说明一切正常；否则，看起来好像是什么事情也没有发生，可终端却可能已经被挂起来了，再按任何键，终端都没有反

应。实际情况是这样的：按下的“**C-s**”字符告诉某些硬件不要再继续接受输入数据。按“**C-q**”组合键退出这种局面，在这期间按下的那些键又都起作用了。

Emacs准备了一个简单的方法来解决这个问题。命令**enable-flow-control**的作用是让Emacs把“**C-s**”和“**C-q**”字符直接传递给底层的操作系统，操作系统将把它们看做是流控制命令并进行相应的处理；需要用“**C-**”（“control”键加反斜线键）和“**C-^**”（“control”键加“shift”键加“6”键）来分别代替原来的“**C-s**”和“**C-q**”组合键——不论可能会在什么地方用到它们。这样，对应于编辑命令**save-buffer**的按键绑定就将变为“**C-x C-**”。

还可以对Emacs做这样的设置：只要使用某种类型的终端，就自动激活流控制功能。这需要把下面这条语句添加到“*.emacs*”文件里：

```
(enable-flow-control-on  termtypes)
```

语句中的“*termtypes*”是一个或者多个字符串，字符串里包含着想为之激活流控制功能的终端类型的**TERM**名称。请看下面这个示例。如果偶尔会在一台DEC VT-320终端上使用Emacs——这种终端上的流控制问题必须由它自己去处理，就要使用下面这条语句：

```
(enable-flow-control-on "vt320")
```

这里介绍的流控制命令已经在Emacs老版本的基础上有了很大的改进，在老版本里解决流控制问题还得再多做很多事呢。

Emacs 变量

下面再来说说如何改变Emacs的行为——而不仅仅是它的用户界面。达到这一目的最简单的方法是对控制着Emacs各种行为的变量进行设置。我们在第二章已经见过**auto-save-interval**等几个例子。要想设置Emacs变量，就需要在“*.emacs*”文件里使用**setq**函数，比如下面这样：

```
(setq auto-save-interval 800)
```

auto-save-interval变量取的是一个整数值；但很多Emacs变量取的是表示“真”或“假”的布尔值。在Emacs LISP里，真值用“**t**”来表示，假值用“**nil**”来表示；但

在很多情况下，任何不是“**nil**”的东西都可以用来代表真值。Emacs 变量还可以取其他类型的值，它们的表达方法如下：

- 字符串值 —— 必须放在双引号里。我们在本章前面介绍按键绑定命令的参数时见过不少的字符串值。
- 字符值 —— 类似于字符串，但字符值必须有一个问号（?）做前缀，而且不能放在双引号里。比如，“**?x**”和“**?\C-c**”就分别是“**x**”和“**C-c**”的字符值。
- 符号值 —— 以单引号（'）打头，后面跟着一个符号名。比如像“'never”这样（参见附录三中的变量 **version-control**）。

附录三对常用的Emacs变量进行了归纳整理，分门别类地列出了它们的说明和默认值。Emacs 变量非常多 —— 附录三里只列出了其中的一部分。可以这么说，只要想对Emacs的某个方面进行定制，就肯定会有一个与之对应的控制变量（特别是当想改变的东西涉及到一个数字或者涉及到真、假条件时更是如此）。可以用将在第十六章介绍的**apropos**命令查出是否有与想定制的东西有关的变量，仔细看看它输出结果里的变量和它们的说明。

有些Emacs变量对不同的编辑缓冲区可以取不同的值（我们称这些值为局部值），与局部值相对的是默认值。如果没有给这些变量设定局部值，它们在编辑缓冲区里的取值就是其默认值。可以分别对这类变量的默认值和局部值做相应的设定。在设置 **left-margin** 或 **case-fold-search** 等变量的值的时候，实际设定的是它们的局部值。如果想设置这类变量的默认值，就必须使用**setq-default**命令而不是使用**setq**命令。如下所示：

```
(setq-default left-margin 4)
```

令人遗憾的是，没有什么通用的方法能够用来判断某个变量是只有一个全局性的取值，还是既有默认值又有局部值（除非阅读该编辑模式的LISP代码）。所以，我们认为最佳策略应该是先尽量使用**setq**命令，如果在实际工作中发现自己用**setq**命令设置给某个变量的值似乎没有起作用，那么再去使用 **setq-default** 命令。比如，假如已经把下面这条语句：

```
(setq case-fold-search nil)
```

添加到“*.emacs*”文件里，但发现Emacs的查找命令还是忽略字母的大小写（就好像这个变量的取值还是“**t**”一样），那么就应该再用**setq-default**命令设置一次。

Emacs的LISP程序包

Emacs包含大量的LISP代码。事实上，就像将在第十三章里看到的那样，Emacs的内置功能基本上都是用LISP语言写成的。Emacs还另外带有一些LISP程序包（或者叫程序库），可以随时用加载LISP程序包的方法来扩展Emacs的功能。为Emacs开发LISP程序包的工作一直在进行着，而且来源不仅限于自由软件基金会；有时候，系统管理员也会安装一些非自由软件基金会来源的程序包。

我们在附录四里用一个表格对比较常用的内置LISP程序包进行了汇总，对它们的使用方法也做了简单的介绍。如果想知道自己的系统上都有哪些程序包可用，请按下“**C-h p**”组合键（命令名是**finder-by-keyword**）。简单地说，Emacs内置的LISP程序包可以用来完成以下一些工作：

- 支持用C、LISP、Perl、FORTRAN及另外几种程序设计语言编写程序（参见第十二章）。
- 支持用**nroff**、**troff**、TEX、LATEX和HTML做文字排版处理（参见第九章）。
- 仿真其他编辑器（**vi**、**EDT**和Gosling Emacs等）。
- 提供到其他UNIX实用程序的接口，比如shell和mail（参见第五章和第六章），以及FTP、Telnet、Kermit和MH电子邮件系统等。
- 提供辅助编辑功能，比如拼写检查（参见第三章）、大纲编辑（参见第八章）、文本排序、编辑命令的历史记录和设置Emacs变量（参见附录三）等等。
- 提供各种游戏或者其他形式的娱乐等。

详细资料请参考附录四。

自动模式的定制

Emacs能够根据准备打开的文件名后缀自动进入相应的主编辑模式，附录四的表格

里就列出了一些具备这种功能的主编辑模式。表格最右一栏给出了许多Emacs在默认情况下设定的文件名后缀与主编辑模式的关联关系。这些关联关系由专用的Emacs 变量 **auto-mode-alist** 负责管理。这个变量是一个“(*regexp* . *mode*)”形式的数据组列表，其中“*regexp*”是一个正则表达式(参见第三章和第十三章)，“*mode*”是一个用来启动某个主编辑模式的函数的名字。Emacs在打开文件的时候，会先到这个列表里（从头开始）查找有没有与文件名匹配的正则表达式。如果找到一个，就会运行与该正则表达式关联的函数以启动相应的主编辑模式。注意，文件名的任何一个部分（不仅仅是它的后缀名）都可以用来与某个主编辑模式建立关联关系。

也可以把自己的关联关系添加到**auto-mode-alist**列表里；不过，如果不熟悉LISP，最好还是别这么做（参见第十三章）。假设正在用Ada语言写程序，而Ada编译器要求文件名后缀都是“*.a*”（有些Ada编译器要求使用“*.ada*”）。如果想让Emacs做到只要访问的是Ada文件，它就自动把它们放到Ada模式里，那么就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq auto-mode-alist (cons '("\\.a$" . ada-mode) auto-mode-alist))
```

请注意，在“cons”的后面必须有一个单引号（'），在“"\\.a$"”和“ada-mode”之间必须有一个英文句号（.）。记号“'(x . y)”是用来“把x和y配对组成一个数据组”的LISP语法。字符串“"\\.a$"”是个正则表达式，能够匹配“以 *.a* 结尾的任何东西”；“$”的意思是匹配字符串的结尾（注意不是匹配文本行的结尾——那是在正则表达式查找－替换操作中才需要匹配的东西）。整个LISP语句的意思是“把数据组（"\\.a$", 'ada-mode）插入到**auto-mode-alist**列表的开始位置处”。注意，因为Emacs是从头开始检索**auto-mode-alist**变量的，并且会在找到一个匹配之后停止前进，所以可以用上面这种“cons”结构覆盖现有的编辑模式关联关系（注6）。

再来看看另外一个例子。假设把某些电子邮件保存在文件名以“*msg-*”打头的文件里。如果想在文本模式里对这些文件进行编辑，就可以像下面这样做：

```
(setq auto-mode-alist (cons '("^msg-" . text-mode) auto-mode-alist))
```

注6： LISP程序员应该知道还有其他方法可以用来往**auto-mode-alist**列表里添加数据组，比如“**append**（追加）”。

注意，在这个例子里，需要匹配的是文件名的开头而不是文件名的结尾。正则表达式操作符（^）表示匹配字符串的开始，所以整个正则表达式的意思是“以 msg- 开头的任何东西”。

最后，如果准备打开的文件没有匹配上**auto-mode-alist**里的任何一个正则表达式，那么 Emacs 就会把它放到 **default-major-mode** 变量的值所指定的那个主编辑模式里。这个模式通常被设置为基本模式，这是一个不带任何特殊功能的基本编辑模式。不过，有许多人喜欢把他们的默认编辑模式设置为文本模式，下面这条语句就是用来完成这件事的：

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

虽然这一章介绍了不少对Emacs进行定制的好方法，但这些还只是一些皮毛。如果各位想做更进一步的研究，请进入第十三章学习 Emacs 中的 LISP 程序设计；如果想让 Emacs 完完全全地按照自己的意愿进行工作，打开这扇大门的钥匙就是 LISP 程序设计。

本章内容：

- 语言编辑模式
- C和C++模式
- LISP模式
- FORTRAN模式
- 对程序进行编译

第十二章

程序员的Emacs

许多程序员都知道，程序设计工作通常可以分解为思考－编写－调试等三个环节组成的周期性循环。如果你曾经用UNIX（或者其他操作系统）进行过程序设计，那么你可能已经习惯于在程序开发周期的各个阶段，使用一组彼此互不衔接的工具；比如，一个用来写代码的文本编辑器、一个用来编译程序的编译器、再用操作系统本身来运行程序。如果有一种能够消除开发周期各阶段以及各阶段所用工具之间界线的开发环境，那么它无疑会提高效率。

Emacs在程序的编写、运行和调试方面为很多种编程语言提供了强有力的支持，而且它把这种支持集成在一个平滑的框架里。既然在开发程序的时候不用退出Emacs，我们就可以把注意力集中在真正的程序设计任务（即程序开发周期的“思考”部分）上，因为我们的时间将不再会浪费在挑选各种程序开发工具方面。

如果打算用Emacs来编写代码，那么可以利用Emacs针对某种程序设计语言的编辑模式来帮助更有效率地完成任务。针对程序设计语言的编辑模式能够把Emacs从根本上改造成一个“语法指导的”或“语言敏感的”（即能够识别和理解某种程序设计语言的语法的）编辑器，从而帮助用户按自己的风格写出格式整齐、方便阅读的代码。Emacs为很多种程序设计语言准备有相应的编辑模式。

Emacs还支持程序的运行和调试工作。第五章介绍的shell模式允许把Emacs用做一种窗口系统，它使用户能够在编写代码的同时对它进行运行和调试。Emacs还为很多种编译器和UNIX操作系统的**make**命令提供了一个功能强大的接口：Emacs可

以对编译器给出的出错信息进行分析，并且能够根据出错语句的行号迅速到达文件中的出错地点。

在这一章里，我们将对各种程序设计语言编辑模式的共同特点做概括性的探讨，然后有针对性地重点介绍三种程序设计语言 ——C 和 C++、LISP 以及 FORTRAN ——的编辑模式。在讨论完 C/C++ 模式之后，我们还要介绍一下 **etags** 工具；对工作在文件数量众多的大型软件开发项目上的C/C++ 程序员来说，这种工具能够带来极大的帮助。最后，我们将对 Emacs 与编译器 /**make** 命令之间的接口进行探讨。

语言编辑模式

我们已经见过一些Emacs的编辑模式了，比如文本模式（参见第二章和第八章）和 shell 模式（参见第五章）等。**list-buffer**（参见第四章）和 **dired**（参见第五章）等特殊功能其实也属于编辑模式的范畴。各种编辑模式都是由两大部分组成的：一个是用来实现这个编辑模式的 Emacs LISP 程序包，另一个是用来启动这个编辑模式的函数。

这本书主要以 Emacs Version 19 为讨论对象，这一版 Emacs 目前能够为以下一些程序设计语言提供相应的编辑模式：Ada、汇编语言、C、C++、Common LISP、FORTRAN、ICON、LISP、MIM、Modula-2、Objective-C、Pascal、Perl、PROLOG、Scheme、SGML 和 Simula，今后的版本还将会有所增加。有许多 —— 但不是全部 —— 语言编辑模式都在Emacs里“设好了挂钩”，如果你打开的文件带有适当的文件名后缀，Emacs就会让你自动进入正确的编辑模式里。可以从附录四中的Emacs LISP程序包速查表里查出，Emacs是否为所使用的程序设计语言准备了相应的编辑模式。如果表格最右一栏里列出了一个或更多的文件名后缀，就表示Emacs在遇到带有这些后缀名的文件时，都会自动启动相应的编辑模式。

如果文件名后缀没有列在那个表格里（或者，如果编译器允许使用没有在表格里列出的其他文件名后缀），那么就得对 Emacs 进行一些设置才能让它在访问源代码文件时自动启动相应的编辑模式。需要做两件事：首先，在程序包速查表的最右栏里找到使用的程序设计语言，然后查出用来启动对应编辑模式的函数的名字（比如 **ada-mode** 和 **modula-2-mode** 等）。其次，给“*.emacs*”文件增加几行代码，以便 Emacs能够在用户访问未在附录四的速查表中，列出的文件时自动加载相应的程序包。

需要为此写两行代码。第一行代码使用**autoload**函数，其作用是告诉Emacs在遇到没有见过的命令时应该到什么地方去查它。这就在函数和实现函数的程序包之间建立起一个关联关系；这样，当这个函数第一次被调用时，Emacs就会自动加载相应的程序包来载入该函数的代码。具体到我们的例子，需要在一个用来启动某语言编辑模式的函数和把实现该编辑模式的程序包之间建立起关联关系。**autoload**函数的格式如下所示：

```
(autoload 'function "filename")
```

注意“function”前面的单引号和“filename”两端的双引号；LISP语言的详细语法请参考第十三章内容。Ada程序员可把下面这条语句添加到“*.emacs*”文件里：

```
(autoload 'ada-mode 'ada")
```

这条语句的作用是让Emacs在第一次调用**ada-mode**函数的时候加载*ada*程序包。

再看第二行代码，它将在源代码文件的后缀名和用来启动对应编辑模式的函数之间建立起一个关联关系；这样，当访问一个带有给定后缀名的文件时，那个用来启动对应编辑模式的函数将被自动调用执行。这需要用到第十一章里介绍的Emacs全局变量**auto-mode-alist**。Emacs将根据这个列表中的关联关系，把被访问的文件放到与它的后缀名相对应的编辑模式里。下面是一个为**ada-mode**创建这种关联关系的例子，Emacs将把后缀名是“*.a*”的文件都放到这个编辑模式里（更进一步的解释请参考第十一章内容）：

```
(setq auto-mode-alist (cons '(" .a$" . ada-mode) auto-mode-alist))
```

这行LISP代码将在用户打开一个后缀名表明它是用某种编程语言写的源代码文件时，引发下述一连串事件。假设准备访问的文件是“*pgm.a*”：首先，Emacs读入这个文件。接着，它发现**auto-mode-alist**里有一个与文件名后缀“*.a*”对应的设置项，于是试图调用与之关联的**ada-mode**函数。它注意到函数**ada-mode**并不存在，但找到了一个把该函数与*ada*程序包关联起来的**autoload**设置项。于是，Emacs将加载这个程序包，从中找到**ada-mode**命令，然后运行这个命令。经过这样一番周折之后，编辑缓冲区就被设置为**ada-mode**编辑模式。如果把刚才介绍的那两行代码放到了“*.emacs*”文件里，Emacs就能在今后所有的编辑工作中识别出所使用的程序设计语言。

语法

不同的语言编辑模式在具体功能方面当然是有区别的，但它们都支持相同的基本概念。其中最重要的一点是，不同的语言编辑模式都能识别和理解与之对应的程序设计语言的语法，即能够识别和理解该语言的字符、词汇和某些语法现象。在对以前各章的讨论中，我们已经见识过Emacs在人类语言语法现象方面的“学问”了。在对普通文字材料做编辑处理的时候，Emacs能够知道单词、句子和段落都是怎么一回事：它允许以这些语言学元素为单位，来移动光标或者删除文本。它还知道某些标点符号的正确用法，例如括号：只要一输入右括号，它就会“闪现”与之配对的左括号——把光标移到那里停一秒钟再返回来（注1）。这为检查括号的配对情况是否正确提供了方便。

Emacs对程序设计语言语法的了解，与它在人类语言语法方面的情况差不多。总的来说，它能够识别和理解以下一些基本的语言元素：

- 单词——对多数程序设计语言来说就是标识符和数字。
- 标点符号——包括操作符（如“+”、“-”、“<”、“>”等）和语句分隔符（如分号“;”）。
- 字符串——由字符组成的序列，通常放在一组配对的分隔符（比如引号）里。
- 括号——包括方括号（“[”和“]”）、花括号（“{”和“}“）和普通的圆括号。
- 空白——比如空格和制表符，用来分隔其他语言元素，通常不具备语言学方面的含义。
- 注释——被括在注释分隔符之间的字符序列，不同的语言有不同的注释分隔符(比如C语言中的“/*”和“*/”,LISP语言中的分号“;”和换行符**LINEFEED**)。

Emacs把语法信息保存在各种“语法表（syntax tables）”里。类似于把按键绑定信息保存在键位映射图（keymap，参见第十一章）里的情况，Emacs有一个供全体编

注1：事实上，Emacs向回寻找配对左括号的搜索距离是有一个最大限度（以字符为计算单位）的；它是变量**blink-matching-paren-distance**的值，默认设置为12 000个字符。左括号的“闪现”持续时间也是可设置的，它是变量**blink-matching-delay**的值（以秒为计算单位），默认设置为1。

辑缓冲区使用的全局语法表，每个编辑缓冲区再有一个局部语法表，局部语法表会随编辑缓冲区所处的编辑模式而变化。此外，程序设计语言编辑模式还能识别和理解依赖于具体语言的更复杂的语言学概念，比如语句、语句块、函数、子例程和LISP中的S-表达式等等。

格式

除语言学方面的知识以外，Emacs的语言编辑模式还有许多能帮助写出格式整齐的代码的功能。这些功能涉及到缩进、注释等与程序设计风格有关的各个方面，从而保证程序有良好的一致性和可读性，让注释排列的整齐划一等等。更重要的是它们能够减轻工作负担，用户不用再去操心语句的缩进量是否正确，甚至不用记住当前的缩进量是多少。这些功能最好的地方是它们基本上都可以由用户来自行定制。

我们在文本模式里见过这样的情况：如果在行尾输入的是换行符**LINEFEED**（或者“**C-j**”组合键）而不是回车键，Emacs就会自动对下一行进行正确的缩进。这种缩进由变量**left-margin**控制，这个变量的取值就是以字符计算的文本缩进量。程序设计语言的编辑模式也具备相同的功能，只是这种处理将更加灵活和复杂。

类似于文本模式中的情况，语言编辑模式里的**LINEFEED**也会对下一行进行正确的缩进。也可以在语句输入完毕之后再对它进行缩进，即把光标放在这一行的任意位置再按下**TAB**键即可。

有些语言编辑模式还为代码语句的结束符（比如分号或者右花括号）特别准备了一项功能：输入这些字符的时候，Emacs将自动对当前行进行缩进。Emacs文档把这种行为叫做“自动缩进”（英文原文是“electric”）。大多数语言编辑模式还提供有成套的变量组合来控制缩进样式（这些变量也都能够进行定制）。

表12-1列出了其他一些与缩进有关的命令，它们的具体工作情况是由具体程序设计语言的有关规则来决定的。

表12-1：基本缩进命令速查表

键盘操作	命令名称	动作
ECS C-	**indent-region**	对光标和文本块标记之间的每一行进行缩进

表 12-1: 基本缩进命令速查表（续）

键盘操作	命令名称	动作
ECS m	**back-to-indentation**	把光标移到当前行的第一个非空白字符上
ECS ^	**delete-indentation**	把当前行合并到上一行去

下面是一个“**ECS C-**”命令的用法示例。例子中的程序代码是用 C 语言写的，我们以后的例子也会用到这段程序。本节各示例所阐明的概念也适用于大多数其他程序设计语言中；LISP和FORTRAN语言中与此相当的操作将在讨论其语言编辑模式的章节里进行介绍。

假设已经输入下面这段 C 语言代码:

```
int times (x, y)
int x, y;
{
int i;
int result = 0;

for (i = 0; i < x; i++)
{
result += y;
}
}
```

现在，把文本块标记设置在这段代码的开始，把光标放在它的末尾，然后按下“**ECS C-**”组合键。Emacs 将把它排成如下所示的样子:

```
int times (x, y)
      int x, y;
{
  int i;
  int result = 0;

  for (i = 0; i < x; i++)
    {
      result += y;
    }
}
```

也可以方便地用“**ECS C-**”组合键把整个文件按你选定的缩进样式进行缩进：先按下“**C-x h**”组合键（命令名是 **mark-whole-buffer**），再按下“**ECS C-**”组合键即可。

“**ECS m**”命令可以迅速把光标移动到某行代码真正开始的地方。请看下面的例子。假设光标现在的位置是:

```
int res lt = 0;
```

如果你按下“**ECS m**”组合键，则光标将移动到“int”的第一个字母上，如下所示:

```
nt result = 0;
```

我们来看一个使用“**ESC ^**”命令的例子。假设想让for语句的左花括号与关键字for出现在同一行上。把光标放在这个左花括号所在文本行上的任意位置，按下“**ESC ^**”组合键，则该语句将变成如下所示的样子:

```
for (i = 0; i < x; i++) {
      result += y;
    }
```

语言编辑模式通常还特别提供一些与该语言某些特定功能有关的缩进命令。

因为程序设计语言全都有注释语法，所以Emacs准备了几个通用的注释处理命令;它们在不同的语言编辑模式里，会根据该语言的具体规定产生不同的效果。适用于各种语言编辑模式的通用注释命令是“**ESC ;**”(命令名是**indent-for-comment**)(注2)。按下“**ESC ;**”组合键的时候，Emacs将按变量**comment-column**的值进行缩进;如果这一行上的文本已经超出了这一列的位置，它将从最后一个字符的位置再往右移一个空格。然后，Emacs插入一个注释分隔符(或者一组配对的注释分隔符，比如C语言中的“/*”和“*/”)，再把光标放在左分隔符的后面。

请看下面的例子。如果想给上面那个函数中for循环的循环体语句加上一条注释，请把光标放的这个语句所在行的任意位置上，再按下“**ESC ;**”组合键，结果将是:

```
result += y;                /*  */
```

这就能够顺势把注释输入到两个分隔符之间。如果想对一个更长的语句行做同样的事，比如下面这条语句:

```
q_i = term_arr[i].num_docs / total_docs;
```

注2: LISP程序员应该比较容易记住这个命令，因为LISP语言中的注释就是以分号(;)开始的。

则结果将是:

```
q_i = term_arr[i].num_docs / total_docs; /* ▌*/
```

comment-column 变量是可以定制的，只要把相应的代码添加到“*.emacs*”文件里即可（如果想对该变量做永久性设置，这是最有效的方法）。但如果只是想在当前编辑缓冲区里对**comment-column**变量做临时性的设置，那么只需把光标移动到想让注释开始的位置再按下“**C-x ;**”组合键（命令名是 **set-comment-column**）即可。注意，这个命令只影响当前编辑缓冲区里的**comment-column** 变量的值；而该变量在其他编辑缓冲区（即使是同一编辑模式下的其他编辑缓冲区）里的值却不会改变。

如果注释的内容用一个文本行容纳不下，就需要在下一行继续写完注释；“**ESC j**”组合键（命令名是 **indent-new-comment-line**）就是用来做这件事的。这个命令将在下一行开始一条新的注释（有些语言编辑模式允许把它定制为继续刚才的注释继续往下写，而不是另外开始一个新的注释行）。假设已经给“times”函数里 for 循环体中的语句输入了一些注释（但还没有输入完）；光标目前停在文本的末尾处，如下所示:

```
result += y;                /* add the multiplicand▌*/
```

现在，想在第二行上继续输入注释。如果按下“**ESC j**”组合键，就将得到如下所示的效果:

```
result += y;                /* add the multiplicand */
                            /*  ▌ /
```

接下来，可以立刻开始输入第二行注释。也可以用“**ESC j**”命令把现有的注释文本拆分为两行。假设光标是在如下所示的位置:

```
result += y;                /* add the ▌ultiplicand */
```

如果此时按下“**ESC j**”组合键，结果将是:

```
result += y;                /* add the */
                            /* ▌ultiplicand */
```

如果想把一小段代码整个地设置为注释，可以使用 **comment-region** 命令（除在少数几种语言编辑模式里以外，这个命令没有绑定到任何组合键上）。假设有一段下面这样的代码:

```
this = is (a);
section (of, source, code);
that += (takes[up]->a * number);
of (lines);
```

像平常那样把这段代码定义为一个文本块，然后输入“**ESC x comment-region**”命令，结果将是：

```
/*      this = is (a); */
/*      section (of, source, code); */
/*      that += (takes[up]->a * number); */
/*      of (lines); */
```

“**ESC x kill-comment RETURN**”命令（它也没有默认的按键绑定）可以把当前行上的注释都清除掉，很适合用来清除程序中自成一行的“单行”注释。光标不必非得处于注释文本之中。每一种语言编辑模式都有一些与该语言注释密切相关的功能，其中往往包括有用来对缩进样式进行定制的各种变量。

在介绍完适用于各种语言编辑模式的通用概念之后，我们将对C、C++、LISP和RORTRAN等程序设计语言的编辑模式做专门的探讨。随后的三个小节将对这几种语言编辑模式依次进行讨论；在关于C和C++语言编辑模式的小节末尾，我们还要研究一下**etags**工具，它对编写大型、多文件程序的C / C++程序员有很大的帮助。

如果你只对一两种程序设计语言感兴趣，就没有必要阅读以下所有的内容。如果你是用其他程序设计语言来写程序的，而Emacs又恰好提供有相应的语言编辑模式，那你可以从下面这三个小节里挑选一个阅读，体会一下语言编辑模式的“味道”；虽然具体的语言编辑模式会有所不同，但都具有相同的基本概念，起码能让你开个好头。

C 和 C++ 模式

如果打开后缀名是“*.c*”、“*.h*”、“*.y*”（用在**yacc**语法方面）或“*.lex*”（lex定义文件）的文件，Emacs就会自动进入C语言编辑模式，简称C模式。如果打开后缀名是“*.C*”、“*.H*”、“*.cc*”、“*.hh*”、“*.cpp*”、“*.cxx*”、“*.hxx*”、“*.c++*”或“*.h++*”的文件，Emacs就会自动进入C++语言编辑模式，简称C++模式。还可以利用“**ESC x c-mode RETURN**”命令把任意文件手动地放到C模式里。类似地，“**c++mode**”命令将把编辑缓冲区放到C++模式里。

C模式和C++模式是用同一个Emacs LISP程序包实现的，它的名字是“cc-mode”；除了这两种语言编辑模式，该程序包还提供一个与用在NextStep环境里的Objective-C语言相对应的编辑模式。C模式能够识别和理解ANSI C语法和“古老的”Kernighan-Ritchie C语法。虽然我们的介绍以C模式为主，但它们全都适用于C++模式。C++模式只比C模式额外多出很少几项功能。我们将在这一小节的末尾对它们进行探讨。

顺便说一句，Emacs的Perl语言编辑模式（这种程序设计语言由于广泛应用于WWW服务器的程序设计中而正变得越来越流行）是从C模式的一个老版本演化而来的。如果你使用的程序设计语言是Perl，就会发现C模式里的光标移动命令、缩进命令和排版命令等都能原封不动地应用到Perl模式里，惟一需要变化的就是得把命令名中的“c-”前缀改为“**perl-**”。Emacs会在遇到后缀名为“*.pl*”的文件时自动启动Perl模式。

在C模式里，Emacs能够识别和理解本章前面提到的各种语言学元素。语句中的分号（;）、冒号（:）、逗号（,）、花括号（{和}）以及井字号（#，用在C语言的预处理器命令中）等字符都是自动缩进的。也就是说，输入这些字符的时候，Emacs将自动对当前行进行缩进。

移动命令

除了以单词、句子等语言元素为单位进行移动的Emacs标准命令（这些命令在C模式里的用武之地，可以说只剩下大段的注释了）之外，C模式还另外准备了一些能够以语句、函数（注3）以及预处理器条件为单位进行移动的高级命令。我们把C模式这些高级的移动命令汇总在表12-2里。

表12-2：C模式高级移动命令速查表

键盘操作	命令名称	动作
ESC a	**c-beginning-of-statement**	移动到当前语句的开头
ESC e	**c-end-of-statement**	移动到当前语句的末尾

注3：与C语言函数有关的Emacs编辑命令在它们的命令名里都有“**defun**”字样，这是从LISP模式借用过来的用法。在LISP语言里，一个**defun**就是一个函数定义。

表 12-2: C 模式高级移动命令速查表（续）

键盘操作	命令名称	动作
ESC q	**c-fill-paragraph**	如果光标在注释文本中间，则进行段落重排，保留缩进和前导字符
ESC C-a	**beginning-of-defun**	移动到光标所在函数的开头
ESC C-e	**end-of-defun**	移动到光标所在函数的末尾
ESC C-h	**c-mark-function**	把光标放到函数的开头，把文本块标记放到函数的末尾 —— 即把函数整个选取为一个文本块
C-c C-q	**c-indent-defun**	按缩进样式对整个函数进行缩进
C-c C-u	**c-up-conditional**	移动到当前预处理器条件的开始位置
C-c C-p	**c-backward-conditional**	移动到上一个预处理器条件
C-c C-n	**c-forward-conditional**	移动到下一个预处理器条件

请注意，以语句为单位的移动命令其按键绑定与 Emacs 的 **backward-sentence** 和 **forward-sentence** 命令的完全相同。事实上，在 C 语言的注释里，它们的使用效果完全相同。

类似地，“**ESC q**”正是原来的 **fill-paragraph** 命令；只不过 C 模式把它的功能扩大为能够保留语句开头的缩进和前导字符而已。举个例子，如果光标位于下面这段注释里的某个任意位置：

```
/* This is
 * a
 * comment paragraph with wildly differing right
 *  margins.
 * It goes on     for a while,
 * then stops.
 */
```

那么，按下“**ESC q**”组合键时，它将变成下面这样：

```
/* This is a comment paragraph with wildly differing right margins.
 * It goes on for a while, then stops. */
```

如果不得不通读别人的大段代码以对之进行调试或者修改，就会发现以预处理器条件为单位的移动命令是一种上帝的恩赐。当面对的代码是准备运行在多种系统

(Emacs就是一个这样的软件)上的时候，情况就更是如此；对于这种情况，你最需要答案的问题是："到底有哪些代码是真正会被编译的？"

有了"**C-c C-u**"命令，就能立即找出某行代码具体是由哪一个预处理器条件控制的。请看下面的例子：

```
#define LUCYX
#define BADEXIT -1

#ifdef LUCYX
    ...
    *ptyv = open ("/dev/ptc", O_RDWR | O_NDELAY, 0);
    if (fd < 0)
        return BADEXIT;
    ...
#else
    ...
    fprintf (stderr, "You can't do that on this system!");
    ...
#endif
```

假设代码里的省略号（...）都代表上百行代码，而现在需要找出文件*/dev/ptc*在什么条件下才会被打开。于是，把光标移到这一行上并按下"**C-c C-u**"组合键，光标将移动到"**#ifdef LUCYX**"语句处：表示它只有在LUCYX系统上时才会被编译。如果想跳过不会被编译的代码，而直接到达这个编译条件的结尾，请按下"**C-c C-n**"组合键。我们在本小节的后面还会再介绍一个与C语言预处理器有关的命令。

C语言中的语句分隔符和语句块分隔符被绑定为一组不仅能正确插入字符、还能正确缩进有关语句的命令上。这些字符是左花括号"{"、右花括号"}"、分号"；"和冒号"："（用来分隔**switch**语句的标号）。比如，如果准备结束一个语句块或者一个函数体，就可以按下**LINEFEED**（或**RETURN**）和"}"；Emacs将把它和与它配对的"{"对齐，也就不用再去查看前面的代码以确定那个"{"到底是在哪一列上了。

"}"是一种括号字符，所以Emacs会在输入"}"的时候"闪现"与它配对的"{"。如果配对的"{"不在窗口所显示的文本范围内，Emacs就将把包含"{"的那条语句显示在辅助输入区里；如果那一行除"{"外只有空白（即空格或制表符），Emacs会先显示一个"^J"（即"**C-j**"或**LINEFEED**字符），然后把下一行显示出来，这是为了让用户更好地把握"{"字符处的上下文。

我们再去看看前面给出的“times”函数的例子。假设准备在这个函数的结尾输入一个“}”字符，而这个函数体开始位置上的“{”字符已经不在屏幕上了。因为在左花括号“{”所在的那一行上没有其他的代码，所以输入“}”后会在辅助输入区里看到如下所示的内容：

```
Matches  { ^J  int i;
```

对代码的缩进样式进行定制

C 语言（或者其他程序设计语言）的编程风格（coding style）是很个性化的。C 程序员是通过各种书籍或者别人写的各种代码片段来学习这种语言的，但他们迟早会形成自己的风格，而这些个性化的风格不见得会与他们在学习时接触到的一模一样。

C 模式的缩进行为恰恰是这种语言学习手段的体现，它还提供很多用来对其缩进行为进行定制的功能。在最简单的层次上，可以挑选一种现成的缩进样式来使用；然后，如果还不满意，就可以对挑选出来的样式进行定制，甚至可以从点滴开始创建自己的样式。后一种做法需要对 Emacs LISP 程序设计知识（参见第十三章）有比较深的掌握，还需要有一点勇敢精神。

“**ESC x c-set-style**”命令的作用就是让从现成的缩进样式里挑出一个来使用。这个命令会提示用户输入自己想要的缩进样式的名称，此时最简单的方法是按下**TAB**键——Emacs 的自动补足命令（参见第六章），它将打开一个“*Completions*”编辑缓冲区，那里面将列出所有的可选项。如果想选择某种缩进样式，输入它的名称再按下回车键即可。

Emacs 已经为我们提前预备了一些缩进样式，它在默认情况下加载的缩进样式见表 12-3 所示。

表 12-3：C 模式缩进样式速查表

缩进样式	说明
CC-MODE	默认的编程风格；其他样式都是从它推导出来的
GNU	Emacs 自身和其他 GNU 程序所使用的 C 语言书写风格
K&R	Kernighan 和 Ritchie 合著的《The C Programming Language》一书中使用的编程风格，这本书是 C 语言的开山之作

表 12-3: C 模式缩进样式速查表（续）

缩进样式	说明
BSD	BSD 系列的 UNIX 版本使用的编程风格
Stroustrup	Bjarne Stroustrup 所著的《The C++ Programming Language》一书中使用的编程风格，这本书是 C++ 语言的标准参考书
Whitesmith	Whitesmith 公司在他们的 C 和 C++ 编译器软件的文档里使用的编程风格
Ellemtel	瑞典 Ellemtel 电讯系统实验室的 C++ 文档里使用的编程风格

为了让大家对这些编程风格有一些了解，我们用本章前面给出的 C 函数为例做一下演示:

```
int times (x, y)
int x, y;
{
int i;
int result = 0;

for (i = 0; i < x; i++)
{
result += y;
}
}
```

如果把这段代码选取为一个文本块，并按下“**ESC C-**”组合键（命令名是 **indent-region**），那么 Emacs 将按其默认的缩进样式，把这段代码重新排版为下面这个样子:

```
int times (x, y)
int x, y;
{
    int i;
    int result = 0;

    for (i = 0; i < x; i++)
        {
            result += y;
        }
}
```

用“**ESC x c-set-style**”命令选择使用“**K&R**”风格，然后对段落进行重排，代码将排列为下面这个样子:

```
int times (x, y)
int x, y;
{
     int i;
     int result = 0;

     for (i = 0; i < x; i++)
     {
          result += y;
     }
}
```

如果想切换到GNU风格的缩进样式，请在选中风格“**GNU**”后对代码块进行重排；结果将是：

```
int times (x, y)
     int x, y;
{
  int i;
  int result = 0;

  for (i = 0; i < x; i++)
    {
      result += y;
    }
}
```

选好编程风格之后，把下面这条语句添加到“*.emacs*”文件里就可以永久地设置它：

```
(add-hook 'c-mode-hook
      '(lambda ()
         (c-set-style " stylename ")))
```

这条语句到底做了些什么还得等学到下一章时才能讲明白。就目前而言，大家只要别忘了在第2行的“(lambda”的前面加上一个单引号就可以了。

每种编程风格都有它自己的特点，而让Emacs把这些特点都实现出来，可并不是件轻而易举的事情。虽然Emacs的早期版本定义了几个用来控制缩进层次的变量，但数量并不是很多；它们不仅不容易使用，而且说老实话，也很难百分之百地从细节上区分出不同的编程风格。与此形成鲜明对比的是，Emacs最新版本里的C模式使用了一个规模相当巨大的变量集。这个变量集实在是太大了，除了那些爱钻牛角尖的Emacs LISP高手之外，一般人根本对付不了。

换句话说，如今的C模式是靠把成组的变量，和它们的值分门别类地归纳为缩进样式而实现出各种编程风格的。Emacs用了一个非常庞大的变量（它的名字是**c-style-alist**）来容纳全体缩进样式和它们的关联信息。各位可以通过修改现有缩进样式里的变量值，或者自行添加缩进样式这两种方法来对这个大家伙进行定制。具体细节请查阅自己系统的Emacs LISP目录里的*cc-mode.el*文件（详见第十三章）。

C和C++模式的附加功能

C模式还有很多非常有价值的功能，有的用途广泛，有的仅适用于极其罕见的情况。自动开始新行（auto-newline）和“饥饿的删除键”（hungry-delete-key）可以说是这些功能中最有意思的两个了，它们是两种用来给按键操作增加自动缩进功能的办法（注4）。

启用了自动开始新行功能之后，只要一输入分号“;”、花括号“{”和“}”或者（在某些特定场合下）逗号“,”及冒号“:”，Emacs就会自动加上一个换行符并对新行做正确的缩进。这项功能可以节省一些时间，而且能够帮助保持代码书写风格的一致性。

自动开始新行功能在默认的情况下是处于禁用状态的；可以用“**C-c C-a**”组合键（命令名是**c-toggle-auto-state**）启用它。（重复这个命令将再次禁用这项功能）。状态行上的“(C)”标志改变为“(C/a)”。我们以*times()*函数为例来看看它究竟是怎样工作的。先输入头两行代码，当输入了第2行上的“y”时，我们看到的是：

```
int times  (x, y)
int x, y
```

现在，按下分号键；Emacs将插入一个换行符并把光标移到下一行，如下所示：

```
int times (x, y)
int x, y
▮
```

输入左花括号，同样的事情将再次发生，如下所示：

注4：　它们可以用来仿真早期Gosling Emacs中的**electric-c-mode**模式。

```
int times (x, y)
int x, y;
{
    ▮
```

当然，Emacs在输入“{”之后的缩进量要取决于正在使用的缩进样式。

另一种可选用的自动缩进功能即“饥饿的删除键”，它在默认的情况下也是处于禁用状态的。可以用“**C-c C-d**”组合键（命令名是**c-toggle-hungry-state**）启用之。状态行上的“(C)”标志改变为“(C/h)”；或者，如果已经启用了自动开始新行功能，状态行上的“(C/a)”标志将改变为“(C/ah)”。

“饥饿的删除键”功能将使**DEL**键具备删除光标左面全部空白的能力。我们还用刚才的例子接着往下说。假设刚输入那个左花括号。现在，如果按下**DEL**键，则Emacs将一直删到这个左花括号处，如下所示：

```
int times (x, y)
int x, y;
{▮
```

自动开始新行和“饥饿的删除键”这两种功能的启用/禁用状态可以同时用“**C-c C-t**”组合键（命令名是**c-toggle-auto-hungry-state**）进行切换。

如果让这两项功能中的某一种在启动Emacs的时候自动处于启用状态，就需要把下面这样的语句添加到“*.emacs*”文件里：

```
(add-hook 'c-mode-hook
          '(lambda ()
             (c-toggle-auto-state)))
```

如果想用另外的C模式定制来合并这个定制（如前例中的缩进样式），就需要将Emacs LISP代码行合并如下：

```
(add-hook 'c-mode-hook
          '(lambda ()
             (c-set-style " stylename ")
             (c-toggle-auto-state)))
```

这条关联语句的含义将在第十三章里的“对现有编辑模式进行定制”一节里进行探讨。

C模式还提供一些与程序注释有关的支持功能。在本章前面的内容里，我们已经对这方面的支持功能做过概括性的讨论，但并没有对某种编辑模式在这方面的具体功能做细致的说明。大家可以对“**ESC j**”（命令名是**indent-new-comment-line**）的执行情况进行定制，从而使Emacs在下一行继续同一条注释，而不是另外创建一组新的注释分隔符。这项功能是由变量**comment-multi-line**控制的：如果它设置为“**nil**”（这是它的默认值），Emacs就将在下一行创建一条新的注释，就像在本章前面的示例中所给出的那样，如下所示：

```
result += y;                    /* add the multiplicand */
                                /*  */
```

在输入单词“multiplicand”之后，立刻按下“**ESC j**”组合键就会得到这样的结果；现在的光标位置表示可以立刻开始输入第二个注释行上的文本。如果把变量**comment-multi-line**设置为“**t**”（或者任意一个不是“**nil**”的值），将得到如下所示的结果：

```
result += y;                    /* add the multiplicand
                                   */
```

我们将要介绍的最后一个功能是“**C-c C-e**”组合键（命令名是**c-macro-expand**）。和以预处理器条件为单位的移动命令（比如绑定在“**C-c C-u**”组合键上的**c-up-conditional**命令）一样，**c-macro-expand**命令也能帮助回答“到底有哪些代码是真正会被编译的?”这个通常很难回答的问题。

在使用**c-macro-expand**命令之前，必须先定义一个文本块。接着，按下“**C-c C-e**”组合键时，这个文本块中的代码将被送往实际的C语言预处理器去进行处理，其结果将被放到一个名为“*Macroexpansion*”的窗口里。

我们用本章前面那个带有C预处理器指令的代码为例，来看看这一过程的具体工作情况，如下所示：

```
#define LUCYX
#define BADEXIT -1

#ifdef LUCYX
    *ptyv = open ("/dev/ptc", O_RDWR | O_NDELAY, 0);
    if (fd < 0)
        return BADEXIT;
#else
```

```
    fprintf (stderr, "You can't do that on this system!");
#endif
```

如果把这段代码定义为一个文本块并按下“**C-c C-e**”组合键，那么就会出现下面这条消息：

```
Invoking /lib/cpp -C on region . . .
```

过一会儿，出现

```
done
```

接下来，出现一个“*Macroexpansion*”窗口，窗口里是这次处理的结果：

```
*ptyv = open ("/dev/ptc", O_RDWR | O_NDELAY, 0);
if (fd < 0)
     return -1;
```

如果想让**c-macro-expand**调用另外一个C预处理器命令而不是默认的“**/lib/cpp -C**”（选项“**-C**”表示“把注释保留在输出里”），就需要对变量**c-macro-preprocessor**做相应的设置。比如，如果想试用文件名是“*/usr/local/lib/cpp*”的预处理器，就需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq c-macro-preprocessor "/usr/local/lib/cpp -C")
```

我们强烈建议大家保留“**-C**”选项以免预处理器删掉代码中的注释。

C++ 模式的差异

我们在前面已经说过，C++ 模式与C模式使用的是同一个 Emacs LISP 程序包。在 C++ 模式里的时候，Emacs 将使用 C++ 语法，而不再是 C 语言（或 Objective-C 语言）的语法。我们刚才介绍的命令中有几个会因此而导致其行为发生变化。

C++ 模式和C模式并没有明显的差异。它们之间最重要的区别在于为了定制C++ 模式，而必须添加到“*.emacs*”文件里的Emacs LISP代码：必须把语句中的“**c-mode-hook**”字样替代为“**c++-mode-hook**”。比如说，如果想把C++ 模式的缩进样式从默认情况设置为 Stroustrup，就必须把下面这条语句添加到“*.emacs*”文件里：

```
(add-hook 'c++-mode-hook
      '(lambda ()
```

```
        (c-set-style "Stroustrup")
        (c-toggle-auto-state)))
```

请注意，可以用这种方法为C模式和C++模式分别设置其各自的关联关系；也就是说，如果你会同时使用这两种程序设计语言，就完全可以给它们分别设置不同的缩进样式。

C++模式还提供了一个附的命令“**C-c :**”（命令名是**c-scope-operator**）。这个命令的作用是插入C++语言中的域操作符——双冒号（::）。之所以需要有这样一个命令是因为单冒号（:）通常会绑定有自动缩进功能——一输入冒号就会对代码行进行缩进，可本意却并非如此。域操作符可能出现在C++代码中的任何位置，而单冒号却往往用来指示一个case语句的标号，因此它需要特殊的缩进。“**C-c :**”命令乍看起来好像有些多余，但它却是为了避免C++语言中的语法冲突所必须的折中办法。

最后，C模式和C++模式中都有**c-forward-into-nomenclature**和**c-backward-into-nomenclature**命令，它们在默认的情况下没有绑定到任何按键上。这两个命令与**forward-word**和**backward-word**命令分别有几分相似，但它们会把单词中间的大写字母看做是新单词的开始。比如说，它们会把“*ThisVariableName*”看做是3个单词，而标准的**forward-word**和**backward-word**命令会把它看做是一个单词。用“*ThisTypeOfVariableName*”形式的变量名是C++程序员比较习惯的编程风格，与此相对的是“*this_type_of_variable_name*”，这种形式的变量名是C语言代码里比较常见的。

C++程序员可能需要把**c-forward-into-nomenclature**和**c-backward-into-nomenclature**命令绑定到原来绑定为标准的单词移动命令的按键组合上。我们将在第十三章的“对现有编辑模式进行定制”一节介绍有关的具体做法。

我们已经把C模式和C++模式的主要功能介绍得差不多了，但这两种编辑模式还有很多我们没有讲到的其他功能。这些功能中有很多是一般人很少会用到的，只有那些专家级的Emacs LISP定制者才有可能需要它们。详细情况请参考Emacs LISP程序包**cc-mode.el**，它可是一个有将近5 000行代码的大家伙。

etags

Emacs 为 C 和 C++ 程序员准备的另外一个功能是 **etags** 工具程序（注 5）。**etags** 也能用来对付许多以其他程序设计语言写成的代码，其中包括 FORTRAN、Perl、Pascal、LaTeX、LISP 以及多种汇编语言。如果你的软件项目规模较大，涉及到的文件也比较多，就会发现 **etags** 确实会给你带来非常大的帮助。

从原理上讲，**etags** 不过是一个面向多文件的搜索工具，它既了解 C 语言和 Perl 语言中的函数定义结构，又具备通用性的查找功能。有了它，就能从整个目录的范围内把某个函数找出来而无需记住定义此函数的文件；查找和查询－替换操作也将延伸到多个文件。**etags** 使用“函数标签表（tag tables）”，其中包含着对应于目录中每个文件的函数名单，和各个函数的定义在文件的定义位置等信息。与 **etags** 有关的许多命令都需要在查找字符串里使用正则表达式（参见第三章）。

为了使用 **etags** 工具，需要先在当前目录里调用运行 **etags** 程序以创建函数标签表，这个程序的参数是打算收集其函数标签信息的文件。最常见的情况是用“*etags *.[ch]*”命令来调用 **etags** 程序，意思是为全体以“*.c*”和“*.h*”结尾的文件建立一个函数标签表。**etags** 程序的输出（也就是函数标签表）将保存在一个名为 *TAGS* 的文件里。在书写代码的时候，可以随时调用 **etags** 程序刷新函数标签表，让它能及时反映出新文件里的新函数定义。

建立了函数标签表之后，还需要让 Emacs 知道它；“**ESC x visit-tags-table RETURN**”命令就是用来完成这项工作的。它会提示输入函数标签表文件的名字 —— 默认为当前目录里的 *TAGS* 文件。完成了这个步骤之后，就可以使用各种各样的 Emacs 函数标签命令了。

最重要的函数标签命令是“**ESC .**”（命令名是 **find-tag**）。这个命令提示输入一个用来在函数标签表里查找某个函数的字符串，而这个字符串会出现在某个函数的名字里。给出查找字符串，Emacs 将找出与之匹配的函数名，并把包含该函数的文件打开在当前窗口里，并且会移动到该函数定义部分的第一行。“**ESC .**”有一个变体是“**C-x 4 .**”（命令名是 **find-tag-other-window**），它将另外使用一个窗口而不是替换当前窗口里的文本。

注 5: **ex** 和 **vi** 用户可以把 **etags** 看做是 **ctags** 的一个功能更为强大的“近亲”。

“**ESC .**”的特点是能把光标位置上的单词用做默认的查找字符串。比如说，如果光标在字符串“my_function”的某个字母上，“**ESC .**”就会把“my_function”当作默认的查找字符串。换句话说，如果你正在读一条调用某个函数的C语句，直接按下“**ESC .**”组合键就能查看到该函数的代码。

如果好几个函数同名，“**ESC .**”找出的将是按字母表顺序最先出现的那个函数。要想查找其他的同名函数，可以用“**ESC ,**”（命令名是**tags-loop-continue**）找到下一个（如果找不到下一个，Emacs将报告出错）。这项功能特别适用于子目录里有不止一个程序的情况；也就是说，如果有不止一个函数叫做“*main*”，“**ESC ,**”组合键就会有其他的用法。我们马上就会讲到。

函数标签表还可以用来查找函数定义以外的其他东西。命令“**ESC x tags-search RETURN**”将提示输入一个正则表达式；它将以列在函数标签表里的全体文件（比如所有的“*.c*”文件和“*.h*”文件）为对象查找有无与该正则表达式匹配的事物，不论它是否是一个函数。这项功能与我们将在本章结尾处介绍的**grep**工具程序有异曲同工之效。在调用执行“**tags-search**”命令之后，能够用按下“**ESC ,**”组合键的方法把其他的匹配情况都找出来。

etags还有一项能够仿真查询－替换操作的功能。命令“**ESC x tags-query-replace RETURN**”将以列在函数标签表里的全体文件为对象做一次正则表达式查询－替换操作。对正则表达式查询－替换操作的讨论请参考第三章。类似于正常的“**query-replace-regexp**”命令，如果给**tags-query-replace**加上一个前缀（比如像“**C-u ESC x tags-query-replace RETURN**”那样），Emacs就只替换与整个单词完全匹配的情况。如果想在不影响“fprintf”语句的前提下替换“printf”，这个功能就非常有用。如果用“**ESC**”键或“**C-g**”组合键退出了一次**tags-query-replace**操作，就还可以通过按下“**ESC ,**”的方法从刚才的退出位置重新继续往下进行。

命令“**ESC x tags-apropos**”对**etags**的查找功能做了进一步的补充。如果给它一个正则表达式参数，它就会一个关联窗口里打开一个“*Tags List*”编辑缓冲区，函数标签表里匹配该正则表达式的所有标签（包括文件名和函数名）都列在其中。比如说，如果想在由多个文件组成的C程序里找出输出例程的名字，就可以用“print”和“write”为参数来调用**tags-apropos**命令。

最后，可以用命令“**ESC x list-tags RETURN**”把函数标签表里的全体标签——即某个给定C文件的全体函数都列出来。在提示符处给出文件名，就会在“*Tags List*”编辑缓冲区里看到在该文件里定义的全部函数的名称及其返回值的类型(如果有)。如果把光标移动到这个名单里，就可以用“**ESC .**”组合键来查看函数的实际代码。这是因为“**ESC .**”能够把光标位置上的单词用做默认的函数名称，只需把光标移动到想查看的函数的名称上再按下“**ESC .**”和回车键就可以看到它的具体内容。

LISP 模式

总共有3种LISP编辑模式，我们按它们“**ESC x**”命令名称的先后顺序列在下面:

emacs-lisp-mode

用来编辑Emacs LISP代码，请参考第十三章中的讨论(文件名以“*.emacs*”或“*.el*”结尾)。

lisp-mode

用来编辑供其他LISP系统使用的LISP代码(文件名以“*.l*”或“*.lisp*”结尾)。

lisp-interaction-mode

用来编辑和运行Emacs LISP代码。

这3种编辑模式的基本功能都是一样的，它们之间的区别体现在它们各自对LISP代码运行的支持功能上。

这3种编辑模式能够识别和理解各种语言编辑模式所共有的基本语法元素。除此之外，它们还专门为S-表达式、列表和函数定义等高级语法概念准备了各种命令。S-表达式(syntactic expression，语法表达式)指的是任意一个没有语法错误的LISP表达式，它可以是一个原子项(数字、符号、变量，等等)或者一个放在括号里的列表。列表(list)是S-表达式的特殊形式，而函数定义(defun)又是列表的特殊形式。这些LISP语言所独有的语法概念需要一些专门的命令来处理。对一般的使用情况而言，只要掌握这些命令的一个子集就足能应付了。

表12-4列出了对S-表达式进行处理的有关命令。

表 12-4: S- 表达式处理命令速查表

键盘操作	命令名称	动作
ESC C-b	**backward-sexp**	移动到上一个 S- 表达式
ESC C-f	**forward-sexp**	移动到下一个 S- 表达式
ESC C-t	**transpose-sexps**	交换光标前后的两个 S- 表达式的位置
ESC C-@	**mark-sexp**	把文本块标记设置在当前 S- 表达式的末尾，把光标设置在当前 S- 表达式的开头
ESC C-k	**kill-sexp**	删除光标后面的那个 S- 表达式
（无）	**backward-kill-sexp**	删除光标前面的那个 S- 表达式

因为 S- 表达式可以是各种各样的东西，所以对 S- 表达式进行处理的命令到底会采取什么动作，要取决于调用命令时光标的具体位置。如果光标在一个左括号“(”或它前面的一个空白位置上，将被处理的 S- 表达式就是以这个左括号“(”开始的列表。如果光标是在字母或数字等其他字符（或该字母或数字前面的一个空白位置）上，将被处理的 S- 表达式就是一个原子项（符号、变量或常数等）。

请看下面这个例子。假设光标停在如下所示的位置上:

```
(mary bob (dave (pete)) ed)
```

按下“**ESC C-f**”组合键时，光标将移动到如下所示的位置上:

```
(mary bob (dave (pete)) ed)
```

也就是说，光标将移过 S- 表达式“(dave (pete))”——这一项本身是一个列表。但如果光标位置是下面这样的:

```
(mary bob (dave (pete)) ed)
```

按下“**ESC C-f**”组合键时，它就将移动到:

```
(mary bob (dave (pete)) ed)
```

第二个例子里的 S- 表达式是原子项“bob”。

表 12-5 列出了对列表进行处理的有关命令。

PDG

表 12-5: 列表处理命令速查表

键盘操作	命令名称	动作
ESC C-n	**forward-list**	移动到上一个列表
ESC C-p	**backward-list**	移动到下一个列表
ESC C-d	**down-list**	向前移动，进入下一级括号层次
(无)	**up-list**	向前移动，退出当前的括号层次
ESC C-u	**backward-up-list**	向后移动，退出当前的括号层次

为了便于记忆，可以把列表想像为文本行，把 S- 表达式想像为字符；因此，“**C-n**”和“**C-p**”出现在列表移动命令清单里，而“**C-f**”和“**C-b**”则出现在 S- 表达式移动命令清单里。“**ESC C-n**”和“**ESC C-p**”的工作情况，分别类似于“**ESC C-f**”和“**ESC C-b**”，但在光标的前或后必须有一个可以让光标移动经过的列表项。换句话说，光标位置上、光标后面或光标前面必须有一个左括号或右括号。如果没有括号，Emacs 就会报告出现一个错误。比如说，如果把光标放在如下所示的地方:

```
(fred bob (dave (pete))ed)
```

并按下“**ESC C-n**”组合键，Emacs 就会给出如下所示的一条出错信息:

```
Containing expression ends prematurely
```

但如果把光标放在如下所示的位置上:

```
(fred bob (dave (pete)) ed)
```

则“下一个列表”其实就是“(dave (pete))”。如果在这个时候按下“**ESC C-n**”组合键，光标就会到达如下所示的位置:

```
(fred bob (dave (pete))ed)
```

以列表项为单位进行移动的命令，使光标能够进入或者退出列表。比如，假设光标位于:

```
(fred bob (dave (pete)) ed)
```

则按下“**ESC C-d**”组合键将使光标移动到:

```
(fred bob (dave (pete)) ed)
```

出现这种结果的原因是“fred”就是括号中列表项的下一级层次。再按下“**ESC C-d**”组合键将使光标移动到:

```
(fred bob (dave (pete)) ed)
```

现在进入到列表“(dave (pete))”里。此时，按下“**ESC C-u**”组合键将得到与“**ESC C-d**”组合键效果正好相反的结果，即让光标回退到这两级列表的外面。但如果输入的是“**ESC x up-list RETURN**”命令，就将向前越过这两级列表，得到如下所示的结果:

```
(fred bob (dave (pete)) ed)
```

表 12-6 对以函数定义（defun）为单位进行移动的命令进行了汇总，它们并不难理解。

表 12-6: 以函数定义（defun）为单位进行移动的命令速查表

键盘操作	命令名称	动作
ESC C-a	**beginning-of-defun**	移动到当前函数的开头
ESC C-e	**end-of-defun**	移动到当前函数的结尾
ESC C-h	**mark-defun**	把光标放到函数的开头，把文本块标记放到函数的结尾

这些命令只有在当前函数的“(defun”关键字出现在某个代码行的最开始处时才能正常工作。

LISP 编辑模式提供了配对左括号的快速“闪现”功能；如果配对的左括号不在当前窗口的显示范围内，它所在的那一行语句将显示在辅助输入区里。LISP 编辑模式还为 **TAB** 键和 **LINEFEED**（“**C-j**”组合键）提供了换行加缩进的自动功能（但 LISP 交互编辑模式不在此列，参见下面的内容）。

LISP 编辑模式所支持的缩进样式“知道”很多关于 LISP 关键字和列表语法方面的知识，但遗憾的是它并不那么容易定制（注 6）。

注 6: 缩进样式是由实现Emacs LISP模式的LISP代码决定的。如果你是一位有经验的LISP老手，就可以对 Emacs LISP 目录里的 **lisp-mode.el** 文件中的代码进行检查，并决定如何按自己的意愿对缩进样式进行定制。一个比较好的出发点是函数 **lisp-indent-line**。

请看下面的例子。这是本章前面内容中的C语言函数“times”的LISP写法，它展示了LISP在处理缩进样式方面的各种功能:

```
(defun times (x y)
  (let ((i 0)
        (result 0))
    (while (< i x)
      (setq result (+ result y)
            i (1+ i)))
    result))
```

基本的缩进量是2；只要下一行上的代码又向下嵌套了一级，Emacs就将使用这个值对它进行缩进。请看上面的例子，函数体（即从“defun”语句的下一行开始的代码块）向右缩进两个字符。而“(while . . . and result))”之间的语句又相对于“let”向右缩进了两个字符，这是因为它们是“let”引入的另外一个语句块。

虽然在行为上看起来像是关键字，可defun、let和while之类其实都是函数调用。函数调用所使用的缩进样式是这样的：如果函数有一个以上的参数，则该函数的函数名及其第一个参数，将出现在第一个代码行上，其他参数分别列在后续的代码行上，并且要与第一个参数对齐。换句话说，带有多个参数的函数应该写成下面这个样子:

```
(function-name arg1
               arg2
               arg3
               ...)
```

刚才定义的times函数里，带有多个参数的setq就是一个很好的例子。

但语句“(result 0)”所在的那一行的缩进格式，却表明非函数调用的列表遵守的是不同的规则。问题中的列表其实是“((i 0) (result 0))”，即由两个元素（这两个元素都是列表）组成的一个列表，所以LISP编辑模式所支持的缩进样式将把这两个元素对齐摆放。

虽然类似关键字的let和while等术语实际上是函数调用，可LISP编辑模式却能根据自己对这类函数的“理解”对它们进行特殊的缩进安排。比如，如果我们打算把while循环的条件判断语句放到另外一行，并用**TAB**键对它进行准确的缩进，就会得到下面这样的结果:

```
(while
        (< i x)
      (setq result (+ result y)
            i (1+ i)))
```

类似的事情也会发生在if和cond等控制结构身上；第十三章里有一些很好的缩进示例。

缩进方面另外一个值得注意的事项是：LISP编辑模式习惯于把多个右括号都连续地写在同一行的最末尾，而不是把它们分别写到不同的代码行上。比如，在上面的函数定义中，语句“i (1+i)))”里就包含着“1+”函数、setq语句和while语句等3项内容的右括号。如果愿意，当然可以把这些右括号分别写到不同的代码行上；但如果用**TAB**键来对它们进行缩进，那么它们一般不会和与它们配对的左括号排列为一条直线 —— 必须亲自动手来对它们进行缩进。

LISP 编辑模式不仅可以用 **TAB** 和 **LINEFEED** 命令进行缩进，还支持命令“**ESC C-q**”（命令名是**indent-sexp**），它的作用是对跟在光标后面的S-表达式的每一行进行适当的缩进。可以用这条命令对整个函数定义进行缩进 —— 只要把光标放在“defun”的右面再按下“**ESC C-q**”组合键即可。

LISP编辑模式里的注释由通用的注释命令“**ESC ;**”负责处理，它会按变量**comment-column**所设定的缩进量（或者，如果代码语句超出了这个位置，就缩进到语句最后一个字符再往右一个字符的位置）对代码中的注释进行缩进，插入一个分号（;），然后把光标放到分号的后面。如果想让注释占据整个一行（或者想从**comment-column**以外的其他任意位置开始写注释），就得亲自动手把光标移动到想让注释开始的地方并输入一个分号。如果注释内容占据了整个一行，并且又在这一行上按下了 **TAB** 键，则注释将被移动到 **comment-column** 所指定的位置。为了避免出现这种情况，可以用两个或者更多个分号来代替单个的分号。**TAB**会把这样写的注释留在原处而不对它们进行缩进。LISP 编辑模式还支持在本章前面介绍的其他注释命令，比如“**ESC j**”命令（用来在下一行继续写注释）和“**ESC x kill-comment RETURN**”命令（清除一条单行的注释）等。以上介绍的这些功能适用于全部三种 LISP 编辑模式；接下来，我们将介绍它们各自所独有的功能。

Emacs LISP模式是为运行于Emacs内部的LISP代码准备的；所以它支持直接运行输入的代码。因为LISP是一种解释型语言（与此相对的是纯粹的编译型语言），所以 LISP 程序设计过程中代码的书写和运行 / 调试阶段之间的界线是比较模糊的；

Emacs LISP模式利用了LISP语言的这个特点，而稍后将要介绍的LISP交互模式就更胜一筹。在Emacs LISP模式里，命令“**ESC C-x**”（命令名是**eval-defun**）能够对光标周围或者光标后面的函数定义进行求值，也就是说，它将对该函数进行语法分析并把它保存起来，以便Emacs在用户调用这个函数的时候“知道”有这样一个函数。

Emacs LISP模式还准备了一个“**ESC TAB**”命令（命令名是**lisp-complete-symbol**），它的作用是自动补足光标前面的符号（比如变量、函数名等），自动补足功能在第六章已经介绍过了。也就是说，如果输入某符号的前几个字符并按下“**ESC TAB**”组合键，Emacs就会设法自动补足这个符号的名字。如果它能自动补足符号名，那么当然是继续工作；可如果它无法自动补足符号名，就说明刚才输入的前几个字符不能独一无二地确定一个符号名。需要再多输入几个字符（以进一步确定那个独一无二的符号）；或者，也可以再按一次“**ESC TAB**”组合键，屏幕上将弹出一个帮助窗口，里面是各种可供挑选的符号。接下来，可以多输入一些字符以亲自补足那个符号，或者再让Emacs去自动补足。

但LISP语言的处理工具并非只有Emacs LISP解释器一种，LISP模式（相对于Emacs LISP模式而言）就是为使用其他LISP语言处理工具的用户而准备的。它提供了几个用来与外部LISP解释器进行通信的接口命令。LISP模式里的“**C-c C-z**”命令（命令名是**run-lisp**）将把用户系统的LISP解释器启动为一个子进程，并且会创建一个名为“`*lisp*`”的编辑缓冲区（以及一个与之关联的窗口）供输入和输出之用（注7）。如果已经有一个LISP子进程存在，“**C-c C-z**”命令就会使用它而不是另外再创建一个。如果把光标放在某个函数定义当中再按下“**ESC C-x**”组合键（此时它代表的是**lisp-send-defun**命令），就能把该函数定义送往LISP子进程中。这个过程使定义的函数在LISP解释器那里挂上号，今后就可以调用这些函数了。

如果编辑的文件内容全部都是Emacs LISP代码（例如正在设计编写自己的编辑模式（参见第十三章），或正在对某个现有的编辑模式进行修改），那么Emacs LISP模式是最佳的工具。但如果只是想编辑一点“小”LISP代码——比如往“*.emacs*”文件里添加语句或者对它稍做修改，就可以选用Emacs作为另外准备的功能更强大的编辑功能，这些编辑功能将进一步模糊编写和运行代码两项工作之间的界线。

注7：LISP编辑模式的这条命令（**run-lisp**）是为BSD UNIX系统上的franz LISP系统准备的，但它也能用来启动其他的LISP解释器。

这些更强大的功能中的第一个是“**ESC :**”命令（命令名是**eval-expression**）。这个命令可以在辅助输入区里输入任何一种形式的单行LISP表达式，然后对该表达式进行求值，再把其结果显示在辅助输入区里。这种方法特别适合用来检查Emacs变量的值，和试用那些没有按键绑定，或需要额外参数的Emacs“内部”函数。在使用**eval-expression**命令的时候，还可以利用符号自动补足命令“**ESC TAB**”。

不幸的是（或者说幸运的是，这取决于不同的观点），Emacs在一般情况下不可以使用**eval-expression**命令。如果试着按“**ESC :**”组合键，就会在辅助输入区里看到一条“loading novice . . .”消息，接着会弹出一个窗口询问“你并不是真的想输入这条命令，对吗?”这条消息。它下有3项选择：按空格键只执行这条命令一次；“**y**”执行这条命令并激活它以便今后继续使用，而且今后也不会再询问刚才的问题；“**n**”则代表什么也不做。

如果真的想使用**eval-expression**命令，请回答“**y**”。这个命令的执行效果是把下面这条语句添加到“*.emacs*”文件里：

```
(put 'eval-expression 'disable nil)
```

如果你是一位老练的LISP程序员，就会知道添加到“*.emacs*”文件里去的这条语句将把符号“**eval-expression**”的“**disabled**”属性设置为“**nil**”。换句话说，Emacs认为某些命令是不适合初级用户使用的，因此它会在默认的情况下禁用这些命令。如果你想跳过这些步骤而直接使用**eval-expression**命令，只要简单地把上面这条语句放到“*.emacs*”文件里去即可（那可别忘了输入几个单引号）。

能够帮助用户熟悉Emacs LISP代码的命令是“**C-x C-e**”（命令名是**eval-last-sexp**）。这个命令的作用是运行光标所在的代码行上的那条LISP语句，然后把其执行结果显示在辅助输入区里。“**C-x C-e**”非常适合用来测试Emacs LISP文件里的单条语句。

LISP交互模式的功能就更强大了。这是“*scratch*”编辑缓冲区默认进入的编辑模式。不带后缀名的文件名一般都会让Emacs进入LISP交互模式（可以用修改**auto-mode-alist**变量的方法来改变Emacs的这种默认行为），有关操作步骤请参考本章前面的内容，更详细的讨论见第十一章。也可以用“**ESC x lisp-interaction-mode RETURN**”把任意一个编辑缓冲区放到LISP交互模式里；如果想另外创建一个处于LISP交互模式的编辑缓冲区，可以按下“**C-x b**”组合键（命令名是**switch-to-buffer**），提供一个编辑缓冲区名字，最后把它放到LISP交互模式下即可。

除了把**LINEFEED**（即“**C-j**”组合键）绑定为**eval-print-last-sexp**命令之外，LISP交互模式与Emacs LISP模式可以说是一模一样。这个命令的作用是对光标前面的S-表达式进行求值，然后把其结果显示在该S-表达式所在的编辑缓冲区里。要想获得其他编辑模式绑定在**LINEFEED**上的换行及缩进的**newline-and-indent**命令效果，必须先按回车键再按**TAB**键。

请记住，LISP语言里的S-表达式可以是任何一个语法正确的表达式。因此，可以在LISP交互模式里利用**LINEFEED**检查变量的值、输入函数定义和运行函数等等。举例来说，如果在输入变量**auto-save-interval**后按下了**LINEFEED**，就能查看到这个变量的值（默认值是300）。如果输入一个函数定义后在它最后一个右括号的后面按下了**LINEFEED**，Emacs就会把这个函数保存起来（以便今后调用）并打印出它的名字；就这种情况而言，**LINEFEED**的作用类似于“**ESC C-x**”组合键（命令名是**eval-defun**），只不过它要求光标必须位于定义的函数的后面（而不是放在函数定义的前面或中间）。如果输入的代码是对某个函数的调用，Emacs就会求值（运行）这个表达式并把该函数的返回值都显示出来。

LISP交互模式中的**LINEFEED**是试用、渐进式开发和调试LISP代码的绝佳手段。同时，因为Emacs LISP本身就是“真正的”LISP语言，所以甚至可以用它来为其他的LISP系统开发一些代码。

FORTRAN模式

当访问后缀名是“*.f*”或“*.for*”的文件时，Emacs就会自动进入FORTRAN模式。FORTRAN模式有几项功能专门用来处理这种面向字符列位置的格式。它不仅能识别和理解其他语言编辑模式所共有的语言学元素，还支持一些专门用来对语句和子程序进行处理的命令；我们把这些命令列在表12-7里。

表12-7：FORTRAN模式中的移动命令速查表

键盘操作	命令名称	动作
C-c C-n	**fortran-next-statement**	向前移动一个语句
C-c C-p	**fortran-previous-statement**	向后移动一个语句
ESC C-a	**beginning-of-fortran-subprogram**	移动到当前子程序的开头

表 12-7: FORTRAN 模式中的移动命令速查表（续）

键盘操作	命令名称	动作
ESC C-e	**end-of-fortran-subprogram**	移动到当前子程序的结尾
ESC C-h	**mark-fortran-subprogram**	把光标放到子程序的开头，把文本块标记放到子程序的结尾

“**C-c C-n**”和“**C-c C-p**”命令与普通的“**C-n**”和“**C-p**”命令是不一样的，它们能跳过程序中的续行和注释行。

FORTRAN 模式也支持其他语言编辑模式所支持的缩进和注释命令，但在做法上有一些我们稍后就会介绍给大家的变化。除通用的注释命令“**ESC ;**”外，FORTRAN 模式还准备了一个对包围光标的子程序进行适当缩进的“**ESC C-q**”命令（命令名是**fortran-indent-subprogram**），和一个把光标和文本块标记之间的区域里的语句全部设置为注释行的“**C-c ;**”命令（命令名是 **fortran-comment-region**）。

FORTRAN 模式在缩进、续行和注释样式等问题上的每一个侧面几乎都能通过相应的 Emacs 变量来进行定制（注 8）。我们把这些变量列在表 12-8 里，然后给出它们的几个用法示例。

表 12-8: FORTRAN 模式中的 Emacs 变量

变量	默认值	说明
fortran-minimum-statement-indent	6	开始输入语句之前需要保留的缩进量（例如：缩进量为 6 时语句将从第 7 列开始输入）
fortran-do-indent	3	do 语句块里的语句还需要增加的缩进量
fortran-if-indent	3	if 语句块里的语句还需要增加的缩进量
fortran-continuation-char	$	续行时放在第 6 列里的续行标记字符
fortran-continuation-indent	5	语句续行时的缩进量（比如从第 6 列开始再缩进 5 个字符的位置）

注 8: 事实上，甚至可以把FORTRAN模式定制得让编译器认不出那是FORTRAN代码来！

表 12-8 FORTRAN 模式中的 Emacs 变量（续）

变量	默认值	说明
fortran-comment-line-column	6	占据一整行的注释所使用的缩进量；“**ESC ;**”命令使用的就是这个值
fortran-comment-indent-style	'fixed（固定值）	这个变量的值可以取“**nil**”、“**'fixed**”或“**'relative**（相对值）”，注意后两个值的前面带有单引号；请参见下面的用法示例
comment-start	nil	与程序代码同在一行上的注释的前导字符；请参见下面的用法示例。如果这个变量取值为“**nil**”，则表示不允许这样的注释
fortran-line-number-indent	1	行号的最大缩进量，用来保证行号不会延伸到第 5 列（续行标记列）上。请参考后面对语句自动编号功能的讨论
fortran-comment-region	“c$$$”	由 **fortran-comment-region**（“**C-c ;**”）命令插入到每一行前面去的字符串

下面用几个具体的例子来演示一下这些功能的用法。假设用FORTRAN语言写出一个“times”函数，而它的功能与本章前面给出的那个同名的C语言函数是一样的：

```
      integer function times (i, j)
      times = 0
      do 10 k = 1, i
         times = times + j
10    continue
      return
      end
```

输入这些代码行的时候，按下 **TAB** 键会把光标先移动到第 7 列 —— 也就是开始输入语句的地方（这个缩进量由 **fortran-minimum-statement-indent** 变量控制）。如果输入的是 do 或 if 语句，Emacs 还将根据变量 **fortran-do-indent** 和 **fortran-if-indent** 的取值情况再对它们做相应的缩进（注 9）。请注意，输入“continue”的时候，它先会像 do 循环的循环体语句那样被缩进，所以必须再按下 **TAB** 键才能让

注 9： FORTRAN 模式的这项功能只有在关键字 do 或 if 之后，至少有一个空格的情况下才能发挥作用，可大多数编译器却没有这样的要求。

Emacs 调整其缩进量以对齐与之配对的“do”。用来结束 if 循环的“continue”语句也需要做同样的处理。(FORTRAN 模式还能识别和理解用来结束 do 循环的“end do”和“enddo”以及用来结束 if 循环的“else”、“elseif”、“endif”和“end if”。)另一个必须注意的地方是语句块不能共用同一个“continue”语句，每个 if 或 do 都必须有它自己的“continue”。因为只有这样，FORTRAN 模式的自动缩进功能才能正确工作。

FORTRAN 模式提供了一个名为“自动编行号（electric line numbering）”的功能来处理FORTRAN程序里的行号问题。如果在开始输入某行代码之前先输入了一个数字，FORTRAN 模式就将认为输入的是一个行号，而它就将据此对这个数字以及后续输入的其他数字进行正确的缩进 —— 它会把这个数字挪到第 2 列（这个缩进量由变量 **fortran-line-number-indent** 控制）上，然后再把光标放到第 7 列上，以便直接开始输入这一行上的FORTRAN语句。后续输入的数字将“加入”到行号里，而光标仍将返回到第 7 列。

请看下面这个用法示例。假设想给代码行加上一个行号“10”；光标现在位于这一行的最开始处。于是，输入数字“1”的时候，结果将是这样的：

```
1      ▌
```

接着输入数字“0”的时候，它会和数字“1”结合在一起构成行号“10”，如下所示：

```
10     ▌
```

注意光标会回到第7列等待输入这一行的语句。最多可以输入一个5位数的行号；输入行号的第5位数字时，行号的缩进量将被调整为1以留出必要的空间，如下所示：

```
12345 ▌
```

如果试图输入一个超过 5 位数字的行号，FORTRAN 模式将在辅助输入区里显示一条警告信息。

FORTRAN 模式支持的注释样式有两种：一种是标准样式，它要求注释必须自成一行，而且必须在第 1 列里加上一个字母“c”作为标志；另一种样式允许注释与语句出现在同一行上，语句后面的惊叹号字符“!”表示随后的文字内容是一条注释。值得注意的是很多编译器并不支持后一种风格的样式。在默认的情况下，FORTRAN

模式也不允许使用后一种注释样式，但可以通过把变量**comment-start**设置为惊叹号字符（"!"）的方法来启用它。

如果已经把**comment-start**设置为惊叹号字符（"!"）了，那么，当用“**ESC ;**”命令准备在已经输入有语句的代码行上输入注释的时候，Emacs将根据变量**comment-column**的取值对它进行缩进（或者，如果那个位置已经有了文本，就缩进到该文本最后一个字符再往右一个格的位置），再插入一个惊叹号字符（!），然后把光标放在惊叹号的后面。此外，如果用“**ESC j**”组合键把注释延伸到下一行，Emacs就将插入一个新行，缩进到变量**comment-column**设定的位置，再插入另外一个惊叹号字符（!）。如果变量**comment-start**设置为“**nil**”，那么，当用“**ESC ;**”命令准备在已经输入有语句的代码行上输入注释的时候，Emacs就会在当前行的上面插入一个新行，然后在新行的第1列放上一个字母“c” —— 也就是插入一个整行的注释。

可以用变量**fortran-comment-indent-style**来控制整行注释的缩进样式。在一般情况下，如果在一个空白行上按下“**ESC ;**”组合键，Emacs就将自动插入一个字母“c”并缩进到第7列去。如果试图在第7列之前开始输入注释文本，那么Emacs会在按下**TAB**键的时候（或者在用“**ESC C-q**”组合键对整个子程序进行缩进的时候）把它重新缩进到第7列。这种行为是由变量**fortran-comment-indent-style**的默认值设定的，这个默认值是“**'fixed**”（注意它前面的单引号）。

不过，也许有的编译器允许注释文本占据第7列之前的位置。如果想利用这一点，就需要把变量**fortran-comment-indent-style**设置为“**nil**”。此后，就可以在字母“c”标志后面的任意位开始输入注释文本；而且，即使按下了**TAB**键或“**ESC C-q**”组合键，Emacs也不会改变缩进设置。另外一种选择是：如果想让注释文本的缩进量超过7个字符，就需要把变量**fortran-comment-indent-style**设置为“**'relative**”（注意它前面的单引号），这将把注释文本的缩进量设置为7加上变量**fortran-minimum-statement-indent**的值；也就是从第7列开始再多缩进6列（以变量的默认值为例），总的缩进量将达到13列。

FORTRAN模式的另一个特色是它的缩略词汇（参见第三章）表，这里所说的缩略词汇是FORTRAN语言中的关键字的缩写，可以利用它们加快代码的输入速度。FORTRAN模式的缩略词汇表其实是利用分号（;）字符对Emacs原有的简写词功能进行的扩展。为了避免与设置的其他简写词发生冲突，FORTRAN模式的缩略词

汇全都以分号（；）打头；如果输入“；?”这两个字符，Emacs就会把全体FORTRAN缩略词汇列成一份清单显示在一个“*Help*”窗口里。

要想在FORTRAN模式里使用缩略词汇，需要先用“**ESC x abbrev-mode RETURN**”命令启用Emacs的简写词功能。然后，就可以用输入一个简写词再按空格键的方法来快速输入。每一个FORTRAN关键字都有相应的缩略形式；表12-9里列出的只是其中的一部分。

表12-9：FORTRAN模式下的一些缩略词汇

缩略词汇	关键字
;c	continue
;dp	double precision
;dw	do while
;f	format
;fu	function
;g	goto
;in	integer
;p	print
;rt	return
;su	subroutine

我们来看几个例子。如果想定义一个函数（并且已经在简写模式里），则直接输入“**;fu**”再按下空格键即可。如果输入“**;fu**”之后停了下来，Emacs会把刚输入的字符显示在辅助输入区里以提醒用户当前正在使用一个缩略词汇；按下空格键之后，辅助输入区里的“**;fu**”字样就会消失，取而代之的将是关键字“function”。

最后再介绍几个用在FORTRAN模式里的杂项命令。“**ESC C-j**”组合键（命令名是**fortran-split-line**）所做的事正好与“**ESC ^**”（把这一行和上一行合并为一行）相反：它把当前行从光标位置一分为二，给第二行加上一个续行标记字符，再让它多缩进6列。“**C-c C-r**”组合键（命令名是**fortran-column-ruler**）在当前行的上方显示一把由两行字符组成的标尺，如下所示：

```
0   4 6  10        20        30        40        50        60        70
[   ]|{   |    |    |    |    |    |    |    |    |    |    |    |    |}
```

标尺是临时性的，它会在按下任意一个键之后消失。最后，“**C-c C-w**”组合键（命令名是**fortran-window-create**）会把当前窗口在水平方向上收缩为72列宽，这可以帮助控制语句不至于太长。标尺命令可以帮助完成用“**ESC C-j**”组合键分断当前行的操作。

对程序进行编译

我们在本章的开头已经讲过，Emacs向程序员提供的支持并不仅限于写代码的时候。当工作在大型程序设计项目上的时候，一个典型的Emacs使用策略是登录上机，进入存放有源代码文件的目录，启动Emacs来编辑所有的源代码文件（比如C程序员使用的“**emacs Makefile *.[ch]**”命令）。在对源代码进行编辑的过程中，可以用下面将要介绍的命令对它进行编译（正如大家将要看到的那样），根本不用考虑保存修改。还可以利用Emacs的shell模式（参见第五章）在一个shell窗口里对经过编译的代码进行测试。总之，就整个上机过程而言，你几乎完全不用退出Emacs。

Emacs准备了一个面向编译器和UNIX的**make**工具的接口；与shell模式相比，这个接口更直接、更强大。这个接口的核心是命令“**ESC x compile RETURN**”，它将引发一连串的事件。

首先，它会提示输入一个编译命令。默认的编译命令是“**make -k**”（注10），但如果输入了另外一个命令，新的命令就将在本次Emacs会话过程中以后的编译操作里取代原来的默认编译命令。如果想改变默认的编译命令，可以在“*.emacs*”文件里对变量**compile-command**进行设置。

输入编译命令之后，Emacs会自动把尚未存盘的编辑缓冲区保存起来；而保存尚未存盘的修改本来是个人的责任，Emacs会在这方面减轻用户的负担。接着，它在一个关联窗口里创建一个名为“`*compilation*`”的编辑缓冲区。然后，它运行编译命令（编译命令是作为一个子进程来运行的，就像shell模式里的shell一样），编译器的输出显示在“`*compilation*`”编辑缓冲区里。在编译命令执行期间，在状态行上显示“`Compiling: run`”字样；编译作业结束时，则显示“`exit`”字样。

注10：“**-k**”选项的作用是取消**make**在某个作业返回一个错误之后就停止执行的默认行为。加上这个选项之后，即使依赖树（dependency tree）的某个分支出了错，**make**也不会停止执行，它会继续对不依赖于该出错分支的其他分支进行处理。

下面的事情就更有意思了。如果编译结果报告说代码里有错误，那么可以用“**C-x `**”组合键（命令名是**next-error**，注意第二个字符是一个反引号而不是一个单引号）来检查代码；Emacs会读入第一条出错信息，把出现错误的文件和行号找出来，然后把光标移到该文件和该行号指定的地方。修改完错误之后，如果再次按下“**C-x `**”组合键，就会来到下一个出现错误的地方。每按一次“**C-x `**”组合键，Emacs就会卷动“*compilation*”窗口里的内容，让当前的出错信息显示在该窗口的顶部。

如果想从第一条出错信息处重新开始，就需要给“**C-x `**”加上一个前缀（即按下“**C-u C-x `**”组合键。“**C-x `**”组合键还有这样一个好处：可以在看到出错信息之后立刻按下它，用不着等到整个编译工作都结束之后再使用它。

“*compilation*”编辑缓冲区的编辑模式（即Emacs的编译模式）还支持其他一些用来浏览出错信息的命令；我们把这些命令汇总在表12-10里。

表12-10：编译模式命令速查表

键盘操作	命令名称	动作
C-x ‘	**next-error**	移动到下一条出错信息并访问与之对应的源代码
ESC n	**compilation-next-error**	移动到下一条出错信息
ESC p	**compilation-previous-error**	移动到上一条出错信息
C-c C-c	**compilation-goto-error**	访问对应于当前出错信息的源代码
SPACE	**scroll-down**	下卷屏幕显示内容
DEL	**scroll-up**	上卷屏幕显示内容

SPACE键和**DEL**键是各种Emacs的“只读”模式——比如RMAIL模式（参见第六章）所使用的卷屏命令，它们用起来很方便。

请注意，“**ESC n**”和“**ESC p**”并不是用来访问对应于出错信息的源代码的；它们的操作对象是出错信息本身。如果出错信息的内容多于一行，用它们来移动会比较方便些。如果想访问对应于某条出错信息的源代码，就需要使用“**C-c C-c**”组合键。

那么，Emacs又是如何对出错信息进行分析的呢？它使用变量**compilation-error-regexp-alist**对出错信息进行分析；这个变量是一个由正则表达式构成的列表，它的作用是对很多种C / C++编译器以及C代码检查程序**lint**所给出的出错信息进行匹配（注11）。这个变量与Emacs其他语言编辑模式所对应的程序设计语言（比如FORTRAN、Ada和Modula-2等）的编译器也能配合工作。Emacs会用该列表里的每一条正则表达式对一条出错信息进行尝试性分析，直到它找到那条能够用来准确地提取出错误发生地点的文件名和行号的正则表达式为止。

尽管已经尽了很大的努力，Emacs的出错信息分析器无法与某些编译器配合工作的情况还是有可能发生；尤其是在一个非UNIX系统上使用Emacs的时候更容易出现这种情况。拿一些你知道肯定包含有某些特定错误的程序代码来试试“**ESC x compile**”命令就可以做出判断：如果Emacs在按下“**C-x `**”组合键时报告说“no more errors（没有其他错误）”，就说明**next-error**命令不能与编译器配合工作。

如果Emacs的出错信息分析器不能与编译器配合使用，则可以采取下面这种方法来解决这一问题：给变量**compilation-error-regexp-alist**增加一条符合编译器的出错信息格式的正则表达式。我们将在第十三章内容里介绍一个具体的例子。

compile程序包还包括有为UNIX的**grep**（查找文件）命令提供的类似支持，这就使Emacs具备了在多个文件里进行查找的能力。如果输入“**ESC x grep**”命令，Emacs就会提示输入一些准备传递给**grep**的参数——即一个查找模板（search pattern）和一组文件名。然后，Emacs会调用执行带“**-n**”选项的**grep**命令，这个选项的作用是告诉**grep**把匹配到的代码行的文件名和行号显示出来（注12）。以后的事情就和“**ESC x compile**”命令的后续操作情况一样了；可以用按下“**C-x `**”组合键的方法让Emacs访问**grep**所查找的文件里的下一个匹配行。

注11：令人遗憾的是，Emacs无法对关于**make**自身的出错信息进行分析——比如制作文件（makefile）中的语法错误。

注12：如果只用“**grep -n**”命令对一个文件进行操作，它就将只给出行号；但Emacs会通过给**grep**命令增加一个哑文件参数*/dev/null*的办法，来强制它在这种情况下也把文件名显示出来。

PDG

第十三章

用LISP语言对Emacs做进一步开发

本章内容:

- LISP语言简介
- LISP语言的基础函数
- Emacs的内部函数
- 主编辑模式程序设计实例
- 对现有编辑模式进行定制
- 建立自己的LISP开发库

Emacs并不是万能的。如果你使用Emacs已经有了一段时间并且用过它的一些高级功能，就肯定会发现一些Emacs无法实现的功能。虽然Emacs有好几百条内部命令、好几十个程序包和编辑模式以及许多诸如此类的东西，但终将会出现这样一种情况：你需要的某些功能是Emacs不具备的——或者说至少在缺省情况下是如此。不论你发现它缺少的是什么样的功能，都可以自己动手，用Emacs LISP程序实现它。

但在进入正题之前，我们想提醒大家一句：并非每个人都适合学习这一章内容。它是为那些已经能熟练使用Emacs的人们和那些有程序设计经验（并不非得是LISP程序设计经验）的人们准备的。如果你没有这样的经验，那最好还是跳过本章；如果你就是想用Emacs来完成某些事情，那么可以去找个友好的Emacs LISP老手来帮忙，让他替你写出必要的程序代码。

再者，我们并不打算在这一章里对LISP程序设计做全面细致的讨论，那是另外一本书的事。我们的侧重点在于介绍这种语言的基本概念和编写Emacs代码时常用的命令和技巧。如果你想在本章的基础上做进一步的学习，可以去研读自由软件基金会（他们的地址见附录一）提供的《Gnu Emacs LISP Reference Manual》（GNU Emacs LISP参考手册）；或者任何一本LISP教科书（Winston和Horn合著的《LISP》是一个不错的选择；这本书是由Addison-Wesley出版社出版的）。

Emacs LISP 是一种全功能的 LISP 实现（注 1）；因此，与大多数文本编辑器所普遍使用的宏（macro）或脚本语言相比，它提供了更强大的功能。（本书作者之一曾经完全依靠 Emacs LISP 写出过一个小型的专家系统。）事实上，完全可以把 Emacs 看做是一个带有很多内部函数的 LISP 系统，而其中许多函数是用来实现文本处理、窗口管理、文件 I/O 和其他各种与文本编辑有关的功能的。Emacs 的源代码是用 C 语言写的，它只实现了 LISP 解释器、LISP 指令和一些最基本的文本编辑命令；Emacs 的其他功能则是在它们的基础上用一个规模巨大的内部 LISP 代码层实现的。Emacs 的最新版本大约带有 200,000 行 LISP 代码。

LISP 语言在许多方面与 C 或 Pascal 等其他常用的程序设计语言是相似或者相通的，这一章就将从程序设计语言的共通之处开始讲起。但这些基础性的功能已经足以让人们写出许多 Emacs 命令了。接下来，我们再一起去探讨怎样才能把 LISP 代码与 Emacs 衔接起来，让编写的 LISP 函数能够成为 Emacs 的编辑命令。我们会见到许多能帮助写出 Emacs 命令的 LISP 内部函数，其中有不少都用到了正则表达式；我们已经在第三章里对正则表达式做过简单的介绍，在此基础上，我们将在本章以 LISP 程序设计为侧重点对它做更进一步的探讨。在此之后，我们将回到 LISP 的基本概念上，把这种语言在列表（list）处理方面的独到之处介绍给大家；然后，我们将通过实例向大家演示一下怎样才能写出一个主编辑模式的程序代码，虽然这个主编辑模式很简单，但本书各章所介绍的概念都将集中体现在里面。然后，我们将介绍如何在不改动（甚至用不着查看）用来实现 Emacs 内部的各种主编辑模式的源代码的前提下，对它们进行定制，你们将会看到这有多么简单。在本章的结尾，我们将介绍如何建立自己的 LISP 程序包。

LISP 语言简介

大家可能听说过 LISP 是一种用在人工智能（artificial intelligence，AI）方面的程序设计语言。即使你不了解人工智能，也没有什么好担心的。LISP 语言的语法可能不同寻常，但它的基本功能与 C 或 Pascal 等较为常见的程序设计语言却没什么不同；而这一章的学习重点正是这些基本的功能。在介绍完 LISP 语言的基本概念之后，我

注 1：有经验的 LISP 程序员会发现 Emacs LISP 最接近 MacLISP，但增加了一些 Common LISP 的功能。更完整的 Common LISP 仿真效果可以通过加载 cl 扩展包实现（参见附录四）。

们将编写出各种示范性的函数，而你完全可以把它们用到Emacs里。为了能够试用这些示范性的函数，你需要熟练掌握Emacs中的Emacs LISP模式和LISP交互模式，而这两种编辑模式都已经在第十二章介绍过。

LISP 语言的基本元素

函数、变量和原子项是LISP语言最基本的元素，也是每一个想要学习LISP语言的人所必须熟悉和掌握的概念。函数是LISP语言中惟一的程序单元；其地位相当于其他程序设计语言中的各种过程（procedures）、子例程（subroutines）、程序（programs）甚至是操作符（operators）。

LISP语言中的“函数”定义为由上述基本元素组成的各种“列表”（list），而且这种列表往往还嵌套调用着其他现有的函数。一切函数都有“返回值”（就像C或Pascal语言中的函数一样）；一个函数的返回值就是该函数最后一个列表项的值，它通常就是最后一个函数调用所返回来的值。嵌套在另外一个函数里的某个函数调用相当于其他程序设计语言中的一条“语句（statement）”，而我们将在本章内容里不加区别地使用“语句”和“函数调用”进行讨论。函数调用的语法是：

```
(function-name argument1 argument2 ... )
```

它相当于C或Pascal语言中的：

```
function_name (argument1, argument2, ... )
```

这条语法适用于一切函数，那些相当于其他程序设计语言中的算术或比较操作符的函数也不例外。比如说，如果想对2和4做加法，在C或Pascal语言中要使用表达式“2 + 4”；但在LISP语言里，必须把它写为：

```
(+ 2 4)
```

类似地，如果想表达“4 >= 2”（大于或等于），在LISP语言里就必须把它写为：

```
(>= 4 2)
```

LISP语言中的“变量”与其他程序设计语言中的变量概念很相似，但LISP变量没有类型（type）的说法。一个LISP变量可以取任意类型的值。

“原子项”是任意类型的值，它可以是整数、浮点数（即实数）、字符串、布尔真/假值、符号，也可以是编辑缓冲区、窗口、进程等特殊的Emacs类型。下面是与这些原子项有关的语法规定:

- **整数** —— 取值范围在 -2^{27} 到 $2^{27}-1$ 范围间的所有数字。
- **浮点数** —— 即数学上的实数，它们可以用小数点和科学计数法（用小写字母“e”来表示10的幂次）来表示。比如说，数值5489可以被表示为5489、5.489e3、548.9e1 等等。
- **字符** —— LISP 语言中的字符必须带有一个前导的问号，比如像“**?a**”这样。**ESC**、**LINEFED** 和 **TAB** 分别用“**\e**”、“**\n**”和“**\t**”来表示；其他控制字符要加上前缀“**\C-**”。比如说，“**C-a**”字符表示为“?\C-a”（注 2）。
- **字符串** —— 必须放在一对双引号中间；字符串中的引号和反斜线字符必须用一个反斜线字符来引导。比如说，"Jane said, \"See Dick run.\"" 是一个合法的字符串。
- **布尔值** —— “**t**”表示真，“**nil**”表示假；在大多数需要使用一个布尔值的场合，任何一个不是“**nil**”的值都意味着真。“**nil**”在很多场合还用来表示一个空值（null）或无取值，我们马上就能看到具体的例子。
- **符号** —— 即 LISP 语言中各种事物的名字，比如变量或者函数的名字。有时候，我们需要使用的是事物的名字而不是它的值，这就需要在该事物的名字前面加上一个单引号（'）。举个例子，我们在第十一章介绍的**define-key** 函数使用的就是命令的名字（做为一个符号）而不是该命令本身。

一个能够集中体现这些 LISP 基本概念的简单例子是函数 **setq**（注 3）。大家可能已经从前面各章里总结出来，即**setq**是对变量进行赋值的一个方法，比如下面这条语句:

```
(setq auto-save-interval 800)
```

请注意，**setq** 是一个函数，这与其他程序设计语言使用“=”或“:=”之类的特殊

注 2: 字符也可以用整数来表示。此时使用的是 ASCII 代码（在大多数机器上）。

注 3: 我们希望 LISP 的卫道士们能够原谅我们把**setq** 称为一个函数。我们这样做是为了能够不拘形式地把问题讲清楚，而这也正是它在技术方面的意义所在。

语法进行赋值的做法是不一样的。**setq** 要使用两个参数：一个变量名和一个值。在上面的这个例子里，变量**auto-save-interval**（两次自动保存操作之间等待的按键次数）的值被设置为800。

setq 可以一次对多个变量进行赋值，请看下面这个例子：

```
(setq thisvar thisvalue
      thatvar thatvalue
      theothervar theothervalue)
```

setq的返回值就是最后一个赋值，对上例而言就是“theothervalue”。对变量进行赋值还可以使用其他方法，但 **setq** 是适用面最宽的。

函数的定义

我们一起来看一个简单的函数定义示例。如果你使用的是终端，则可能想不带任何参数地启动Emacs；这将使你进入编辑缓冲区“*scratch*”，它是一个LISP交互模式（参见第十二章）下的空白编辑缓冲区，你可以在这个编辑缓冲区里试用我们在本章给出的程序示例。

在介绍程序示例之前，大家有必要再多了解一些LISP的语法规定。首先，短划线字符（-）在变量名、函数名等语言元素里用做“间隔”字符。这是LISP程序设计中一种广泛使用的记号写法；它与C语言和Ada语言中下划线字符（_）的作用是一样的。另外一个更值得重视的问题是LISP程序代码中的括号。LISP是一种“古老”的程序设计语言，在它诞生的年代里，人们对程序设计语言的语法问题还很少做细致的考虑，因此它的语法对程序员并不是非常友好。而用LISP语言写出的程序会使用大量的列表——也就是会大量地使用括号；从程序设计的角度看，这种做法有它的优点，大家学到这一章的末尾就会看到。

但从程序员的角度看，如何一个不少地把那么多的括号都正确地配上对却是个令人头疼的问题。使这一问题更加复杂的是，人们习惯于把多个右括号集中放置到代码行的末尾，而不是把它们分别放在一行并与对应的左括号对齐——后者的可读性当然要好得多。Emacs LISP模式提供的支持功能（特别是它的**TAB**键缩进功能和配对括号“闪现”功能）为我们准备了最好的解决手段。

现在，来看看我们的示例函数。假设你是一名学生或记者，需要把一篇论文或者稿

件中的单词个数统计出来。Emacs没有提供统计编辑缓冲区中单词个数的内部函数，所以我们打算写一个 LISP 函数来完成这项工作:

```
1. (defun count-words-buffer ()
2.  (let ((count 0))
3.    (goto-char (point-min))
4.    (while (< (point) (point-max))
5.      (forward-word 1)
6.      (setq count (1+ count)))
7.    (message "buffer contains %d words." count)))
```

我们来对这个函数一行一行地进行分析，看它到底做了些什么。(当然，如果你是在一个终端上试用这个函数的，可别把行号也输入进去。)

第 1 行上的 **defun** 定义了一个函数的名字和它的参数。别忘记 **defun** 本身也是一个函数 —— 它会在自己被调用时定义一个新的函数。(**defun**的返回值是一个符号，这个符号就是它所定义的函数的名字。)函数的参数都放在括号里，形成了一个由变量名组成的列表。这个例子里的函数没有参数。如果在某个参数的前面加上一个关键字“**&optional**”，就可以把这个参数设置为可选参数。如果一个参数是可选的并且没有在函数调用的时候给出来，它的取值就将是“**nil**”。

第 2 行上是一个 **let** 结构，这个结构的基本格式是:

```
(let ((var1 value1) (var2 value2) . . . )
  statement-block )
```

let 做的第一件事是对变量 var1、var2 等进行定义，把它们的初始值设为 value1、value2 等等。接着 **let** 将执行它的语句块部分，语句块是顺序排列的一组函数调用或者值，就像函数的函数体一样。

为了便于理解，我们可以认为 **let** 做了 3 件事情:

- 定义（或者叫“声明”）了一个由变量组成的列表。
- 给变量设置初始值，就像 **setq** 那样。
- 创建一个“知道”那些变量的语句块；**let** 语句块也叫做变量的“作用域(scope)”。

对一个变量来说，如果它是用 **let** 定义的，就允许以后在 **let** 语句块里用 **setq** 对它的

值重新进行设置。再深入一些，可以给用**let**定义的变量起一个与某个全局变量相同的名字；在该**let**语句块里对该变量进行的一切**setq**操作都只作用于那个局部变量，那个同名字的全局变量不会受到任何影响。但如果用**setq**赋值的变量不是用**let**定义的，该变量就将被看做是一个全局变量。要尽可能地避免使用全局变量的名字做为局部变量的名字，因为这很有可能会引起局部变量与同名的全局变量发生冲突。

回到我们的示例函数上来。我们用**let**定义了一个局部变量**count**，并把它的初始值设为0。我们马上就会看到，这个变量将用做循环计数器。

第3行到第7行是**let**语句块里的语句。第一条语句（第3行）调用了Emacs的内部函数**goto-char**，这个函数我们最早是在第二章里看到的。**goto-char**函数的参数是一个（嵌套着的）函数调用，被调用的函数也是Emacs的一个内部函数，它的名字是**point-min**。我们在前面已经讲过，“point（插入点）”是Emacs内部对光标位置的称呼，我们将在本章以后的内容里有时会用“插入点”来称呼“光标”。**point-min**的返回值是当前编辑缓冲区中第一个字符的位置，它的取值几乎总是1；于是，**goto-char**以数值1为参数被调用，整个语句的执行效果是把光标移动到编辑缓冲区的开头。

第4行安排了一个**while**循环，它与C语言或Pascal语言中的**while**循环没什么两样。LISP语言中的**while**结构的基本格式是这样的：

```
(while condition
   statement-block)
```

与**let**的情况相类似，**while**也设置了一个语句块。循环条件**condition**是一个值（一个原子项、一个变量或者是有一个返回值的某个函数）。**while**将对这个值进行测试；如果它是“**nil**”，循环条件将被认为是假，这个**while**循环也就到此为止。如果这个值不是“**nil**”，循环条件将被认为是真，语句块将被执行，然后再次对这个变量进行测试，整个过程就这样循环下去。

当然，写出一个无限循环的可能性是存在的。如果你写的LISP函数里有一个**while**循环，而你的终端在运行这个函数的时候又被挂起来了，那就极有可能是错误地写出了一个无限循环——这种错误并不少见。按下“**C-g**”组合键可以结束无限循环。

在我们的示例函数里，用来判断循环条件的函数是“**<**”，这是一个需要有两个参数

的“小于”函数，它与C语言或Pascal语言中的“<”操作符的作用是一样的。第一个参数又是一个函数，它负责返回插入点的当前字符位置；第二个参数返回的是编辑缓冲区里的最大字符位置，也就是编辑缓冲区的长度。“<”函数（以及其他条件比较函数）返回的都是一个布尔值，即“**t**”或“**nil**”。

while循环的语句块由两条语句组成。第5行把插入点向前移动一个单词（相当于按下“**ESC f**”组合键）。第6行给循环计数器加上一个1，函数“**1+**”是“**(+ 1 variable-name)**”的简写形式。注意第6行上的第3个右括号是与单词**while**前面的那个左括号配对的。这个**while**循环会使Emacs以每次一个单词的间隔走过当前编辑缓冲区，同时完成统计单词个数的工作。

示例函数中的最后一条语句利用内部函数 **message** 在辅助输入区里显示一条消息，把编辑缓冲区里有多少个单词报告出来。C程序员对 **message** 函数的格式应该是比较熟悉的。**message** 函数的第一个参数是一个输出格式字符串，它是由普通文本和“*%x*”形式的排版指令组成的，其中“*x*”是下列几个可用的字母之一。**message** 函数将依次把格式字符串中的排版指令套用到后续的参数上，并根据百分号后面的字母对相应的后续参数做出解释。下面是被用做排版指令的各个字母的含义：

%s 字符串或符号

%c 字符

%d 整数

%e 用科学计数法表示的浮点数

%f 用十进制小数表示的浮点数

%g 用最短字符串表示的浮点数

比如，下面这条语句：

```
(message "\"%s\" is a string, %d is a number, and %c is a character"
         "hi there" 142 ?q)
```

将把下面这条消息：

```
"hi there" is a string,142 is a number, and q is a character
```

显示在辅助输入区里。它相当于下面这条C语言代码：

```
printf ("\"%s\" is a string, %d is a number, and %c is a character\n",
        "hi there", 142, 'q')
```

浮点格式的输出字符情况稍微有些复杂。如果没有特别指定，它们将保留到小数点后面的6位数字。比如下面这条语句:

```
(message "This book was printed in %f, also known as %e." 1994 1994 )
```

将产生如下所示的输出:

```
This book was printed in 1994.000000, also known as 1.994000e+03.
```

可以对小数点后面的保留位数进行设定，具体做法是在百分号“%”和字母“e”、“f”或“g”之间插入一个小数点和打算保留的小数位数。比如下面这条语句:

```
(message "This book was printed in %.3e, also known as %.0f." 1994 1994)
```

将在辅助输入区里显示下面这条消息:

```
This book was printed in 1.994e+03, also known as 1994.
```

把 LISP 函数转变为 Emacs 命令

我们刚才编写的**count-words-buffer**函数确实可以工作，但它还有几个小问题。如果已经把它输入进去，就可以在LISP交互窗口里输入“**(count-words-buffer)**”以执行之。把光标移动到该函数最后一个右括号的后面再按下“**C-j**”(或**LINEFEED**)组合键，即LISP交互模式下的“求值”键就可以运行它了。

如果窗口里的光标(即按下“**C-j**”组合键的位置)后面还有一些文本，就会看到这个函数会让Emacs把插入点移动到编辑缓冲区的最末尾，它不会再返回到它原来的位置。这是Emacs LSIP命令普遍存在的一个问题，这个问题有个简单的解决办法，那就是**save-excursion**函数。这个函数的语法定义如下所示:

```
(save-excursion
  statement-block)
```

用了这个函数之后，语句块中的语句在运行时引起的光标移动在Emacs的内部进行，不会显示在屏幕上；当语句块全部执行完毕之后，插入点和文本块标记将出现

在它们原先的位置。除了预防出现刚才讲的不便以外，在LISP函数有多个嵌套层次的情况下，**save-excursion** 函数还可以作为一个“堆栈”来保存、和检索各嵌套层次的插入点和文本块标记的位置。**save-excursion** 函数的返回值是语句块中最后一条语句的返回值。

因此，如果你不想让用户看到某个函数中的光标移动命令把光标在屏幕上移来移去的，就可以把它们放在一个 **save-excursion** 块里，比如下面这样：

```
(defun count-words-buffer ()
  (save-excursion
    (let ((count 0))
      (goto-char (point-min))
      (while (< (point) (point-max))
        (forward-word 1)
        (setq count (1+ count)))
      (message "buffer contains %d words."count))))
```

现在，既然已经能够让这个函数在 LISP 交互窗口里正确地执行，不妨像执行其他 Emacs 命令那样用“**ESC x**”执行这个函数。输入“**ESC x count-words-buffer RETURN**”命令，可看到的却是一条出错消息“[No match](不匹配)”。出现这条出错消息的原因是还没有把这个函数“登记”到 Emacs 里（必须先对函数进行注册才能在LISP交互模式里使用）。用来完成这项工作的函数是 **interactive**，下面是它的语法定义：

```
(interactive "prompt-string")
```

这条语句必须是函数里的第一条语句，也就是说，它必须紧跟在包含 **defun** 和文档字符串（参见下一页）的那个代码行的后面。**interactive** 的作用是把函数注册为 Emacs 的一个命令，这样，当执行它的时候，Emacs 会提示用户输入你在 **defun** 语句里声明的参数。这个函数中的提示字符串是可选的。

提示字符串的格式有特殊要求：你必须为每一个你想提示用户输入的参数准备一个提示字符串的子字符串。子字符串之间要用 **LINEFEED**（\n）字符来分隔。每个子字符串的第一个字母必须是一个用来表示该参数类型的代码。参数类型代码有很多选择，我们把比较常用的列在表 13-1 里。

表 13-1：交互式函数的参数类型代码

参数类型代码	参数类型
b	一个现有编辑缓冲区的名字
e	事件（鼠标动作或功能键动作）
f	一个现有文件的名字
n	数字（整数）
s	字符串
这些代码还有相应的大写变体：	
B	一个可能不存在的编辑缓冲区的名字
F	一个可能不存在的文件的名字
N	如果命令在调用时带有一个前缀参数，按前缀参数指示的情况办理；否则就是数字
S	符号

对“**b**”和“**f**”选项来说，当用户给出的编辑缓冲区或文件不存在时，Emacs 会报告出错。**interactive**函数还有一个很有用的选项是“**r**”，我们稍后再对它进行介绍。我们还将在第十四章介绍一个“**e**”选项。还有很多其他的选项字母，具体细节请查阅**interactive**函数的有关文档。子字符串的剩余部分是显示在辅助输入区里的实际内容。

用**interactive**来填充函数参数的具体方法是比较复杂的，我们最好还是用一个例子来做一下说明。函数 **goto-percent**（我们马上就会看到它）里有一个简单的例子，它包含的语句是：

```
(interactive 'nPercent: ")
```

提示字符串里的字母“**n**”告诉Emacs提示用户输入一个整数；字符串“`Percent:`”是实际出现在辅助输入区里的提示文字。

再来看一个稍微复杂点的例子。假设我们想写出一个自己的**replace-string**命令。下面是我们写的用户提示部分：

```
(defun replace-string (from to)
  (interactive "sReplace string:  \nsReplace string %s with: ")
  ...)
```

这个提示字符串是由两个子字符串组成的，一个是“sReplace string:”，另一个是“sReplace string %s with:”，两个子字符串之间用一个**LINEFEED**字符隔开。两个子字符串的第一个字母都是“s”，这表示预期用户输入字符串；“%s”是一个格式操作符（参见前面介绍**message**函数的有关内容），Emacs会用用户在第一条提示下输入的响应替换它。

当这个命令被调用的时候，首先出现在辅助输入区里的是第一条提示信息“Replace string:（置换字符串）”。假设用户输入“fred”作为响应。当用户按下回车键之后，第二条提示信息“Replace fred with:”将出现在辅助输入区里。用户接着输入替换字符串并再次按下回车键。

用户输入的两个字符串将（按其输入的先后顺序）被分别用做函数参数“**from**”和“**to**”的值，整个命令将继续执行直到结束。这就是说，**interactive**是按照提示字符串中子字符串的先后顺序来为函数参数提供值的。

interactive不仅能够作为独立的函数来使用，我们还可以从其他LISP代码里对这个函数进行调用；在这种场合里，对它进行调用的那个函数必须负责为它准备好全体参数的值。请看下面的例子。如果我们想在另外一个LISP函数里调用刚才编写的replace-string命令，以便把某个文件里出现“Bill”的地方都替换为“Deb”，就可以用下面这条语句来实现：

```
(replace-string "Bill" "Deb")
```

在这个例子里，因为函数并不是在交互方式下被调用的，所以**interactive**语句没有任何作用；参数“**from**”被设置为字符串“Bill”，参数“**to**”被设置为字符串“Deb”。

再回到我们的**count-words-buffer**命令上来：它本身没有参数，所以它里面的**interactive**命令也就不需要有提示字符串。最后，我们还想给此命令增加一个“文档字符串”（*documentation string*，或者简写为“*doc string*”），这是用户发出**describe-function**（“**C-h f**”组合键）等在线求助命令时将会看到的帮助信息。文档字符串是正常的LISP字符串；它们是可选的，其长度可以是任意多行。但传统上，它的第一行应该是用来对命令功能进行介绍的一个完整而又精练的句子。需要提醒大家注意的是：字符串里的双引号（"）前面必须加上一个反斜线字符（\）作为前缀。

在综合考虑了以上介绍的各种因素之后，我们最终得到的函数是下面这样的：

```
(defun count-words-buffer ()
  "Count the number of words in the current buffer;
print a message in the minibuffer with the result."
  (interactive)
  (save-excursion
    (let ((count 0))
      (goto-char (point-min))
      (while (< (point) (point-max))
        (forward-word 1)
        (setq count (1+ count)))
  (message "buffer contains %d words." count))))
```

LISP 语言的基础函数

在介绍完自行编写工作函数的方法之后，我们再来看看LISP语言的基础函数。它们是你用来打造自己的工作函数的建筑材料。正如我们前面讲的那样，LISP在其他程序设计语言使用操作符（用于算术运算、比较运算和逻辑运算）的地方使用的是函数。表 13-2 列出了一些相当于其他程序设计语言中各种操作符的 LISP 基础函数。

表 13-2：LISP 语言的基础函数

算术运算	+、−、*、/（加、减、乘、除） %（求余数） 1+（递增） 1−（递减） max（最大值）、min（最小值）
比较运算	>、<、>=、<=（大于、小于、不小于、不大于） /=（不等于） =（等于，用于数字和字符） equal（等于，用于字符串和其他复杂的数据对象）
逻辑运算	and（与）、or（或）、not（非）

除“**1+**”、“**1−**”和“%”以外的其他算术函数都可以有任意多个参数；逻辑函数“**and**”和“**or**”也是如此。算术函数只有在其参数中至少有一个是浮点数的时候才会返回浮点值；也就是说，函数**(/ 7.0 2)**的返回值是 3.5，而**(/ 7 2)**的返回值是 3。注意整数除法将舍去余数。

有些工作用函数来做未免效率不高，而且满纸全都是函数的程序也很难看。可话又说回来，LISP语言的主要优点之一就是它的核心代码少而精，高效率的解释执行很容易实现。而且，有 Emacs 的各种 LISP 模式作为程序设计支持工具，语法也不会是一个多么严重的问题。

语句块

我们见过用 **let** 函数定义的语句块，也见过 **while** 结构里的语句块。LISP 语言中还有其他一些能够定义语句块的构造：**progn** 和 **let** 的其他形式。

progn 是最基础的，它的语法定义是：

```
(progn
  statement-block)
```

progn 是个让一个语句块看起来就像是一条语句的简单方法，它的作用与 Pascal 语言中的 **begin** 和 **end** 或者 C 语言中的花括号有些相似。**progn** 的返回值就是语句块中最后一条语句的返回值。**progn** 特别适合用在像 **if** 这样的控制结构里（参见下面的内容），因为这些控制结构不像 **while** 那样允许使用语句块。

let 函数还有其他一些变体形式。其中最简单的是：

```
(let (var1 var2 ...)
  statement-block)
```

上面这个语法定义没有使用“**(var value)**”这样的列表项，它只有一个由变量名组成的列表。和 **let** 函数的其他形式一样，用这种方法定义的变量也是一些局部变量，只能在这个语句块里访问。因为没有把它们初始化为给定的值，所以这些变量都被初始化为“**nil**”。在实际工作中，完全可以把这两种形式的 **let** 函数混在一起使用，比如像下面这样：

```
(let (var1 (var2 value2) var3 ...)
  statement-block)
```

在我们见到的第一种 **let** 格式里，局部变量的初始值可以是函数调用（记住，函数都有返回值）。在开始对任何一个变量进行赋值之前，必须先把这些函数的返回值全都求出来。但有些场合需要用某些局部变量的值，来计算另外一些局部变量的值，这就需要用到 **let** 函数的最后一种形式 —— **let***。

我们来看一个例子。假设我们想编写一个**goto-percent**函数，它的作用是能够前进到用百分比表示的文本在编辑缓冲区里的某个位置上。下面是这个函数的一种写法：

```
(defun goto-percent (pct)
  (interactive "nGoto percent: ")
  (let* ((size (point-max))
         (charpos (/ (* size pct) 100)))
    (goto-char charpos)))
```

我们在前面已经讲过，**interactive**函数是用来提示用户输入参数值的；在上面这个例子里，它将提示用户为参数**pct**输入一个整数值。接下来，**let***函数先把**size**初始化为编辑缓冲区里的字符总数，然后计算出一个字符位置**charpos**，让**charpos**、**size**和**pct**正好是一个百分比的关系。最后，我们用**goto-char**函数把光标移动到当前窗口里的指定字符位置。

这段代码最重要的一点是：如果我们用**let**来代替**let***，在需要计算**charpos**的值时，**size**的值还没准备好。**let***也可以用在**(var1 var2 . . .)**结构里，就像**let**一样。

下面是**goto-percent**函数更有效率的一种写法：

```
(defun goto-percent (pct)
  (interactive "nPercent: ")
  (goto-char (/ (* pct (point-max)) 100)))
```

控制结构

我们已经见过**while**函数用做控制结构的情况——与其他程序设计语言里类似语句的情况差不多。LISP语言还有另外两种重要的控制结构，它们是**if**和**cond**。

if函数的语法定义是：

```
(if condition
    true-case
  false-block )
```

它的执行情况是这样的：先对condition进行求值，如果它不是“**nil**”，就将执行**true-case**部分；如果它是“**nil**”，就将执行**false-block**部分。**true-case**必须是单独一条语句，而**false-block**则是一个语句块；**false-block**可以省略。

我们来回忆一下第十一章里对终端的特殊键进行定制的例子。如果你使用着一个以

上的终端，就应该给“*.emacs*”文件增加一些代码，让它能够判断出你当前正使用着哪一个终端并以此为根据来设置相应的按键绑定。

这段代码同时还演示了其他几个重要的 Emacs LISP 功能的用法。为简单起见，我们的函数将以 DEC VT100 / 200 系列的 ANSI 终端和 X 窗口系统中的 **xterm** 终端仿真器等3大类终端上的方向键为目标进行设计。表13-3列出了各种方向键所输出的字符代码。

表 13-3: ANSI 方向键输出的字符代码

箭头键	字符代码
上	**ESC O A**
下	**ESC O B**
右	**ESC O C**
左	**ESC O D**

下面是用来设置键位映射图和按键绑定的有关代码。代码中的分号是注释标记；分号之后直到行尾的一切文本都是注释内容。如下所示：

```
(setq termtype (getenv "TERM"))

(if (or (equal termtype "vt200")          ;这 3 种终端都有 ANSI 小键盘
        (equal termtype "vt100")
        (equal termtype "xterm"))

    (progn
      (setq term-file-prefix nil)         ;覆盖默位的键位映射
      (send-string-to-terminal "\e=")     ;强制应用小键盘
      (setq SS3-map (make-keymap))        ;建立小键盘的键位映射
      (define-key global-map "\eO" SS3-map)

      (define-key SS3-map "A" 'previous-line)   ; 上箭
      (define-key SS3-map "B" 'next-line)       ; 下箭
      (define-key SS3-map "C" 'forward-char)    ; 右箭
      (define-key SS3-map "D" 'backward-char))  ; 左箭

  (if (equal termtype other-term-type...)
      (progn
        code for binding keys on other terminal)))
```

第一条语句使用了内部函数 **getenv**，它是一个面向 UNIX 操作系统的接口，作用是把某给定环境变量的值（参见第十一章）以字符串的形式返回来。上面例子里使用

的 UNIX 环境变量 **TERM** 描述的是正在使用的终端的类型（如果你的终端设置正确……）。

接下来，**if** 子句的条件判断部分检查 **termtype** 是否是 3 个给定的终端类型之一。如果是，就将执行 **progn** 语句块里的代码。记住：**if** 函数的“真”部分只能有一条语句，所以我们在这里用 **progn** 定义了一个语句块。这个语句块中的前 4 条语句是我们在第十一章里见过的；而语句块里的其他语句则把方向键绑定到相应的光标移动命令上。

第二个 **if** 语句实际上是第一个 **if** 语句的“false（假）”子句。就如同 Pascal 或 c 语言中的 `else if`。在这个 **if** 子句中可以检测其他终端类型并在 **progn** 语句块中为该类型绑定按键。

更通用的条件控制结构是 **cond** 函数，它的语法格式如下所示：

```
(cond
 (condition1
  statement-block1)
 (condition2
  statement-block2)
 ...)
```

C 和 Pascal 程序员可以把 cond 函数看做是由一连串“*if then else if then else if...*”组成的语法结构；也可以把它看做是一组“case”或“switch”语句。**cond** 函数中的条件将按其出现的先后顺序依次被求值，如果某个条件的求值结果不是“**nil**”，就将执行相应的语句块；而 **cond** 函数也将在执行这个语句块之后结束运行，它的返回值就是该语句块中最后一个语句的值（注 4）。

我们可以用 **cond** 函数重写前面的 **if** 函数示例。假设我们还是以刚才那 3 类不同的终端类型为目标，最后一个条件保留给缺省终端使用：

```
(setq termtype (getenv "TERM"))

(cond ((or (equal termtype "vt200")              ;这 3 种终端都有 ANSI 小键盘
           (equal termtype "vt100")
```

注 4： **cond** 函数里的语句块其实是可以省略的；有些程序员喜欢省略最后一个语句块，把最后一个“条件”留做它前面的其他条件计算值都是“**nil**”时的“例外”子句。如果省略语句块，则 **cond** 函数的返回值就简单地取为与之对应的条件的计算值。

```
        (equal termtype "xterm"))
(setq term-file-prefix nil)                ;覆盖默位的键位映射
(send-string-to-terminal "\e=")            ;强制应用小键盘
 (setq SS3-map (make-keymap))               ;建立小键盘的键位映射
 (define-key global-map "\eO" SS3-map)

 (define-key SS3-map "A" 'previous-line)   ;上箭
 (define-key SS3-map "B" 'next-line)       ;下箭
 (define-key SS3-map "C" 'forward-char)    ;右箭
 (define-key SS3-map "D" 'backward-char)) ;左箭

((equal termtype other-term-type ...)
 (code for binding keys on other terminal))

(t
 (code for binding keys of third terminal type)))
```

第 3 个条件表达式很简单，就是一个原子项“**t**”（真）；它的含义是：如果终端不属于前两大类，就认定它是第 3 种（缺省的）终端类型并执行相应的代码。

Emacs 的内部函数

许多现成的Emacs函数和一些自行开发的函数都涉及到对编辑缓冲区里的文本进行查找和替换之类的操作。这些函数在某些特定的编辑模式（比如 RMAIL 模式和上一章介绍的各种语言编辑模式）里非常有用。Emacs 有许多内部函数都与字符串和编辑缓冲区里的文本的处理工作有关；其中最值得一提的是那些利用Emacs的正则表达式功能的函数。对正则表达式的介绍请参考第三章。

我们先介绍一些与编辑缓冲区和字符串有关却又没有使用正则表达式的Emacs基本函数。然后，我们将在第三章内容的基础上对正则表达式做更深入的研究；讨论重点将集中在对LISP程序设计最有帮助的那些功能上。最后，我们将介绍一些对正则表达式本身进行操作的 Emacs 函数。

编辑缓冲区、文本和文本块

表 13-4 列出了一些与编辑缓冲区、文本和字符串有关的Emacs 基本函数，它们是一些通常只有LISP程序员才会用到的函数，所以没有被绑定到任何按键上。我们已经在刚才编写的 **count-words-buffer** 函数里见过它们当中的几个了。

表 13-4：与编辑缓冲区和文本有关的函数

函数名称	返回值或执行动作
point	光标的字符位置
mark	文本块标记的字符位置
point-min	最小字符位置（通常是 1）
point-max	最大字符位置（通常是编辑缓冲区的长度）
bolp	光标是否位于行首（取值为“t”或“nil”）
eolp	光标是否位于行尾
bobp	光标是否位于编辑缓冲区的开始
eobp	光标是否位于编辑缓冲区的末尾
insert	把任意个数的参数（字符串或字符）插入到编辑缓冲区光标位置之后
number-to-string	把一个数值参数转换为一个字符串
string-to-number	把一个字符串转换为一个数字（整数或浮点数）
char-to-string	把一个字符参数转换为一个字符串
concat	把任意个数的字符串合并到一起
substring	给定一个字符串及两个整数索引 *start* 和 *end*，返回从 *start* 指示的位置开始到 *end* 指示的位置前结束的子字符串。下标从 0 开始计算。比如说，函数“(substring “appropriate” 2 5)”将返回子字符串“pro”
aref	数组索引函数，它可以用来从字符串里取出单个的字符；它的输入参数是一个整数，从函数返回的是一个以整数表示的字符，返回值（在大多数机器上）使用的是 ASCII 码。比如说，函数“(aref “appropriate” 3)”将返回数字“114”，即字母“r”的 ASCII 码

很多没有出现在上面这个表格里的函数 —— 包括一些大家都很熟悉的用户级命令在内 —— 也能对编辑缓冲区和文本进行处理。有几个比较常用的 Emacs 函数是以文本块（*region*，即编辑缓冲区中的文本区域）为操作对象的。在使用 Emacs 做文字编辑工作时，要用先设置文本块标记、再移动光标的方法来选取文本块。可如果想用面向文本块的函数（比如 **kill-region**、**indent-region** 和 **shell-command-on-region** 等等。总之，任何名字里带有“*region*”字样的函数）来编写 Emacs LISP 代码，就应该采用一种更灵活的方法来选取文本块 —— 这些函数也确实有这种灵活性。面向文本块的函数一般都有两个整数参数，它们的作用是给出文本块前后边界的字符位置；如果函数是在交互方式下调用的，这两个参数的缺省值将是 *point* 和 mark 函数的返回值。

很明显，让Emacs自动把point和mark函数的返回值设置为这些参数的缺省值，要比由用户自己动手来输入这些值方便得多（也更符合人们的习惯）。**interactive**函数的“**r**”选项使这种设想成为现实。我们来看一个例子。我们打算编写一个能够把文本块翻译为德文的**translate-region-into-German**函数，于是，写出下面这些代码：

```
(defun translate-region-into-German (start end)
  (interactive "r")
  ...
```

当这个函数以交互方式调用时，**interactive**的“**r**”选项将自动把参数**start**和**end**的值设置好；但如果从其他LISP代码里对它进行调用，就必须给出这两个参数。在LISP代码里调用这个函数最常见的方法是：

```
(translate-region-into-German (point) (mark))
```

事实上，并不是非得像这样来调用它。如果想用这个函数写一个用来把整个编辑缓冲区翻译为德文的**translate-buffer-into-German**函数，就只需给它加上一个下面这样的“包装”：

```
(defun translate-buffer-into-German ()
  (translate-region-into-German (point-min) (point-max)))
```

事实上，除非是确有必要，否则最好不要在LISP代码里直接使用point和mark函数；应该使用局部变量来选取文本块。写出的LISP函数不应该是Emacs编辑命令的堆砌——后者更适合用Emacs的宏（参见第十章）来实现。

正则表达式

正则表达式（简称regexp）提供了更强大的文本处理功能。虽然有许多Emacs用户不喜欢使用**replace-regexp**和**re-search-forward**等需要由他们给出正则表达式的命令，但编写LISP代码时却经常要大量地使用正则表达式。缺少了正则表达式，RMAIL、Dired以及各种语言编辑模式将是无法想像的。学习和掌握正则表达式的使用方法需要时间和耐心，但作为一个LISP程序员，你付出的努力是很值得的，因为你程序里的正则表达式只能由你本人来编写和调试（如果你是一位普通用户，使用的是用户级的Emacs编辑命令，就不需要也不能对命令中的正则表达式进行调试）。

我们将用一些查找和替换方面的例子来介绍正则表达式的各种功能；这些例子都是比较容易理解的，没有过多地纠缠在晦涩难懂的细节方面。在此之后，我们将介绍几个高级的LISP函数，它们能够用正则表达式完成更加复杂的查找和替换操作。在下面这几个需要使用查找和替换操作的情况里，简单的查找/替换命令或者根本无法完成，或者完成得不好：

1. 在用C语言开发代码，想把例程 *read* 和 *readfile* 的功能合并到一个名为get的新函数里去。需要把程序代码里凡是出现“read”和“readfile”的地方都替换为“get”。

2. 按第八章介绍的方法用Emacs的大纲模式撰写着一篇 **troff** 文档。在大纲模式下，文档中的段落标题是一些以一个或者多个星号（*）开始的文本行；可是并不想让这些星号也出现在文档的打印稿里。于是，决定自己写一个 **remove-outline-marks** 函数去掉文档里的这些星号，然后再用 **troff** 对文件进行排版。

3. 想把文档里的“*program*”、“*programs*”和“*program's*”分别替换为“*module*”、“*modules*”和“*module's*”，但不能让“*programming*”和“*programmer*”也类似地被替换为“*moduleming*”和“*modulemer*”。

4. 正在为以Ada语言重写的某个C语言软件编制文档。因为Ada编译器要求使用“*.a*”做为文件名后缀，所以需要把文档里的文件名都从“*filename.c*”改为“*filename.a*”。

5. 新安装的C++编译器产生的出错信息是德文。于是，想修改Emacs的 **compile** 程序包，好让它能够正确地对出错信息进行分析（参见第十二章结尾部分）。

我们马上就会介绍利用正则表达式来解决上面这些问题的方法，为了便于查阅，我们给这些例子都编了号。请大家注意，虽然这一章对正则表达式的讨论比第三章的内容更加深入，但并没有涉及到它的全部功能；那些没有在此介绍的功能或者与我们介绍的功能有所重叠，或者与本书的讨论范围没有直接的关系。更需要大家注意的是，这里探讨的正则表达式语法只适用于LISP字符串；我们很快就会看到，LISP字符串所使用的正则表达式语法与用户级Emacs命令（比如 **replace-regexp**）所使用的正则表达式语法两者之间有一个很重要的区别。

基本操作符

正则表达式最初只是计算机科学在理论方面的一个想法，但它如今已经深入到计算

领域的每一个角落。用来表示它们的语法可能会千差万别，但其核心都是一样的。某些正则表达式的表示法应该是大家都很熟悉的了，比如UNIX操作系统的shell用来匹配文件名的那些通配符。但Emacs采用的表示方法与UNIX的文件名通配符稍微有些区别；它与 **ed** 和 **vi** 等编辑器以及 **lex** 和 **grep** 等 UNIX 软件工具所使用的记号方法比较接近。我们先从与UNIX的shell通配符功能相近的Emacs正则表达式操作符开始讲起，这些操作符都列在表 13-5 中。

表 13-5: 正则表达式基本操作符

Emacs 操作符	UNIX shell 操作符	作用
.	?	匹配任意一个字符
.*	*	匹配任意一个字符串
[abc]	[abc]	匹配字母 a、b 或 c
[a-z]	[a-z]	匹配任意一个小写字母

比如，如果想匹配所有以“program”开头的文件名，在 UNIX shell 里得使用“program*”；在Emacs里就得使用“program.*”。如果想匹配所有以字母“a”到“e”开头的文件名，在UNIX里要使用“[a-e]*”或“[abcde]*”；在Emacs里就要用“[a-e].*”或“[abcde].*”。换句话说，方括号里的短划线字符（-）代表字符范围（注5）。我们过一会儿再来对字符范围和允许放在方括号里的字符集做进一步的探讨。

如果需要在正则表达式里放上一个本身被用做操作符的字符，就必须在它的前面加上两个连续的反斜线字符，比如“*”将匹配一个星号字符。为什么要使用两个反斜线字符呢？答案是这与Emacs LISP读取和解码字符串的操作方法有关。在把一个字符串读到一个LISP程序里的时候，Emacs会对用反斜线字符转义的字符进行解码，即把两个反斜线字符转换为一个单个的反斜线字符。如果字符串被用做一个正则表达式——即当它被传递到某个需要正则表达式作为参数的函数里的时候，那个函数会使用这个单个的反斜线字符作为正则表达式语法的一部分。请看下面这行LISP代码:

注 5: Emacs使用ASCII码（在大多数机器上）来确定字符范围，但你可别依赖这个事实；还是用可靠的东西——比如小写字符集、大写字符集或用来表示数字的“[0-9]”比较保险。要避免使用不具备移植性的表示方法，比如“[A-Z]”和包含标点符号的字符范围等。

```
(replace-regexp "fred\\*" "bob*")
```

LISP解释器会先把字符串“fred*”解码为“fred*”，然后再把它传递到**replace-regexp**命令。**replace-regexp**命令知道“fred*”表示的是“fred”后面加一个星号。注意，**replace-regexp**命令的第二个参数不是一个正则表达式，所以“bob*”中的星号不需要用反斜线字符进行转义。还要注意的是：如果把这条语句当做一个用户级命令来调用，就不需要输入两个反斜线字符。也就是说，只需输入“**ESC x replace-regexp RETURN**”，然后是“**fred***”和“**bob***”。Emacs采用另外一种方法来解码从辅助输入区里读入的字符串。

Emacs里的“*”正则表达式操作符与UNIX shell使用的“*”通配符含义是不一样的：它的含义是“星号（*）前面任何东西的零次或多次出现”。于是，因为“.”能匹配任意一个字符，“.*”就表示“任意字符的零次或多次出现”，即任意一个字符串。“*”前面可以是任何东西：比如，“read*”将匹配“rea”后面跟有零个或者多个字母“d”的情况；“file[0-9]*”将匹配“file”后面跟有零个或者多个数字的情况。

有两个操作符与“*”的关系非常密切。第一个是“+”，它将匹配“+”前面任何东西的一次或者多次出现。于是，“read+”将匹配“read”和“readddddd”，但不匹配“rea”；而“file[0-9]+”要求“file”后面至少要跟有一个数字。第二个是“?”，它的含义是“?”前面任何东西的零次或者一次出现（即使之成为可选项）。于是，“read?”将匹配“rea”和“read”；而“file[0-9]?”则要求“file”后面最多跟有一个数字。

在介绍其他操作符之前，我们先说说字符集和字符范围。首先，允许在一个字符集里指定一个以上的字符范围。因此，字符集“[A-Za-z]”代表的是字母表中的全体字符；这比不具备移植性的“[A-z]”要好得多。把字符范围和字符集中的字符的列表组合在一起也是允许的；比如，字符集“[A-Za-z]_”代表的是字母表中的全体字符加下划线字符，这正是允许用在C语言标识符名称里的全体字符。如果在某字符集第一个字符的前面加上字符“^”，它的作用就相当于“not（取反）”操作符；该字符集将匹配所有没有出现在“^”字符之后的那些字符。比如，字符集“^[A-Za-z]”将匹配全体非字母表字符。

如果字符“^”没有出现在字符集第一个字符的前面，它就没有特殊的含义；它只是

一个上箭头罢了。与此相对的是，如果短划线字符（-）出现在字符集的最开始，它就将失去特殊含义；右方括号字符（]）也遵守同样的规定。但我们不推荐把这些用做操作符的字符放在字符集里——这是一种投机取巧的做法；为安全起见，最好是把它们放在字符串里，而且要用两个反斜线字符对它们进行转义。给一个非特殊字符加上一个双反斜线前缀得到的通常还是那个字符——但最好别这么做！正则表达式的操作符还包括其他一些字母和标点符号，我们将在以后的内容里再介绍几个。有些字符在用双反斜线前缀进行转义后会变成操作符，我们也会在后面的内容里把它们介绍给大家。“^”字符如果被放在字符集的外面会有另外一种含义，我们马上就会看到。

归组操作符和二选一操作符

如果想让“*”、“+”或“?”对一个以上的字符进行操作，就需要使用“\\(”和“\\)”操作符对字符进行归组。注意，在这两个操作符（以及此后介绍的其他操作符）里，那两个反斜线字符是操作符的组成部分。（为了避免出现过多的“特殊”字符，除了“*”、“+”或“?”等正则表达式基本操作符之外，其他一切字符（包括反斜线字符在内）都必须遵守这条规定。）大家刚才也看到了，如果想让“*”、“+”或“?”作为字符出现在字符串里，就必须用两个反斜线字符对它们进行转义，这样才能使 Emacs 能够对它们正确地解码。如果某个基本操作符紧跟着出现在“\\)”的后面，它就将把“\\(”和“\\)”之间的内容做为一个整体来对待。比如，“\\(read\\)*”将匹配空字符串、“read”和“readread”等等；而“read\\(file\\)?”将匹配“read”或“readfile”。现在，我们可以开始着手解决这一节开始时列出来的“疑难杂症”了。下面这行 LISP 代码将解决“情况 1”：

```
(replace-regexp "read\\(file\\)?" "get")
```

二选一操作符“\\|”是一个“非此即彼”操作符；它的作用是匹配“\\|”前后两样事物中的一个。对于用归组操作符（“\\(”和“\\)”）归为一组的多个字符（即字符组）来说，“\\|”操作符对它们的处理与正则表达式的基本操作符不同。它并不挑剔字符组是否是由一个以上的字符所组成的子表达式，其“效力”将沿它自己的左、右方向伸展到极至，直到它到达正则表达式的开头、正则表达式的末尾、一个“\\(”、一个“\\)”或者另外一个“\\|”。请看下面这几个例子，它们应该能把问题说得更清楚：

- "read\\|get"将匹配"read"或"get"
- "readfile\\|read\\|get"将匹配"readfile"、"read"或"get"
- "\\(read\\|get\\)file"将匹配"readfile"或"getfile"

在第一个例子里，"\\|"的效力伸展到了正则表达式的开头和末尾。在第二个例子里，第一个"\\|"的效力向左伸展到了正则表达式的开头、向右伸展到了第二个"\\|"。在第三个例子里，"\\|"的效力向左、向右分别伸展到了"\\("和"\\)"。

与上下文有关的正则表达式操作符

正则表达式另一组重要的操作符与字符串的上下文有关，所谓"上下文(context)"指的是字符串前后的文本。我们在第三章讨论递增查找操作时介绍过一个**word-search**命令，它在执行时就需要依赖于特殊的上下文设置；**word-search**命令所依赖的上下文就是出现在字符串前、后的单词分隔字符，比如空格和标点符号等。

正则表达式中最简单的上下文操作符是"^"和"$"，它们也是正则表达式的基本操作符，使用时必须分别放置在正则表达式的开头或者末尾。"^"操作符的作用是让正则表达式的后续部分只匹配它出现在行首的情况；而"$"操作符则是让正则表达式的前导部分只匹配它出现在行尾的情况。在"情况2"里，我们需要让函数匹配一个或者多个星号出现在行首的情况，下面这段代码将解决这一问题：

```
(defun remove-outline-marks ()
  "Remove section header marks created in outline-mode."
  (interactive)
  (replace-regexp "^ \\*+" ""))
```

这个函数会找出那些以一个或者多个星号开头的文本行("*"表示匹配"*"字符本身，"+"表示"一个或一个以上")，然后它会把那些星号替换为空字符串("")——也就是删除它们。

请注意，如果想匹配的字符串可能延伸到两个或者更多个文本行，千万不要把"^"和"$"放在正则表达式的半中间，这是不允许的。正确的做法是在正则表达式里的适当位置加入"\n"(作用是匹配**LINEFEED**字符)来匹配这样的字符串。另一个有类似功效的字符是用来匹配制表符**TAB**的"\t"。当"^"和"$"出现在对字符串(而不是对编辑缓冲区中的文本行)进行查找的正则表达式里时，它们将分别匹

配字符串的开头和结尾；本章后面将要介绍的**string-match**函数能够对字符串进行正则表达式查找。

下面这个复杂的正则表达式是一个实实在在的例子，它里面用到了我们此前介绍的各种操作符：**sentence-end** —— Emacs 用来识别句子结尾的变量，诸如**forward-sentence**（“**ESC e**”组合键）等，以句子为单位的光标移动命令就是靠这个变量来完成操作的。它的值是：

```
"[.?!][]\"')}]*\\($\\|\t\\|  \\)[ \t\n]*"
```

让我们逐段分析一下这个正则表达式。第一个字符集即“[.?!]”，能匹配一个英文句号、一个问号或者一个惊叹号（英文句号字符“.”和问号字符“?”也是正则表达式的操作符，但它们在字符集里没有特殊含义）。接下来是“[]\"')}]*”，这个字符集里包含有右方括号、双引号、单引号、右圆括号和右花括号。紧跟在这个字符集后面的星号(*)表示匹配这个字符集中任何一个字符的一次或者多次出现的情况。截止到这里，这个正则表达式将匹配一个表示句子结束的标点符号，以及一个或者多个出现在句尾标点符号后面的引号、括号或者花括号。再往后是一个用来进行多选一匹配的字符组“\\($\\|t\\| \\)”，它将匹配三样东西：“$”（行尾符）、制表符 **TAB** 或两个空格中的任何一个。最后，“[\t\n]*”匹配零个或者多个空格、制表符或换行符 **LINEFEED**。也就是说，句子结尾字符后面还可以有行尾符、或者任意组合的空格（至少两个）、制表符和换行符 **LINEFEED**。

上下文操作符并非只有“^”和“$”两个；我们再介绍两个可以利用正则表达式来进行单词查找的上下文操作符。操作符“\\<”和“\\>”分别匹配单词的开始和结尾。有了这两个操作符，我们就可以把“情况 3”里的问题解决一大半了。正则表达式“\\<program\\>”将匹配“program”，但不匹配“programmer”或“programming”（它也不会匹配上“microprogram”）。问题差不多解决了；但它还不能匹配上“program's”或“programs”。为了把这些情况都匹配上，我们需要一个更复杂点的正则表达式：

```
\\<program\\('s\\|s\\)?\\>
```

（出现在正则表达式最后面的“\\>”可以省略。）这个正则表达式的含义是“一个以 program 开始，后面可能还跟有一个's 或 s 的单词”。这个正则表达式已经把“情况 3”在匹配正确单词方面的问题都解决了。

截取匹配目标的部分内容

离成功只差一步了："program"后面可能带有"s"或"'s"，可我们的正则表达式现在还无法做到在不触及"s"或"'s"的同时把"program"修改为"module"。这就引出了我们准备在这一章里探讨的最后一个正则表达式功能：截取匹配目标（即匹配操作找到的字符串）的部分内容以便对之做进一步处理。我们刚才得到的正则表达式本身是正确的，但它只是**replace-regexp**命令的查找字符串而已。至于替换字符串，答案是"module\\1"；换句话说，完整的LISP代码应该是：

```
(replace-regexp "\\<program\\('s\\|s\\)?\\>' "module\\1")
```

从最终效果上讲，"\\1"的含义是"只对匹配目标中与"\\("和"\\)"之间的子表达式相匹配的部分内容进行替换"。这是替换操作中惟一允许使用的与正则表达式有关的操作符。就"情况3"而言，它的含义是：如果匹配目标是"program's"，就先把替换字符串中的"\\1"替换为"'s"，然后再对匹配目标进行替换；如果匹配目标是"programs"，就得先把替换字符串中的"\\1"替换为"s"；如果匹配目标只是"program"，就先把替换字符串中的"\\1"替换为空字符（即去掉替换字符串中的"\\1"）。最终的结果是皆大欢喜："program"将被替换为"module"、"programs"将被替换为"modules"，"program's"将被替换为"module's"。

"情况4"也需要用这个功能来解决。为了匹配"*filename.c*"并把它替换为"*filename.a*"，我们需要使用下面的LISP代码：

```
(replace-regexp "\\(filename\\)\\.c" "\\1.a")
```

注意，"\\."代表的是一个"真正"的英文句号(.)；而给查找字符串里的"filename"两端加上"\\("和"\\)"的惟一目的是为了让替换字符串里的"\\1"能够截取出句号来。

事实上，"\\1"操作符只是正则表达式一种更强大的功能的一个特例。一般说来，如果用"\\("和"\\)"对正则表达式里的某一小段进行了归组，那么，与这个子表达式相匹配的字符串就会被保存起来；接着，当设计替换字符串的时候，就可以用"*N*"来截取那些被保存起来的子字符串，其中"*N*"是被归组的子字符串的编号，字符组从左向右顺序编号，编号从1开始。被归组的表达式可以嵌套；与之对应的"*N*"编号将按"\\("出现的先后顺序从左向右进行赋值。

完整地使用这项功能的LISP代码，通常都包含着相当复杂的正则表达式。Emacs自

身的LISP代码里就有一个绝佳的例子，它就是（我们在上一章快结束时介绍的）**compile** 程序包在分析编译器给出的出错信息时，所使用的正则表达式列表 **compilation-error-regexp-alist**。我们把这个列表变量从Emacs的源代码里节选出来列在下面，有兴趣的读者可以好好研究一下：

```
(
 ;; 4.3BSD grep, cc, lint pass 1:
 ;;   /usr/src/foo/foo.c(8): warning: w may be used before set
 ;; or GNU utilities:
 ;;  foo.c:8: error message
 ;; or HP-UX 7.0 fc:
 ;;  foo.f          :16    some horrible error message
 ("\n\\ ([^:( \t\n]+\\)[:(][ \t]*\\([0-9]+\\)[:) \t]" 1 2)

 ;; 4.3BSD lint pass 2
 ;;  strcmp: variable # of args. llib-lc(359)   :: /usr/src/foo/foo.c(8)
 ("[ \t:]\\([^:( \t\n]+\\){:(](+[ \t]*\\([0-9]+\\))[:) ]*$" 1 2)

 ;; 4.3BSD lint pass 3
 ;;  bloofle defined( /users/wolfgang/foo.c(4) ), but never used
 ;; This used to be
 ("([ \t]*\\([^:( \t\n]+\\)[:(][ \t]*\\([0-9]+\\))" 1 2)

 ;; Ultrix 3.0 f77:
 ;;  Error on line 3 of t.f: Execution error unclassifiable statement
 ;; Unknown who does this:
 ;;  Line 45 of "foo.c": bloofel undefined
 ("\n\\(Error on \\)?[Ll]ine[ \t]+\\([0-9]+\\)[ \t]+\
of[ \t]+\"?\([^\"\n]+\\)\"?:" 3 2)

 ;; Apollo cc, 4.3BSD fc:
 ;;  "foo.f", line 3: Error: syntax error near end of statement
 ;; IBM RS6000:
 ;;  "vvouch.c", line 19.5: 1506-046 (S) Syntax error.
 ;; Unknown compiler:
 ;;  File "foobar.ml" , lines 5-8, characters 20-155: blah blah
 ("\"\\([^,\" \n\t]+\\)\", lines? \\([0-9]+\\)[:.,-]" 1 2)

 ;; MIPS RISC CC - the one distributed with Ultrix:
 ;;  ccom: Error: foo.c, line 2: syntax error
 ("rror: \\([^,\" \n\t]+\\), line \\([0-9]+\\):" 1 2)

 ;; IBM AIX PS/2 C version 1.1:
 ;;  ****** Error number 140 in line 8 of file errors.c ******
 ("in line \\([0-9]+\\) of file \\([^ \n]+[^.\\n]\\)\\.? " 2 1)

 )
```

这个列表里的元素都是由三个部分组成的：一个正则表达式和两个数字。正则表达

式的作用是按特定编译器所使用的格式对出错信息进行匹配，它被归组为几个子表达式。第一个数字告诉Emacs出错信息里的文件名被保存在第几个子表达式里；第二个数字则用来指明行号保存在第几个子表达式里。

比如说，列表的最后一项包含着正则表达式：

```
"in line \\([0-9]+\\) of file \\([^ \n]+[^. \n]\\)\\.?"
```

然后是数字2和1，这就意味着被归为第二组的子表达式包含着文件名，被归为第一组的子表达式包含着行号。如果你使用的是这个编译器（运行着IBM版UNIX操作系统的Intel x86个人电脑上的C语言编译器），你看到的出错信息就将是下面这样的：

```
****** Error number 140 in line 8 of file errors.c ******
```

正则表达式对出错信息中“in line”之前的内容不做处理。接下来，它找到了与第一个子表达式“[0-9]+”（一个或更多个数字）相匹配的行号“8”，以及与第二个子表达式“[^ \n]+[^. \n]”（一个或更多个不是空格也不是换行符的字符，后面跟着一个不是英文句号，不是空格，也不是换行符的单个字符）相匹配的文件名“errors.c”。

现在可以着手解决“情况5”中的问题了。为了使Emacs能够对新C++编译器给出的德文出错信息进行分析，我们决定给compilation-error-regexp-alist列表增加一个元素。新编译器给出的出错信息采用的是下面这样的格式：

```
Fehler auf Zeile linenum in filename: text of error message
```

下面是我们准备添加到 **compilation-error-regexp-alist** 里去的元素：

```
("Fehler auf Zeile \\([0-9]+\\)in\\([^:\t]+\\:" 21)
```

注意观察编译器报告的出错信息，我们发现它和刚才那个例子一样：被归为第二组的子表达式将负责匹配文件名，被归为第一组的子表达式负责匹配行号。

为了把这个元素添加到变量**compilation-error-regexp-alist**里，我们需要把下面这些语句添加到“*.emacs*”文件里：

```
(setq compilation-error-regexp-alist
  (cons '("Fehler auf Zeile \\([0-9]+\\) in \\([^: \t]+\\):" 2 1)
    compilation-error-regexp-alist))
```

这个例子与我们往变量**auto-mode-alist**里添加一种新的语言编辑模式的支持（参见上一章内容）有很多相似之处。

正则表达式操作符汇总

表13-6为我们对正则表达式操作符的讨论画上了句号，我们在这一章讲过的所有操作符都列在这个表格里。

表 13-6: 正则表达式操作符速查表

操作符	作用
.	匹配任意一个字符
*	匹配其前面的字符或字符组出现零次或更多次的情况
+	匹配其前面的字符或字符组出现一次或更多次的情况
?	匹配其前面的字符或字符组出现零次或一次的情况
[. . .]	字符的集合，参见下面的说明
\\(	开始一个字符组
\\)	结束一个字符组
\\\|	匹配“\\\|”前、后的子表达式
^	如果出现在正则表达式的开始，匹配文本行首或字符串的开始
$	如果出现在正则表达式的末尾，匹配文本行尾或字符串的末尾
\n	在正则表达式里匹配换行符 **LINEFEED**
\t	在正则表达式里匹配制表符 **TAB**
\\<	匹配单词的开始
\\>	匹配单词的结尾
下面的操作符出现在字符集里时的含义:	
^	如果出现在字符集的开始，则该字符集将被用做不匹配字符集
-	用来设定字符范围
下面的操作符出现在 *regexp* 替换字符串里时的含义:	
\\N	替换第 N 个“\\(”和“\\)”之间的部分匹配，从左归组操作符“\\(”向右编号，编号从 1 开始

最后，如果用两个反斜线字符对以下字符进行转义，它们将转变为正则表达式的操

作符（我们没有对这些操作符进行介绍）：“b”、“B”、“w”、“W”、“s”、“S”、“'”（单引号）和“`”（反引号）。请注意这些“反斜线字符转义陷阱”。

使用正则表达式的函数

使用正则表达式的用户级命令有 **re-search-forward**、**re-search-backward**、**replace-regexp**、**query-replace-regexp**、**isearch-forward-regexp**和**isearch-backward-regexp**，它们也都能用来编写LISP代码（虽然在LISP代码里使用递增查找是难以想像的事）。本章后面关于定制主编辑模式的内容里，有一个使用**re-search-forward**函数的例子。

还有一些函数虽然也使用正则表达式，但不能做为用户级命令来使用。这些函数里最常用的大概是**looking-at**。这个函数的输入参数是一个正则表达式，它的作用是：如果光标后面的文本匹配上该正则表达式，返回“**t**”（否则返回“**nil**”）；如果真的有一个匹配，它会截取“\\(”和“\\)”之间的片段并保存起来供今后使用，就像我们刚才见过的那样。**string-match**函数也有类似的行为：它需要两个参数，一个正则表达式和一个字符串。如果字符串的某个片段能够与正则表达式匹配，**string-match**函数将返回匹配片段的起始位置在该字符串里的索引；如果没有匹配，则返回“**nil**”。

match-beginning和**match-end**函数可以用来检索被保存起来的字符串匹配片段。它们以正则表达式中的子表达式的编号（比如**replace-regexp**命令的替换字符串里的“*N*”）为输入参数，返回的是字符串匹配片段的起始点（**match-beginning**）或结束点（**match-end**）在编辑缓冲区里的字符位置。如果输入参数为数字“0”，就把用正则表达式匹配到的整个字符串的起始点，和结束点的字符位置做为返回值。

上面介绍的这些函数还需要另外两个函数的帮忙才能真正起作用：我们必须知道怎样才能把编辑缓冲区里的文本转换为一个字符串。没问题：**buffer-string**能够把整个编辑缓冲区做为一个字符串返回；**buffer-substring**利用两个整数参数（用来给出子字符串在编辑缓冲区里的起始位置和结束位置）把想截取的子字符串返回。

有了这些函数，我们就能用下面这样的LISP代码把正则表达式的第*N*个子表达式，在编辑缓冲区里匹配到的片段放在一个字符串里返回：

```
(buffer-substring (match-beginning  N)
                  (match-end N))
```

需要使用这种结构的地方实在是太多了，所以 Emacs 特意准备了一个内部函数 **match-string** 作为这个结构的快捷方式；**(match-string** *N*)的返回值与上面这条 LISP 语句的完全一样。

为了把这个函数的工作情况讲清楚，我们一起来看一个例子。接着刚才编写用来分析编译器出错信息的 LISP 代码的例子往下说。你的代码依次使用 **compilation-error-regexp-alist**列表的每一个元素，来检查编辑缓冲区中的文本是否与其中的某条正则表达式匹配。如果出现了匹配，你的代码将把文件名和行号提取出来，然后打开相应的文件，再到达指定的行号。

遍历列表中每一个元素的代码超出了我们目前的学习范围，但不要紧。这个例程的基本框架是下面这样的：

```
for each element in compilation-error-regexp-alist
  (let ((regexp the regexp in the element)
        (file-subexp the number of the filename subexpression)
        (line-subexp the number of the line number subexpression))
    (if (looking-at regexp)
        (let ((filename (match-string file-subexp))
              (linenum (match-string line-subexp)))
          (find-file-other-window filename)
          (goto-line linenum))
      (otherwise, try the next element in the list)))
```

第二条 **let** 语句通过编号为 **file-subexp** 的子表达式，把文件名从出错信息所在的编辑缓冲区里提取出来；然后再用同样的方法根据编号为**line-subexp**的子表达式提取出行号（然后把行号从字符串转换为一个数字）。接下来，这段代码打开相应的文件（在另外一个窗口里，不是出错信息编辑缓冲区所在的窗口）并到达出现错误的行号。

本章后半部分内容里用来实现计算器模式的代码，包含另外几个使用 **looking-at**、**match-beginning** 和 **match-end** 函数的例子。

找出其他的内部函数

Emacs 包含数以百计的内部函数，它们都可以用在编写的 LISP 代码当中。要想找出哪个函数最适合用来完成某项工作也不是很困难。

需要弄清楚的第一件事是：最常用的函数差不多都能通过键盘命令来访问。如果想使用某个函数，可以通过“**C-h k**”（命令名是**describe-key**）命令查出其函数名（参见第十六章）。用这个方法可以查到待查命令的完整文档；与此相对的是“**C-h c**”（命令名是**describe-key-briefly**）命令，它只给出待查命令的名字。请注意，在某些场合，一些常见的键盘命令用做LISP函数时需要带参数。**forward-word**就是一个例子；为了得到与按下“**ESC f**”组合键同样的效果，必须使用“**(forward-word 1)**”。

为某种操作寻找适用函数的另一个强大工具是**command-apropos**（“**C-h a**”组合键）帮助函数。这个帮助函数能够根据给定的正则表达式，把所有与之匹配的命令都查出来，然后把这些命令的按键绑定（如果有）和文档显示在一个“help”窗口里。它特别适合用来寻找一个能够完成某项“基本”工作的命令。例如，如果想知道哪些命令是对单词进行操作的，请按下“**C-h a**”组合键、再输入单词“*word*”，几十个与单词有关的编辑命令和它们的文档就会显示出来。

但**command-apropos**命令只能查出用做键盘命令的函数的资料。**apropos**命令的功能更强，但它没有被绑定为一个帮助热键（只能用输入“**ESC x apropos RETURN**”的方法来使用它）。给定一个正则表达式，**apropos**将把与之匹配的各种函数、变量以及其他符号的资料都显示出来。但我们也必须提醒大家：如果让**apropos**查找的是一个比较基本的概念，那么它可能会执行很长的时间，而且会生成一个非常长的清单。

查找能力最强的命令是**super-apropos**，它也没有被绑定到任何按键上。**super-apropos**命令不仅具备**apropos**命令所具备的功能，还能对函数、变量或其他符号的文档字符串（documentation strings）进行搜索，把与给定正则表达式匹配的东西都找出来。当然，如果不注意节制，**super-apropos**命令的执行时间要比**apropos**命令更长（而且会产生长度令人难以置信的输出）。

如果需要使用**apropos**命令来查找关于函数的资料，千万要把选好关键字。不过，需要**apropos**出马的场合并不是很多。这是因为如果想找的函数是比较常用或比较基本的，那么多半Emacs已经内置，因而也就用不着使用**apropos**命令了。

即使找到了感兴趣的函数，**apropos**或**super-apropos**命令给出的资料也有可能不足以让你彻底了解该函数的功能、参数情况、使用方法或者其他事项。此时，最好

的解决方法就是直接研读 Emacs 的 LISP 源代码，从中找出那个函数的使用示例。Emacs的LISP源代码一般保存在各位系统中一个名为“*emacs-source/lisp*”的子目录里，其中的“*emacs-source*”是系统上Emacs源代码的根目录名称。先用**grep**或其他查找工具查出需要的函数示例保存在哪个文件里，再打开那个文件看看函数周围的上下文情况。虽说 Emacs 的内部 LISP 源代码中的文档注释并不是很充足，但它提供的函数用法示例还是很有帮助意义的 —— 甚至能对大家自己编写的函数有所启发。

以上介绍的各种 Emacs LISP 基本概念和知识已经足以让大家编写出很多有用的 Emacs命令了。在我们分析过的函数示例中，既有LISP语言本身的基础函数，也有 Emacs的内部函数。各位已经具备了运用在这一章里学习的概念和技巧解决其他问题的能力。换句话说，各位已经踏上了成为一名优秀EmacS LISP程序员的道路。为了检查自己的学习情况，希望大家能够在**count-words-buffer**函数代码的基础上写出下面这几个函数：

count-lines-buffer	统计出编辑缓冲区里有多少个文本行
count-words-region	统计出文本块里有多少个单词
what-line	给出光标当前位置的行号

主编辑模式程序设计实例

对 Emacs LISP 程序设计比较熟悉之后，你可能发现自己想用一种主编辑模式让 Emacs做一点“额外工作”。在以前各章里，我们对文本输入、文字排版和程序设计语言等工作所使用的主编辑模式进行了探讨。这些主编辑模式大都相当复杂，普通程序员很难写出这样的程序来。因此，我们将提供一个简单的主编辑模式程序设计实例，希望大家能够从中学习到编写自己的主编辑模式所需要的概念。然后，我们将在下一节介绍，如何在不对用来实现现有主编辑模式的有关LISP代码做任何改动的前提下对它们进行定制。

我们准备开发的**calc-mode**主编辑模式，将实现一个与UNIX的**dc**（desk calculator，桌面计算器）命令功能近似的计算器。它是一个使用逆波兰表示法（一种基于堆栈的表示法）的计算器，这种表示法因惠普公司（Hewlett Packard）的推广而变得很流行。在介绍完主编辑模式的几个重要组件，和计算器模式几个有趣的功能之后，我们将把这个主编辑模式完整的 LISP 代码呈现给大家。

主编辑模式的组件

就一个主编辑模式而言，它是由各种把它集成到Emacs里的组件构成的。这些组件包括:

- 符号 —— 即用来实现这个主编辑模式的函数的名称。
- 名字 —— 这个主编辑模式准备显示在状态行括号里的名称。
- 局部键位图 —— 用来定义这个主编辑模式里各种命令的按键绑定。
- 变量和常数 —— 为了实现这个主编辑模式，LISP代码需要使用一些局部变量和局部常数。
- 专用编辑缓冲区 —— 这个主编辑模式下有特殊用途的编辑缓冲区。

我们来一个一个地解决这些问题。先设置主编辑模式的符号，方法是把用来实现这个主编辑模式的函数的名字赋值给全局变量**major-mode**，如下所示:

```
(setq major-mode 'calc-mode )
```

类似地，设置主编辑模式的名字，需要把相应的字符串赋值给全局变量**mode-name**，如下所示:

```
(setq mode-name "Calculator")
```

局部键位映射图需要用第十一章介绍的函数来定义。因为我们的计算器模式只需绑定一个按键（用来输入换行符**LINEFEED**的“**C-j**”组合键），所以我们决定使用**make-keymap**命令一个特殊形式 —— 如果需要绑定的按键数量不大，使用**make-sparse-keymap**命令将更有效。为了把一个键位映射图设置为某主编辑模式的局部键位映射图，我们需要调用函数**use-local-map**，如下所示:

```
(use-local-map calc-mode-map)
```

我们已经见过几种定义变量的方法，比如用**setq**给它们赋一个值，或者用**let**来定义函数中的局部变量等。更正规的变量定义方法是使用**defvar**函数，它允许程序员把变量的文档资料集成到“**C-h v**”组合键（命令名是**describe-variable**）等在线帮助功能里。这个函数的语法格式是:

```
(defvar varname initial-value "description of the variable")
```

defvar 函数还有一个变体叫做 **defconst**，可以用它来定义常数（即不会发生变化的值）。请看下面的例子：

```
(defconst calc-operator-regexp "[-+*/%]"
  "Regular expression for recognizing operators.")
```

这条语句定义了一个用来查找算术运算符的正则表达式。正像大家将会看到的那样，我们将给为计算器模式所定义的一切函数、变量和常数都加上一个“**calc-**”前缀。这是主编辑模式普遍采用的习惯做法；比如说，C++ 模式里的一切名字就都是以“**c++-**”开头的。这种做法是一个很好的习惯，它能帮助避免使用的名字与 Emacs 里数以千计的其他函数、变量或者诸如此类的东西发生重名冲突。

接下来的是把变量设置为主编辑模式的局部变量，这样，运行在某主编辑模式下的编辑缓冲区才会知道有哪些东西可用（注6）。设置局部变量的工作要用**make-local-variable** 函数来完成，如下所示：

```
(make-local-variable  calc-stack )
```

请注意，这里使用的是变量的名字而不是它的值；所以需要在变量名的前面加上一个单引号，使它成为一个符号。

最后，各种主编辑模式会用到一些特殊的编辑缓冲区，这些编辑缓冲区与任何文件都没有关联关系。比如，“**C-x C-b**”（命令名是 **list-buffers**）命令会创建一个名为“*Buffer List*”的编辑缓冲区。要想在新窗口里创建一个编辑缓冲区，需要使用 **pop-to-buffer** 函数，如下所示：

```
(pop-to-buffer "*Calc*")
```

pop-to-buffer函数还有几个很有用的变体。我们这里的主编辑模式程序设计实例没有使用它们，但它们在其他场合是很有用的：

switch-to-buffer　与第四章介绍的“**C-x b**”命令作用相同，也可以用来编写LISP代码，但需要给出一个编辑缓冲区名字做为参数。

注 6：　遗憾的是，因为这些变量是先进行定义，然后才被设置为主编辑模式的局部变量，所以与全局变量产生重名冲突的可能性是存在的。因此，给变量起名字时一定不要用全局变量已经使用的名字，这一点非常重要。避免出现重名冲突的应对策略之一就是让变量名以主编辑模式的名字开头。

set-buffer 只能用在LISP代码里，作用是将编辑缓冲区指定为文本编辑工作使用；最适合用来在LISP函数里创建一个临时性的“工作”编辑缓冲区。

列表：LISP语言的基本概念

采用逆波兰表示法的计算器使用一种名为“堆栈”的数据结构。可以把堆栈想像为咖啡店里摞在一起的许多碟子。向采用逆波兰表示法的计算器输入一个数字时，等于是把这个数字“压入”堆栈。输入一个运算符（比如加号或减号）时，等于是把堆栈最顶部的两个数字“弹出”堆栈，对它们进行加法或者减法操作，然后再把计算结果压入堆栈。

列表是LISP语言中的一个基本概念，它充分体现了堆栈结构本质特征。列表是把LISP与其他程序设计语言区分开来的主要概念。这种数据结构由两个部分组成：头元素和尾元素。纯粹出于历史沿革方面的原因，它们在LISP语言里又被分别称为**car**和**cdr**。可以把这两个术语看做是“列表里的第一样东西”和“列表里的其余东西”。如果给**car**和**cdr**函数提供一个列表做为参数，它们就将分别返回该列表的头元素和尾元素（注7）。人们经常用两个函数来构造列表。**cons**（构造）函数带两个输入参数，而这两个参数将分别被设置为列表的头元素和尾元素。**list**函数以多个数据元素为输入参数，用它们构造出一个列表。请看下面的例子：

```
(list 2 3 4 5 )
```

上面这个函数将构造出一个由从2到5的四个数字组成的列表，而

```
(cons 1 (list 2 3 4 5 ))
```

将构造出一个由从1到5的五个数字组成的列表。如果以这个列表为输入参数，**car**函数将返回“**1**”，**cdr**函数将返回列表“**(2 3 4 5)**”。

这些概念是非常重要的，因为堆栈（我们的计算器模式就使用了堆栈概念）可以很容易地被实现为列表。为了把值“**x**”压入堆栈**calc-stack**，我们只需写出下面这条语句：

注7： 有经验的LISP程序员会注意到Emacs LISP没有提供**cadr**和**cdar**等标准的列表操作函数。

```
(setq calc-stack (cons x calc-stack))
```

如果我们想取出堆栈最顶部的值，就可以用下面这条语句得到它:

```
(car calc-stack)
```

如果想把最顶部的值弹出堆栈，我们只需写出:

```
(setq calc-stack (cdr calc-stack))
```

请记住，列表的元素可以是包括其他列表在内的任何东西，这就是人们把列表称为一种“递归”数据结构的原因。事实上，对LISP语言里的任何一样东西来说，如果它不是一个原子项，就会是一个列表。这里面也包括函数，它可以看做一个由函数名、参数、准备求值的表达式等元素组成的列表。把函数看做是一种列表的观点很快就会显出它的优越性来。

程序设计示例：计算器模式

我们把用来实现计算器模式的完整的LISP代码列在这一小节的最后部分；大家在阅读下面的解释内容时可以随时去查阅它们。如果把那些代码都输入或扫描到机器里，就可以通过输入“**ESC x calc-mode RETURN**”命令的方法来使用这个计算器。光标将移动到“`*Calc*`”编辑缓冲区里。可以输入一行数字和运算符，按下“**C-j**”组合键的时候，计算器会对这一行进行求值。我们的计算器有 3 个特殊的命令，它们是:

= 显示堆栈最顶部的那个值

p 显示整个堆栈的内容

c 清除堆栈

数字之间用空格分隔，而运算符与数字之间则可以没有空格。请看下面的例子。如果输入:

```
4 17*6-=
```

再按下“**C-j**”组合键，我们的计算器就会开始计算(4*17)-6，并把结果 62 显示出来。

计算器模式的核心代码是函数**calc-eval**和**calc-next-token**。（从本节结尾部分里的源代码可以更清楚地看出这一点。）**calc-eval**函数被绑定到计算器模式下的“**C-j**”组合键上。计算工作将从“**C-j**”之前的行首开始进行，它将调用**calc-next-token**函数，依次取出该行中的每一个记号（数字、运算符或者命令字母）并对它进行求值。

calc-next-token函数使用一个**cond**结构，这个**cond**结构将根据正则表达式**calc-number-regexp**、**calc-operator-regexp**和**calc-command-regexp**来检查光标位置上是否有一个数字、运算符或者命令字母。根据得到匹配的是哪一个正则表达式，它将把变量**calc-proc-fun**设置为应该被调用执行的函数（**calc-push-number**、**calc-operate**或**calc-command**）的名字（符号），并且会把变量**tok**设置为正则表达式匹配的结果。

再说说**calc-eval**函数，它体现了把函数看做是一种列表的观念。**funcall**函数把LISP语言中代码和数据之间几乎没有区别的特点显露无余。我们可以用一个符号和一组表达式构造出一个列表，然后再把这个列表当做一个函数来进行求值。那个符号就是函数名，而那些表达式就是这个函数的参数；**funcall**正是这样做的。请看：

```
(funcall calc-proc-fun tok)
```

这条语句把**calc-proc-fun**的符号值看做将被调用的函数的名字，然后以**tok**做为其参数进行调用。接下来，这个函数将进行下面这3种操作中的一种：

- 如果记号是一个数字，**calc-push-number**将把这个数字压入堆栈。
- 如果记号是一个运算符，**calc-operate**将对堆栈最顶部的两个数字进行相应的操作（参见下面的解释）。
- 如果记号是一个命令字母，**calc-command**将执行相应的命令动作。

calc-operate函数把函数也是数据列表的概念又往前推进了一步，它把用户输入的记号直接转换为一个函数（对计算器来说，就是某个算术运算符）。这项工作是由**read**函数完成的，这个函数以一个字符串为输入参数并把它转换为一个符号。这样，函数**funcall**和**read**在**calc-operate**函数里的合作情况如下所示：

```
(defun calc-operate (tok)
  (let ((op1 (calc-pop))
```

```
        (op2 (calc-pop)))
    (calc-push (funcall (read tok) op2 op1))))
```

这个函数以一个算术运算符的名字（当做一个字符串）做为自己的输入参数。从前面的讲解里我们已经知道，字符串 **tok** 是从“*calc*”编辑缓冲区提取出来的一个记号；对我们的计算器来说，这个记号就是一个算术运算符，比如“+”或“*”。**calc-operate** 函数通过 **pop** 函数把堆栈最顶部的两个数字弹出堆栈，这与前面介绍的 **cdr** 函数的工作情况有些相似。**read** 函数把记号转换为一个符号，即某个算术函数的名字。假设这个运算符是“+”，则 **funcall** 的调用情况就将是下面这个样子：

```
(funcall '+ op2 op1)
```

这相当于以两个参数来调用加法函数，这与“正宗“的“**(+ op2 op1)**”效果完全相同。最后，加法函数的计算结果被压入堆栈的最顶部。

这套把戏是很有必要的，它使用户输入的算术运算符（比如加号“+”）被 LISP 自动转换为一个加法函数。我们也可以不那么精细，当然，也就不那么有效率；比如说，我们完全可以用一个 **cond** 结构写出 **calc-operate** 函数，就像下面这样：

```
(defun calc-operate (tok)
  (let ((op1 (calc-pop))
        (op2 (calc-pop)))
   (cond ((equal tok "+")
          (+ op2 op1))
         ((equal tok "-")
          (- op2 op1))
         ((equal tok "*")
          (* op2 op1))
         ((equal tok "/")
          (/ op2 op1))
         (t
          (% op2 op1)))))
```

最后，请记住计算器模式的代码是函数 **calc-mode**，我们编写的这个主编辑模式将由它来启动。它先创建（并弹出）“*Calc*”编辑缓冲区。然后，它会清除该编辑缓冲区中已经存在的所有局部变量，把堆栈初始化为“**nil**”，再创建一个局部变量 **calc-proc-fun**（参见前面的讨论）。最后，它把 **calc-mode** 设置为主编辑模式，设置好这个主编辑模式的名称，并启用计算器模式的局部键位图。

计算器模式的 LISP 代码

现在，相信大家已经能够看懂计算器模式的全部代码了。你会发现代码根本不算多！这要完全归功于强大的 LISP 语言和多功能的 Emacs 内部函数。一旦掌握了这些代码的工作原理，你就可以开始编写自己的主编辑模式。不多说了，下面就是我们编写的计算器模式的代码：

```
;;    Calculator mode.
;;
;;    Supports the operators +, -, *, /, and % (remainder).
;;    Commands:
;;    c       clear the stack
;;    =       print the value at the top of the stack
;;    p       print the entire stack contents
;;

(defvar calc-mode-map nil
  "Local keymap for calculator mode buffers.")

; set up the calculator mode keymap with
; C-j (linefeed) as "eval" key
(if calc-mode-map
    nil
  (setq calc-mode-map (make-sparse-keymap))
  (define-key calc-mode-map "\C-j" 'calc-eval))

(defconst calc-number-regexp
  "-?\\([0-9]+\\.?\\|\\.\\)[0-9]*\\(e[0-9]+\\)?"
  "Regular expression for recognizing numbers.")

(defconst calc-operator-regexp "[-+*/%]"
  "Regular expression for recognizing operators.")

(defconst calc-command-regexp "[c=ps]"
  "Regular expression for recognizing commands.")

(defconst calc-whitespace "[ \t]"
  "Regular expression for recognizing whitespace.")

;; beep and print an error message
(defun calc-error-message (str)
  (beep)
  (message str)
  nil)

;; stack functions
(defun calc-push (num)
  (if (numberp num)
      (setq calc-stack (cons num calc-stack))))
```

```
(defun calc-top ()
  (if (not calc-stack)
      (error-message "stack empty.")
    (car calc-stack)))

(defun calc-pop ()
  (let ((val (calc-top)))
    (if val
      (setq calc-stack (cdr calc-stack)))
    val))

;; functions for user commands:
(defun calc-print-stack ()
  "Print entire contents of stack, from top to bottom."
  (if calc-stack
      (progn
        (insert "\n")
        (let ((stk calc-stack))
          (while calc-stack
            (insert (number-to-string (calc-pop)) " "))
          (setq calc-stack stk)))
    (error-message "stack empty.")))

(defun calc-clear-stack ()
  "Clear the stack."
  (setq calc-stack nil)
  (message "stack cleared."))

(defun calc-command (tok)
  "Given a command token, perform the appropriate action."
  (cond ((equal tok "c")
         (calc-clear-stack))
        ((equal tok "=")
         (insert "\n" (number-to-string (calc-top))))
        ((equal tok "p")
         (calc-print-stack))
        (t
         (message (concat "invalid command: " tok)))))

(defun calc-operate (tok)
  "Given an arithmetic operator (as string), pop two numbers
off the stack, perform operation tok (given as string), push
the result onto the stack."
  (let ((op1 (calc-pop))
        (op2 (calc-pop)))
    (calc-push (funcall (read tok) op2 op1))))

(defun calc-push-number (tok)
  "Given a number (as string), push it (as number)
onto the stack."
  (calc-push (string-to-number tok)))
```

```
(defun calc-invalid-tok (tok)
  (error-message (concat "Invalid token: " tok))
  (sit-for 0))        ;make sure message is displayed

(defun calc-next-token ()
  "Pick up the next token, based on regexp search.
As side effects, advance point one past the token,
and set name of function to use to process the token."
  (let (tok)
    (cond ((looking-at calc-number-regexp)
           (goto-char (match-end 0))
           (setq calc-proc-fun 'calc-push-number))
          ((looking-at calc-operator-regexp)
           (forward-char 1)
           (setq calc-proc-fun 'calc-operate))
          ((looking-at calc-command-regexp)
           (forward-char 1)
           (setq calc-proc-fun 'calc-command))
          (t
           (setq calc-proc-fun 'calc-invalid-tok)))
    ;; pick up token and advance past it (and past whitespace)
    (setq tok (buffer-substring (match-beginning 0) (point)))
    (if (looking-at calc-whitespace)
      (goto-char (match-end 0)))
    tok))

(defun calc-eval ()
  "Main evaluation function for calculator mode.
Process all tokens on an input line."
  (interactive)
  (beginning-of-line)
  (while (not (eolp))
    (let ((tok (calc-next-token)))
      (funcall calc-proc-fun tok)))
  (insert "\n"))

(defun calc-mode ()
  "Calculator mode, using H-P style postfix notation.
Understands the arithmetic operators +, -, *, / and %,
plus the following commands:
    c   clear stack
    =   print top of stack
    p   print entire stack contents (top to bottom)
Linefeed (C-j) is bound to an evaluation function that
will evaluate everything on the current line. No
whitespace is necessary, except to separate numbers."
  (interactive)
  (pop-to-buffer "*Calc*" nil)
  (kill-all-local-variables)
  (make-local-variable 'calc-stack)
  (setq calc-stack nil)
  (make-local-variable 'calc-proc-fun)
```

```
(setq major-mode 'calc-mode)
(setq mode-name "Calculator")
(use-local-map calc-mode-map))
```

下面是计算器模式还值得改进或扩展的一些方面，我们把它们作为练习留给大家。这些练习不仅能够加深你对计算器模式的代码的理解，还能全面提高你的 Emacs LISP 程序设计水平。

- 增加一个求幂运算符"^"（比如45^ 等于 1024）。Emacs LISP 没有提供内建的求幂函数，但可以利用内建函数 **expt** 来做这个练习。
- 让此计算器能够支持八进制和/或十六进制数字。八进制数字用一个"0"开头，十六进制数字用一个"0x"开头；这样，017 等于十进制的 15，而 0x17 等于十进制的 23。
- 增加两个运算符"\+"和"*"，让它们对堆栈里的数字进行连加或者连乘；而不仅仅是对最顶部的两个数字进行运算（比如，"4 5 6 \+"等于 15，而"4 5 6 *"等于 120）（注 8）。
- 做为考察大家 LISP 知识水平的附加题，请解决本章开始部分给出的"情况 5"里的问题，即在 Emacs 变量 **compilation-error-regexp-alist** 里为编译器出错信息找出一个匹配来。（提示：先给这个列表复制一个副本，然后反复弹出最顶部的元素，直到找到一个匹配或者列表已经穷尽。）

对现有编辑模式进行定制

在掌握了主编辑模式的程序设计方法之后，你可能觉得对现有的主编辑模式进行定制更方便些。很幸运，对大多数定制情况来说，根本不用修改任何主编辑模式现有的LISP代码；甚至无需查看那些代码。Emacs的各种主编辑模式都有一些可以把你编写的代码添加到它上面去的"挂钩"。它们的名字恰好叫做"**mode-hooks**"（英文"hook"的意思就是"挂钩"）。Emacs内建的各主编辑模式都有一个名为"mode-name-**hook**"的编辑模式挂钩，其中的"*mode-name*"是该编辑模式的名称或启动它的函数的名字。比如说，C 模式就有一个 **c-mode-hook**，而 shell 模式则有一个 **shell-mode-hook** 等等。

注 8：　APL 程序员会看出这两个运算符是 APL 语言"扫描"操作符的变体。

那么，这个挂钩到底是什么呢？它是一种特殊的变量，这些变量的值是一些LISP代码，这些代码会在相应的编辑模式启动时得到运行。启动一个编辑模式就等于运行了一个LISP函数，这个函数通常要完成很多事情（比如为特殊命令设置按键绑定、创建编辑缓冲区和局部变量等等）；如果编辑模式有挂钩，那么负责启动编辑模式的函数所做的最后一件事情通常就是运行这些挂钩。总而言之，挂钩是负责实现某个编辑模式的LISP代码留出的一个“位置”，使你有机会改变它已经设置好的一些东西。比如说，你定义的按键绑定就将取代编辑模式的缺省绑定设置。

通过以前的学习，我们知道LISP代码可以是一个LISP变量的值，这一特点特别适合创建挂钩。在介绍如何创建一个挂钩的具体操作之前，我们先要介绍另外一个LISP基本函数：**lambda**。**lambda**也可以用来定义函数，在这一点上它与**defun**很相似；用LISP的术语来说，它们之间的区别是**lambda**定义的函数没有名字，就是用**lambda**定义的函数都是些“匿名函数”。**lambda**的语法格式如下所示：

```
(lambda (args)
 code)
```

其中的*args*是该函数的参数，而*code*是该函数的函数体。如果想把一个用lambda定义的函数作为一个值赋给某个变量，就必须对它进行“转义引用”以防止它被求值（即防止它被调用运行）。这就是说，必须使用下面这样的语句来进行赋值：

```
(setq var-name
      '(lambda ()
         code))
```

于是，在编写编辑模式挂钩的代码时，可以使用下面的格式：

```
(setq mode-name-hook
      '(lambda ()
         code for mode hook))
```

不过，你想定制的编辑模式可能已经有定义好的挂钩了。如果使用**setq**来定义你自己的挂钩，就会覆盖掉原有的挂钩。要想避免出现这样的问题，就应该改用**add-hook**函数：

```
(add-hook 'mode name-hook
      '(lambda ()
         code for mode hook))
```

编辑模式挂钩最常见的用途是为某个编辑模式的特殊命令修改一个或者多个按键绑

定。先来看一个例子：我们在第八章介绍了一种可以用来绘制简单线条图形的工具－图形模式。图形模式里有几个用来设置缺省绘制方向的命令。把缺省绘制方向设置为“下”的命令是**picture-movement-down**，它被绑定到“**C-c .**”（“**C-c**”后面跟着一个英文句号）组合键上。与对应**picture-movement-left**命令的“**C-c <**”组合键和对应**picture-movement-up**命令的“**C-c ^**”组合键相比，“**C-c .**”组合键不太容易记忆；所以我们现在想把**picture-movement-down**命令绑定到“**C-c v**”组合键上。图形模式所使用的键位图叫做**picture-mode-map**，所以用来设置这个按键绑定的代码是：

```
(define-key picture-mode-map "\C-cv" 'picture-movement-down )
```

图形模式的挂钩叫做**edit-picture-hook**（因为启动图形模式的命令是**edit-picture**）。于是，为了把这个代码挂到图形模式的挂钩上去，就必须把下面这条语句添加到“*.emacs*”文件里：

```
(add-hook 'edit-picture-hook
      '(lambda ()
         (define-key picture-mode-map "\C-cv" 'picture-movement-down)))
```

这条指令创建了一个lambda函数，而它的函数体是一个按键绑定命令。以后，只要进入图形模式（从下次启动Emacs开始），这个绑定就会生效。

再来看一个稍微复杂点的例子，假设你正使用**nroff**或**troff**对文本进行排版。你进入了nroff模式（参见第九章），但发现Emacs没有提供用来输入标准的**nroff / troff**字体设置命令（即分别对应于粗体、斜体和罗马体（即“正常”体）的“`\fB`”、“`\fI`”和“`\fR`”）的编辑命令。于是，你决定自行编写一个函数来插入这几个字符串，并且想把它们绑定到nroff模式下的几个键盘操作命令上。

为了实现你的想法，你先得写出一个用来插入字体设置命令字符串的函数。这个函数还是比较简单的，如下所示：

```
(defun nroff-insert-bold ()
  (interactive)
  (insert "\\fB"))

(defun nroff-insert-italic ()
  (interactive)
  (insert "\\fI"))

(defun nroff-insert-roman ()
```

```
    (interactive)
    (insert "\\fR"))
```

请注意，作为 **insert** 函数的参数的字符串里必须加上双反斜线字符，这样才能让Emacs把它们解释为单个的反斜线字符；对“**(interactive)**”函数的调用也是必不可少的，这样 Emacs 才会把这些函数看做用户级命令。

下一个步骤是用第十一章介绍的技术，编写能把这几个函数绑定到nroff模式键位映射图中的按键动作代码；nroff 模式的键位图叫做 **nroff-mode-map**。假设你想把这几个函数分别绑定到“**C-c C-b**”（粗体）、“**C-c C-i**”（斜体）和“**C-c C-r**”（罗马体）组合键上。“**C-c**”是Emacs的许多内建编辑模式—比如我们在上一章里见到的各种语言编辑模式 —— 所使用的前缀键。这段代码写起来也不困难，如下所示：

```
(define-key nroff-mode-map "\C-c\C-b" 'nroff-insert-bold)
(define-key nroff-mode-map "\C-c\C-i" 'nroff-insert-italic)
(define-key nroff-mode-map "\C-c\C-r" 'nroff-insert-roman)
```

最后，需要把这些LISP语句转换为**nroff-mode-hook**的一个值。下面是有关的代码：

```
(add-hood 'nroff-mode-hook
      '(lambda ()
         (define-key nroff-mode-map "\C-c\C-b" 'nroff-insert-bold)
         (define-key nroff-mode-map "\C-c\C-i" 'nroff-insert-italic)
         (define-key nroff-mode-map "\C-c\C-r" 'nroff-insert-roman)))
```

如果把上面这些函数的代码都添加到“*.emacs*”文件，以后再使用nroff模式的时候就会得到想要的功能。

下面是第三个例子。假设你正使用C语言进行程序设计，想用一个LISP函数来统计某个文件里有多少C语言的函数定义。下面那个函数将实现你的想法；它与我们在本章前面看到的**count-lines-buffer**例子有几分相似。这个函数将遍历当前编辑缓冲区，通过查找出现在行首的左花括号“{”来找出（和统计）其中到底有多少C语言的函数定义。如下所示：

```
(defun count-functions-buffer ()
"Count the number of C function definitions in the buffer."
  (interactive)
  (save-excursion
    (goto-char (point-min))
    (let ((count 0))
      (while (re-search-forward "^{" nil t)
        (setq count (1+ count)))
      (message "%d functions defined." count))))
```

这个函数的**re-search-forward**调用中额外多了两个参数；其第三个（即最后一个）参数的含义是“如果没有找到，就只返回**nil**而不发出错误信息”。而它的第二个参数则必须设置为它的缺省值“**nil**”，这样才能引入第三个参数（注 9）。

接下来，我们想把这个函数绑定到C模式里的“**C-c f**”组合键上。下面是我们用来给**c-mode-hook**进行赋值的语句：

```
(add-hook 'c-mode-hook
      '(lambda ()
         (define-key c-mode-map "\C-cf" 'count-functions-buffer)))
```

把这两个函数定义添加到“*.emacs*”文件里，就可以在C模式里得到预期的功能。

做为编辑模式挂钩的最后一个例子，我们将完成在第十二章里许下的一个诺言。我们在讨论C++模式时曾经介绍过两个可以用来替补**forward-word**和**backward-word**命令的编辑命令，它们分别是**c-forward-into-nomenclature**和**c-backward-into-nomenclature**命令；这两条替补命令可以把诸如“*WordsLikeThis*”之类的字符串当做三个而不是一个单词来对待，这项功能对C++程序员来说是很有用的。摆在我们面前的问题是怎样才能让通常用来执行**forward-word**和**backward-word**命令的按键动作去执行这两个新命令。

最容易想到的答案是为C++模式创建一个挂钩，把**forward-word**和**backward-word**命令的缺省绑定，即“**ESC f**”和“**ESC b**”组合键重新绑定到那两个新命令上，就像下面这样：

```
(add-hook 'c++-mode-hook
      '(lambda ()
         (define-key c++-mode-map '\e"
           'c-forward-into-nomenclature)
         (define-key c++-mode-map '\eb'
           'c-backward-into-nomenclature)))
```

（请注意，我们的按键绑定操作是在C++模式的局部键位映射图**c++-mode-map**上进行的。）可遇到这两个组合键已被Emacs重新绑定过的情况该怎么办？遇到**forward-word**和**backward-word**命令被绑定到其他组合键上的情况（这种可能性

注 9： **re-search-forward**函数以及其他查找函数的第二个参数是用来对该查找操作进行限制的：如果给它一个整数值*N*，就意味着跳过*N*个字符位置不做查找。如果是缺省值“**nil**”，就意味着不对查找操作加以限制。

是存在的）又该怎么办？我们需要查明这两个函数到底被绑定在哪些组合键上，再把它们都重新绑定为新的函数。

幸好有一个不太常用的函数可以帮我们查出这些信息，它就是**where-is-internal**函数。这个函数实现求助命令**where-is**（我们将在第十六章对**where-is**命令进行介绍）的功能。**where-is-internal**函数的作用是返回一个由LISP原子项构成的列表，该列表的每个元素对应着一个组合键，而作为其输入参数的函数就是绑定在这些组合键上的。我们可以用一个**while**循环来遍历这个列表以完成必要的重新绑定操作。下面是有关的代码：

```
(add-hook 'c++-mode-hook
      '(lambda ()
         (let ((fbinds (where-is-internal 'forward-word))
               (bbinds (where-is-internal 'backward-word)))
           (while fbinds
             (define-key c++-mode-map (car fbinds)
               'c-forward-into-nomenclature)
             (setq fbinds (cdr fbinds)))
           (while bbinds
             (define-key c++-mode-map (car bbinds)
               'c-backward-into-nomenclature)
             (setq bbinds (cdr bbinds))))))
```

let语句的头两行代码把**forward-word**和**backward-word**命令的各种绑定都找出来并分别放到局部变量**fbinds**和**bbinds**里。

接下来是两个**while**循环，它们的各种情况与本章前面用来实现计算器模式的LISP代码中的**print-stack**函数很相似。采用**while**结构来遍历列表元素的这种方法在LISP程序设计工作中是很常见的：先对列表的第一个元素（即列表的头元素**car**）进行处理，然后用“(**setq** *list* (**cdr** *list*))”语句把它从列表里删除；而整个循环将在列表变为空列表（nil）——即**while**语句的循环条件不成立时停止执行。

就上面这段代码而言，第一个**while**循环将逐个地取出**where-is-internal**函数找到的**forward-word**命令的每个绑定，并在C++模式局部键位图**c++-mode-map**中创建一个对应于新命令**c-forward-into-nomenclature**的绑定。第二个**while**循环将对**backward-word**命令和**c-backward-into-nomenclature**命令做同样的处理。

周围的代码将这些循环作为挂钩到安装C++模式，因此仅当C++模式被调用并在编辑缓冲区（也处于该模式）是活动的时，才能进行重新绑定。

大家可能已经注意到了，我们在以前各章里给出的编辑模式定制示例有的使用了挂钩，可有的却没有使用挂钩。比如说，上一章中用来设置C或C++缩进样式的代码里就使用了一个挂钩:

```
(add-hook 'c-mode-hook
      '(lambda ()
         (c-set-style "stylename")
         (c-toggle-auto-state)))
```

而用来给 **c-macro-expand** 命令另外设置一个C语言预处理器命令名的代码却没有使用挂钩:

```
(setq c-macro-preprocessor "/usr/local/lib/cpp -C")
```

为什么会这样的呢？按照规定，定制任一编辑模式的正确做法是必须通过它的挂钩来进行 ——刚才的例子本应该用下面这条语句来完成:

```
(add-hook 'c-mode-hook
      '(lambda ()
         (setq c-macro-preprocessor "/usr/local/lib/cpp -C")))
```

如果只是给变量重新设置一个值，就可以不使用挂钩；但如果想运行函数，例如 **c-set-style** 或者那些用来绑定按键的函数，就必须使用挂钩。这种“两面派”做法的具体原因涉及到 LISP 语言深层次的设计理念，它的基本原理是这样的:

诸如**c-macro-preprocessor**之类的变量是属于编辑模式局部范畴之内的，如果没有启动对该变量做出定义的编辑模式，它们就不会存在。因此，如果不是正在编辑C或C++代码，变量 **c-macro-preprocessor** 就不会存在于当前运行的Emacs中，因为还没有加载C模式（参见下面的说明）。但只要“*.emacs*”文件包含一条给这个变量赋值的 **setq** 语句，那么不管是否使用C模式，这个变量就已经存在了。Emacs能够很好地应付这种情况：当加载C模式时，它会注意到已经设置过这个变量的值，于是就不会再覆盖掉它。

但函数的情况就不同了。如果“*.emacs*”文件里有一个属于某个编辑模式的局部函数（比如**c-set-style**），那么Emacs多半会给出一条出错信息“`Error in init file`（初始化文件中发现错误）”，因为它此时根本不知道这个函数属于哪个编辑模式，更不用说它的作用。因此，必须把这个函数挂到C模式的一个挂钩上：等Emacs运行挂钩时，因为它已经加载C模式，所以就会知道这个函数的作用。

我们给出的这些关于挂钩的示例只是为了让大家了解在定制Emacs主编辑模式的道路上能够走多远。总而言之，因为有了挂钩，你能够在不触动实现编辑模式的LISP代码的前提下对它们进行各种各样的定制，其数量之多绝对超乎你的想像。

建立自己的 LISP 开发库

在做了大量的Emacs LISP程序开发工作之后，你可能想把自己写的LISP函数和程序包总结为一个开发库，这样，你就能在必要时按自己的意愿来调用它们。如果你只定义了几个“小”函数，把它们都放到*.emacs*”文件里当然不会有问题；可如果你正在为比较专业的用途开发规模较大的代码，把它们都放到“*.emacs*”文件里就不是个好主意——你肯定不想让Emacs在你每次启动它的时候都因为要处理这些代码而花费大量的时间。解决这一问题的办法就是建立自己的LISP开发库，就像Emacs自带的*lisp*目录一样，Emacs的内建LISP代码都放在这个目录里。一旦你建立起自己的开发库，就能做到只在必要的时候加载必要的LISP程序包，不用再把它们全都塞到“*.emacs*”文件里。

只需两个简单的步骤就能创建一个开发库。首先，创建一个目录（用UNIX操作系统的**mkdir**命令），你的LISP代码就将存放在这个目录里。大多数人会在他们的主目录里创建出一个名为“*lisp*”的下级子目录。LISP文件的名字通常是以“*.el*”结尾的（你的“*.emacs*”文件是个例外）。第二步是让Emacs知道有这么一个目录，这样，当你试图加载某个LISP程序包时，Emacs才会知道能够在什么地方找到它。Emacs把这类目录都记录在全局变量**load-path**里，这个变量的值是一个由字符串元素组成的列表，每个字符串是一个目录名。

load-path的初始值只包含Emacs自带的LISP目录的名字，比如“*/usr/local/emacs/lisp*”。你需要把自己的LISP目录的名字添加到**load-path**变量里。这个操作可以用LISP函数**append**来完成，它的作用是把任意个数的列表参数合并到一起。假设你的LISP目录名是“*~yourname/lisp*”，那么就需要把下面这条语句添加到你的“*.emacs*”文件里：

```
(setq load-path (append load-path (list "~yourname/lisp")))
```

函数**list**不可缺少，因为**append**函数的参数必须是列表。在“*.emacs*”文件里，这行代码必须出现在任何一条从LISP目录里加载程序包的命令之前。

当你试图加载一个开发库的时候，Emacs 会按目录在 **load-path** 变量里出现的先后顺序对它们进行查找；就我们刚才的设置情况而言，Emacs 会先查找它自己缺省的 LISP 目录。如果想让 Emacs 先查找 LISP 子目录，就应该用前面介绍的 **cons** 函数代替 **append** 函数，如下所示：

```
(setq load-path (cons "~yourname/lisp" load-path))
```

当想用自己的程序包代替 Emacs 标准程序包时，这种格式很有用。例如，如果你自己编写了一个 C 模式版本，并想用它代替标准包，你就可以用这种形式。注意，这里的目录名不需要通过 **List** 函数调用，因为 **cons** 函数的第一个参数可以是一个原子项（在这种情况下是一个字符串），正如前面描述的计算模式，这种情况与使用 **cons** 函数将值压入堆栈类似。

如果想让 Emacs 查找你当前的工作目录，给 **load-path** 变量加上一个“**nil**”即可，可以用 **cons** 函数把它加在列表的开始，也可以用 **append** 函数把它加在列表的末尾。它的效果与把英文句号“**.**”添加到 UNIX 环境变量 **PATH** 里的情况差不多。

在创建了个人的 LISP 目录并告诉 Emacs 可以在什么地方找到它之后，就可以加载和使用自己开发的 LISP 程序包。有好几种方法可以用来把 LISP 程序包加载到 Emacs 里。下面给出了四种方法；我们在第十一章里曾经用过其中的第一种方法：

1. 输入用户级命令“**ESC x load-library RETURN**”；参见第十一章。
2. 在 LISP 代码（通常就是你的“*.emacs*”文件）里加上语句“**(load** “*package-name*”)”。如果这条语句是加在你的“*.emacs*”文件里的，Emacs 就会在你启动它的时候加载指定的程序包。
3. 给用来启动 Emacs 的命令（emacs）加上一个命令行参数“-l *package-name*”。这个操作将加载指定的程序包 *package-name*（注 10）。
4. 在 LISP 代码（通常就是你的“*.emacs*”文件）里加上语句“**(autoload** ’*function* “*filename*”)”，就像第十二章中介绍的那样。这个操作将在执行给定函数 *function* 时让 Emacs 自动加载相应的程序包。

注 10：另一个命令行选项“-f *function-name*”会让 Emacs 在启动时自动运行函数 *function-name*，此函数没有参数。

对 LISP 文件进行字节编译

在创建了自己的LISP子目录之后，如果你还对LISP文件进行了字节编译（byte-compiling），即把文件中的LISP转换成字节代码，就能更有效地加载和使用它们；字节代码是一种更加紧凑的机器阅读格式。对LISP文件*filename.el*进行字节编译将创建出一个字节代码文件*filename.elc*。与没有进行字节编译处理的LISP文件相比，字节代码文件的长度通常会减少到原来的40%到75%。

虽然经过字节编译的文件运行效率更高，但严格说来并不是一个必要的步骤。当**load-library**命令遇到参数“filename”时，它将先去查找是否存在一个名为“*filename.elc*”的文件。如果这个文件不存在，它就去查找“*filename.el*”文件（即未经字节编译的版本）是否存在。如果这个文件也不存在，它才会去查找“*filename*”文件。对“*.emacs*”文件也可以进行字节编译；如果你的“*.emacs*”文件比较大，字节编译之后的启动速度会有明显提高。

在编写LISP代码的过程中，可以对编辑缓冲区中的某个函数单独进行字节编译：把光标移到该函数定义里的任意位置，再输入“**ESC x compile-defun**”命令即可。如果想对整个LISP代码文件进行字节编译，请输入“**ESC x byte-compile-file RETURN**”命令和它的文件名。如果省略了“*.el*”后缀，Emacs会自动加上它并请求确认。如果修改了文件但还没来得及存盘，Emacs会先对它进行存盘。

当字节编译工作正在进行的时候，你将看到正被编译的函数的名字从辅助输入区里一闪而过。由字节编译器生成的文件其名字与原始的LISP文件相同，但后缀名中增加了一个字母“*c*”；也就是说，“*filename.el*”将变为“*filename.elc*”，而“*.emacs*”将变为“*.emacsc*”。

最后，如果你的LISP目录里有很多个LISP文件，而你只对其中的几个进行了修改，就可以用**byte-recompile-directory**命令只对那些从上次字节编译之后被修改过的文件重新进行编译（做法类似于UNIX操作系统的**make**工具）。你只要先输入“**ESC x byte-recompile-directory RETURN**”命令，再给出LISP目录的名字（或直接按下回车键以确认当前目录为缺省项）即可。

本章内容：

- Emacs 的 X 界面
- 让 Emacs 使用 X 字体和颜色
- 定制 Emacs 在 X 环境中的显示情况
- 通过 .Xdefaults 文件进行定制
- 属性、窗格、菜单和鼠标事件
- 与 X 服务器进行通信
- 良好的X程序设计风格

第十四章

Emacs 编辑器和 X 窗口系统

虽然Emacs能够运行在字符终端上，可它的本质却是X窗口系统下的一个客户软件，用不着借助 *xterm* 之类的终端仿真器就能运行。这就使它能够使用多种字体、能够解释鼠标事件、能够显示在多个X窗口里……总之，能够使用X窗口系统全套的功能。在这一章里，我们将介绍 Emacs 的 X 界面。

Emacs 的 X 界面

启动Emacs的时候，X窗口系统的许多功能就已经处于随时待命的状态。我们先介绍一些马上就能使用的功能，然后再介绍一些关于Emacs定制和程序设计方面的内容。

窗口卷屏条和 X 状态行

运行在X窗口系统下的Emacs与运行在字符终端上的Emacs之间最明显的区别是每一个 Emacs的X窗口都有它自己的X卷屏条。这些卷屏条与 *xterm* 窗口的卷屏条在行为上非常相似。可以用鼠标拖动卷屏块来连续卷动窗口内容，也可以在卷屏条里点击鼠标使窗口内容快速上卷或下卷。还可以点击或拖动窗口的状态行，其效果是对它上面或者下面的窗口进行移动以及调整窗口尺寸。

菜单条

Emacs窗口最顶部的那一行与Macintosh的菜单条很相似，上面依次排列着“**Buffers**（编辑缓冲区）”、“**Files**（文件）”、“**Tools**（工具）”、“**Edit**（编辑）”、“**Search**（查找）”和“**Help**（帮助）”等菜单标题。如果在这些标题上点击鼠标，就能打开各个菜单并通过鼠标轻松地选择执行各种常用的Emacs编辑命令。（从Emacs 19.30版开始，运行在字符终端上的Emacs也有菜单条了，但还是得通过键盘操作而不是鼠标来使用它们。）

比如说，“**Buffers**”菜单会给出一个Emacs编辑缓冲区的清单，用鼠标点击某个条目就可以进入相应的编辑缓冲区。“**Search**”菜单则把没有按键绑定的查找命令（标准的或使用了正则表达式的查找命令）都列了出来，可以通过鼠标来选择和执行各种查找操作。“**Edit**”菜单提供了Macintosh风格的剪切/粘贴操作。学习这些菜单的最佳方法就是多做练习。

如果不喜欢下拉菜单，想回到屏幕以前的显示方式，也可以禁用菜单条功能；具体做法我们将在后面讨论定制问题的章节里进行介绍。

注意：根据Emacs 19.30（1996年6月推出）的情况看，自由软件基金会好像还没有把Emacs菜单条的设计方案最终确定下来（这也是我们为什么没有在此介绍大量细节的原因）。因此，如果你的菜单条看上去与我们这里介绍的不一样、或者子菜单和菜单项发生了变化，请不必惊讶。

鼠标的其他用途

我们完全可以把鼠标点击动作绑定为任意的Emacs LISP函数，从而使鼠标的点击、拖放动作就像是输入了相应的键盘命令一样（我们将在本章后面的内容里对此做更详细的讨论）；不过，最常见的鼠标动作绑定情况还是帮助设置光标和文本块标记的位置。

如果在一个普通的Emacs编辑缓冲区里点击鼠标左键，Emacs将会把插入点（即Emacs编辑缓冲区里的光标）移动到X光标位置。如果点击鼠标的地方没有文本（比如因为在该位置的左边已经有了一个换行符），Emacs编辑缓冲区里的光标将被移动该行文本的最右位置（这类似于上、下方向键的动作效果）。

如果点击并拖动鼠标，就又是另外一种情况了。Emacs 会在发生点击动作的地方设置一个文本块标记，把插入点留在松开鼠标键的地方。也就是说，只需用一个点击-拖放命令就可以（为“**C-w**”等文本块编辑命令）定义一个文本块。这个文本块将立刻被复制到 X 窗口系统的选取缓冲区（selection buffer）和 Emacs 的删除环里。

当点击并拖动鼠标的时候，文本块标记和插入点之间的文本区域反显为灰色背景。当松开鼠标左键以设定插入点位置的时候，这个文本区域仍将呈灰色反显状态（注 1）。通常，反显区域会在下一个输入事件发生时消失。可以让 Emacs 进入一个能够把反显区域保留在屏幕上（具体做法将在后面的内容里介绍）的编辑模式；但最好不要把它用做缺省的编辑模式，因为反显区域将随着插入点的移动而收缩或者扩大，从而给正常工作造成干扰。

鼠标中键缺省的绑定动作是插入 X 剪切缓冲区里的内容（与 *xterm* 窗口里的情况差不多）。

鼠标右键的作用是把文本块标记和插入点之间的文本复制到删除环里。如果在同一位置再次按下鼠标右键，文本块将会被删除。因此，用它来剪切和粘贴文本是很方便的。

按住**ALT**键再按下鼠标左键将对X窗口系统的辅助选取缓冲区（secondary selection buffer）进行操作，缺省的主选取缓冲区（primary seiection buffer）不受影响。不过，知道辅助选取缓冲区的 X 应用程序并不很多，所以这个细节最好不用。

出于教学的目的，以上内容都稍微做了一些简化。它们没有提到双击或三击鼠标键时的各种特殊操作，也没有介绍各种特殊的文本块选取情况（这些细节可以通过 Emacs的在线帮助系统查到）。在本书写作期间，自由软件基金会还在对鼠标接口进行微调；因此，鼠标的操作细节还有可能会发生变化，但我们在这里介绍的操作行为应该不会再有变化了。

Emacs 的删除环和 X 的选取缓冲区

Emacs 内部的删除和恢复命令能够与 X 窗口系统的选取机制互通有无。用“**C-w**”

注 1：在 Emacs 19 某些早期的版本里，松开鼠标键的时候，灰色反显区域会肖失。如果你遇到了这样的情况，就快去升级吧！

或“**C-k**”等命令删除Emacs文本时，被删除的文件也将被复制到X的选取缓冲区里；可以再通过标准的鼠标命令把它粘贴到其他X窗口里。

反过来说，X选取的东西Emacs也看得见。对于遵守X窗口系统关于选取操作有关规定的应用程序来说，从它们的X窗口里剪切到的文本也将放入Emacs的删除环里，并且马上就能用“**C-y**”组合键或Emacs的其他删除恢复命令进行粘贴。

窗格的使用

在Emacs18版本里，所有的Emacs窗口都只能显示在一个字符方式下的X窗口里。现在则不同了，Emacs能够在一个或者多个窗格里进行显示；“窗格”是彼此分开的X窗口。

全体窗格都是从同一个给定的Emacs衍生出来的，驱动它们的也都是同一个基本进程；它们共享Emacs的二进制执行代码和LISP库的执行代码。因此，用一个Emacs打开多个窗格要比用多个Emacs分别打开它们更合算，前者加给机器的负载要比后者少。

Emacs拥有一些对窗格进行操作的命令，我们把其中最常用的命令列在表14-1里。

表14-1：Emacs的窗格操作命令速查表

键盘操作	命令名称	动作
C-x 5 f	**find-file-other-frame**	访问另一窗格里的某个文件
C-x 5 d	**dired-other-frame**	在另一个窗格里运行Dired模式
C-x 5 0	**delete-frame**	删除当前窗格
C-x 5 b	**switch-to-buffer-other-frame**	切换到另一个窗格

正如大家看到的那样，这些命令与正常的文件和编辑缓冲区访问命令很相似（它们与文件访问命令的不同之处，只是多了一个不同的命令前缀）。

与窗格有关的其他函数和变量能帮助管理好X显示画面上的各种窗格，也能帮助Emacs的编辑模式用好窗格。如果你对这些函数有兴趣（它们都没有缺省的按键绑定），可以用“**ESC x apropos frame**”命令来查阅；它们每一个都有在线帮助信息。

让 Emacs 使用 X 字体和颜色

X 窗口系统下的Emacs能够以多种固定宽度的字体来显示文本。它对比例间隔字体的处理目前还做得不是很好，希望以后的版本能够解决这一问题。X 窗口系统下的 Emacs 还能在 X 服务器的支持下以多种前景色和背景色的组合来显示文本。

按住**SHIFT**键再点击鼠标左键将打开一个字体菜单。在做出选择的同时，当前窗格里的Emacs字体会立即发生改变并重新刷新这个窗格。可以用这个方法来检查不同字体在屏幕上的显示情况，看哪几种字体的可读性最好。

如果想查看可用颜色的范围，请从菜单条上选择“**Edit**”菜单，打开“**Text Properties**（文本属性）”子菜单，再选择“**Display Colors**（显示颜色）”项。虽然这个菜单项不会把所有可能的前景色和背景色组合都显示出来（它们实在是太多了，没有几千，也有几百），但已经足够为文本挑选前景色和背景色之用。

自动反显与着色功能

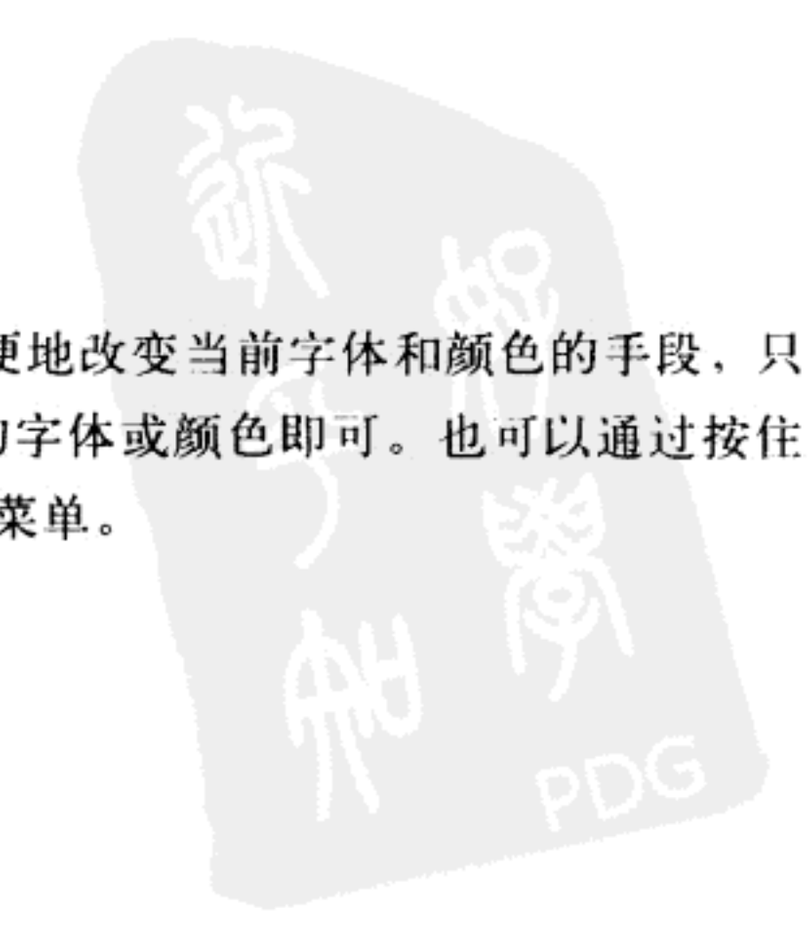
要想使用字体和颜色，最简单的方法就是加载LISP程序包**font-lock.el**（这是Emacs发行版本自带的一个程序包）。这个编辑模式可以用不同的颜色和字体把文本编辑缓冲区里的不同部分标识出来。比如说，可以把C和LISP编辑缓冲区里的程序注释都挑出来，给它们加上颜色，让它们与黑色文字的程序代码有所区别。如下所示：

```
;; Turn on font lock mode every time Emacs initializes a buffer
;; for Lisp or C.
;;
(add-hook 'emacs-lisp-mode-hook 'turn-on-font-lock)
(add-hook 'c-mode-hook 'turn-on-font-lock)
```

font-lock模式特别适用于给程序代码或大纲模式文本着色，对其他高度结构化的文本也非常有用，比如 **troff** 稿件的源代码等。

改变文本的字体和颜色

Emacs 菜单里的“**Edit**”下拉菜单提供一种方便地改变当前字体和颜色的手段，只需从“**Text Properties**”子菜单里选一个新的字体或颜色即可。也可以通过按住 **CTRL** 键再按下鼠标第二键的方法来打开这个菜单。



为了用好“**Text Properties**”菜单，需要知道Emacs在其内部是用术语“字型”(face)来描述字体和颜色的。一种字型就是字体和颜色的一种组合。“**Text Properties**”菜单准备了一些事先安排好的字型和一个用来通过名字指定其他字型的选项。

我们将在本章后面的内容里对字型、给字型起名字以及有关的LISP程序设计结构做进一步的讨论。就目前来说，可以简单地认为编辑缓冲区里的每个字符都有一个不同的字型以不可见的方式与它关联即可（当然，在实际工作中，字型很少会变化得如此频繁）。

保存设置了字体和颜色的文本

细心的读者可能已经注意到了，虽然着色机制允许我们给编辑缓冲区里的文本加上颜色，可我们却没有介绍怎样才能把文本属性保存起来供今后的编辑工作使用。这是一个很重大的问题。如果没有办法把属性和文本一块儿保存起来，那么再好的字体和着色机制也都不过是画面显示方面的一种噱头而已，用来装饰编辑缓冲区倒是不错，可在加强 Emacs 编辑能力方面却并没有多大功效。

我们需要这样一种方法来解决这个问题：把文本属性当做一种文本排版标记（text-markup）保存起来；然后，当下次编辑文件的时候，再把文本排版标记恢复为文本的属性。

在本书的写作期间，这方面的试验性代码已经添加到Emacs里。一个名为**enriched-mode**的程序包允许人们把文本属性保存为一种符合因特网技术标准RFC 1563文档有关规定的MIME扩展文本格式（注2），然后再把这种格式的文件按照原来的文本和文本属性导入Emacs编辑缓冲区。这个编辑模式听起来很不错，但要想让它达到实用要求并做到能够与 Emacs 的其他编辑模式（比如邮件处理器和 WWW 浏览器等）完善地集成在一起，在其设计和开发方面还有很多工作有待完成。等大家读到这本书的时候，这样的程序包可能已经有好几个了，每个程序包支持一种不同的扩展格式，比如 HTML 格式等。这类编辑模式的发展方向是让 Emacs 具备支持 WYSIWYG 编辑能力，甚至具备支持多媒体编辑的能力。

注 2: RFC 1563 已经被 RFC 1896 取代。Emacs 使用的标记方法看来应该是遵守 RFC 1896 的。

定制 Emacs 在 X 环境中的显示情况

让 Emacs 给编辑缓冲区里的文本进行着色的方法我们已经介绍过了。X 环境下的 Emacs 还有其他一些值得一用的定制方法。如果想把屏幕上原来是菜单条的那一行省下来以增加一个文本行，可以把下面这条语句添加到“*.emacs*”文件里：

```
(menu-bar-mode nil )
```

当然，也可以通过辅助输入区完成这一定制工作。

还可以让 Emacs 为每个编辑缓冲区分别创建一个它自己的窗格。为了实现这个定制，需要把下面这条语句添加到“*.emacs*”文件里：

```
(setq pop-up-frames t )
```

是否需要做这样的定制完全取决于个人。在 X 桌面空间的使用习惯方面，我们基本上可以划分出（至少）两种不同的风格。第一种风格的用户总是在相对固定的位置打开几个大的窗口：一个窗口里可能是个编辑器，另一个窗口里可能是个时钟等等。采用这种桌面布局的人通常喜欢让桌面保持整洁。如果你把自己归入这类人群中，把这个变量设置为“**t**”的效果大概不会让你喜欢。而第二种风格的用户似乎更喜欢桌面堆满文件。如果你就是一名这样的用户，那么**pop-up-frames**就是为你准备的。

函数调用

```
(transient-mark-mode t )
```

将改变Emacs在选取文本块时的反显行为，使灰色反显背景总是出现在插入点和文本块标记之间的区域里。在这种模式下，只要再开始对编辑缓冲区做修改，它的文本块标记就将被取消；这是为了让屏幕上不至于总是有一大块灰色的区域干扰用户的正常工作，特别是当我们知道文本块的边界已经不再有用的时候。

通过 .Xdefaults 文件进行定制

还可以用在*.Xdefaults*文件（或者是*.Xresources*文件）里设置X资源的方法对Emacs进行定制。我们准备介绍几个比较重要的X资源；如果想查看允许进行设置的资源的清单，请参考 GNU Emacs 的使用手册页。

要想对X资源进行设置，必须编辑资源文件。需要为想设置的每一项资源添加一条下面这样的语句:

```
emacs.<resource >:<value >
```

如果想对geometry（几何）资源（即窗口的尺寸和坐标）进行设置，就必须使用下面这样的语句:

```
emacs.geometry := 80x49 + 270 + 0
```

这条语句的作用是把geometry资源设置为值“=80x49+270+0”。它告诉Emacs使用一个80个字符宽、49个字符高的窗口，窗口从屏幕点阵坐标（270, 0）处开始显示。在一个分辨率为1024x768的显示器上，如果使用14磅字体并且窗口不带边框，这样的设置将使Emacs窗口占据整个显示器右面几乎四分之三的面积，垂直方向上的文本行数则是尽可能的多。可以根据自己的具体情况来修改这个例子里的数值。几何资源的详细计算方法请参考X的使用手册页。

font资源允许选择一种显示字体供X环境下的Emacs使用。注意只有固定宽度的字体才能正常工作。请看下面的例子:

```
emacs.Font: *fixed-bold-r-*-14-*-*
```

这个例子设置的显示字体是14磅、每英寸75点（即75dpi），这种字体在17英寸、0.31点阵、1024x768分辨率的显示器上工作得很好。如果点阵数更小（比如0.28点阵）或者显示器更大，就可以设置一个更小的磅值，Emacs窗口将能容纳更多的文本行。如果你有一台高级的21英寸、0.25点阵的显示器，就可以试试下面这样的设置:

```
emacs.Font: -*-courier-*-r-*-*-16-*-100-100-*-*-*-*
```

这是专为点阵数非常小的显示器而准备的一种100dpi的字体。如果你觉得小字体读起来费眼，就挑选一种大字体好了。

通过**title**资源可以设置显示在Emacs窗口标题栏里的文字。窗口标题一般不用设置；如果进行了设置，所有的Emacs窗格就都会是同样的名字，即设置的标题；如果没有设置，那么，在缺省的情况下，打开的第一个窗格的名字是

"emacs@*hostname*"（"*hostname*"将被替换为你机器的主机名），以后创建的窗格将使用其中的编辑缓冲区的名字。

foreground（前景）和**background**（背景）资源允许对Emacs窗口里的文本颜色和背景颜色进行设置。资源值必须是标准的X颜色的名称。如果临时标记的文本块反显功能看着不对劲，就可能需要对这两项资源进行设置。

修改*.Xdefaults*文件之后，需要在主目录里运行下面这条命令：

```
xrdb .Xdefaults
```

或者在任何一个目录里运行下面这条命令：

```
xrdb ~/.Xdefaults
```

如果你觉得采用命令行参数来设置这些资源的方法更方便，也可以那样做。比如说，**-font**选项所使用的字体设置参数与我们刚才介绍的font资源所使用的完全一样（但必须给它加上引号以免shell对其中的元字符进行扩展）。可以用**-geometry**选项设置几何资源，用**-fg**和**-bg**选项设置前景色和背景色。其他可用的选项请参考Emacs的使用手册页。

属性、窗格、菜单和鼠标事件

下面，我们将展开关于使用Emacs LISP进行X程序设计方面的讨论，这部分内容与第十三章是相互衔接的。通过前面对X定制方面的介绍，大家可能已经对Emacs的X接口代码的工作情况有了一个直观的了解。为了把X环境下的Emacs LISP程序设计讲明白，我们对这些概念的探讨还需要再稍微正式一些、细致一些。

在这一节里，我们将对几个新出现的与Emacs的X支持功能有关的基本概念进行讨论。如果你对Enacs LISP程序设计没有兴趣，则本章从这里开始一直到结尾的内容都可以略过不读。

在阅读这一节内容的时候，需要时刻记住下面这3件事情：

- 我们将要介绍的有些概念在X或GUI环境以外的地方也有意义（特别是菜单和

文本属性)。但在我们写作本书的期间，字符终端上的Emacs还只能零星地提供一点对它们的支持——如果真的能支持它们（注3）。

- 每个新概念通常都会引入一个新的Emacs LISP数据结构。弄清每种数据结构都有哪些操作具有实际使用价值，这对加深我们理解其行为是很有帮助的，弄清哪些操作开销小，哪些程序开销大也很有必要。
- 自从Emacs 19的首次发行以来，某些与X有关的函数的行为和某些与X有关的数据结构的形状已经发生了变化（出现了一些不互相兼容的情况）。虽然未来版本发生巨大变化的情况就目前看不太容易发生，但多加注意和勤查《Emacs LISP Reference Manual》(Emacs LISP参考手册，自由软件基金会出版）或Emacs的在线文档是很值得推荐的做法，如果你是在这本书最后一次再版好几年之后才读到它的，就更要注意这一点。

窗格

X环境下的Emacs最简单的基本构造是“窗格”。如果你在阅读此前各章节的时候动了脑筋，对窗格及其功能就应该已经有了一个大致的了解。X环境下的Emacs有一个缺省的窗格。一个窗格就是一个显示一个或者多个Emacs窗口的X窗口。全体窗格共享同一个编辑缓冲区清单和其他Emacs上下文。

窗格是用一种基本数据类型实现的，这种数据类型的名字就叫做窗格。用来创建新窗格的函数只有一个，**make-frame**。它的输入参数是一个给定格式（具体格式可以在《Emacs LISP Reference Manual》里查到）的关联列表，它对窗格的大小、位置、X字体以及各种其他参数做了定义。缺省值取自一个Emacs LISP全局变量**default-frame-alist**和X的资源系统。与窗格有关的其他函数通常会以这些窗格对象为参数（而且往往是第一个参数）。

函数**redraw-frame**的作用是绘制窗格，而**delete-frame**函数的作用是删除窗格。如果想查看所有窗格，可以检查**frame-list**和**visible-frame-list**函数的返回值（但不能用修改这些列表返回值的方法来创建或者删除窗格，它们只是Emacs内部窗格列

注3: 不要坐等这种情况会发生变化，因为已经出现了适用于Linux和其他免费UNIX的XFree86实现，自由软件基金会和其他一些Emacs程序员现在都把X支持看做理所当然，基本上不再为纯字符终端做开发了。

表的副本)。为了掌握实际窗格列表(Emacs Lisp的内部列表、**frame-list**是它的一个副本)中的窗格次序,可以使用**next-frame**和**previous-frame**。

在任一给定的时刻,只能有一个窗格被选中。只有被选中的窗格才能接受输入事件。各种窗格函数通常用一个取值为"**nil**"的窗格对象参数来代表当前的选中窗格。可以用窗口管理器为X窗口系统提供的任意一种方法选中一个新的窗格,Emacs会知道发生了什么变化。还可以通过**select-frame**函数以程序控制的手段强行切换窗格,强制X把输入焦点交给新窗格。

窗格可以处于可见状态、不可见状态或图标化状态(被缩小为一个X图标,点击图标将使它还原为窗格)。可以通过窗口管理器以交互方式让窗格进入这三种状态。Emacs LISP代码可以用函数**make-frame-visible**、**make-frame-invisible**和**iconify-frame**来改变窗格的显示状态。窗格的显示状态可以用**frame-visible-p**函数测试出来。

窗格的大小可以用**frame-height**、**frame-width**、**frame-pixel-height**和**frame-pixel-width**等函数分别查到;**set-frame-size**函数可以设置窗格的大小。各种其他参数可以通过**frame-parameters**函数所返回的关联列表查出来;其格式与**make-frame**函数的参数格式完全一样。可以通过**modify-frame-parameters**函数改变这些参数中的任何一个。各种窗口参数的含义和作用在《Emacs LISP Reference Manual》(Emacs LISP参考手册)里有详细的说明。

窗格的程序设计其实挺简单(但所涉及的鼠标和焦点事件方面的理论却相当复杂,大家学到后面就能亲眼见到)。最重要的事情是必须记住有哪些Emacs全局变量是适用于全体窗格的,又有哪些是各窗格专用的局部参数。一般说来,只要是能够在屏幕上看到的任何东西,或者说与X有关的任何东西,各个窗格就都会分别有一个彼此互不影响的值,这些值是创建窗格时从系统的某个缺省设置值那里复制过来的。

文本属性与覆盖

Emacs 19表示文本属性(text properties)的基本表示法可以附着在编辑缓冲区或者字符串里的字符上。当字符被移动或者被复制时,它的文本属性也会随之转移。编辑缓冲区里的文本的属性被认为是编辑缓冲区内容的组成部分;对它们进行的修改也可以像被改动的文本那样通过撤销操作恢复回来。

Emacs还允许你定义自己的文本属性，这对某些场合来说是非常有用的。在使用文本属性方面的功能时，很多用法都涉及到文本属性的一个子集，这个子集里的文本属性对Emacs的显示管理器（display manager）和编辑命令有特殊的含义。我们将在本章后面的内容里对这些属性做进一步的讨论。

在Emacs的内部，字符属性是用一个由3个元素构成的列表来表示的，这3个元素分别是：一个把文本属性的名称和值关联起来的列表，一个给出文本属性作用范围起始点的编辑缓冲区索引或字符串索引，和一个给出文本属性作用范围结束点的编辑缓冲区索引或字符串索引。之所以采用这种表示形式的列表（而不是把每个字符位置上的文本属性分别保存起来），部分原因是为了避免用来保存文本属性的列表体积过大，或者说是为了避免让指向该列表的假名指针过多，同时还能做到把查找某种属性的平均开销控制在可接受的范围内；但更主要的原因是为了能够迅速查找到文本属性的值发生了修改的地方。

与文本属性有关的基本函数

为了对文本属性进行测试和设置，Emacs准备了一套完备的函数。函数**get-text-property**允许查询某已知属性在给定位置处的值；而**set-text-property**函数则允许设置一项属性；函数**all-text-properties**允许查看由给定位置处的全体属性组成的一个列表。在程序设计实践中，需要对编辑缓冲区中的某个区域或者某些字符串位置的属性值进行检查和设置的情况往往更多见一些；这些工作可以用**add-text-properties**、**put-text-property**、**remove-text-properties**和**set-text-properties**等几个函数来完成。关于这几个函数的详细资料请查阅《Emacs LISP Reference Manual》。

有时候，我们需要找出编辑缓冲区或字符串某个给定的属性值是从什么地方开始发生变化的。事实上，因为这个操作对Emacs显示管理器来说的确是太重要了，所以人们对根据编辑缓冲区的内容而提取其文本属性列表的操作进行了彻底的优化。如果你想看看这些优化的效果，可以用函数**get-text-property**和其他有关函数，特别是**next-property-change**和**previous-property-change**函数，或者**next-single-property-change**和**perevious-single-property-change**函数对编辑缓冲区进行遍历。这些函数的细节也可以从《Emacs LISP Reference Manual》中查到。

PDG

特殊属性

最重要同时也是最常用的文本属性是字符的“字型（face）”。不带字型属性的字符将被显示为当前Emacs窗格所使用的缺省X字体（可以随时用“**C-mouse-2-down**”命令改变这个X字体）。通过对字符的字型属性进行设置，就可以改变它的X字体、颜色或者有没有下划线等属性。我们已经介绍过从“**Edit**”菜单对这些属性进行设置的方法；还可以利用Emacs LISP提供的一些函数在程序控制下实现同样的设置功能。

local-keymap属性允许把一个局部键位映射图和一个字符序列关联起来。这个属性最适合应用于某种文本属性在编辑缓冲区内的作用范围的起始点索引或结束点索引，这样，就可以创建一些在编辑缓冲区的不同区域里有不同绑定的特殊热键。

属性 **read-only** 可以禁止对某个字符或某个区域进行写操作；如果有人想对它进行修改，就会引发一个LISP错误。

特殊属性 **modification-hooks**、**point-entered** 和 **point-left** 允许对挂钩函数做这样的设置：每当某字符范围被修改或者被插入时，就执行这些挂钩函数。此外，为了能够以间接方式对文本属性进行设定，Emacs还专门准备了一个 **category** 挂钩。这些特殊属性的细节超出了我们现在对Emacs的X界面的讨论范围；如果读者需要了解这方面的详细资料，请自行阅读《Emacs LISP Reference Manual》（Emacs LISP参考手册）。

字型的设置和命名

创建新字型对象的基本函数是 **make-face**。创建新字型之后，还需要对它的属性进行设置，其中最重要的一个属性是字体。给字型设置字体的工作可以用函数 **set-face-font** 来完成，这个函数的输入参数是一个字符串，而这个字符串必须是一个合法的X字体设置项（即该字符串的语法必须与X资源文件所使用的有关语法一致）。

我们还能对字型某些其他特征进行定制，其中包括字型的前景色、背景色、前景点阵、背景点阵以及该字型是否带有下划线等。这些属性中的每一个都有一个与之对应的**set-face**函数，而这些函数的详细资料都可以通过Emacs的在线求助系统查到。

在引用某个字型对象的时候，既可以使用一个字符串形式的名字（即给 **make-face**

函数准备的参数），也可以直接使用该字型对象的一个副本；这与编辑缓冲区的情况有些类似。我们推荐使用第一种方法（用名字来引用字型对象）。Emacs的有关文档没有对“给某个类型为字型的值设置的属性是否会影响到与之同名的其他字型对象”这一问题做出明确的说明。

注意：在我们写作本书的时候（1996年6月），这里探讨的某些细节还没有收录到当时的《Emacs LISP Reference Manual》里。请大家留意可能的修订，千万不要麻痹大意！

覆盖

对编辑缓冲区里的某些文本区域而言，它们的一些字型属性不会被看做编辑缓冲区内容的一部分（因此，移动和复制操作也不会保留它们的字型属性），我们把这些字型属性称为“覆盖（overlay）”。比如，在某些主要用途是编辑表格或者编辑列表清单的主编辑模式里，为了以多种反显效果和多种颜色绘制出这些编辑模式的显示画面，就可能需要用到覆盖。

一个编辑缓冲区可能有多种覆盖；每种覆盖都有一个优先级，而高优先级的覆盖将取代低优先级的覆盖。版式的优先级高于文本属性。你可以设定某种覆盖只在某个特定的窗口里起作用，而不是访问同一编辑缓冲区的所有窗口。

函数**make-overlay**和**delete-overlay**的作用分别是创建和删除覆盖。查看和设置覆盖属性的函数分别是**get-overlay**和**put-overlay**。函数**overlay-start**、**overlay-end**和**overlay-buffer**允许对覆盖的作用范围进行检查；**move-overlay**则用来改变其作用范围。最后，**overlays-at**和**next-overlay-change**可以帮助查找到指定的覆盖和它们的边界。

X窗口系统中的输入事件

在X环境下，Emacs LISP必须面对的最重要和最复杂的问题是如何表达和处理输入事件。在基于字符终端的Emacs早期版本里，输入事件简单说来就是字符，只要是Emacs收集到的输入事件，就都可以用一个简单的字符串来表示。

但X环境下的Emacs却必须对付种类更多的输入事件，比如鼠标点击动作、鼠标移

动动作、切换窗格焦点等等。相应地，输入事件的内部表达方式也就越来越复杂，适用面也越来越宽。除了能够把函数绑定为普通的组合键，或者组合键加修饰键之外，现在还可以把函数绑定到用来表示X输入事件的向量上，而每一种输入事件可能都有一种它自己的结构。以前只接受按键组合为参数的函数（比如 **define-key**、**global-set-key** 和 **local-set-key** 等）现在也开始接受这些输入事件向量了。

输入事件主要分为4大类：按键组合、功能键、鼠标事件和窗格焦点事件。我们将在下面依次对它们进行讨论。

按键组合

输入数据中的 ASCII 字符是用 ASCII 字符来表示的。但字符编码的宽度却是 16 位（因而允许使用诸如UNICODE之类的大字符集），而且还可能会额外带有几个修饰位，这些修饰位包括 **META**、**CTRL**、**SHIFT**、**HYPER**、**SUPER** 和 **ALT**。

有几个修饰位的名称似乎暗示着它们携带的信息已经被编码到输入事件的ASCII部分里，而你也许正在奇怪 **META** 和 **ALT** 到底有哪些修饰作用；如果真是这样，就说明你被一些聪明但却完全错误的推论误导了——但这并不是你的过错。为了把这个问题解释清楚，我们需要回溯一下“古老的”历史。

很久以前，当Richard Stallman还在麻省理工学院的人工智能实验室里工作的时候，一种被人们称为LISP机器的分析引擎出现了。用LISP语言实现的Emacs早期版本就运行在这些机器上，它们对此后（包括 GNU Emacs 在内）的 Emacs 版本的设计有着巨大的影响。这些计算机的基本数据类型是32位的标记字（tagged word），其中 8 位给出类型信息或其他特殊信息，另外 24 位给出数据或地址。LISP 机器上的每一个按键都会在输入流（input stream）里被转换为一个 32 位的数据项，数据项的扩展位表示它们是字符。很明显，ASCII字符只能填满24位符号空间的一个小角落。

后来，人们又为这些LISP机器发明了一种专用键盘。除常见的**CONTROL**和**SHIFT**修饰键以外，这种键盘还另外多出5个修饰键（**META**、**HYPER**、**SUPER**、**TOP**和**FRONT**）。这些修饰键中的每一个都对应着基本ASCII按键所产生的代码中的一个或者多个修饰位。

PDG

专用键盘上的许多按键都在键帽上画有3个符号：顶上是一个字母和一个符号，前面是一个希腊字母。比如说，“L”键的顶上画着一个字母“L”和一个左右方向的箭头符号，前面是希腊字母“λ”。如果在左手按住某个修饰键的同时再用右手按下这个键，就可以输入箭头符号或拉丁字母“L”与希腊字母“λ”的大、小写形式。**CTRL**、**META**、**HYPER**和**SUPER**键的任意组合都能输入这几样东西。

事实上，这种专用键盘可以产生八千个不同的字符；这就使它们能够轻松地输入复杂的数学公式或者代表上千种单字符命令。为了节省输入时间，很多程序员都愿意多记住一些按键的命令含义（Emacs的操作界面就在这种氛围里形成了）。

修饰键实际上控制着机器字数据部分高8位（第17到第23位、统称为“修饰位”）的翻转状态。以“a”键为例，是否需要把带有“control”修饰位的“a”键解释为ASCII字符“CTRL-A”完全取决于操作系统。

到了20世纪80年代后期，LISP逐渐退出了历史舞台；通用机器已经能够以更快的速度运行LISP。但对LISP机器专用键盘的记忆却延续了下来，X窗口系统的**keysym**程序在设计思路上就受到了它的影响，可以利用这个工具把任意一个键定义为修饰键，也可以利用它来定义各种古怪的组合键。当Richard Stallman准备重新布置用于X环境的Emacs键盘命令时，LISP机器专用键盘的影子又出现了。

这就是把输入事件代码的第18到第23位分别设定为**META**、**HYPER**、**SUPER**、**SHIFT**和**ALT**按键修饰位，并且仅在输入事件无法用ASCII字符表示的时候才使用的原因。比如说，代表ASCII大写字母“A”的输入事件就不需要对**Shift**修饰位进行置位，代表“**C-a**”组合键的事件也不需要对**CTRL**修饰位进行置位；可如果按下的是“**C-M-A**”（即**control-meta-shift-a**）组合键，则因为ASCII无法对它进行表示，所以**META**、**CTRL**和**SHIFT**修饰位就都将被置位。

这段历史也道出了Emacs文档混用“**M-**”和“**ESC**”的原因（为了避免出现这种混乱，本书统一使用“**ESC**”来描述组合键；这与自由软件基金会文档里的做法正好相反，那些文档里到处都是“**M-**”）。用来设计Emacs的机器原本有一个**META**键，整个操作接口就是围绕着它而设计的。当Richard Stallman和朋友们把Emacs移植到没有**META**键的机器上去的时候，他们采用了让“**ESC**”前缀等同于一个**META**换档修饰位的简单做法，但保留了用来描述按键组合的老记号办法。

各修饰键的完整定义可以在Emacs的在线文档里查到。随着时间的推移和程序设计技术的发展，LISP专用键盘上需要使用4个修饰键的命令变得不那么流行了；现在的程序员可能永远不用了解这些修饰键的细节了。

功能键

功能键在按下时将被表示为一个符号而不是一个字符，而这个符号就是该功能键的名称。大多数名称符号的含义都是很明显的（比如4个方向键分别代表着上、下、左、右四个方向，“f1”代表着第一个功能键等等）。

字符终端下的Emacs 19就采用了这种做法。Emacs会根据终端的termcap设置项对功能键的定义情况做出分析，再把它们（虽然你看不见）添加到自己的输入分析表里；这样，那些功能键就能返回正确的符号。但在Emacs 18里，与终端有关的LISP程序包都是自己提供各种终端转义序列，这些程序包现在几乎都被淘汰了。

如果需要表示功能键和某个修饰键的组合，可以给功能键的符号加上相应的前缀，比如**META**用“**M-**”来表示，**SHIFT**用“**S-**”来表示，**ALT**用“**A-**”来表示。**CTRL**用“**C-**”来表示等等。这些前缀的顺序无关紧要，比如说，“**C-S-f1**”与“**S-C-f1**”的效果完全相同。

鼠标事件

鼠标事件是用列表表示的。列表的头元素是一个鼠标按键符号，即“mouse-1”、“mouse-2”或“mouse-3”（在用**define-key**或其他有关函数设置鼠标绑定时，只需给出这3个符号就足够了）。这些鼠标符号也可以像键盘按键那样加上修饰位前缀。

用来表示鼠标事件的列表里可能还包含一个或者多个地址元素。地址元素也是一个列表，它负责给出以下细节：发生鼠标事件的窗口和编辑缓冲区，事件发生时的鼠标位置，本次事件与上次事件之间的时间间隔（以微秒为单位）等。

最简单的鼠标事件是在不移动鼠标的情况下按下并释放鼠标——即“点击事件”，列表头元素符号的后面将只有一个地址元素。这个事件将在你松开鼠标按键的时候被发送出来。

如果在按住鼠标键的同时还拖动了鼠标，就会在松开鼠标时得到一个“拖动事件”。拖动事件有两个地址元素(分别对应于事件的起始点和结束点)，并且会在头元素符号的修饰位前缀后面再加上一个“**drag-**”前缀。不过，如果没有把拖动事件绑定为某种操作，Emacs就会丢弃第二个地址元素并把它表示为一个点击事件。这就是说，不用你来区分短距离的拖动事件与点击事件，除非你真的想这么做。

还有一个环节。按下一个鼠标键的时候，将得到一个“按下事件”。它与点击事件很相似，但事件列表头元素符号在鼠标键符号，和它的修饰位前缀之间会多出一个“**down-**”前缀。鼠标事件通常都是配对出现的，所以按下事件的后面要么会跟有一个点击事件，要么（比较少见）会跟有一个拖动事件。如果需要对这些细节进行处理，可以给函数“`(track-mouse ...)`”包上一段LISP代码做为“外套”，用这个“外套”来处理鼠标移动事件；表示鼠标移动事件的列表有一个头元素符号“**mouse-movement**”和一个地址元素。

最后，我们再说说关于键位映射图的事。以鼠标事件开始的按键组合将根据发生鼠标动作的编辑缓冲区的键位映射图来做出解释，而这个编辑缓冲区并不一定是当前选中的编辑缓冲区。这条规定的效果是很直观的：可以点击另外一个编辑缓冲区里的文本并执行一些特殊的命令，而这些操作将在不丢失或者不需要改变编辑缓冲区焦点的情况下得以完成(除非鼠标绑定被显性地设置为改变编辑缓冲区的焦点——而这是比较常见的做法)。

为了把这些概念解释得既明白又简单，我们在这里省略了许多具体的细节。这些细节大多与**read-key-sequence**函数针对输入事件的解释方法、或者说与Emacs自身的向后兼容性有关，这是为了保证Emacs LISP代码中那些包含“**\M-**”修饰位的按键组合字符串仍然能够使用。如果需要了解这方面的知识，请查阅《Emacs LISP Reference Manual》(Emacs LISP参考手册)。

窗格焦点事件

窗格焦点事件并不复杂。当用户把输入焦点从一个Emacs窗格，切换到另外一个窗格里去的时候，就会产生一个窗格焦点事件，用来表示窗格焦点事件的列表由两个元素组成，一个是头元素符号“**switch-frame**”，另一个是新窗格的名字。

当前输入事件只有在整个序列完整地输入之后才会被报告出来。这种延迟是为了避

免出现某些混乱。比如像用户开始输入一个复杂的序列却在半中间不小心改变了当前窗格的场合。按键组合将按“老”窗格的键位映射图进行解释，除非事先发送过焦点改变事件。

菜单

菜单是一种特殊结构的键位映射图，这种键位映射图允许为有关操作提供提示字符串。换句话说，它不仅能够把一个函数或者另外一个键位映射图绑定到一个按键组合上，还可以把一个函数和一个提示字符串同时绑定到一个按键组合上。当菜单显示时，那个提示字符串将成为菜单项的名字。每个菜单键位映射图还可以有一个总的提示字符串，它将用做该菜单的标题。

要想激活一个菜单，就必须把它绑定到一个起操作前缀作用的组合键上。菜单键位映射图在工作原理方面与普通的键位映射图没什么两样，但在动作行为上却有所不同：跟在操作前缀后面的各输入事件将弹出相应的菜单（或子菜单）。既可以用传统方法（键盘操作）选择菜单项，也可以用鼠标点选。

对菜单功能的开发目前仍在进行当中。除了用做菜单项名字的提示字符串之外，还可以给每个菜单项关联上一个帮助字符串；也就是说，在线求助功能可以用来给每一个菜单项提供相应的帮助信息。

Emacs 不仅准备了几种用来布置菜单格局的方法，还准备了一种能够把你的菜单添加到 Emacs 窗格顶部的菜单条上的方法。如果你对这些方法感兴趣，或者想了解键位映射图和菜单的 LISP 表示法方面的细节，请自行查阅《Emacs LISP Reference Manual》（Emacs LISP 参考手册）或 Emacs 发行版本自带的 */etc/LNEWS* 文件。

与 X 服务器进行通信

Emacs LISP 还提供了很多用来与 X 服务器进行通信的函数。想用这么短的篇幅把它们都介绍给大家是不现实的，所以我们只拣其中最重要的几个说说，好让大家对它们有一个初步的了解。

要想查出关于 X 服务器显示能力方面的信息，可以使用的函数有 **x-display-pixel-**

height、**x-display-pixel-width**、**x-display-mm-height**、**x-display-mm-width**、**x-display-planes**、**x-display-backing-store**、**x-display-color-cells** 和 **x-display-color-p** 等等。还可以用 **x-rebind-key** 函数修改键盘按键所发送的字符序列。

一般说来，与X环境有关的函数在名字里都有一个“**x-**”前缀。所以，如果把“**x-**”提供给Emacs帮助系统的**apropos**函数做为其参数的前缀，就能查出与X环境有关的函数的完整清单。

良好的X程序设计风格

在前面介绍窗格对象的小节里，我们提到过这样一件事：有几个与窗格有关的Emacs LISP函数完成的其实是一些通常由X窗口管理器负责的操作，比如窗口的图标化。如果你去学习《Emacs LISP Reference Manual》(Emacs LISP参考手册)，就会发现还有其他一些这样的函数我们没有在这里讲过，例如，从Emacs LISP直接读、写X服务器的选取缓冲区是完全能够做到的。出于其自身的设计理念，Emacs LISP提供一些这样的函数，但如果你真的想把这些函数用在一个本身并不是由X输入事件循环驱动的Emacs LISP程序里，就一定要三思而后行。传统交互式程序(包括运行在字符终端上的Emacs在内)的I/O风格都是程序化的：它们被设计为每次只使用一个，而且只有在它们工作正常并准备接受输入数据时才明确地请求用户进行输入。而程序则会根据用户在整个执行过程中的具体位置划分为许多个状态。这类程序的交互式执行过程，可以看做由许多依次进行的I/O小循环构成的集合体，每个I/O小循环面对的是全部输入事件的一个子集，而当前的输入事件子集应该与前一个有所区别。

但X世界里的情况就不一样了，几乎所有的程序(还都是些“好”程序)都是由一个巨大的随机事件循环驱动的，这个循环能够接受的X输入事件范围相当广泛，不仅包括普通的按键组合，还包括鼠标点击，窗口的打开与关闭事件，窗口尺寸调整等许多内容。只有很少的程序状态被编码为全局性的执行序列；与程序运行状态有关的变量就更多了，与这些变量有关的内部检查也更多了。如果用户准备采取某项行动，可它的某些先决条件还没有准备好，那些内部检查就会发出各种相应的通知。(出于同样的理由，苹果公司Macintosh机器上的程序也是这样组织的。)

这两种风格是根本对立的。如果从 Emacs LISP 里对窗口进行图标化或者对 X 的选取缓冲区做明确的设置(或者是明确地调用一些通常是由鼠标驱动的窗口管理器函数完成的操作),就极有可能陷入一种不正常的程序化交互风格(相对于随机风格);与通过X完成这些操作的情况相比,画面可能会因窗口边框的意外消失而变得很难看。

所以要多想想“事件驱动”。尽量做到不依赖于具体的程序运行状态;不要在没有考虑设计冲突的前提下用 Emacs LISP 强制执行 X 操作。Emacs 的“把输入事件绑定到函数上”的设计思想能够帮助定义隐含的全局性事件处理循环;尽量使用这种手段,要把序列化的程序运行状态(特别是那些不可见的)减到最少。

第十五章

Emacs 下的版本控制

本章内容:

- 版本控制的用途
- 版本控制的有关概念
- VC 对基本操作的辅助作用
- 修改注释的编辑
- VC 命令汇总
- VC 模式的标志
- 使用哪一种版本控制系统
- VC 命令细说
- 对 VC 进行定制
- 对 VC 进行扩展
- VC 的不足之处
- 有效地使用 VC

版本控制的用途

如果你曾经编写过大型的程序或者很长的文档，就肯定遇见过这样的情况：以前做的某些修改到最后反而不适当，更让人着急的是你还不知道怎样才能撤消那些修改并回到以前的某个“好”版本上。或者，你为别人开发了一个程序或文档，可因为不知道怎样才能回到别人使用的那个老版本上，也就没有办法把新的程序补丁或注释集成到它里面去。如果你是某个程序或文档开发团队的一员，也许就需要有一种能够把修订史记载下来的办法，方便日后核查哪些修改应该由哪些人来负责。

这些常见的问题都可以用“版本控制系统（version control system）”来解决。版本控制系统能够把某个文件或者某些文件的修订历史自动记录下来。它允许返回到这个历史记录中的任何一个阶段，并且能方便地制作出关于各版本之间差异比较的报告。

在能够运行 Emacs 大多数版本的 UNIX 机器上，有 3 种广泛应用的版本控制系统 —— SCCS、RCS 和 CVS。这几种版本控制系统都能很好地完成它们的工作；但

由于历史方面的原因，在很多本应该使用它们的场合却见不到它们的身影。造成这种局面的原因之一是这3种版本控制系统的操作界面都比较复杂和琐碎。而且，因为它们还没有被广泛地集成到编辑器里，所以，如果你想使用它们对文件进行检索、编辑、保存或者修改，就必须经过两个烦琐的过程才能完成工作。

在这一章里，我们将介绍一个被称为“VC”（Version Control，版本控制）的Emacs工具，它是一个能够让你轻松使用这3种版本控制系统的Emacs LISP辅助编辑模式。VC能够执行各种版本控制命令（通过Emacs的子进程功能，与运行邮件处理器模式的方法一样）。VC隐藏绝大多数的面向版本控制系统的接口细节；你只需掌握一个命令，就可以用它来完成那些最基本的版本控制操作。

我们假设你使用的是RCS系统，这是因为：(a)人们普遍认为它比出现较早的SCCS系统更先进，(b)它是一种来源广泛的免费软件，(c)它的安装和使用要比CVS系统简单。因此，在对Emacs的VC功能进行介绍时，我们将主要采用RCS系统的术语，并且会把RCS系统的行为当做典型的用法示例。

版本控制的有关概念

处于版本控制下的每个文件都有一份修订记录，这份记录由一个原始版（initial version）和一系列（有时还会有分支）以此为基础的修订版（revisions）组成。

要想对一个文件进行版本控制，就必须先把它“注册”到版本控制系统上；也就是让版本控制系统把注册的文件内容看做原始版本，然后开始记录它的修订历史（注1）。

如果想对一个已经注册的文件进行修改，就必须“取出（check out）”这个文件；也就是通知版本控制系统准备对它进行修改。SCCS或RCS系统还会给文件加锁；在完成修订工作之前，其他人是无法再取出它的（但仍可查看它的内容）。（CVS系统没有使用这种加锁机制，个中的原因我们在此后的第三个段落里进行了解释。）

在SCCS或RCS等使用加锁机制的版本控制系统里，如果有人已经给某个文件加上

注1：如果你想用VC来管理某个文件，倒不必非得从VC里对它进行注册。VC对已经存在的文件版本控制树也能工作得很好。

了锁，你就无法再收它。如果别人取出了文件之后又忘记了这码事，锁就会一直加在文件上。你也许想把锁“偷卸”下来，也就是把工作文件的控制权以及别人对文件的修改都转移到自己手里，再由你负责“签入（check in）”一个经过修改的版本。（如果责任心不强，这样做将很危险。）

在对工作文件（work file，即由你签出的工作用文件副本）进行修改的过程中，你可以随时“复原（revert）”工作文件，即丢弃所做的修改并撤销取出操作。可如果你想把自己对文件的修改保存起来，就必须“签入”那些改动。这将使它们成为该文件修订记录中一个永久性的修订版。在RCS和SCCS系统下，签入操作还将释放工作文件上的锁；这样，别人就可以签出这个文件并对之进行编辑。在CVS系统下，文件是不会加上锁的；CVS系统会尽量让你所做的修改与你取出这份文件期间别人所做的修改和平共处，如果它发现其中有所冲突，就会向你求助（即要求由你来进行人工协调）。

注册、签出、复原、签入等操作是版本控制系统最基本的操作。还可以利用版本控制系统做一些其他的事情：可以检索出任何一份已经记录在案的修订版，可以生成一份关于任意两个修订版之间或者关于任何一个修订版与自己（可能已经被改动过的）工作文件之间的差异报告，甚至能够把不想保留的某个修订版删除掉（虽然这种情况非常少见）。

大多数版本控制系统（包括SCCS、RCS和CVS这三个在内）能够给每一份修订版加上一些“修改注释（change comments）”。也就是说，在签入一个已被注册过的文件时，还可以在修改记录里写上一些关于所做的修改的解释，这些解释不会成为文件本身的一部分。每个修订版都有一个“修订号（revision number）”，其作用是把它自己在修改记录中的位置标识出来。SCCS、RCS和CVS的修订号都是从1.1开始的。如果文件的修改历史呈线性发展（小型项目修订历史的典型结构），修订号就是一个由两个数字组成的序列，两个数字之间用小数点隔开。

但文件的修改历史完全可能出现旁支，文件的变体版本可以平行地发展。对于这种情况，文件修改历史的主干仍由两段（two-field）修订号组成，但旁支的修订号就需要用更多的数字来表示。SCCS、RCS和CVS在旁支的具体命名规则方面各有各的做法；如果想详细地了解这方面的资料，请自行查阅版本控制系统的有关文档。版本控制系统还有很多我们来不及在这里介绍的功能。O'Reilly出版公司出版过一

本关于版本控制系统的好书《Applying RCS and SCCS》，它的作者是Don Bolinger和Tan Bronson。

VC 对基本操作的辅助作用

由于历史上的原因，如果你想进行一些基本的版本控制操作（即注册、签出、签入、复原），就不得不掌握三个或四个不同的shell命令，而且这些操作还都必须在你的编辑器之外（最好的情况是在编辑器的一个shell子进程里）进行。整个过程既复杂又繁琐，许多人就是因为这个原因才不使用版本控制系统的。

VC的操作界面就简单得多。这种简单性来源于这样一个事实：不管你的版本受控文件处于什么状态，合乎逻辑的后续操作通常就只有一种。下面是有关的规则：

- 如果你的文件还没有被置于版本控制之下，那么接下来的逻辑步骤就是注册它并签出一份允许你修改的副本。
- 如果文件被注册过但还没有人取出它，接下来的逻辑步骤通常就是签出它并对它进行修改。
- 如果你已经把文件签出，接下来的逻辑步骤就是签入它。
- 如果已经有别人签出了文件，除了等待，你还能做的事情就只有卸下锁了（锁的主人会收到一个锁已被你卸下的通知）。

事实上，VC模式只有一个基本命令“**C-x v v**”（命令名是**vc-next-action**），它的含义是“对这个文件做下一个合乎逻辑的操作”；说得更精确点，“把当前访问的文件带到下一个正常的版本控制状态”。整个流程将沿图15-1里的箭头方向运转，这个图给出了正常的版本控制流程（注2）。这个命令在Emacs 19的各个版本里都能使用；调用它的时候，它会自动加载VC模式的其他功能并完成它的工作。

注2：小技巧：在一个新的工作文件上，发出的第一个**vc-next-action**命令通常会从“尚未注册”状态进入“已注册，未加锁”状态，再到“已加锁，可编辑”状态——如果只想对文件进行注册但不打算签出它（这是相当典型的情况），就得用第二条命令从“已加锁，可编辑”状态退回到“已注册，未加锁”状态。这未免有些费力不讨好。如果只想注册不想编辑，可以使用“**C-x v i**”命令（命令名是**vc-register**）。

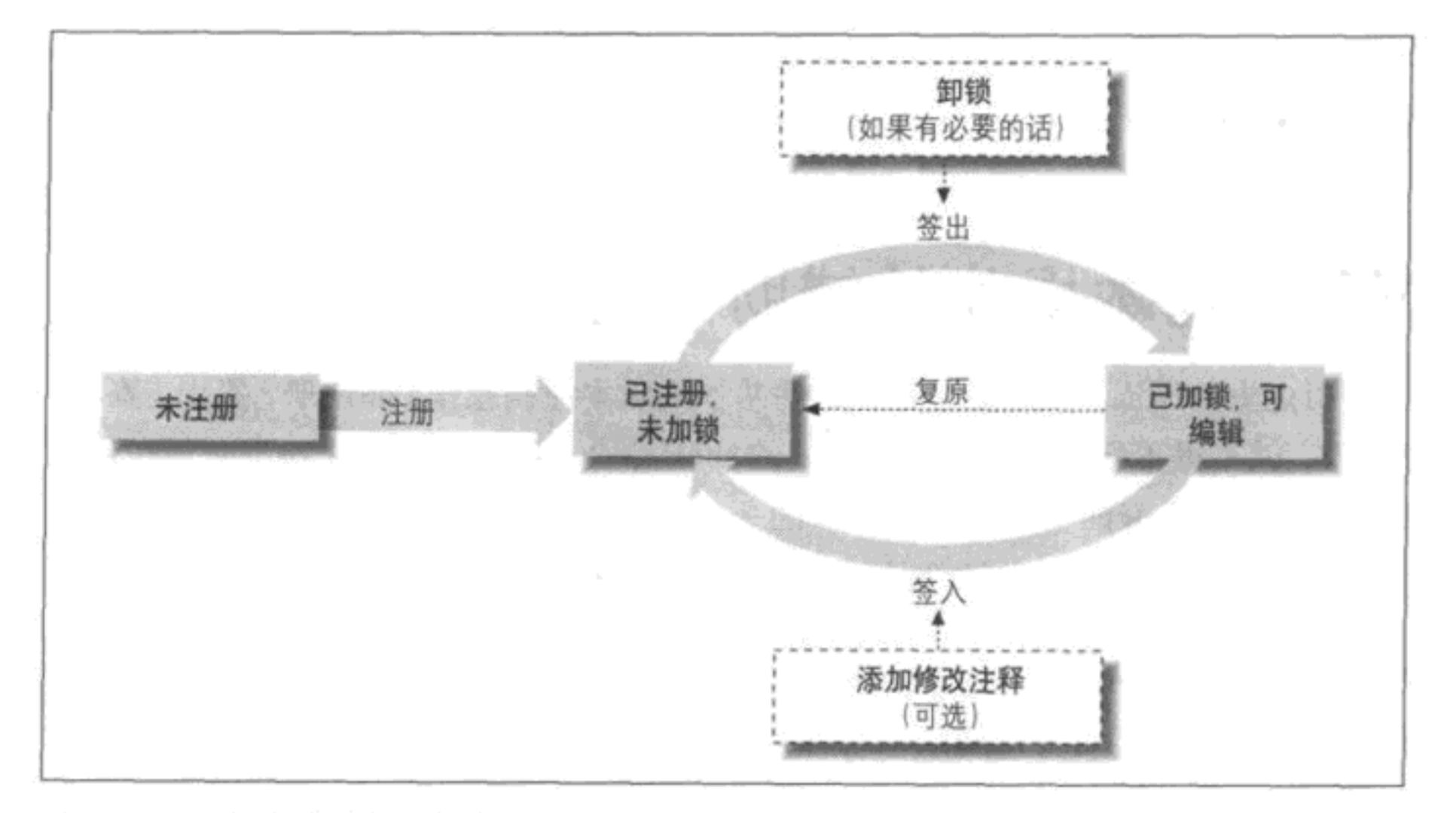


图 15-1：正常的版本控制流程

事情当然没有这么简单。归还某个文件的一批修改时，VC会弹出一个编辑缓冲区以供输入一条修改注释。类似地，卸下一个锁时，VC会弹出一个编辑缓冲区要求提供一个解释；此解释将电邮给锁的主人。

VC模式还提供一个复原操作："**C-x v u**"（命令名是**vc-revert-buffer**）。在实际操作中，负责实现**vc-next-action**命令的函数会先检查编辑缓冲区自上次签出之后是否被修改过。如果没有修改过，那么它只复原编辑缓冲区并卸下工作文件上的锁而不会签入一个空白的修改。

修改注释的编辑

在VC模式里，有3个操作会弹出一个编辑缓冲区以接受注释或通知文本，这3个操作是签入、卸锁和文件注册（只发生在某些特定的场合；详见本章后面内容里的解释）。在每一种情况里，有关操作会暂停下来，等你在注释编辑缓冲区中按下"**C-c C-c**"组合键之后再继续完成。可以立即开始输入注释以尽快完成有关操作，也可以去先做点别的事情。VC不关心程序运行状态（还记得我们在第十四章对程序化接口和非程序化接口的讨论吗？）；它会耐心地等你做完其他事情后再回来完成注释编辑缓冲区中的工作。如果删除弹出来的编辑缓冲区，有关操作就会无声无息地结束。

注释编辑缓冲区是一个明文编辑缓冲区。但是，每打开一个注释编辑缓冲区，其内容就将被保存到注释编辑缓冲区环的一个新位置。可以用“**ESC p**”组合键沿这个环上溯、用“**ESC n**”组合键下溯、用“**ESC r**”沿这个环向回查找文本，或者用“**ESC s**”向前查找文本。根据设计，这些命令与漫游 Emacs 辅助输入区的命令历史记录时所使用的组合键是一样的。这些命令中最常用的是“**ESC p**”。能够调出和编辑上一条修改注释通常是很有用的。

VC 命令汇总

为了让大家对 VC 模式的其他功能有一个基本印象，这里把 VC 命令汇总在表 15-1 里。我们对每个命令都做了详细的解释；其实，从这些命令的名称上就能把它们的功用猜出来。

表 15-1：VC 命令速查表

键盘操作	命令名称	动作
C-x v v	**vc-next-action**	前进到下一个合乎逻辑的版本控制状态
C-x v d	**vc-directory**	列出某个子目录下所有的已注册文件
C-x v =	**vc-diff**	生成一份版本差异报告
C-x v u	**vc-revert-buffer**	丢弃上次签入修订版之后的一切修改
C-x v ~	**vc-version-other-window**	在另外一个窗口打开指定的修订版
C-x v l	**vc-print-log**	显示某个文件的修改注释和修改记录
C-x v i	**vc-register**	把文件注册到版本控制系统
C-x v h	**vc-insert-headers**	给文件加上供版本控制系统使用的标题块
C-x v r	**vc-retrieve-snapshot**	签出一个已命名的项目快照
C-x v s	**vc-create-snapshot**	创建一个项目快照并给它起个名字
C-x v c	**vc-cancel-version**	丢弃一个已经保存起来的修订版
C-x v a	**vc-update-change-log**	刷新一个 GNU 格式的 ChangeLog 文件

这些命令大致按使用频率由高到低的顺序排列在表格里。这也是我们将在以后的章节里对它们进行讨论的顺序。VC 命令都有相同的前缀“**C-x v**”。你的手指很快就会习惯于这个前缀，所以只要记住那些单个字符的命令后缀即可。VC模式还有两个辅助性的命令**vc-rename-file**和**vc-clear-context**，它们都没有对应的按键绑定。我们稍后再对这两个命令进行介绍。

VC 模式的标志

如果在某个编辑缓冲区里打开的是一个注册过的文件，VC模式就会占用其状态行的一点空间以给出该文件的版本控制状态。我们将会看到，如果编辑缓冲区正访问一个版本受控文件，模式标签部分（状态行上的括号部分）将给出版本控制系统的名称和文件的修订号。

如果这两部分是用一个短划线隔开的，就说明文件还没有被签出；如果它们是用一个冒号隔开的，就说明文件已经被签出，而显示在状态行上的修订号是签出该文件时它所拥有的修订编号。

如果没有看到这些标志，就说明文件尚未注册。

使用哪一种版本控制系统

我们前面讲过，VC模式至少可以使用3种版本控制系统（今后可能还会增加更多的版本控制系统）。对一个给定的文件来说，VC将根据与之对应的“主控文件（master file）”来选择一种版本控制系统来打开它；主控文件就是保存修改记录的那个文件。

如果使用的是RCS系统，那么每个项目的目录通常会有一个子目录，RCS主控制文件就在这个子目录里；如果使用的是SCCS系统，那么就会是些SCCS子目录。CVS系统的情况有点特殊：每个项目的目录有一个CVS子目录，控制信息就保存在其中，但CVS主控文件通常都集中保存在某个目录里，这个位置通常由环境变量**CVSROOT**给定。

如果VC在这些专用目录里找不到它需要的主控文件，就会到工作文件所在的目录里去查找（因此，如果你不介意自己的工作目录里充斥着各种主控文件，就不必专门创建SCCS或RCS子目录）。VC将检查各种可能性（因此，你完全可以在同一个目录里同时使用多个不同的版本控制系统；但我们并不推荐这种做法）。

如果VC始终没有找到它需要的主控文件，就会依次对RCS目录、SCCS目录，最后是CVS中央目录进行查找。如果它在RCS目录里找到了主控文件，它就会把新文件注册到RCS版本控制系统里，以此类推。如果它在这些目录里也没有找到主控

文件，而你又在让它注册一个文件，它就会认为你打算使用的是RCS系统并创建出相应的主控制文件和你的工作文件。

如果不清楚SCCS、RCS、CVS等版本控制系统是否能够在你的系统上使用，你可以不带参数地分别执行一次**comb**、**rcs**和**cvs**命令。如果看到的是出错信息或使用方法信息，就说明对应的版本控制系统能够在你的系统上使用；如果看到的是"command not found（命令未找到）"，就说明对应的版本控制系统不能在你的系统上使用。

VC 命令细说

vc-next-action是最基本的VC命令，它的用途我们已经介绍过了。现在，我们将对VC模式的其他命令做详细的介绍。我们选择的讨论顺序是把比较常用和比较简单的命令放在前面，把比较罕见和比较复杂的命令放在后面。

根据你的具体情况，只要你认为已经学到了足够的东西，就可以随时跳到本章的结尾。不过，虽然越靠后面的命令越不常用，但有关的讨论没准能启发你想出一些记录和组织项目文件的新点子，谦虚好学总是件好事。

文件组和子树

一般说来，打算用版本控制系统进行管理的项目肯定不会只包含一个文件；比较常见的做法是把某个目录及其子目录里的所有文件都交给版本控制系统去管理。所以，如果能够查看到当前工作目录下所有版本受控文件的清单，就能给工作带来很多便利。要是还能把它们当做一个整体来操作就更有好处了。

这些想法都能用VC模式实现。可以用命令"**C-x v d**"（命令名是**vc-directory**）打开一个编辑缓冲区对目录进行编辑，这个编辑缓冲区运行在一种定制出来的Dired（directory editing，目录编辑）模式下。这个编辑缓冲区不仅列出当前目录下的全部已注册文件，还提供（如果有）哪些文件已被签出，谁是签出人等信息。从Emacs 19.30开始，签出、签入操作对文件状态的影响会立刻在这个清单里反应出来。

如果先在这个Dired编辑缓冲区里（用正常的Dired文本块选取命令）选取了多个文件，然后执行**vc-next-action**或**vc-revert-buffer**命令，VC模式就会对选取的全

体文件执行这个操作。这种做法最适合用在需要同时签入多个文件的场合，而且还有另外一个好处：VC将只弹出一个编辑缓冲区以输入一次修改注释，输入的注释将施用到签入操作创建的每一个修订版。

但 **vc-revert-buffer** 命令相对要保守一些：它会为每个文件提问是否真的想丢弃所做的修改。

从 19.30 开始，有些Dired命令在VC模式下的Dired里被绑定为一些与版本控制有关的操作。比如说，VC模式下的“=”命令进行的是VC模式的 **diff** 操作而不再是Dired的 **diff** 操作；而“**g**”命令刷新的是目录里的全部VC状态区。

差异报告

我们前面讲过，版本控制系统可以生成版本之间的差异报告。VC模式用来完成这一工作的命令是“**C-x v =**”（命令名是 **vc-diff**）。这个命令的常见用法是查看当前工作文件与上一次签入的修订版之间都有哪些不同：让你对自己将会签入哪些修改做到心中有数。

如果你给这个命令加上一个前缀参数，即按下“**C-u C-x v =**”组合键，那么它将提示输入一个文件名和两个修订号，然后把该文件那两个修订号之间的差异查出来并报告给你。如果较早的修订号是空的（即在VC提示输入它时直接按下了回车键），它将默认使用最近一次签入的修订版；如果较晚的修订号是空的，它将默认使用当前的工作文件。

VC还能给出整个项目树文件的差异报告。如果在“**C-u C-x v =**”命令提示输入文件名的时候输入了一个目录名，那么VC将把该目录里全部已注册文件的指定版本之间的差异都报告出来。

根据设计要求，VC生成的差异报告可以通过Larry Wall开发的**patch**软件工具（现代的UNIX操作系统里都有这个工具）转换为一个补丁包。如果你所在的软件合作项目主要以电子邮件为联络手段，这项功能就将是一个莫大的帮助：你可以下载源代码，注册它，进行修改，然后只需一条命令——把所做的修改转换为一个完整的补丁包，再利用电子邮件把它发给其他合作者。

VC给出的差异报告其实会随版本控制系统的不同而变化，因为VC使用了每个系统自己的差异报告器组件，所以这些报告的具体格式会因版本控制系统的不同而有所变化（注3）。但总的说来，这些报告与UNIX命令**diff**的输出格式都比较相似。在本章稍后的内容里，我们将介绍如何对差异报告进行定制。

早期版本的检索

“**C-x v ~**”命令（命令名是**vc-version-other-window**）能够把指定文件的任何一份早期修订版检索出来。修订版将打开这样一个工作文件：工作文件的文件名与注册文件名相同，文件名后缀则给出它的修订号（后缀名的格式是一个英文句号“.”、一个波浪号“~”、修订号，然后又一个波浪号“~”）。换句话说，可以同时打开多个修订版，它们彼此之间不会发生混淆。如果不满足于仅看到差异报告给出的比较结果，可以用这个命令查看早期修订版的全文。

如果你曾设置过Emacs LISP的全局变量**version-control**（VC出现之后，还使用这个变量的人已经很少了），就会激活Emacs的编号备份功能；而Emacs的编号备份功能生成的后缀名与VC修订版工作文件的后缀格式是很接近的。两者之间的区别是**version-control**保存的备份其后缀名只是一个从1开始顺序递增的数字，而修订号的中间却有英文句号等其他字符。

查看修改记录

如果在一个已注册文件上使用“**C-x v l**”（命令名是**vc-print-log**）命令，VC就会弹出一个内容为该文件修改记录的编辑缓冲区。可以用这个命令查看文件各修订版的修改注释。

注册文件

可以用“**C-x v v**”（命令名是**vc-next-action**）命令把文件注册到版本控制系统里，但这个命令会在注册的同时取出一份可编辑的副本。但有时候只想对文件进行注册而不想签出它，这时就需要使用“**C-x v i**”命令（命令名是**vc-register**）。

注3：这个问题稍微有些复杂。事实上，为了让SCCS系统的**sccsdiff**命令在调用次序上与RCS系统的**rcsdiff**命令保持一致，VC用自己的一个脚本程序对前者进行了打包。

插入版本控制块

大多数版本控制系统都鼓励你在自己的文件里嵌入一些能够在签出、签入以及复原操作中自动更新的字符串。这些字符串的作用是在文件里自动插入一些信息，比如文件的当前修订号，最后一次修改文件的人，以及文件的最后一次签入时间等。

文件内容里的版本控制字符串大部分照抄自 VC 放在状态行上的版本信息，剩下的则来自“**C-x v l**”命令（命令名是 **vc-print-log**）。这个功能看起来似乎没有什么用处，但嵌在文件里的版本控制字符串能够出现在经编译得到的程序代码里，对今后的调试和升级会有帮助。

此外，还可以让 VC 根据嵌在文件里的版本控制字符串而不是根据主控文件分析该文件的版本信息（如果主控制文件很大，这种做法可以加快打开文件的速度）。我们将在后面讨论 VC 模式定制办法的有关章节里介绍如何强制实现这种操作。

另外一种情况是，可能需要经常使用Emacs以外的其他软件查看版本受控文件，而在其他软件里是看不到显示版本控制信息的Emacs状态行的。对于这种情况，嵌在文件里的版本控制字符串的价值就体现出来了。为了插入这些字符串，VC专门准备了一条命令。

正在访问一个加有锁的可编辑文件的时候，如果按下“**C-x v h**”组合键（命令名是**vc-insert-headers**），VC就会根据该文件的语法推断出怎样才能把相应的版本控制信息以适当的注释格式插入到文件里并插入这些字符串。VC 对 C 语言代码和**nroff/troff/groff**代码特别熟悉，而且还经常能根据Emacs各主编辑模式所设置的全局变量**comment-start**和**comment-end**做出正确的推断。如果它无法做出更好的判断，就使用“`#-to-\n`”这样的注释格式（shell、awk、Perl 和 tcl 等许多 UNIX 语言的注释使用的都是这种格式）。

C语言代码在插入版本控制信息方面的行为是比较特殊的。在默认的情况下，Emacs不会把C文件的版本控制信息放在程序中的注释里。Emacs会把这些信息保存在一个名为**vcid**的静态字符串变量里。这将使版本控制信息以字符串的形式出现在编译器生成的目标文件里；可以通过C语言的字符串命令查出二进制执行文件是用哪个版本的源文件生成的。

创建和检索项目快照

软件项目往往包含有许多文件，而这些项目文件在开发过程中又会经过多次修订，但并非每次修订都会对全体文件进行修改。这样，在经过一个时期之后，有些文件的修订号已经很大；而有些文件因为不需要经过那么多次的修订，因而修订号还很小。当我们需要把项目交付给顾客或者准备请第三方对项目进行评估的时候，就必须把这些修订号参差不齐的项目文件都放到一个“项目快照（snapshot）”里。换句话说，项目快照就是软件开发项目的阶段性成果。软件的“升级版”或“测试版”其实就是一些项目快照。

随着开发工作的进行，项目文件的修订号会越来越多，也就越来越不好记忆。等到想对项目文件的全体或者一部分做阶段性总结的时候，就会感到无从下手。幸好大多数版本控制系统都已经考虑到这个问题，它们允许给软件项目的阶段性成果起一个符号名称，用这个符号名称把有关的文件全都关联在一起；此后，就可以用这个符号名称来完成检索修订版或者生成差异报告等版本控制工作了。

RCS 和 CVS 本身就有项目快照功能。SCCS 虽然本身不具备项目快照功能，但 VC 模式用自己的代码在 SCCS 系统下进行了仿真。在实际工作中，版本控制系统给项目快照起的符号名称与VC给项目快照起的符号名称看不出有什么不同。用VC仿真出来的项目快照功能只有一个美中不足：如果在VC以外的地方使用SCCS系统，那些 SCCS 工具就不会知道项目快照的符号名称到底代表什么。

“**C-x v s**”命令（命令名是 **vc-create-snapshot**）会提示输入一个符号名称。然后，VC 会把当前目录里所有已注册文件的当前修订版都与这个名称关联起来。

对需要用到修订号的其他 VC 命令来说，用 **vc-create-snapshot** 命令创建出来的符号名称将成为它们的合法参数。项目快照对 **vc-diff** 命令来说特别有用，它意味着将能够生成项目快照彼此之间或者与你签出的工作文件之间的差异报告。“**C-x v r**”（命令名是 **vc-retrieve-snapshot**）以一个项目快照的符号名称为输入参数，把当前工作目录里每一个已注册文件关联在该名称下的修订版取出来。

如果当前目录中某个已注册文件被别人签出，两个快照命令就都会执行失败，快照创建命令会报告无法创建快照，而快照检索命令会报告没有检索到文件。**vc-create-snapshot** 命令执行失败的原因是不想创建出一个无法完整恢复当前状态的项目快

照。**vc-retrieve-snapshot**命令执行失败的原因是不想做无用功，因为肯定需要在签入或复原工作文件之后再进行一次检索。

刷新 ChangeLog 文件

自由软件基金会（Free Software Foundation，FSF）在项目管理方面使用的某些记录方法与SCCS、RCS和CVS等版本控制系统有所不同，为了让它们互通有无，VC模式准备了一条“**C-x v a**”命令（命令名是**vc-update-change-log**）。按照FSF项目开发的要求，每修改一次文件，就应该写下一条修改注释；这些修改注释将按时间顺序记载到一个文件里以备查考。因此，FSF项目的每个目录通常都会有一个名为*ChangeLog*的文件，其内容就是该目录里每个文件的修改注释，而且这些注释都加有时间戳（timestamp）。简单地说，*ChangeLog* 文件就是FSF项目的修改历史；而VC虽然也有类似的功能，但那是通过版本控制系统的修改注释功能来实现的。

用不着把每个修改注释输入两次，VC提供的一个挂钩能够把新添加的修改注释从当前目录下的主控文件里复制出来，并把它们以正确的格式追加到*ChangeLog*文件的末尾。

vc-rename-file 命令：重新命名版本受控文件

重新命名版本受控文件是一件比较麻烦的事。在RCS或SCCS系统里，只把工作文件的名字改过来还不够，还必须把它所关联的主控文件的名字也改过来。在CVS系统下，由于一些很难用三言两语解释清楚的原因，要想不弄坏点东西就给文件改了名字可以说根本就不可能。

vc-rename-file 试图隐藏这个操作的具体细节，并且能够捕获各种可能发生的错误并发出通知。它的用法很简单：先提示输入旧文件名和新文件名，然后尝试完成操作，如果不成功就发出通知。

警告：重命名操作与SCCS系统下仿真出来的项目快照功能很难协调工作。这也是我们推荐使用RCS和CVS系统的重要理由之一。

---

vc-clear-context 命令：当 VC 被弄糊涂时

需要根据文件的版本控制状态决定如何继续执行的文件系统操作其开销往往很大，而且速度也会很慢；在 NFS 等网络化的环境里，这一问题尤其突出。如果用 VC 来做这些事，情况也差不多（除非你不让它这么做，我们马上就会看到）。

它有两个主要的方法：（1）把每个文件的信息（比如锁的主人或修订号等）都放到内存中的缓存里，不采用每次都得用运行版本控制工具把它们从对应的主控文件里分析出来的办法；（2）设法从某个已注册文件的写权限里推导出它的版本控制状态。简单地说，如果某个已注册文件的写权限是“可写”，VC 就将认为它正处于“已签出并已加锁”状态；如果某个已注册文件的写权限是“不可写”，它就不是当前编辑的取出版本。

在多用户环境里，VC缓存的信息和它根据文件访问权限做出的判断往往会让它走上歧途。因为经常有人以手动的方式不经过 VC 就改变文件的权限，所以这种情况常常发生。

如果认为发生了这样的情况，就应该立刻使用 **vc-clear-context** 命令。这个命令将强制 VC 丢弃内存中与当前工作文件的版本控制状态有关的缓存信息。

此外，在理论上还存在着这样一种可能：因为出现了竞争现象（race condition）而把VC弄得不知所措，这种竞争可能出现在两个或者更多个VC之间，也可能出现在 VC 与别人单独运行的 SCCS、RCS 或 CVS 工具程序之间。这种问题不仅 VC 会遇到，两个或者更多单独运行的工具程序也有可能出现竞争现象（虽然概率不大）。不过，这种竞争现象即使是 VC 也很少遇到；在笔者多年的程序设计工作中，从没听说发生过这样的事情。

如果你对这个问题感兴趣，请查看VC的源代码（Emacs LISP源代码目录中的*vc.el*文件）里的一段注释，这段注释对多用户环境下的冲突和竞争现象做了细致深入的分析。但大家应该有充分的信心，VC 是相当成熟和强大的。

对 VC 进行定制

作为 Emacs 的一种辅助编辑模式，VC 的行为方式有很多是可以定制的，这需要我们对某些属于 VC 模式的 Emacs 变量进行设置。我们将介绍几个最重要的。

vc-default-back-end

这个变量负责设定VC缺省使用的版本控制系统，如果它找不到想签入的文件的主控文件，就会使用这个变量设置的版本控制系统进行处理。它的缺省设置是“RCS”，其他合法值是“CVS”和“SCCS”。如果你手里只有SCCS系统或者你的站点要求必须使用CVS系统，就需要修改这个变量。

vc-display-status

如果这个变量的取值不是“nil”，就会在编辑缓冲区的状态行上显示出文件的修订号和版本控制状态。如果你正在通过很慢的网络链接运行VC，就可以考虑把这个开关变量关掉以减少开销巨大的主控文件查询操作。

vc-header-alist

这个变量的作用是通过一个关联列表把版本控制系统后端映射到由**vc-insert-headers**命令插入的字符串上。它的缺省值可以在VC源代码里查到；如果想让自己的版本号标题使用另外一种格式，可以修改这个变量。

vc-keep-workfiles

在正常情况下，VC每执行一次签入操作就会留下工作文件的一个只读副本。这个功能是很有用的，因为**make**和其他工具能够在预期的地方找到工作文件。要是硬盘空间很紧张，就可以考虑关掉这个变量；但以后每当有VC以外的其他工具需要用到这个工作文件时，就不得不明确地执行一次签出操作。(Emacs能够通过一小段长驻内存的VC代码掌握版本控制方面的情况；它能在必要时自动执行取出命令，取出一个只读副本，并且不用加锁。)

vc-mistrust-permissions

这个变量一般设置为“**nil**”。把它设置为“**t**”等于告诉VC不要根据文件的访问权限或属主权限判断它的版本控制状态。这个改变会大大降低VC的执行速度，但如果（比如说）开发工作正在多个不同的目录里同时展开并且工作文件的访问通过符号链接（symbolic link）来实现，可能就需要把这个变量设置为“**t**”。此后，链接的访问权限和属主权限与工作文件的版本控制状态将没有任何关系。

vc-suppress-confirm

这个变量的缺省值是“**nil**”。如果它的取值不是“**nil**”，以后再使用**vc-revert-buffer**复原命令丢弃所做的修改时，VC就不再提示进行确认。

vc-initial-comment

大多数版本控制系统允许（但不要求）对文件进行注册操作时输入一条初始注释，也就是修改记录中的第一句话。如果这个变量的取值不是“**nil**”，VC将在注册过程中弹出一个编辑缓冲区以输入这条注释，就像它在签入操作中要求输入修改注释时的做法一样。

diff-switches

Emacs的*diff.el*模式将在需要生成修改报告的时候把这个全局变量里的命令行开关传递到UNIX操作系统的**diff**命令。VC也以同样的方式使用这个变量。这个变量的缺省值是单个开关“**-c**”，即使用**context-diff**格式；用来设定使用**unified-diff**格式的开关“**-u**”也比较常见。

vc-consult-headers

如果这个变量（通常是“**nil**”）设置为“**t**”，VC就将从嵌入在每个文件里的版本控制字符串处获取文件的状态和版本编号信息，不再查看对应的主控文件。（如果主控文件很大，这种设置将提高VC打开文件的速度，但在激活这项功能之前一定要保证所有的文件都包含正确的版本控制信息块）。

Emacs的在线帮助系统对另外一些不这么重要的全局变量做了详细的说明。

对VC进行扩展

VC的设计思想从一开始就是让它成为多种版本控制系统的一个操作前端。为了使接插新系统的工作容易进行，人们对实际运行版本控制工具的代码与用户级组件逻辑进行了仔细的隔离。VC的作者最初只写了它与SCCS和RCS的接口，对CVS系统的支持是由另外一个人后来添加进去的，其难度也确实不大。

在我们编写这本书的时候，听说有人正在编写VC到ClearCase的接口；ClearCase是一种流行的商业化项目控制系统。考虑到自由软件基金会对非自由软件的敌视态度，那些代码能否被接受到GNU Emacs的发行版本里还是一个未知数。不过，即使自由软件基金会不接受ClearCase所做的修改，它们也会通过别的渠道逐渐进入VC的覆盖范围。

本书只介绍了3种版本控制系统；但等大家读到这本书时，你的VC可能已经能够很好地处理这3种以外的其他版本控制系统了（但在可预见的未来，这3种版本控制系统却极可能仍然是因特网/UNIX世界里最流行的）。如果你是一位熟练的EmacS LISP程序员（或者如果你想成为一个这样的人）并拥有一个你最喜欢的版本控制系统，请钻研它的源代码并对VC进行扩展，让VC也能够使用你的版本控制系统，然后把你的成果拿出来与大家分享。

VC的不足之处

VC决不是项目管理问题的终极解决方案。它能够对独立作者的程序设计或文档管理工作提供极大的帮助，也能够对有多名程序员参加的中小型项目提供重要的帮助。可大型项目往往会涉及到许多组件和许多目录，单凭VC本身的能力是无法胜任的。当用它来管理大型项目时，它的某些不足就会显露出来，其中包括：

- 它没有集成修改请求或问题报告系统。不能及时发现潜在的需要修改的地方。
- 它对项目文件的组织管理手段仅限于目录子树。这种局限性在面对涉及到多个目录的大型项目时可能会出现一些问题，特别是当两个或者更多目录需要共享一个开发库或者子树的时候。

这些不足在小型项目上很容易得到解决，比如说，变体版本可以用编译条件来对付，比如C语言中的“**#ifdefs**”语句；修改请求可以单独保存在某个数据库（比如FSF的GNATS系统）里。程序员可以把需要进行重命名操作的开发阶段条理清晰地记在脑子里。

随着项目在规模和复杂程度上的增加，还想用这些小手段来解决所有的有害的冲突情况就不那么容易了；而小问题积累多了，就会造成致命的损害。要想对超大规模项目的复杂性加以控制，就需要有一个基础更强大（因而约束性和复杂性也更大，这是没有办法的事）、功能比版本控制系统更多的支持环境，即一个完备的项目管理系统。

如果想了解更多关于项目管理系统的设计问题，请阅读《Applying SCCS and RCS》一书的后半部分。

有效地使用 VC

最后，我们希望有版本控制经验的读者留意这句话：要想有效地使用VC，就一定要尽早和及时地签入所做修改。

如果你习惯于笨重而又难用的版本控制接口，比如独立的SCCS、RCS或CVS，那么你过去养成的习惯可能会让你无法充分地享受到VC的好处。你可能不习惯经常存盘；不习惯经常查看某个文件子树的状态报告；也常常想不起利用项目快照之类的功能来总结阶段性的成果。

多花点时间和精力对自己进行再教育是值得的。你将发现，VC决不是一种无关紧要的辅助工具，VC下的版本控制是一种极大的解放。如果你经常签入所做修改，就会发现自己将能承受更多探索性质的尝试，因为你有把握在必要时能够迅速返回到一个已知的良好状态。

不管是在项目快照之间还是在项目快照与修改的工作文件之间，用VC给程序打补丁能够做到又快又完整；当认识到这一点时，你对分布式开发的想法肯定会有所转变。松散型的开发团队往往会因沟通不足而降低效率，尤其是当程序员各自保有一份源代码的本地副本而主要的通信手段又仅仅是电子邮件时将更是如此；VC提供的打补丁功能将大大改善这类开发团队的工作情况。经验表明，虽然VC还远不是这类沟通问题的一个完美解决方案，但与依靠个人技术完成小修小补的旧时代相比，这是前进道路上的重要一步。

第十六章

在线帮助

本章内容:

- Emacs 的自动补足功能
- 帮助命令
- 针对复杂 Emacs 命令的帮助功能

就文本编辑器而言，Emacs 的在线帮助功能是最全面的；即使与其他种类的程序相比，它的帮助功能也算得上是最好的。如果从规模上比较，编写 Emacs 在线帮助文档所花费的时间肯定比我们编写这本书所花费的时间要长。Emacs 的在线帮助功能主要由 3 个部分组成:

- 自动补足功能 —— Emacs 能帮助完成事物名称（比如文件名）的输入工作。
- 帮助键（通常是“**C-h**”）—— 它可以查看关于各种问题的帮助，还能让你使用 **info** 数据库和 Emacs 教程等工具。
- 针对复杂命令的帮助功能 —— 比如 **query-replace** 和 **dired** 等命令。

Emacs 的自动补足功能

我们在第一章就见识过 Emacs 的自动补足功能。Emacs 的自动补足功能远不止是一项功能：应该说它是 Emacs 设计工作中的一项基本原则。我们可以把它表述如下:

> 如果需要输入某个事物的名称，而这个名称是有限个数的可能性中的一种，Emacs 就能在输入可能最少个数的字符后判断出想输入的是什么。

换句话说，只需输入一个“最短无二义前缀”就足够了，名称的剩余部分由 Emacs

来补足。所谓“最短无二义前缀”的含义是“从名称中的第一个字符开始、到足以把该名称与其他可能性区分开来的字符”。Emacs比较重要的几样事物都有从有限个数的可能性中挑选出来的名称，其中包括:

- 命令
- 某给定目录里的文件
- 编辑缓冲区
- Emacs 变量

当Emacs提示需要在辅助输入区里输入某个事物的名称时，几乎总是可以利用它的自动补足功能。在输入事物名称的过程中，如果想让Emacs来帮助补足它，那么有3个字符可供选用，它们是制表符（**TAB**）、空格（**SPACE**）和问号（**?**）。这3个字符的作用是:

TAB 尽可能多地补足这个名称

SPACE 补足这个名称直到下一个标点符号

? 以此前输入的字符为依据，把可用选择列在“*Completions*”窗口里

这3个字符里最有用的是**TAB**。

举个常见的例子。假设已经按下“**C-x C-f**”组合键以访问一个文件，准备访问的文件是一个名为“*program.c*”的C语言程序。输入“**pro**”之后按下**TAB**键; Emacs会自动把这个文件名补足为“*program.c*”。如果按下的是空格键，Emacs就只补足为“*program*”。在Emacs补足这个文件名之后，按下回车键后就可以打开这个文件。

在使用自动补足功能前需要输入多少个字符呢？这要看当时的其他可能性情况。如果“*program.c*”是目录里惟一的文件，输入字母“**p**”后马上按**TAB**键都行（注1）。如果目录里还有其他文件，可它们的名字都不是以字母“**p**”开头的，也可以像刚才那样做。但要是还有一个名为“*problem.c*”的文件，就只能先输入“**prog**”后才能按下**TAB**键;“**prog**”正是此例中最短的无二义前缀。如果只输入“**pro**”就按下**TAB**

注1: 不能没输入字母“**p**”就按**TAB**键，这是因为当前目录和它的父目录也是文件名候选，它们的名字分别是“.”和“..”。可如果你访问的文件是一个目录，Emacs就会运行**dired**命令。

键，Emacs将打开一个“*Completions*”窗口，里面是供自动补足功能选取的各种可能性——就此例而言就是“*program.c*”和“*problem.c*”，然后，Emacs会把光标放回到辅助输入区里以便完成文件名的输入工作。如果按下的是问号而不是**TAB**键，情况将与刚才一样。此时，再输入一个字母“**g**”，然后按**TAB**键；Emacs就能自动补足出文件名“*program.c*”了。

再来看一个例子。假设你为自己的C语言程序编写了一份文档，文档存放在文件“*program.doc*”里；而你现在想访问它。按下“**C-x C-f**”组合键，输入“**prog**”后按下**TAB**键。Emacs自动补足出“*program.*”（注意里面的句点）。此时，可以再输入一个字母“**d**”并按下**TAB**键；Emacs将自动补足出完整的“*program.doc*”。换句话说，在输入事物名称时可以反复使用自动补足功能。

最后，假设你的目录里还有一个就叫做“*program*”的文件，它是你的C文件经过编译得到的结果，但你现在想访问的仍是文档文件。于是，输入“**prog**”后按下**TAB**键；Emacs自动补足出“*program*”。从这里开始，**TAB**键和空格键做的事就不一样了。如果再次按下**TAB**键，Emacs将在辅助输入区里显示一条信息“[Complete, but not unique]”（自动补足，但不惟一）；但如果按下的是空格键，Emacs会认为你对文件“*program*”不感兴趣并将继续进行自动补足。因为你有两个分别叫做“*program.c*”和“*program.doc*”的文件，而此时Emacs只补足出了“*program.*”，所以你必须再敲入字母“**d**”并按下**TAB**键才能打开“*program.doc*”。

自动补足功能也同样适用于编辑缓冲区的名字，比如按下“**C-x b**”组合键以切换到当前窗口另一个编辑缓冲区的时候。它还能用来为“**ESC x**”组合键自动补足Emacs的命令名——但此时它将多出一项功能。当你准备输入的是一个文件名或者一个编辑缓冲区名时，你想打开的文件/编辑缓冲区可能还不存在（比如想创建一个新文件的时候）。如果是这种情况，当然得由你本人来输入文件名或者编辑缓冲区名，然后按回车键。但当你准备用“**ESC x**”组合键来执行Emacs编辑命令时，这个命令肯定是已经存在的。因此，即使按的是回车键，Emacs也将开始对命令名进行自动补足。

比如说，如果想把某个文本文件的编辑缓冲区放到自动换行模式（参见第二章）里，就可以在输入“**ESC x auto-f**”之后按下回车键，不必输入完整的“**ESC x auto-fill-mode**”。如果输入的命令前缀不惟一（有二义性）——假设输入的是“**ESC x aut**”并按下了回车键，则这个回车键的作用就相当于**TAB**键；就这个例子而言，

它将自动补足出“**auto-**”。如果再按一次回车键，Emacs将打开一个“*Completions*”窗口列出可用的选择，也就是“**auto-fill-mode**”和“**auto-save-mode**”。如果想执行的命令是“**auto-fill-mode**”，就需要再输入一个字母“**f**”并再次按下回车键。

针对命令名的回车键自动补足功能会给工作带来很大的方便。在用过Emacs一段时间之后，你对自己常用命令的最短无二义前缀就应该比较熟悉了；只需输入这些前缀而不是完整的命令名将大大减少你的输入工作量（注2）。

Emacs还能自动补足Emacs变量的名字。在第二章和其他一些章节里，我们已经见过很多用“**ESC x set-variable**”命令来改变Emacs变量值的例子。而刚才介绍的回车键自动补足功能不仅适用于命令名，也同样适用于变量名；这就是说，在“**ESC x set-variable**”操作中也能使用自动补足功能。事实上，命令和变量都是特殊形式的Emacs LISP符号，而Emacs能够用回车键对一切类型的LISP符号进行自动补足。当你需要在工作中使用本章介绍的那些帮助命令时，LISP符号上的自动补足功能将给你带来很大的方便。

对自动补足功能进行定制

如果你读过第十一章并且愿意尝试着对Emacs变量进行设置，就应该知道有几个变量是可以用来定制Emacs的自动补足功能的。变量**completion-auto-help**将决定是否要在按下空格键或TAB键而此前输入的却是一个二义性前缀时自动打开一个“*Completions*”窗口。它的缺省值是“**t**”，意思是自动打开这个窗口。如果把它设置为“**nil**”，Emacs就不会自动打开一个“*Completions*”窗口，它将在辅助输入区（minibuffer）里显示“[Next char not unique](下一个字符不惟一)”信息几秒钟。

如果你是一位程序员或者你使用TEX等文本排版软件，你创建的就将是一些不适合人类阅读的文件，比如编译器生成的目标文件或者文本排版软件生成的打印文件等。一般说来，当你使用Emacs的自动补足功能时，你是不想让Emacs把这类文件也列为候选的；比如说，如果你有两个文件“*program.c*”和“*program.o*”（编译器输出的目标代码），那么你通常只想让Emacs列出前者做为候选。Emacs也确实提供了

注2：举个例子。如果你需要经常修改你的*.emacs*文件，就可以用“**ESC x eval-c**”前缀做为“**ESC x eval-current-buffer**”命令的替代品。

这样的功能；事实上，你可能已经注意到这种情况了：如果输入“*program*”并按下**TAB**键，Emacs将忽略“*program.o*”而只补足出“*program.c*”。这种行为是由变量**completion-ignored-extensions**控制的；它是一个由文件名后缀构成的列表，Emacs在补足文件名时将不把以它们为后缀的文件名列为候选。在默认的情况下，这个列表收录着Emacs备份文件使用的波浪符（~）、程序员使用的“*.o*”、TEX用户使用的各种文件名后缀、Emacs LISP程序员使用的“*.elc*”（经过字节编译的Emacs LISP代码），以及其他一些文件名后缀。（当然，如果你确实想查看这些文件，可以亲手输入它们完整的文件名。）

如果想把自己的“忽略”文件名后缀添加到这个列表里，就需要把一条下面这样的语句添加到“*.emacs*”文件里：

```
(setq completion-ignored-extensions
      (cons "suffix" completion-ignored-extensions))
```

比如说，假设你正用一台PostScript打印机对文本进行排版，文本排版软件将生成一个后缀名为“*.ps*”的打印文件。你不想让自动补足功能把这些打印文件也列为候选，所以你必须将下面这条语句添加到“*.emacs*”文件里：

```
(setq completion-ignored-extensions
      (cons ".ps" completion-ignored-extensions))
```

最后，如果把变量**completion-ignore-case**设置为“**t**”（或者任何不是“**nil**”的值），Emacs将在使用自动补足功能时忽略字母的大小写情况。这个变量的缺省值是“**nil**”，意思是Emacs将区分字母的大小写。

帮助命令

Emacs有很多帮助命令，它们有的是标准的Emacs命令，有的是“**C-h**”帮助键下的选项（注3）。它们可以用来查找关于命令、键盘操作、变量、编辑模式，以及各种Emacs事物的信息。最基本的帮助命令是“**C-h C-h**”（命令名是**help-for-help**）；这个命令的第二项必须是“**C-h**”，不管你是否重新绑定过帮助键。“**C-h ?**”组合键也能用来执行**help-for-help**命令。这个命令将使Emacs在一个窗口里打开“*Help*”编辑缓冲区，里面列出的是关于各种帮助命令的说明。可以敲入其中的

注3：如果你重新绑定帮助键，就需要把本章后面内容里的“**C-h**”都看做你的新绑定，但也有几个例外，我们会在遇到它们的时候告诉你。

某个帮助键查看相应的帮助信息；如果按的是空格键，“*Help*”窗口将下卷，就好像按的是“**C-v**”组合键一样。按任意其他键将关闭“*Help*”窗口。如果到达帮助文档的末尾，按一个帮助键或任何其他键都将退出操作。

列在“*Help*”窗口辅助输入区消息里的是这样一些按键：如果在敲入帮助键之后又按下这些键，就会执行相应的Emacs帮助命令。我们将依次对这些按键进行说明。帮助命令又可分为两大类：一类负责提供特定问题的答案，另一类负责给出关于Emacs的概括性资料。

在对Emacs比较熟悉之后，你将发现第一类帮助命令对自己更有帮助。因为它的规模是如此的巨大而功能又是如此的丰富，所以你经常会遇到这样的情况：需要在里面查找某个键盘操作命令或编辑命令的细节、或者想用Emacs完成某项工作，可又不知该如何下手。正如我们在书中反复强调的那样，你想要的东西几乎都能在Emacs里找到，但问题的关键是怎样才能找到它们。帮助命令把这种查找工作变得既快捷又简单，你既不需要离开Emacs，也不必做一名参考手册的奴隶。

详细资料

我们从用来查找某些细节的帮助命令开始讲起。表16-1里是一些最常用的帮助命令。

表16-1：用来查找详细资料的帮助命令

键盘操作	命令名称	回答的问题
C-h c	**describe-key-briefly**	这个按键组合将运行哪个命令
C-h k	**describe-key**	这个按键组合将运行哪个命令？这个命令的作用是什么
C-h l	**view-lossage**	最近输入的100个字符是什么
C-hw	**where-is**	这个命令的按键绑定是什么
C-h f	**describe-function**	这个函数的作用是什么
C-h v	**describe-variable**	这个变量的含义是什么？它有哪些可取值
C-h m	**describe-mode**	查看当前编辑缓冲区所在编辑模式的有关资料
C-h b	**describe-bindings**	这个编辑缓冲区都有哪些按键绑定
C-h s	**describe-syntax**	这个编辑缓冲区使用的是哪个语法表

最需要使用在线帮助功能的场合大概是按错键的时候：编辑缓冲区里出了点问题，可你却不知道下一步该怎么办。通常，最安全的方法是按下“**C-x u**”组合键（命令名是 **undo**）撤销刚才的操作。但有几种情况（例如一个与你愿望背道而驰的 **replace-string**命令）是这个命令也帮不上忙的。如果你还能想起自己刚敲入的组合键是什么，就可以用“**C-h c**”组合键（命令名是 **describe-key-briefly**）来看看它到底执行了哪个命令；在提示符处再按一次刚才误按的组合键，Emacs将把绑定到这个组合键上的命令显示在辅助输入区里。如果仅知道命令名还不够，请按下“**C-h k**”组合键（命令名是 **describe-key**），它将弹出一个“*Help*”窗口，里面除命令的名称和按键绑定之外还会给出一个比较详细的说明。要是你想不起自己刚才误按了哪些键，就需要求助于“**C-h l**”组合键（命令名是 **view-lossage**）。这个命令将弹出一个“*Help*”窗口，里面是最近输入的100个键；引发错误的（组合）键应该位于比较靠后的位置；接下来，你可以用“**C-h c**”和“**C-h k**”组合键来查出它的功用。

如果你想了解一个没有按键绑定的命令的情况，请按下面的方法操作。按下“**C-h f**”组合键（命令名是 **describe-function**）并在提示符处输入命令的名称；Emacs将打开一个“*Help*”窗口，里面是关于这条命令的文档。如果你还记得命令名却忘了它的按键绑定，请按下“**C-h w**”组合键（命令名是 **where-is**）。它的作用正好与“**C-h c**”组合键“相反”，如果给定命令有按键绑定，它就会把那个绑定显示在辅助输入区里；如果给定命令没有按键绑定，就在辅助输入区里显示下面这条信息：

```
command-name is not on any keys
```

有时候，你可能忘了某个变量应该如何设置，比如说，哪个值会让Emacs的查找操作区分或者忽略字母的大小写（由变量 **case-fold-search** 控制）？我的编辑缓冲区每隔多长时间会自动存盘一次（由变量 **auto-save-interval** 控制）？如果你按下“**C-h v**”组合键（命令名是 **describe-variable**）后再输入那个变量的名字，Emacs就会把它的可取值和它的文档显示在一个“*Help*”窗口里。“**C-h f**”、“**C-h w**”和“**C-h v**”都允许在输入命令名/变量名的时候利用Emacs的自动补足功能。“**C-h f**”和“**C-h v**”组合键对Emacs LISP程序员也特别有用；特别是“**C-h f**”，它会把全部函数的资料都查出来，而不仅限于绑定到按键组合上的那些命令。

另一种需要帮助的情况是：你正在某个特殊的编辑模式中进行着工作，比如shell模式、某种程序设计语言或字处理器的专用编辑模式等，可怎么也想不起它的某个专

用命令或者其他诸如缩进样式之类的细节了。如果在运行某个编辑模式的编辑缓冲区里按下“**C-h m**”组合键（命令名是**describe-mode**），Emacs就会弹出一个“*Help*”窗口，里面显示着这个编辑模式的文档。编辑模式的文档通常包括以下内容：它的局部按键绑定（即该编辑模式的专用命令和与之关联的按键组合）、它的定制变量以及其他一些有意思的东西。

表16-1中的最后两条帮助命令普通用户很少会用到，但它们对技术娴熟的Emacs定制者却有很大的帮助。“**C-h b**”组合键（命令名是**describe-bindings**）会把活跃在当前编辑缓冲区里的各种按键绑定（包括局部（即该编辑缓冲区专用的）绑定和全局绑定）都显示在一个“*Help*”窗口里；如果你的Emacs是运行在X窗口系统里的，则鼠标动作、菜单项选取等操作的绑定（参见第十四章）也都会显示在这个“*Help*”窗口里。

“**C-h b**”的输出内容相当多。如果只想查看那些有特定前缀的按键绑定的资料，就需要在“**C-h**”的前面输入相应的前缀。比如说，“**C-x C-h**”组合键打开的“*Help*”窗口里将只列出以“**C-x**”开头的全部按键绑定。

“**C-h s**”组合键（命令名是**describe-syntax**）打开的“*Help*”窗口将给出当前编辑缓冲区所使用的语法表（*syntax table*，参见第十二章）的有关说明。

apropos 命令

当你想用Emacs来完成某项工作却又不能肯定应该使用什么命令或者应该设置哪些变量的时候，就需要使用另一个帮助命令。这个命令就是**apropos**，一个在功能和用法上都与很多软件库所使用的基本信息检索系统非常接近的Emacs帮助命令。**apropos**命令有3种形式，我们把它们汇总在表16-2里。

表16-2：apropos命令

键盘操作	命令名称	回答的问题
C-h a	**command-apropos**	这个概念都涉及到哪些命令
（无）	**apropos**	这个概念都涉及到哪些函数和变量
（无）	**super-apropos**	哪些函数和变量的文档里提到了这个概念

这3个命令都要求用正则表达式（参见第三章）回答它们的提示。按下“**C-h a**”组合键并给出一个正则表达式时，Emacs会把所有与它匹配的命令都找出来；它会把它们的按键绑定（如果有）和文档的第一行显示在一个“`*Help*`”窗口里。（Emacs命令和变量的文档基本上都是这样写的：第一行是一句概括，其余部分是详细说明。）如果不愿意使用正则表达式，也可以使用普通的查找字符串代替，但先决条件是不涉及特殊字符。比如说，如果想知道Emacs都支持哪些替换命令，请按下“**C-h a**”组合键并输入“**replace**”；Emacs会把关于下面这些命令的资料显示出来：

- **query-replace**
- **query-replace-regexp**
- **replace-buffer-in-windows**
- **replace-regexp**
- **replace-string**
- **tags-query-replace**

如果你曾经使用过某种信息检索系统，就该知道要想用好一个这样的系统必须掌握一些技巧。你必须认真挑选概念（即查找字符串），让它们既不至于太宽泛（以免产生太多无用的输出），也不至于太细致（以免因输出结果太少而找不到想要的资料）。准备使用**apropos**命令的时候，这个问题会变得更加突出：**apropos**命令与**command-apropos**原理相同，但信息量更大，它不仅会查找出有关的命令，还会把与此有关的全部函数（包括Emacs的内部函数）和全部变量都找出来。**super-apropos**命令在这个问题上是最突出的，因为它的查找范围在函数和变量之外又增加了它们的文档。

如果输入的查找字符串过于宽泛，不仅Emacs为容纳这些帮助信息而创建的编辑缓冲区会过于巨大，整个操作的时间也会非常的长。比如说，如果给**apropos**命令的参数是“file”或“buffer”，就能让Emacs“昏迷”一段时间（这要看计算机的速度有多快了），而且罗列在输出内容里的Emacs数据对象可能会有100多个。如果用**super-apropos**命令查找关于“buffer”的资料，它的输出将超过6000行！一般来说，可能需要调用好几次**apropos**命令才能找到真正想要的东西。

需要用到**apropos**和**super-apropos**命令的场合并不多，除非你是一位Emacs LISP程序员并且确实需要查找某些非命令函数的资料。但是，因为Emacs并没有提供

variable-apropos 之类的命令，所以如果想查找关于变量的资料，就肯定要使用这两个命令。我们认为 Emacs 应该有一个这样的命令，所以我们自己动手写了一个 LISP 函数，下面就是它的代码；这段代码是从 **command-apropos** 命令的代码里直接节选出来的。我们还给出了把这个命令绑定到“**C-h C-v**”组合键上的 LISP 代码。如果你喜欢，可以把这段代码照抄到你的“*.emacs*”文件里；这样，你就可以在以后的 Emacs 工作中使用这个命令了：

```
(defun variable-apropos (string)
  "Like apropos but lists only symbols that are names of user-modifiable
variables. Argument REGEXP is a regular expression
that is matched against user variable names. Returns list of symbols and
documentation found."
  (interactive "sVariable apropos (regexp): ")
  (let ((message
         (let ((standard-output (get-buffer-create "*Help*")))
           (print-help-return-message 'identity))))
    (if (apropos string t 'user-variable-p)
        (and message (message message)))))

(define-key help-map "\C-v" 'variable-apropos)
```

作为这个命令的一个例子，请按下“**C-h C-v**”组合键，然后在提示符处输入“**auto-save**”；Emacs 将给出关于 **auto-save-default**、**auto-save-interval**、**auto-save-timeout**、**auto-save-visited-file-name** 和 **delete-auto-save-files** 等变量的资料。如果想查看其中某个变量的可取值和完整的说明，请使用“**C-h v**”命令。

一般性资料

剩下的帮助命令负责提供关于 Emacs 的一般资料。我们把这些命令汇总在表 16-3 里。

表 16-3：用来查找一般性资料的帮助命令

键盘操作	命令名称	动作
C-h t	**help-with-tutorial**	运行 Emacs 教程
C-h i	**info**	启动文档阅读器 **Info** 程序
C-h C-f	**Info-goto-emacs-command-node**	启动文档阅读器 **Info** 程序，并前进到指定的结点；这个帮助命令的参数是一个命令名

表 16-3: 用来查找一般性资料的帮助命令（续）

键盘操作	命令名称	动作
C-h C-k	**Info-goto-emacs-key-command-node**	启动文档阅读器 **Info** 程序，并前进指定的结点；这个帮助命令的参数是对应于某个命令的按键组合
C-h n	**view-emacs-news**	查看关于Emacs新增和改进功能的信息
C-h F	**view-emacs-FAQ**	查看 Emacs 的常见问题答疑文件
C-h p	**finder-by-keyword**	沿着这个命令弹出的菜单可以查到关于安装在本系统上的 Emacs LISP 程序包的资料
C-h C-c	**describe-copying**	阅读 Emacs 的“通用公共许可证”（General Public License，参见附录六）
C-h C-d	**describe-distribution**	阅读从自由软件基金会订购Emacs的信息（参见附录一）
C-h C-p	**describe-project**	查阅关于 GNU 软件开发项目的信息
C-h C-w	**describe-no-warranty**	查阅Emacs的免责条款（参见附录六中的“NO WARRANTY”部分）

对初学者来说，这些命令中最重要的是“**C-h t**”（命令名是**help-with-tutorial**），它会把屏幕上多余的窗口都删除（只留下一个）并开始教授使用 Emacs。事实上，它会在窗口里显示一个名为“*TUTORIAL*”的文件。Emacs 初学者可以通过这个教程熟悉以下一些 Emacs 功能的使用方法:

- 基本的光标移动
- 删除和恢复
- 打开和保存文件
- 编辑缓冲区
- 文本模式和自动换行模式
- 递增查找

PDG

- 基本的帮助命令

读者可以把这个教程与本书的第一章和第二章结合起来学习。这个教程很有帮助作用，虽然它只介绍了最基本的操作，但却是非常必要的。

如果想查阅更有深度的一般性文档，请按下“**C-h i**”组合键（命令名是**info**），这个命令将启动文档阅读器**Info**程序。这个浏览器有点类似于现代的超文本系统（注4），可以用来查看Emacs（以及其他GNU软件）的文档资料。它是由信息结点（node）组成的一棵信息“树（tree）”。如果想查阅关于某个问题的资料，就需要选中与之对应的信息树；这棵信息树的结点则包含着这个问题的分论点、分分论点…以此类推；整个文档形成一种树状的结构。

按下“**C-h i**”组合键之后，Emacs将在一个**Info**模式下的窗口里创建一个只读编辑缓冲区，里面是**Info**系统的目录结点（directory node）。虽然有关这个系统详细的使用方法超出了本书的讨论范围，但**Info**系统的自备文档很充足。如果在**Info**系统里按下“**h**”键，就会进入**Info**系统的教程，这个教程与我们刚才介绍的Emacs基本命令教程很相似。在这个编辑缓冲区里按下“**C-h m**”组合键（命令名是**describe-mode**）就能获得关于**Info**模式的帮助信息；它对**Info**模式下的各种命令进行了汇总和介绍。

Emacs帮助命令“**C-h C-f**”（命令名是**Info-goto-emacs-command-node**）和“**C-h C-k**”（命令名是**Info-goto-emacs-key-command-node**）能够进行更有针对性地使用。可以把它们看做“**C-h f**”（命令名是**describe-function**）和“**C-h k**”（命令名是**describe-key**）在**Info**系统里的等效命令；它们的作用是启动**Info**系统并根据给出的参数直接到达**Info**系统的某个结点，“**C-h C-f**”查阅用其参数指定的命令，“**C-h C-k**”查阅绑定到指定按键组合上的命令。

对忠实的Emacs用户和定制者来说，剩下的Emacs帮助命令中最重要的大概就是“**C-h n**”（命令名是**view-emacs-news**）了，它将打开Emacs自带的文件*NEWS*以供查阅。这个文件其实是一个修改记录，自上一个主版本（major version）推出之后又在Emacs上做的改进都会记载在里面，比如Version 18到Version 19之间的全部修改以及Version 19到最新的次版本（minor version，比如笔者使用的Version

注4：事实上，它是具有实用价值的超文本系统中最早出现的成功范例之一。

19.30）之间的全部修改。如果自上一个主版本推出之后曾经有过好几个次版本，这个文件的长度就会相当可观——笔者手里的这个文件有2500多行。如果想从这个文件里找出某个具体的Emacs修改事项，那么还是使用一个适当的查找命令比较好。如果只是随便看看，请注意它是按大纲模式的要求缩进的——主论点全都以“*”开头，分论点全都以“**”开头，以此类推。因此，最好是用大纲模式来浏览，具体做法请参考第八章中的有关内容。

针对复杂Emacs命令的帮助功能

许多比较复杂的Emacs命令都自备有一套专用的键盘命令。而这些命令又几乎都自备有它们自己的帮助功能，但用来启动这类帮助功能的键盘命令是问号（?）而不是标准的Emacs帮助键。我们把“?”在几个常用的复杂命令中的作用汇总在下面：

dired（“C-x d”组合键）

辅助输入区里将列出一些最常用的命令。注意：这个清单并不完整。按下“**C-h m**”组合键（命令名是**describe-mode**）可以看到更全面细致的文档。

query-replace（“ESC %”组合键）

它的子命令清单将显示在一个“*Help*”窗口里。按下“**C-h**”组合键（不管是否重新绑定过帮助键）也有同样的效果。这种做法也适用于**query-replace-regexp**命令。

save-some-buffers（“C-x s”组合键）

与刚介绍的**query-replace**命令中的情况差不多。

list-buffers（“C-x C-b”组合键）

你会在一个“*Help*”窗口里看到关于编辑缓冲区菜单模式的帮助信息。这个命令与按下“**C-h m**”组合键（命令名是**describe-mode**）的效果相同。

自动补足功能

根据Emacs给出的提示在辅助输入区里输入事物的名称时，如果Emacs能够对它进行自动补足，就可以随时输入“?”。这将打开一个“*Completions*”窗口，里面是当时可用的各种候选项。自动补足功能我们已经在本章的开头部分介绍过了。

附录一

如何获得 Emacs 软件

正如我们在本书前言里所说的那样，Emacs 是一个自由软件。这并不意味着可以完全免费地得到它，但自由软件基金会（Free Software Foundation，FSF）努力让大家以最可能容易的方式获得 Emacs。如果从 FSF 那里获得 Emacs，那么它会收取一定的费用 —— 但收费的数额只是同类软件价格的一个零头，其目的是为了支付存储介质、运输和手续方面的成本。如果从朋友那里获得一份复本更方便，FSF 是鼓励你这样做的。如果你的周围环境里有很多机器（比如一家大公司或者大学里的某个系）并且已经有人拥有了 Emacs，从他或她手里获得 Emacs 肯定更受欢迎。

Emacs 能够运行在超过 70 种的 UNIX 版本上；它还已经被移植到 VAX/VMS、NextStep、Macintosh、Microsoft Windows 95 和 NT 等系统上，甚至被移植到 MS-DOS 上。如果你必须从本附录所介绍的这几个来源获得 Emacs，那么，从源代码（即用程序设计语言 C 和 LISP 写出来的程序）开始来建立它的工作可能就需要由你本人来完成。别紧张！这工作并不要求你非得是一名程序员。问题的关键是 Emacs 源代码的规模非常大，而你必须根据自己操作系统和硬件的组合情况对它们进行正确的配置。这个过程很消耗时间，但就大多数情况来讲都不复杂。

用 FTP 从因特网上下载

如果你周围没有方便的 Emacs 来源而你又不想增加开支，那么最简单的办法就是从因特网上下载。如果你能够上网，就可以从众多站点中的某一个站点获得

Emacs —— 不过，你的网络连接必须快得足以传输一个 10 MB 字节的文件才行。你的硬盘至少要有 105 MB 剩余空间；今后发行的Emacs的规模肯定会超过这个数字。因特网上有许多站点都主动向外提供 Emacs 和其他一些 FSF 软件的源代码。

从某个因特网站点下载 Emacs 需要使用 FTP。需要特别说明的是，应该使用 FTP；支持匿名FTP的站点允许任何人连接，不过可能会有一些（可以理解的）访问限制规定。FTP 有 3 种使用方式，可以任选一种：

- 有一个 WWW 浏览器，比如 Mosaic 或 Netscape Navigator 等，它们能够支持 FTP 并提供图形化的用户操作界面。
- 在系统上实现因特网TCP/IP协议的软件自备一个图形化的FTP用户操作界面。
- 作为最低要求，至少应该能够从 UNIX 操作系统的 shell 命令提示符处访问 FTP。

有许多不同的站点都能够提供Emacs的源代码，但最正宗的站点是位于麻省理工学院（MIT）的 *prep.ai.mit.edu*，文件路径是 */pub/gnu* 子目录。你应该先尝试从这个站点进行下载，尤其是当你身处美国的时候。如果你不在美国，或者从这个站点进行下载时遇到了问题，世界各地还有很多 *prep.ai.mit.edu* 的镜像站点。

自由软件基金会有一份所有镜像站点的名单。如果你想获得这份名单以及与获取 Emacs 和其他 FSF 软件有关的其他有用信息，请输入命令“**finger fsf@prep.ai.mit.edu > fsf-info**”。你将得到一个名为 *fsf-info* 的文件，里面包含着全部信息（当然，可以使用一个不同的文件名）。目前，这份名单开列了二十几个 FTP 站点，它们分布在加拿大、欧洲、南美洲、南部非洲和环太平洋地区。

不管访问的是哪一个站点，都必须在一个给定的目录里找到一个名为 *emacs-19.N.tar.gz* 的文件，其中的 *N* 是一个数字。这是 Emacs 发行版本的压缩档案；*N* 是次版本号。如果这个名字的文件不止一个，那么请下载辅版本号最高的文件。这个文件的长度大概是 10 MB。

下面是一个使用WWW浏览器下载Emacs的例子。我们假设使用Netscape Navigator 从站点 *prep.ai.mit.edu* 处进行下载。

如果 Netscape Navigator 还没有启动，请启动之。点击“**Location**（地址）”栏并输

入 "*ftp://prep.ai.mit.edu/pub/gnu*"。这是该 FTP 站点的 URL 地址（Universal Resource Locator，统一资源定位器）；FTP站点的URL地址格式是"*ftp://site.name/directory*"（注 1）。按回车键让 Netscape 链接到这个 URL 地址。稍等一会儿，就应该链接到 *prep.ai.mit.edu* 站点上。

在Netscape完成主页加载工作之后，你将看到一个文件清单。根据我们刚才给出的文件名在清单里找到那个文件。在文件名上点击鼠标键，屏幕上弹出一个对话框，询问是立即运行或保存这个文件、取消操作、还是配置一个"查看器"以运行这个文件。选择保存，然后选择用来保存这个文件的目录。点击"**OK**（确认）"，Netscape 将开始进行文件传输。

如果你没有Web浏览器或其他图形化界面的FTP工具，就只能从UNIX操作系统的 shell命令行来执行FTP操作。下面就是一次完成这项工作的UNIX会话过程；你输入的命令用粗体表示，系统响应则用普通字体表示。此外，我们将假设你的UNIX 提示符是"$"。

注意：作为一个良好的因特网公民，你应该在口令字提示符处输入你真实的电子邮件地址；不要直接按回车键。

```
$ ftp prep.ai.mit.edu
Connected to prep.ai.mit.edu.
220 aeneas FTP server (Version 4.136 Mon Oct 31 23:18:38 EST 1988) ready.
Name (prep.ai.mit.edu:): anonymous
331 Guest login ok, send ident as password.
Password:  your full email address
(An introductory message appears here.)
ftp> binary
200 Type set to I.
ftp> cd /pub/gnu
250 CWD command successful.
ftp> ls emacs-19.[0-9][0-9].tar.gz
200 PORT command successful.
150 Opening data connection for /bin/ls (128.119.40.122,2653) (0 bytes).
emacs-19.28.tar.gz
emacs-19.29.tar.gz
emacs-19.30.tar.gz
226 Transfer complete.
remote: emacs-19.[0-9][0-9].tar.gz
60 bytes received in 0.014 seconds (4.2 Kbytes/s)
```

注 1：　在某些Web浏览器上，这个格式是"*ftp // site. name: /directory*"。

在上面的文件名里，辅版本号（28、29和30）可能会有所不同，但我们想下载的是最新的版本（本例中就是30）。

```
200 PORT command successful.
150 Opening BINARY mode data connection for emacs-19.30.tar.gz (10039387 bytes).
```

当文件传输开始之后，根据你因特网链接的速度，可能需要等待一会儿；可能是几分钟（对T-1专线而言）或超过一个小时（对一台28.8K-bps的调制解调器而言）。

等到文件传输结束，你将看到下面的信息：

```
226 Transfer complete.
local: emacs-19.30.tar.gz remote: emacs-19.30.tar.gz
10039387 bytes received in 1.7e+02 seconds (59 Kbytes/s)
ftp>
```

用“**quit**”命令退出FTP。如果使用的是Web浏览器或其他图形界面的工具，请根据自己的具体情况决定是否要退出程序。

解压缩和拆包

下载这个文件之后，真正开始建立Emacs之前还需要做两件事情：解压缩和拆包。这些工作的第一步需要使用一个名为**gunzip**的程序，但遗憾的是，它不是一个标准的UNIX命令。FSF软件的源代码文件都是用他们自己编写的一个名为**gzip**的程序来压缩的，没有使用UNIX操作系统中标准的**compress**工具。为什么会这样？这是因为标准的**compress**命令所使用的数据压缩算法是有专利的，不符合FSF倡导的自由软件精神（参见附录六）。另外一个原因是**gzip**的压缩效果和速度远远好于标准的**compress**命令。

gunzip是与**gzip**相对的解压缩工具。如果运气够好，你的系统上就应该已经有这个程序了。为了检查一下是否如此，请输入“**gunzip -h**”命令。如果看到**gunzip**的帮助信息和它的各种选项，就说明你系统上已经有这个程序了。否则，你将看到一条出错信息“not found（文件未找到）”；如果真的收到这条出错信息，就必须在做其他事情之前先下载和建立**gunzip**。

gunzip程序的下载过程与刚才介绍的用FTP下载Emacs的过程完全相同，只不过这次要找的是一个名为*gzip-N.N.N.tar*的文件而已。这个文件同时包含**gzip**和**gunzip**

两个程序的源代码。如果必须以命令行方式来完成 FTP 操作，则可以用命令“**ls gzip***”帮忙找到正确的文件。这个文件的长度在 800 KB 左右。

下载 *gzip-N.N.N.tar* 文件之后，需要对它进行拆包和安装。这个过程与 Emacs 的建立过程很相似，这是因为 FSF 制定有一套标准的安装流程，所有 FSF 软件都可以用同一种方法建立。具体步骤请查阅本附录后半部分中的“建立 Emacs”一节。为了把 **gzip** 和 **gunzip** 正确地安装到系统上，你必须请一位系统管理员帮忙。

把 **gunzip** 程序安装到系统上之后，就可以使用它和 **tar** 命令对 Emacs 进行解压缩和拆包操作。进入存放下载的 Emacs 源代码的那个目录。然后输入以下命令，你将看到一个文件清单：

```
$ gunzip -c emacs-19.30.tar | tar xvf -
x emacs-19.30/GETTING.GNU.SOFTWARE, 4737 bytes, 10 tape blocks
x emacs-19.30/INSTALL, 24397 bytes, 48 tape blocks
x emacs-19.30/PROBLEMS, 57880 bytes, 114 tape blocks
x emacs-19.30/README, 3960 bytes, 8 tape blocks
x emacs-19.30/BUGS, 937 bytes, 2 tape blocks
x emacs-19.30/move-if-change, 129 bytes, 1 tape blocks
x emacs-19.30/ChangeLog, 109081 bytes, 214 tape blocks
x emacs-19.30/Makefile.in, 20393 bytes, 40 tape blocks
x emacs-19.30/configure, 108386 bytes, 212 tape blocks
x emacs-19.30/configure.in, 40490 bytes, 80 tape blocks
. . .
```

这个文件清单得在屏幕上走一阵子 —— Version 19.30 大约包含有 1500 个文件。如果不想看到这个文件清单，就可以把上面那条 **tar** 命令里的字母“**v**”（对应 **verbose** 选项）删除。等这个命令执行完毕的时候，获取 Emacs 源代码文件的工作就算告一段落。

自由软件基金会

如果你不能访问因特网，就可以直接从 FSF 购买 Emacs。它们销售的 Emacs 有好几种存储格式，其中包括 CD-ROM 盘、软盘（用于 MS-DOS 版本）以及以下几种数据磁带，它们全都用 **tar** 命令进行了打包：

- 1600bpi 或 6250bpi 的 1/2 英寸双轴数据带
- Sun 公司出品的 DC300XLP 型 1/4 英寸盒式数据带（QIC 24）

- 惠普公司出品的16轨DC600HC型1/4英寸盒式数据带
- IBM公司出品的RS/6000型1/4英寸盒式数据带（QIC 150）
- Exabyte公司出品的8毫米盒式数据带
- DAT公司出品的4毫米盒式数据带

FSF能够根据顾客要求提供多种产品和服务组合，订购服务也算其中的一项。他们通常只提供源代码（DOS软盘版是个例外，它包含一个已经建立好的可执行程序）；但如果用户报出自己的软、硬件配置情况，他们也愿意根据定单替用户建立Emacs。这些产品的价格（在本书写作期间）是：对个人用户，一张CD-ROM盘$60美元；按定单制作的Emacs则收费可能高达$5000美元。CD-ROM盘上还有很多其他的FSF程序和文档，而数据磁带上除Emacs外就只有很少几个关键的程序设计工具和**gzip**软件。

FSF的销售产品还包括《Gnu Emacs Reference Manual》（GNU Emacs参考手册）的副本，各种其他GNU软件和文档等等。

如果想进一步了解FSF或者订购他们的产品，请按下面的地址与他们联系：

Free Software Foundation, Inc.
59 Temple Place, Suite 330
Boston, MA 02111-1307
phone: (617) 542-5942
fax: (617) 542-2652
email: *gnu@prep.ai.mit.edu*

也可以访问FSF位于*http://twws1.vub.ac.be/ studs/ tw45639/ fsf.htm*的Web主页。

其他CD-ROM来源

如果不考虑FSF渠道，你还可以选择一些把公共软件和免费软件打包到CD-ROM盘里再以很合理的价格进行销售的小公司。这些供应商倾向于把能够在大多数比较流行的软、硬件平台上运行的Emacs预装版打包在CD-ROM盘里。如果你有一个*CD-*

ROM 光驱，这类来源可能更值得考虑。如果你手里没有可以用来自行建立 Emacs 的工具（比如C编译器），就只能满足于一个预装版；即使你有必要的工具，这类来源也是一个方便的选择，它们能够节省大量的时间、硬盘空间并减少潜在的麻烦。下面是其他一些 Emacs 的 CD-ROM 发行版本来源：

Prime Time Freeware
370 Altair Way, Suite 150
Sunnyvale, CA 94086
phone: (408) 433-9662
fax: (408) 433-0727
email: *info@ptf.com*
WWW: *http://www.cfcl.com/ptf/*

Ready-to-Run Software
4 Pleasant Street
Forge Village, MA 01886
phone: (800) 743-1723 or (508) 692-9922
fax: (508) 692-9990
email: *info@rtr.com*
WWW: *http://www.rtr.com/*

还可以从 O'Reilly & Associates 出版公司出版的《UNIX Power Tools》一书附赠的 CD-ROM 盘上获得 Emacs，这本书的作者是 Jerry Peek、Tim O'Reilly 和 Mike Loukides。

建立 Emacs

除非你的 Emacs 预装版正好适用于你的系统，要不然，在能够使用之前，你必须从源代码开始对 Emacs 众多的可执行组件进行编译和安装。下面是建立 Emacs 时应该注意的一些问题。

Emacs 源代码有一个顶级目录，它的名字应该是“*emacs-19.30*”。这个目录里有两个名字分别是 *INSTALL* 和 *README* 的文件。先查阅 *README* 文件；它的内容通常

是很有用的一般性信息和“最后一分钟”才确定下来的重要“游戏规则”。接着查阅*INSTALL*文件，它为Emacs每个建立步骤给出了详细的指导；即使你不是一位UNIX专家，也应该看得懂。

Emacs的建立过程要花相当长的时间；大概得两三个小时。希望大家在整个过程中一切顺利。但各种意外情况还是有可能发生，比如编译器报告有个错误并半途而废、原以为有的某个系统文件无法不到、配置文件和系统的具体情况差之毫厘，或者是其他一些不可预见的陷阱等（注2）。如果遇到麻烦，最好找一位身边的专家来帮助你。我们这本书不可能把安装过程中可能遇到的各种问题都面面俱到地讨论到。

FSF的标准安装流程一直在改进，它越来越能干，对意外情况的处理也越来越好。但Emacs建立工作的难易程度主要取决于用户的软、硬件组合情况。FSF的安装脚本里有一个名为**configure**的程序，它的作用是检查用户系统，确定用户所使用的硬件和软件，然后对Emacs做相应的配置。

configure对系统配置情况的正确识别率是很高的。如果你的软、硬件组合属于一种比较主流的配置情况（比如CPU是Sun公司出品的SPARC芯片，操作系统是该公司最新推出的Solaris UNIX标准版本）并且你的系统管理员也没有对系统配置做过重大更改，这个正确率就更有保证。如果情况是这样，你的Emacs就应该能很顺利地建立起来，既不用做很多的配置设置，也不用技术专家来救急。可如果你有一个不同寻常的配置——比如一种很罕见的计算机或操作系统版本，一种不同寻常的软、硬件组合或者一种不常见的系统配置情况——那么除了你自己去配置这个软件之外就没有什么别的选择了。

我们的建议是：如果你了解C语言、是一位有经验的UNIX用户、并且愿意花时间去研究，那就放手大干一场吧！Emacs是聪明能干而又高度复杂的软件中的一个典型例子，这么说吧：UNIX操作环境能够提供的基础功能它几乎都用上了，UNIX软件开发工具能够提供的开发手段它几乎也都用上了。钻研Emacs的源代码可能是你能够感受到的最高深的UNIX教育。

注2：整个建立过程几乎总是会给出几条警告信息。你可以把它们看做坏消息，但如果Emacs建立工作没有因它们而半途而废，也不必太把它们放在心上。

附录二

解除他人对 Emacs 的定制设置

在学习Emacs的时候，最让人头疼的是：当你坐在终端前面时，才发现有位“热心肠”的系统管理员已经把所有的命令都改了。对这种情况你往往是无计可施的：你无法删除全局性的定制文件、你的Emacs与随机手册（或者本书）里讲的不同、而那位“热心肠”的系统管理员可能根本就没想过要把他做的修改记载下来。

如果给你开设账户的系统管理员已经为你准备了一个“有用的”个人“*.emacs*”配置文件，类似的问题就也有可能发生。这个“有用的”配置文件对有经验的用户来说可能是好东西，可如果因为有了它而反让你无法学用Emacs编辑器，那么它对你就没有什么帮助和价值。

还好，有一种办法能够让你从这些困境中解脱出来。如果你发现自己处于这种境地，先删除或者重命名你手里的所有“*.emacs*”文件。然后在你的主目录里创建一个只有一行语句的“*.emacs*”文件，这条语句必须严格按下面这样书写：

```
(setq inhibit-default-init t ); 不进行全局初始化
```

然后，重新启动Emacs。这个文件的作用是制止Emacs读取它的全局初始化文件。

还有一种棘手的情况：如果你正坐在别人的键盘前面，又该怎么办？你启动了Emacs，可出现在你面前的却都是别人“专用的”按键绑定和功能命令。这种困境也有解决办法：

- 试试“**emacs -q**”命令。“**-q**”选项的作用是告诉Emacs在启动之前不读取该用户的“*.emacs*”文件。这将使你避开该用户个人的定制设置。
- 但这个步骤并不会得到你自己的定制。如果想让Emacs读入你自己的“*.emacs*”文件，哪怕你使用的是别人的账户，请输入命令“**emacs -u** *yourname*”。比如，“**emacs -u deb**”将以用户Deb的初始化文件（即*~deb/.emacs*文件）启动Emacs。

但是如果你是在一个网络上，则“**-u**”选项可能不起作用。它要求你在网络的每一个系统上都有同样的主目录，或者是在每一个系统上有不同的主目录但这些主目录里的“*.emacs*”文件都是最新的。这两个条件在某些网络上是无法满足的。况且，网络配置也不在本书的讨论范围之内。如果你对这个问题感兴趣，可以去看看O'Reilly & Associates出版公司出版的《Managing NFS and NIS》一书，它的作者是Hai Stern；这本书对网络上的主目录问题做了详细的讨论。

附录三

Emacs 变量

我们已经在第十一章对 Emacs 变量进行过专门的讨论，本附录是对它的一个补充。我们将把与前十一章内容有关的Emacs变量根据常用、适用的原则汇总在以下的表格里。

表格中的变量是根据其用途分类的，它们的缺省值列在表格的第二栏里。具体到某个变量的详细说明请参考表格标题中给出的有关章节中的讨论。程序设计语言编辑模式所使用的变量的有关资料请查阅第十二章内容。

表 C-1：备份、自动保存和版本控制（参见第二、十五章）

变量	缺省值	说明
make-backup-files	**t**	如果取值为“**t**”，则在当前文件第一次存盘之前先给它创建一个备份版本
backup-by-copying	**nil**	如果取值为“**t**”，则不采用把将要保存的文件重命名为备份版本的做法，而是以复制的方法来创建备份文件。缺省的重命名办法比较有效率，而复制的办法则更安全——复制操作过程中发生的磁盘故障不会有什么损害，但在 **mv** 操作中“及时”出现磁盘故障却可能彻底毁坏备份文件[a]
version-control	**nil**	如果取值为“**t**”，则创建文件的编号版本作为备份（备份的文件名格式是“*filename~N~*”）。如果取值为“**nil**”，则只对已经有编号版本的文件这样做。如果取值为“'**never**”（注意前导的单引号），就不创建任何编号版本

表 C-1：备份、自动保存和版本控制（参见第二、十五章）（续）

变量	缺省值	说明
kept-new-versions	2	Emacs 在创建一个新的编号备份时将为文件保留的最新版本的个数
kept-old-versions	2	Emacs 在创建一个新的编号备份时将为文件保留最老版本的个数
delete-old-versions	**nil**	如果取值为“**t**”，则删除多余备份版本（根据上面两个变量的设置情况而需要保留的不算在内）时不需要用户确认；如果取值为“**nil**”，则需要经过用户确认
auto-save-default	**t**	如果取值为“**t**”，则对每个被访问文件都进行自动保存
auto-save-visited-file-name	**nil**	如果取值为“**t**”，则自动保存为被访问文件而不是保存为另外一个特殊的自动保存文件
auto-save-interval	300	两次自动保存操作之间间隔的按键次数。如果取值为“0”，则关闭自动保存功能
auto-save-timeout	30	Emacs 在用户没有操作情况下执行自动保存操作的时间间隔。如果取值为“**nil**”或“0”，则关闭此项功能
delete-auto-save-files	**t**	非“**nil**”值表示在保存“真正的”文件时删除其自动保存文件
buffer-offer-save	nil	非“**nil**”值表示在退出 Emacs 时提示用户保存当前编辑缓冲区，不论在它里面打开的是不是一个文件
vc-default-back-end	'RCS	**vc** 程序包使用的版本控制系统。它的可取值是符号“'RCS”、“'CVS”、或“'SCCS”
vc-display-status	t	如果取值不为“**nil**”，则在状态行上给出版本号和版本控制状态
vc-keep-workfiles	t	如果取值不为“**nil**”，向版本控制系统注册所做的修改后，Emacs 将不删除工作文件
vc-mistrust-permissions	nil	如果取值不为“**nil**”，则认为文件自身的属主 ID 和访问权限标志不代表版本控制系统对此的看法；VC 将直接从版本控制系统那里获取这些信息
vc-suppress-confirm	nil	如果取值不为“**nil**”，则执行版本控制操作前不要求用户确认
vc-initial-comment	nil	如果取值不为“**nil**”，则在文件第一次向版本控制系统注册时提示输入一条初始注释

表 C-1：备份、自动保存和版本控制（参见第二、十五章）（续）

变量	缺省值	说明
vc-make-backup-files	nil	如果取值不为“**nil**”，则仍为已经用版本控制操作注册过的文件创建标准的 Emacs 备份版本
vc-consult-headers	nil	如果取值不为“**nil**”，则根据嵌在工作文件里的版本控制信息来确定版本号；否则从主控文件里获取此项信息

a. 因为文件备份操作的效果是创建一个新的版本，所以新文件的属主就是你。换句话说，不管原始文件的属主是不是你，只要这个变量的取值是“**nil**”，以重命名方式得到的备份版本的属主也将是你。因此，如果你是以根用户身份进行的登录而你所编辑的文件的属主却不是根用户 —— 比如 **uucp** 的配置文件，就可能会引起一些问题。

表 C-2：查找与替换（参见第三章）

变量	缺省值	说明
case-fold-search	**t**	如果取值不为“**nil**”，则在查找操作中不区分字母的大小写
case-replace	**t**	如果取值不为“**nil**”，则在替换时维持原来的大小写情况（对 **case-fold-search** 也不例外）
search-upper-case	**'not-yanks**	如果取值不为“**nil**”，则查找字符串中的大写字母将压制 **case-fold-search** 变量做出的设置（即强制查找操作区分字母的大小写）。符号“**'not-yanks**”的含义是把替换字符串中的大写字母转换为小写
search-exit-option	**t**	如果取值不为“**nil**”，则任何一个不是递增查找子命令（**DEL**、**LINEFEED**、**C-q**、**C-r**、**C-s**、**C-w**、**C-y**）的控制字符都将退出查找
search-slow-speed	**1200**	如果终端的通信速度等于或小于这个值，就将使用慢速递增查找 —— 用一个小窗口来显示查找操作的部分结果[a]
search-slow-window-lines	**1**	慢速查找所用窗口的高度，计量单位是文本行。如果这个数字是一个负数，就表示要把这个窗口放在屏幕的顶部而不是底部
search-highlight	**nil**	如果取值不为“**nil**”，则反显已经查找到的部分匹配。适用于 X 窗口系统和其他具备反显功能的显示器

表 C-2: 查找与替换（参见第三章）（续）

变量	缺省值	说明
query-replace-highlight	**nil**	如果取值不为“**nil**”，则在查询-替换模式中反显找到的匹配。适用于X窗口系统和其他具备反显功能的显示器

a. 遗憾的是，只有当计算机认为终端的通信速度低时，这个选项才能发挥作用。如果你的终端是通过LAN、终端服务器或其他数据通信设备进行通信的，就可能出现因计算机正在与“你的终端”进行高速通信而“忽略”“你的终端”速度慢的情况。

表 C-3: 屏幕显示（参见第二、四、六章）

变量	缺省值	说明
next-screen-context-lines	2	用“**C-v**”或“**ESC v**”前、后卷屏时，Emacs 保留的上、下文行数
scroll-step	0	当光标沿垂直方向移出当前窗口时，Emacs 将前卷或后卷的文本行数。如果取值为“0”，则卷动足够多的行以便光标在卷动后出现在窗口中央
hscroll-step	0	当光标沿水平方向移出当前窗口时，Emacs 将左卷或右卷的文本列数。如果取值为“0”，则卷动足够多的列以便光标在卷动后出现在窗口中央
tab-width	8	制表位的宽度；如果被设置，将只对当前编辑缓冲区起作用
truncate-lines	**nil**	如果取值不为“**nil**”，则不对超长文本行进行自动换行，截断它并且用“$”表示这一行超出屏幕画面
truncate-partial-width-windows	**t**	如果取值不为“**nil**”，则把显示宽度小于显示器宽度的全部窗口里的超长文本行截断（像上面那样）
window-min-height	4	窗口的最小高度（计量单位是文本行）
window-min-width	10	垂直分割窗口时，分割出来的窗口的最小宽度（计量单位是列）
split-window-keep-point	**t**	分割窗口时，非“**nil**”取值表示使两个窗口里的光标同步移动。如果取值为“**nil**”，则以使重新绘制屏幕画面的工作量最小为原则设置一个光标位置（这是为低速显示器准备的）

表 C-3: 屏幕显示（参见第二、四、六章）（续）

变量	缺省值	说明
resize-minibuffer-mode	**nil**	如果取值不为“**nil**”，则允许辅助输入区增加高度以显示其中的内容
resize-minibuffer-window-exactly	**t**	允许辅助输入区的高度做动态改变，使之刚好能够显示其中的全部内容
resize-minibuffer-frame	**nil**	如果取值不为“**nil**”，则允许（X窗口系统显示器中的）辅助输入区窗格改变高度
resize-minibuffer-frame-exactly	**t**	允许（X窗口系统显示器中的）辅助输入区窗格的高度作动态改变，使之刚好能够显示其中的全部内容
resize-minibuffer-window-max-height	**nil**	**resize-minibuffer-mode** 模式中辅助输入区所能到达的最大高度；在X窗口系统下，如果这个变量的取值小于1或不是数字，则辅助输入区的高度不能超过它所在窗格的高度
ctl-arrow	**t**	非“**nil**”取值表示把控制字符显示为“^*X*”形式，其中的“*X*”是“被控制”的字符。其他取值表示把控制字符显示为八进制数字，比如说，“**C-h**”将被显示为八进制数字“\010”
display-time-day-and-date	**nil**	如果取值不为“**nil**”，则“**ESC-x display-time RETURN**”命令把星期几和日期都显示出来
line-number-mode	**t**	如果取值不为“**nil**”，则把行号显示在状态行上[a]
line-number-display-limit	1,000,000	如果让 Emacs 显示行号，则编辑缓冲区的长度不能超过这个数字（计量单位是字符）
column-number-mode	**nil**	如果取值不为“**nil**”，则把列号显示在状态行上[a]
visible-bell	**nil**	如果取值不为“**nil**”，则在必要时以屏幕闪烁代替蜂鸣报警
track-eol	**nil**	如果取值不为“nil”，则当光标位于某文本行的行尾并做上、下移动时，它仍将移动到前、后文本行的行尾；其他取值表示光标将固定在它此时所在的列位置
blink-matching-paren	**t**	如果取值不为“**nil**”，则用户输入一个需要配对出现的右括号类字符时，Emacs 将快速“闪现”与之配对的左括号类字符

PDG

表 C-3: 屏幕显示（参见第二、四、六章）（续）

变量	缺省值	说明
blink-matching-paren-distance	4000	当用户输入一个需要配对出现的右括号类字符时，向回查找其配对左括号类字符的最大距离（计量单位是字符）
bilnk-matching-delay	1	闪现配对左括号类字符的持续时间
echo-keystrokes	1	如果用户输入命令时停顿了这个变量所设置的的时间（计量单位是秒），则在辅助输入区里显示未完成命令的前缀（比如“ESC-”）作为提示；取值为“0”时表示不提示
insert-default-directory	**t**	如果取值不为“**nil**”，则在要求用户输入文件名时先把当前目录的路径名插入到辅助输入区里
inverse-video	**nil**	如果取值不为“**nil**”，则对整个显示画面进行反显（状态行将呈正常显示）
mode-line-inverse-video	**t**	非“**nil**”取值表示状态行将被反显
highlight-nonselected-windows	**t**	如果取值不为“nil”，则把除当前窗口以外的所有窗口里的文本块反显；适用于X窗口系统和其他具备反显功能的显示器
mouse-scroll-delay	0.25	当用户在某个窗口里按下鼠标并拖动到这个窗口边界以外的地方时，屏显画面将延迟这个变量所设置的时间（计量单位是秒）之后才发生卷动。如果取值为“0”，则表示以最快速度卷动
mouse-scroll-min-lines	1	当鼠标在窗口边界以外的地方被按下并做上、下拖动时，至少要卷动这个变量所设置的文本行数

a. 可以用命令“**ESC-x line-number-mode**”和“**ESC-x column-number-mode**”切换行号、列号显示功能的开关状态。

表 C-4: 编辑模式（参见第二、五、六、九、十一章）

变量	缺省值	说明
major-mode	**fundamental-mode**	新编辑缓冲区的缺省编辑模式，除非根据文件名后缀而另有规定；设置这个变量的时候要注意在编辑模式名称的前面加上一个单引号（这个变量的值必须是一个符号）
auto-mode-alist	（参见第十一章）	设置文件名和主编辑模式之间关联关系的列表
left-margin	0	在基本模式和文本模式里按下“**C-j**”组合键时的缩进量
indent-tabs-mode	**t**	如果取值不为“**nil**”，则用户使用“**C-j**”组合键对文本进行缩进时允许使用制表符（以及空格）
find-file-run-dired	**t**	如果取值不为“**nil**”，则在访问文件时，如果用户文件名是一个目录，就运行 **dired**
dired-kept-versions	2	在 **Dired** 里清理目录时，需要保留的文件版本个数
dired-listing-switches	“**-al**”	生成**dired**文件清单时传递给**ls**命令的选项；至少要包含“**-l**”选项
shell-file-name	**$SHELL**	shell的文件名；如果某个Emacs函数需要调用 shell —— 比如 **list-directory**、**dired** 和 **compile**等，Emacs就会运行这个变量指定的 shell。此缺省值表示将使用 UNIX 环境变量 **SHELL** 的值
load-path		这个列表变量设置的是加载LISP程序包（参见第十三章）时的搜索路径；通常就是Emacs源代码在系统的安装目录下的 *lisp* 子目录

表 C-5: 电子邮件（参见第六章）

变量	缺省值	说明
mail-self-blind	**nil**	如果取值不为“**nil**”，则自动把你的名字加入到“BCC（密抄）”栏以保证能给自己留下一份邮件副本

表 C-5: 电子邮件（参见第六章）（续）

变量	缺省值	说明
rmail-mail-new-frame	**nil**	如果取值不为“**nil**”，则创建一个新窗格来编写外发邮件消息，仅适用于 X 窗口系统
mail-default-reply-to	**nil**	缺省插入到邮件消息“`Reply-to:`（回复地址）”栏里的字符串
mail-use-rfc822	**nil**	如果取值不为“**nil**”，则使用完全符合 RFC 822 标准规定的地址解析器对邮件地址进行解析；这会多花上一点时间，但正确解析出复杂网络地址的概率会有所增加
mail-host-address	**nil**	你的机器的名字；将被用来构造 **user-mail-address**
user-mail-address	（你的邮件地址）	你的完整的电子邮件地址
rmail-primary-inbox-list	**nil**	保存新收（尚未阅读）邮件的文件清单。如果取值为“**nil**”，则使用环境变量 **$MAIL** 的值；如果 **$MAIL** 指定的路径不存在，就使用路径“**/usr/spool/mail/***yourname*”
rmail-file-name	**"~/RMAIL"**	RMAIL 用来保存邮件消息的文件
mail-archive-file-name	**nil**	用来保存所有外发邮件消息的文件名字符串；如果取值为“**nil**”，表示不保存外发邮件
mail-personal-alias-file	**"~/.mailrc"**	用来保存邮件假名的文件名；Emacs 的邮件编辑模式与 UNIX 操作系统标准的 **mail** 和 **mailx** 程序使用的假名格式是一样的
mail-signature	**nil**	准备添加到外发邮件消息末尾的文本
rmail-dont-reply-to-names	**nil**	与这个正则表达式相匹配的名字将被排除在邮件回复地址名单之外；如果取值为“**nil**”，则把你本人排除在回复名单之外
rmail-displayed-headers	**nil**	与这个正则表达式相匹配的邮件消息标题栏将被显示出来；如果取值为“**nil**”，则把 **rmail-ignored-headers**变量没有包括的标题栏都显示出来
rmail-ignored-headers	（复杂正则表达式）	不显示与这个正则表达式相匹配的邮件消息标题栏

表 C-5: 电子邮件（参见第六章）（续）

变量	缺省值	说明
rmail-highlighted-headers	**"^From:\\|^Subject:"**	反显与这个正则表达式相匹配的邮件消息标题栏；适用于 X 窗口系统和其他具备反显功能的显示器
rmail-delete-after-output	**nil**	如果取值不为"**nil**"，则自动删除已经被保存到某个文件里的邮件消息
mail-from-style	**'angles**	Emacs 为"From:（发信人）"栏生成的用户名的格式。如果取值为"**nil**"，则只包括电子邮件地址；如果取值为"**'angles**"，则把电子邮件地址用角括号括起来（例如"*Dave Roberts<droberts@ed.com>*"）；如果取值为"**'parens**"，则把电子邮件地址用圆括号括起来（例如"*Dave Robert(droberts@ed.com)*"）

表 C-6: 文本编辑（参见第二、三、六、九、十一章）

变量	缺省值	说明
sentence-end	（见第十三章）	匹配句尾的正则表达式
sentence-end-double-space	**t**	如果取值不为"**nil**"，则不把句号后面的单个空格看做句尾
paragraph-separate	**"[\t\C-l]"**	匹配段落分隔行行首的正则表达式
paragraph-start	**"[\t\n C-l]"**	匹配段落分隔行或段落第一行行首的正则表达式
page-delimiter	**"^\C-l"**	匹配分页符的正则表达式
tex-default-mode	**'plain-tex-mode**	打开 / 创建 TEX 或 LATEX 文件时将被启动的编辑模式
tex-run-command	**"tex"**	在 TEX模式下，用来运行 TEX程序以排版某个文件的命令字符串；TEX程序将运行在一个子进程里
latex-run-command	**"latex"**	用来运行 LATEX程序的命令字符串；LATEX程序将运行在一个子进程里
slitex-run-command	**"slitex"**	用来运行 SliTEX程序的命令字符串；SliTEX程序将运行在一个子进程里

表 C-6: 文本编辑（参见第二、三、六、九、十一章）（续）

变量	缺省值	说明
tex-dvi-print-command	**"lpr -d"**	在 TEX 模式里，用“**C-c C-p**”组合键打印文件时将调用的命令字符串
tex-dvi-view-command	**nil**	用“**C-c C-v**”组合键查看“*.dvi*”TEX 输出文件时将调用的命令字符串。在 X 窗口系统上，这个变量的值往往设置为“*xdvi*”
tex-offer-save	**t**	如果取值不为“**nil**”，则 Emacs 将在运行 TEX 之前提示用户保存尚未存盘的编辑缓冲区
tex-show-queue-command	**“lpq”**	在 TEX 模式里，用“**C-c C-q**”组合键查看打印队列时将被调用的命令字符串
tex-directory	“.”	TEX 存放临时文件的目录；默认为当前目录
outline-regexp	**“[*\C-l]+”**	在大纲模式里，用来匹配文本标题行的正则表达式
outline-heading-end-regexp	**“[\n\C-m]”**	在大纲模式里，用来匹配文本标题行行尾的正则表达式
selective-display-ellipses	**t**	如果取值为“**t**”，则把大纲模式里的隐藏文本显示为省略号“...”；其他取值表示什么也不显示

表 C-7: 自动补足功能（参见第十六章）

变量	缺省值	说明
completion-auto-help	**t**	如果取值不为“**nil**”，则在自动补足功能（辅助输入区里的 **TAB** 和 **RETURN** 键）无效或有二义时提供帮助
completion-ignored-extensions	（参见第十六章）	Emacs将不对这个列表里的文件名后缀进行补足
completion-ignore-case	**nil**	如果取值不为“**nil**”，则进行自动补足时将忽略字母的大小写

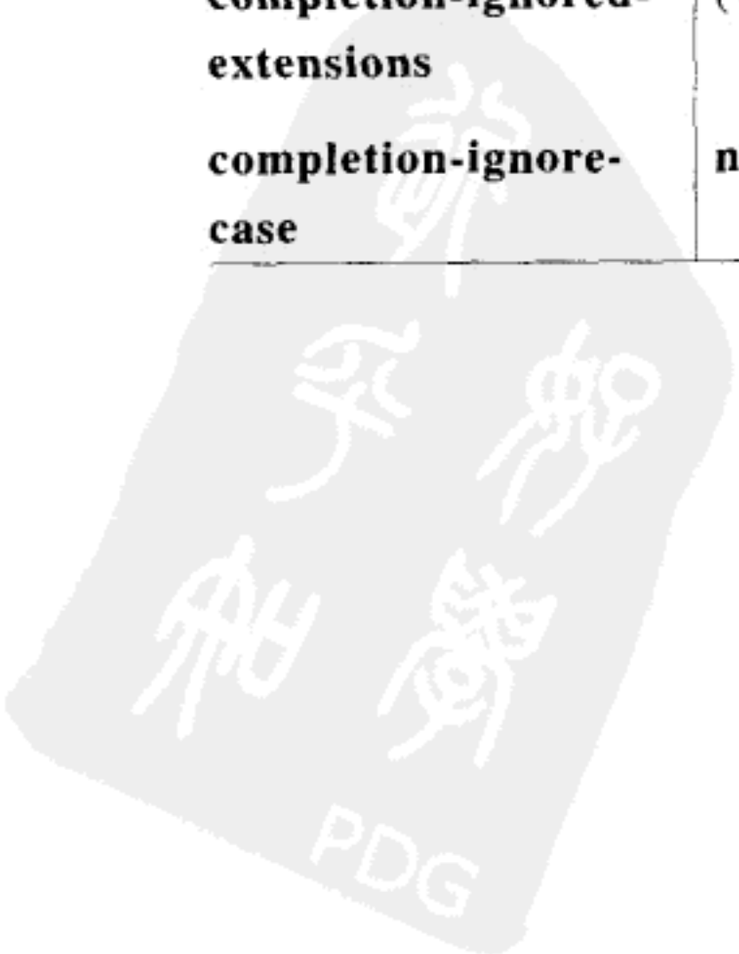


表 C-8: 杂项

变量	缺省值	说明
kill-ring-max	30	保存在删除环里的被删除文本块数；如果没有剩余的空间，就将删除其中最“老”的那个
require-final-newline	**nil**	如果某个已经被保存起来的文件没有最末尾的 **LINEFEED** 字符，则这个变量取值为“**nil**”时不自动添加之；取值为“**t**”时则自动添加之；其他取值时询问用户是否想添加一个 **LINEFEED** 字符[a]
next-line-add-newlines	**t**	如果取值不为“**nil**”，则在编辑缓冲区的末尾执行 **next-line** 命令（按下“**C-n**”组合键或向下方向键）时将插入一个新行而不是报告出错

a. 注意，有些程序（比如 **troff**）要求文件必须以 **LINEFEED** 字符结束。

附录四

Emacs LISP 程序包

本附录对Emacs自带的最有用的LISP程序包进行了汇总。LISP程序包通常都存放在目录“*emacs-source/lisp*”里，而“*emacs-source*”代表存放Emacs源代码的目录。我们略去了那些为Emacs提供“基本”功能支持的程序包；同理，我们也略去了那些功能过时或者不易使用的程序包。

虽然书中已经详细介绍过几个程序包，但数量还是太少；读者必须依靠GNU Emacs的在线帮助功能查找对程序包功能的精确描述。我们在第十六章专门对Emacs的在线帮助功能作了详细的介绍；具体到查找LISP程序包功能方面的资料，最重要的帮助命令是“**C-h p**”（命令名是**finder-by-keyword**）、“**C-h f**”（命令名是**describe-function**）和“**C-h m**”（命令名是**describe-mode**）。

“**C-h p**”的用处非常大。它可以帮助从相关的信息树上把所有能够用在自己系统上的程序包的资料都查出来，其适用面从本附录所汇总的各种基本功能领域的程序包一直延伸到每一种编辑模式的“**C-h m**”信息。因为篇幅的关系，我们在下面那些表格对程序包做的“详细”说明还不够完整，而很多列在表格里的程序包却又可能只有那些坚定的Emacs定制者才感兴趣。

只要合理，我们就会在表格里给出用来“启动”某个程序包的命令。表格中“启动命令”一栏的含义是：

- 如果程序包是实现某个主编辑模式的，则它的启动命令就是把Emacs放到这个主编辑模式里的函数。
- 如果实现主编辑模式的程序包还会在访问带有某个特定后缀名的文件时被自动加载，我们就将在启动命令的后面再开列出“suffix *suffixname*”（普通字体的“suffix”中文意思是“后缀”；表格中的文件名后缀用斜体字表示）。
- 如果程序包是实现某个副编辑模式的，则它的启动命令就是把Emacs放到这个副编辑模式里的函数。
- 如果程序包实现了多个、成系列的通用性函数，我们将尽量把其中最“典型的”函数挑选出来。比如说，*studly* 程序包实现了 3 个命令；而我们将把 **studlify-region** 挑出来作为这个程序包的启动命令。如果没有更合理的选择，我们就写上“many”。

最后，我们还想说说程序包使用方面的问题。有些程序包是在Emacs启动时自动加载的，有些则需要等到你访问某个有特定文件名后缀的文件时才会加载（比如许多程序设计语言专用的编辑模式），有些是在你执行某个特定命令时自动加载的（例如“**ESC x shell RETURN**”加载对应于 **shell-mode** 的程序包 *shell.el*），还有一些则永远也不会自动加载。Emacs 对程序包的加载方式（即哪些程序包会在启动时自动加载，哪些会因用户操作而自动加载，哪些根本就不会被自动加载）事先有一个标准的配置，但你的系统管理员可能已经对这种配置进行了定制。如果是这样的话，你又该做些什么呢？其实，你不用关心这个问题，我们已经把程序包的启动命令列在下面这些表格里；你只要在启动 Emacs 之后发出这个命令（“**ESC x** *startup-command* **RETURN**”）试试就知道了。如果 Emacs 抱怨说“`no match`（找不到匹配）”，就说明这个程序包不能自动加载，必须由你来“亲手”加载它。如果你不想退出 Emacs，就需要使用命令“**ESC x load-library** *name* **RETURN**”，其中的“*name*”是我们在表格第一栏里给出的程序包名称；如果你想让 Emacs 在启动时自动加载这个程序包，就需要把下面这样一条语句添加到“*.emacs*”文件里：

```
(load-library "name")
```

好、不啰嗦了。研究这些表格里的 LISP 程序包去吧。

表 D-1：对 C 和 C++ 程序设计语言的支持

程序包名称	说明	启动命令
cc-mode	主编辑模式：编辑 C、C++ 和 Objective-C 源代码文件	**c-mode, c++-mode, objc-mode**, suffixes *.c, .h, .y, .lex, .cc, .hh, .C, .H, .cpp, .cxx,*
c-fill	副编辑模式：增加了对多行 C 程序注释的排版功能	**c-comment**
cmacexp	函数：功能是用 cpp 扩展 C 源代码中的宏定义的函数	**c-macro-expand**
hideif	副编辑模式：隐藏 C 预处理器条件中的代码	**hide-ifdef-mode**
cpp	主编辑模式：反显和隐藏 C 预处理器条件中的代码；适用于 X 窗口系统和其他具备反显功能的显示器	**cpp-parse-edit**
gud	主编辑模式：与调试器 **gdb**、**sdb**、**dbx** 和 **xdb** 等配合工作	**gud-mode**

表 D-2：对其他程序设计语言的支持

程序包名称	说明	启动命令
ada	主编辑模式：编辑 Ada 源代码	**ada-mode**，suffixes *.ada, .adb, .ads*
fortran	主编辑模式：编辑 FORTRAN 源代码	**fortran-mode**, suffix *.f*
asm-mode	主编辑模式：编辑汇编源代码	**asm-mode**，suffix *.s*
awk-mode	主编辑模式：编辑 awk 源代码	**awk-mode**，suffix *.awk*
f90	主编辑模式：编辑符合 FORTRAN-90 标准的 FORTRAN 源代码	**f90-mode**，suffix *.f90*
fortran	主编辑模式：编辑 FORTRAN 源代码	**fortran-mode**，suffixes *.f, .for*
icon	主编辑模式：编辑 ICON 源代码	**icon-mode**，suffix *.icn*
mim	主编辑模式：编辑 MIM 源代码	**mim-mode**
modula-2	主编辑模式：编辑 Modula-2 源代码	**modula-2-mode**

表 D-2: 对其他程序设计语言的支持（续）

程序包名称	说明	启动命令
perl-mode	主编辑模式: 编辑 Perl 源代码	**perl-mode**, suffix *.pl*
prolog	主编辑模式: 编辑 Prolog 源代码	**prolog-mode**, suffix *.prolog*
simula	主编辑模式: 编辑 Simula-87 源代码	**simula-mode**,
tcl-mode	主编辑模式: 编辑 tk/tcl 源代码	**tcl-mode**, suffix *.tcl*
compile	函数: 编译程序和运行 **make**	**compile**

表 D-3: 对 LISP 程序设计语言的支持

程序包名称	说明	启动命令
lisp-mode	主编辑模式: LISP、Emacs LISP 和 LISP 互动等模式	**lisp-mode,emacs-lisp-mode, lisp-interaction-mode**[a]
scheme	主编辑模式: 编辑 Scheme 源代码	**scheme-mode**, suffixes *.scm, .oak*
cl	函数和宏: 处理 Emacs LISP 和 Common LISP 之间的兼容问题	many
debug	主编辑模式: 调试 Emacs LISP 程序	**debug**
edebug	实现为副编辑模式的 Emacs LISP 调试功能	**edebug**
disass	函数: 对 Emacs LISP 编译代码进行反汇编	**disassemble**
elp	Emacs LISP 的代码检查器	**elp-instrument-package, elp-instrument-function**
trace	生成 Emacs LISP 程序的函数调用顺序表	**trace-function**
cmuscheme	面向 CMU Scheme 解释器的接口; CMU Scheme 解释器运行在一个子进程里	run-scheme
xscheme	面向 MIT Scheme 解释器的接口; MIT Scheme 解释器运行在一个子进程里	run-scheme

a. LISP模式在遇到后缀名为“*.l*”、“*.lsp*”、“*.lisp*”和“*.ml*”的文件时自动启动; Emacs LISP 模式在遇到后缀名为“*.el*”和“*.emacs*”以及任何被命名为“*_emacs*”的文件时自动启动。LISP 互动模式是编辑缓冲区“*scratch*”的编辑模式。

表 D-4: 对文本处理的支持

程序包名称	说明	启动命令
text-mode	主编辑模式: 编辑尚未排版的文本文件	**text-mode**[a]
nroff	主编辑模式: 编辑 **nroff** 和 **troff** 文本文件	**nroff-mode**
scribe	主编辑模式: 编辑 Scribe 文本文件	**scribe-mode**
sgml-mode	主编辑模式: 编辑 SGML 文本文件	suffixes *.sgm, .sgml, .dtd*
tex-mode	主编辑模式: 编辑 TEX 和 LATEX 文本文件	**tex-mode**、**latex-mode**[b]
bibtex	主编辑模式: 编辑 LATEX 目录文件	**bibtex-mode**、suffix *.bib*
refbib	把 **refer** 格式的目录文件转换为 LATEX 格式	**r2b-convert-buffer**

a. 文本模式在遇到后缀名为 "*.text*"、"*.article*" 和 "*.letter*" 以及带有前缀 "*/tmp/Re*"、"*MessageN*"（邮件消息）和 "*/tmp/fol*"（某些新闻阅读器会使用这个前缀）的文件时自动启动。

b. TEX模式在遇到后缀名为 "*.tex*" 和 "*.TeX*" 的文件时自动启动；LATEX模式在遇到后缀名为 "*.ltx*"、"*.sty*"、"*.cls*" 和 "*.bbl*" 的文件时自动启动。

表 D-5: 对其他编辑器的仿真

程序包名称	说明	启动命令
edt	函数: 设置仿真 VAX/VMS 平台上 **EDT** 编辑器的按键绑定	**edt-emulation-on**
mlconvert	函数: 把 Gosling Emacs 的 "伪 LISP" 代码转换为 Emacs LISP 代码	**convert-mocklisp-buffer**
vi	主编辑模式: 仿真 **vi** 编辑器	**vi-mode**
vip	另外一个用来仿真 **vi** 编辑器主编辑模式	**vip-mode**
ws-mode	主编辑模式: 仿真 WordStar 文字处理器	**wordstar-mode**

表 D-6: 面向 UNIX 工具的接口

程序包名称	说明	启动命令
shell	主编辑模式: 与 UNIX 操作系统的 shell 互动	**shell-mode**
diff	用 UNIX 的 **diff** 命令对比两个文件之间的差异	**diff, diff-backup**
find-dired	运行 UNIX 的 **find** 命令并在其结果文件清单上用 **dired** 做进一步的处理	**find-dired**

表 D-6: 面向 UNIX 工具的接口（续）

程序包名称	说明	启动命令
tar-mode	通过一个 **dired** 风格的界面访问 tar 档案文件	**tar-mode**, suffix *.tar*
lpr	打印输出编辑缓冲区或文本块的内容	**lpr-buffer**, **print-buffer**, **lpr-region**, **print-region**
uncompress	函数：访问压缩文件的过程中临时对文件进行解压缩	**uncompress-while-visiting**
spell	函数：检查拼写情况	**spell-word**, **spell-region**, **spell-buffer**

表 D-7: 面向通信程序的接口

程序包名称	说明	启动命令
ange-ftp	函数：通过 FTP 透明地访问远程文件	许多标准的文件操作命令
kermit	函数：在 shell 模式里帮助运行数据通信程序 **kermit**	many
mh-e	与 **MH** 邮件系统配合使用的各种函数	**mh-rmail**, **mh-smail**
rmail	主编辑模式：阅读和编辑电子邮件	**rmail**
gnus	主编辑模式：阅读和投稿 Usenet 新闻	**gnus**
telnet	主编辑模式：使用 Telnet 协议连接远程机器	**telnet**, **rsh**
rlogin	一个用来完成远程操作的 shell 模式的超集	**rlogin**

表 D-8: 通用编辑功能程序包

程序包名称	说明	启动命令
chistory	函数：编辑和重复执行以前执行过的命令	**repeat-matching-complex-command**
echistory	主编辑模式：编辑和重复执行以前执行过的命令	**electric-command-history**
compare-w	函数：比较两个 Emacs 窗口	**compare-windows**
options	函数：查看和设置 Emacs 变量	**list-options**, **edit-options**

表 D-8: 通用编辑功能程序包（续）

程序包名称	说明	启动命令
sort	函数：对各种文本文件进行排序	**sort-lines, sort-columns, sort-fields**
outline	主编辑模式：编辑大纲	**outline-mode**
underline	函数：给编辑缓冲区中的文本加下划线	**underline-region**

表 D-9: 游戏

程序包名称	说明	启动命令
blackbox	主编辑模式：Blackbox 游戏	**blackbox-mode**
cookiel	函数：根据月相算命	**cookie**
dissociate	函数：随机拼凑各种文字	**dissociated-press**
doctor	主编辑模式：Psychiatrist（心理学家）游戏	**doctor**
dunnet	主编辑模式：Adventure（冒险）游戏。	**dunnet**
flame	函数：随机出现的粗话	**flame**
gomoku	主编辑模式：Gomoku 游戏	**gomoku**
hanoi	函数：汉尼塔游戏	**hanoi**
life	主编辑模式：Life 游戏	**life**
mpuz	主编辑模式：随机拼图游戏	**mpuz**
solitaire	主编辑模式：纸牌游戏	**solitaire**
spook	函数：让 NSA（美国国家安全局）注意你的邮件	**spook**
studly	函数：把文章中的小写字母随机地改为大写字母	**studlify-region**
yow	随机打印一个 Zippy the Pinhead（Emacs 中的游戏人物）说的笑话	**yow**

表 D-10: 杂项

程序包名称	说明	启动命令
calendar	主编辑模式：显示一个全功能的日历	**calendar**
emacsbug	函数：向 FSF 报告发现一个程序漏洞	**report-emacs-bug**
time	函数：在状态行上显示当前时间	**display-time**

附录五

软件漏洞及其修补

没有十全十美的程序。虽然GNU Emacs经受了非常彻底的调试，但肯定还能找出不能正确工作的环节。我们就曾在这本书的写作过程中设法让Emacs“死”过一次——不过，和大多数用户一样，我们并没有把这个问题报告给有关方面：我们接着工作去了。

自由软件基金会对上报给他们的问题是很重视的，反应也非常迅速。但他们需要的是真正的问题报告；针对某些具体做法的不同意见不能算做程序漏洞。如果你认为某个命令应该这样或者那样，那么请记住Emacs已经存在了相当长的时间而且已经有了大批的用户——为了迎合一两个人而改变它是不切实际的。为了便于用户反映程序漏洞，自由软件基金会在《GNU Emacs Manual》（GNU Emacs手册）里给出了非常详细的指导，我们把它们总结如下：

- 如果你遇到一个系统级错误（比如Emacs死机、因内存分配问题而意外终止运行或者造成“反动”破坏），就表明Emacs很有可能存在一个程序漏洞。
- 在上报程序漏洞的时候，要把事情的原委尽可能地讲清楚。有些命令可以帮你把Emacs的故障现象描述得更精确。“**C-h l**”命令（命令名是**view-lossage**）能够把最近输入的100个按键查出来；“**ESC :**”命令（命令名是**open-dribble-file** "*filename*"）能够把按下的每一个按键组合存到指定的“*filename*”文件里，而“**ESC :**”命令（命令名是**open-termscript** "*filename*"）能够把你输入的每一个键和发送到屏幕上的每一个字符保存到指定的“*filename*”文件里。

- FSF不鼓励由你本人对程序漏洞报告里的程序漏洞做出解释。你只要能讲清楚“我是这样、这样做的，然后发生了那样、那样的事情”就足够了，尤其是故障现象一再出现的时候；而“我认为终端处理存在这一问题”之类的说法却提供不了任何有用的信息。
- 在程序漏洞报告中，一定要清楚地说明你使用的Emacs的版本。可以用命令“**ESC x emacs-version**”查出有关的信息。
- 你还需要在程序漏洞报告里说明下面几件事：当时正在编辑的文件内容（如果它有特殊之处）、“*.emacs*”文件的内容、当时所在的编辑模式以及错误发生之前刚加载的LISP程序包，等等。

我们再来加上一条重要的指导意见：

- 虽然我们已经尽了最大的努力来保证本书的准确性，但这并不代表它是十全十美的。我们自己也知道它距离完美还有一定的差距。因此，在上报程序漏洞的时候，千万不要以本书做为权威参照。虽然我们没有问过，但自由软件基金会极有可能会拒绝接受根据第三方出版物做出的程序漏洞报告。因此，如果你认为自己遇到了一个程序漏洞，请根据《GNU Emacs Manual》（GNU Emacs手册）或Emacs的在线帮助工具判断给你带来麻烦的命令的实际功用。你可能发现我们这本书讲错了；如果真是这样，请按电子邮件地址*nuts@ora.com*给我们也发一份问题报告。

如果你能够通过因特网发送电子邮件，可以把程序漏洞报告发往：

bug-gnu--emacs@prep.ai.mit.edu

或者，你可以用程序包**emacsbug**提交程序漏洞报告。具体做法是：输入“**ESC x submit-bug-report**”命令，在提示符处输入报告主题，然后按回车键；屏幕上将弹出一个“`Mail`”窗口，其中的“`Subject:`（主题）”栏已经被填写为刚才在提示符处输入的内容。这是将邮件发往正确地址的简便方法。在Emacs里使用电子邮件的细节请参阅本书第六章内容。

附录六

Emacs 的版权文件

本附录将要介绍的两个重要文件可能大家都已经很熟悉了，它们就是《GNU通用公共许可证》（GNU General Public License）和《GNU宣言》（GNU Manifesto）。《GNU通用公共许可证》对GNU Emacs、自由软件基金会的其他产品以及很多由个人编写和发行的程序的版权问题做出了规定；我们把这份许可证的全文收录在本附录里。《GNU宣言》是一份很长的文档，它对自由软件基金会的存在意义做出了声明。

在这些材料之后，我们还将对“自由编程联盟”（the League for Programming Freedom）组织做一个简单的介绍。这个组织与自由软件基金会并没有什么关系，但目标却差不多。我们将告诉大家从哪儿能获得这个联盟两份重要的时势分析报告，并把摘自其中一份报告的会员申请表收录在本附录里。

注意：有关文档完全按原文收录，一字未改。

《GNU通用公共许可证》

GNU Emacs的一切复本都是在《GNU通用公共许可证》的规定下发行的，大家可能更熟悉“*copyleft*”（译注1）的叫法。根据《GNU通用公共许可证》的基本精神，

译注1：这是一个臆造出来的单词，与“版权”的英文单词“copyright”正好相对。

任何一位拥有GNU Emacs的人都有权向他人提供Emacs的复本；而任何一位收到Emacs的人都不得给Emacs发行版本增加任何限制规定；如果有人想发行任何针对Emacs的改进，就必须像原来的程序那样遵守同样的规定——即必须按《GNU通用公共许可证》制度发行他或她所做的改进。为传播Emacs而收取费用是允许的；从这个方面看，“自由软件”并不一定就是廉价的。总之，你不能剥夺任何人（包括你的顾客在内）使用和传播这个程序的权利。这份许可证将保证GNU Emacs和其他程序永远都是自由的。

《GNU通用公共许可证》已经成为UNIX文化的一个重要组成部分。它对计算机软件行业里一种常见的丑陋行径——找个不错的公共域软件，对它做些改进，然后把经过改进的程序视为一项私有财产——进行了抵制。程序员之所以会把他们辛辛苦苦写出来的程序放到公共域里，其目的就是希望它们能够被更多的人所共享；可这种美好的愿望却因为办法的简单而给少数人以可乘之机。通过抵制上述行径，《GNU通用公共许可证》把对软件作者这种意愿的保护向前推进了一大步。如今，希望与他人共享开发成果的程序员对软件进行“*copyleft*”（即采用《GNU通用公共许可证》作为其发行许可证）的现象越来越普遍。

自由软件基金会的这份“通用公共许可证”在拟订时就考虑到要让任何程序都能很容易地使用它。在这份许可证正文的后面，自由软件基金会告诉你如何把它应用到你自己的软件上。FSF会定期对这份许可证进行修订，而它目前正出处于一种流动状态。我们这里提供的Version 2是它最新的版本。在这个版本之前，FSF还曾推出过几个过渡性版本；此外，为解决“*copylefting*”函数库的许可证问题，FSF还曾推出过一份特殊的版本。

《GNU通用公共许可证》是一份非常精炼的法律文书。并且，正像这份许可证所承诺的那样，今后的版本仍将坚持下面这个版本里所体现的精神。它可能会在遣词造句等细节上有所变化，但你做为一名用户的基本权利肯定不会被动摇。

《GNU 通用公共许可证》全文（译注 2）

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

译注 2：《GNU 通用公共许可证》是一份法律文书，理应由自由软件基金会做出解释；为了避免不必要的误导和纠纷，我们没有翻译它。而如果读者想“copyleft”自己的软件，就应该把《GNU 通用公共许可证》——当然是英文的，加入到它里面；这也是我们不翻译它的原因之一。

Also, for each author' s protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone' s free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program' s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution,

a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <19yy> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items — whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit

linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

以上信息直接取自 Emacs 自带的 *COPYING* 文件，除了它的格式，我们没有对它的内容做任何修改。大家可以在Emacs发行版本的*etc*目录里找到这个文件。当然，根据 Emacs 发行版本的具体情况，某些细节可能会与这里有所不同。

《GNU 宣言》

自由软件基金会的另一份基本文件是《GNU宣言》。这份宣言介绍了自由软件基金会存在的原因和目的。更可贵的是，它阐明了FSF对整个软件行业的看法。你可能不赞同它的观点，但你应该读读它。与刚才介绍的《GNU 通用公共许可证》相比，它的长度非同一般（虽然也不是太长），所以我们不准备把它的原文收录在这里。如果你在机器里保留了GNU Emacs的源文件，就可以在文件*"emacs-source"/etc/GNU*里找到这份宣言，其中的“emacs-source”是存放Emacs源代码发行文件的那个目录。在《GNU Emacs Manual》（GNU Emacs 手册）里也能找到这份宣言。

自由编程联盟

如果你关注自由软件，就不会没有听说过自由编程联盟（League for Programming Freedom，简称LPF）组织。这个联盟所关心的问题并不仅仅是自由软件，它针对的是对各种软件的可持续开发造成威胁的一切行为。自由编程联盟反对的是最近一个时期以来为软件申请专利和把命令语言版权化的不良趋势。正如这个联盟在它的一份时势分析报告里阐述的那样：它“与国会制定的法律制度（程序受版权保护）并不对立。我们的目标是扭转那些迎合特殊利益的法官们所造成的最新趋势，这些趋势通常极其明显地违背了国会的公众利益原则。”

LPF认为设计和技术上而不是具体产品上的专利已经使软件的日常开发工作陷入了一种在法律上非常危险的境地，威胁到了程序员开发软件的自由。给技术和方法（而非具体发明创造）颁发专利造成了极为严重和广泛的垄断。大家不妨设想一下，如果有人在20世纪50年代为链接列表（linked list，一种数据结构）申请了专利，那么今天的程序设计将如何发展。让软件开发人员去确定他们自己的设计和技术是否已经被申请专利不仅浪费时间和金钱，而且也极为困难。无论程序员多么聪明，

也不可能写出完全不依赖老技术的新软件，而它们当中的每一个都有可能已经被别人申请了专利。

事实上，软件专利和命令语言版权对一切软件开发工作都造成了威胁，但它们对自由软件的影响则可以用“致命”一词来形容。专利许可证中的条款与自由软件的传播方式是水火不相容的；比如说，如果软件自由传播，就不可能准确地统计出有多少复本正在流通，当然也就无法计算版税。你将无法把一个像 GNU Emacs 这样的程序转送给别人——除非你已经事先向这次交易所涉及的版权和专利拥有人全都做出了通报。更为严重的是，计算机上的公共服务程序有很多都是由自由软件的开发人员研制的；他们当中的很多人可能无法承受强加在他们正在开发的这类程序上的法律开支或软件许可税费。

LPF在1989年5月和1990年8月曾经引起了社会的广泛关注。当时，为了抗议Lotus Development Corporation（即中国读者比较熟悉的美国莲花软件公司，该公司的知名产品是 Lotus 1-2-3 电子表格程序）针对 Paperback Software, Inc. 公司就后者侵犯前者的操作界面版权而提起的法律诉讼，LPF 包围了 Lotus 软件公司在麻萨诸塞州剑桥市的总部。在这类被大众称之为“观感官司”的诉讼活动里，原告方坚持认为两种软件在用户操作界面方面的相似已经足以构成版权侵犯。Lotus不仅打赢了这场官司，还引发了几场同样的诉讼，而且已经有几场官司又取得了相同的胜利。LPF 认为这种诉讼构成了垄断而且威胁到了软件工业的创造性——软件公司将被迫编写出一些与现有软件不相兼容的软件，而且将不愿意再对现有产品进行改进。

以下内容节选自 LPF 章程中阐明的奋斗目标：

1. 证明并引起公众的注意在各类计算机程序上存在的限制和垄断，而这种垄断阻止或限制了人们对特定种类的计算机程序的开发权。
2. 从公众利益出发，研究出必要的针对性办法和手段——包括教育、科研、公众集会、立法听证会以及干涉与公众利益问题有关的司法庭审（但不以法庭为针对目标）等，以有效地阻止、预防或限制这种垄断行为。
3. 支持并参加从事于和涉及上述活动的一切商务或其他活动，但这些活动必须是依麻萨诸塞州州法（General Law）第 180 章组建的公司能够依法实施的。

LPF希望在财务方面能够得到大家的支持并邀请你成为它的会员。在我们写本书的时候，它的会员年费是：程序员、经理和教授每人$42美元，学生每人$10.50美元，

其他职业每人 $21 美元。捐款和年费主要用于“补偿当事人、印制手持标语/示威牌/横幅等有可能劝说法庭、立法者和公众的东西”。LPF 每年还会向它的会员印发一些邮件和宣传品。它已经写出了两份时势分析报告，这些报告对有关事态的现状和应该采取的补救措施做了详细的论述。不过，这两份报告实在是太长了，我们没有办法把它们收录在这里；你可以通过向 *lpf@uunet.uu.net* 发电子邮件的办法来获得其 Texinfo 格式的文档。其他问题请致电（617）621-7084。

如果你愿意加入，请按下页表格中的要求提出申请；这份申请表是从该联盟的一份时势分析报告里摘抄下来的（译注 3）：

为了加入 LPF，请把一张支票和个人资料寄往：

League for Programming Freedom
1 Kendall Square #143
P. O. Box 9171
Cambridge, Massachusetts 02139

（如果在美国以外，请寄送一张在美国有分支机构的银行的美元支票，这可以节省我们的支票兑付费用。）

你的姓名：

你的邮寄地址：LPF 将按这个地址寄送邮件（每年几份）；请注明它是家庭地址还是工作地址。

你的工作单位和职务：

你的电话号码：请注明它是家庭电话还是工作电话。

你的电子邮件地址：便于我们联系你参加示威或写请愿信活动。（如果你不想我们联系你参加这类活动，请注明。但希望你能把电子邮件地址告诉我们，发电子邮件将节省我们的纸张和邮费。）

你是否有LPF可以用来加以宣传以增加公众印象的个人事项？比如说，如果你是或者曾经是一位大学教授、企业执行官，或者写出过有良好声誉的软件，请告诉我们。

你是否愿意为 LPF 的活动提供帮助？

译注 3：为方便中国读者，我们对这份申请表进行了翻译，但在格式上采取了符合中国人习惯的写法。它最终的解释权仍属于 League for Programming Freedom。

附录七

请支持自由软件基金会

即使有自愿者参加，软件开发和软件维护也是代价高昂的。我们必须告诉读者、而你们也应该知道：自由软件基金会并在真空里工作。他们的出版物一直在呼吁以下捐助：

- 计算机硬件
- 劳动（即你的时间）
- 金钱

我们无法预知FSF在某个特定的时期最需要哪些硬件或者劳动；如果你想贡献你的智慧或设备，请直接与他们联系。但作为劳动交换的一种抽象手段、FSF肯定需要支出金钱。如果你想做出捐助，请按下面的地址与他们联系：

Free Software Foundation, Inc.
59 Temple Place, Suite 330
Boston, MA 02111-1307
phone: (617) 542-5942
fax: (617) 542-2652
email: *gnu@prep.ai.mit.edu*
WWW: *http://twws1.vub.ac.be/ studs/ tw45639/ fsf.htm*

附录八

Emacs 编辑命令速查表

以下这些速查表都按用途进行了分类，表格的先后顺序与书中的讨论大致对应。但因为它们只是些速查表，所以不可能同时做到细致和全面；对一个像Emacs这样的规模宏大而又功能全面的文本编辑器来说，这不能不是一个遗憾。我们已经尽了最大的努力让这些速查表能够兼顾完备和简捷；但我们能够保证，即使我们犯了错误，也只错在“简捷”上而不是“完备”上。

表 H-1：对文件和编辑缓冲区进行操作（参见第一、四章）

键盘操作	命令名称	动作
emacs	（无）	在shell提示符处输入，启动Emacs
C-x C-f *Files → Open File*	**find-file**	查找文件并打开它
C-x C-v	**find-alternate-file**	读入另外一个文件，替换掉用“**C-x C-f**”读入的文件
C-x i *Files → Insert File*	**insert-file**	把文件插入到光标的当前位置
C-x C-s *Files → Save Buffer*	**save-buffer**	保存文件
C-x C-w *Files → Save Buffer As*	**write-file**	把编辑缓冲区内容写入一个文件

表 H-1: 对文件和编辑缓冲区进行操作（参见第一、四章）（续）

键盘操作	命令名称	动作
C-x C-c *Files → Exit Emacs*	**save-buffers-kill-emacs**	退出 Emacs
C-z	**suspend-emacs**	挂起Emacs（用**exit**或**fg**命令唤醒）
C-x b	**switch-to-buffer**	移动到指定的编辑缓冲区
C-x C-b *Buffers → List All Buffers*	**list-buffers**	显示编辑缓冲区清单

表 H-2: 移动光标（参见第二章）

键盘操作	命令名称	动作
C-f	**forward-char**	光标前移一个字符（右）
C-b	**backward-char**	光标后移一个字符（左）
C-p	**previous-line**	光标前移一行（上）
C-n	**next-line**	光标后移一行（下）
ESC f	**forward-word**	光标前移一个单词
ESC b	**backward-word**	光标后移一个单词
C-a	**beginning-of-line**	光标移到行首
C-e	**end-of-line**	光标移到行尾
C-v	**scroll-up**	屏幕上卷一屏
ESC v	**scroll-down**	屏幕下卷一屏
ESC <	**beginning-of-buffer**	光标前移到文件头
ESC >	**end-of-buffer**	光标后移到文件尾
C-l	**recenter**	重新绘制屏显画面，当前行放在画面中心处

表 H-3: 删除文本、恢复文本、标记文本块（参见第二章）

键盘操作	命令名称	动作
C-d	**delete-char**	删除光标位置上的字符
DEL	**delete-backward-char**	删除光标前面的字符
ESC d	**kill-word**	删除光标后面的单词
ESC DEL	**backward-kill-word**	删除光标前面的单词

表 H-3: 删除文本、恢复文本、标记文本块（参见第二章）（续）

键盘操作	命令名称	动作
C-k	**kill-line**	从光标位置删除到行尾
C-w *Edit → Cut*	**kill-region**	删除文本块
ESC w 或 **C-Insert** *Edit → Copy*	**kill-ring-save**	复制文本块（以便用“**C-y**”命令粘贴）
C-y 或 **S-Insert** *Edit → Paste Most Recent*	**yank**	恢复被删除的文本
ESC y	**yank-pop**	在用过“**C-y**”命令以后粘贴更早删除的文本
C-@ 或 **C-SPACE**	**set-mark-command**	标记文本块的开始（或结束）位置
C-x C-x	**exchange-point-and-mark**	互换光标和文本标记的位置

表 H-4: 命令的中止执行和编辑操作的撤销（参见第二章）

键盘操作	命令名称	动作
C-g	**keyboard-quit**	放弃当前命令
C-x u	**advertised-undo**	撤销上一次编辑（可以重复施用）
C-_ 或 **C-/** *Edit → Undo*	**undo**	撤销上一次编辑
（无） *Files → Revert Buffer*	**revert-buffer**	把编辑缓冲区恢复到上次对文件进行存盘（或者自动存盘）时的状态

表 H-5: 交换文本位置和改变文本的大小写（参见第二章）

键盘操作	命令名称	动作
C-t	**transpose-chars**	交换两个字符的位置
ESC t	**transpose-words**	交换两个单词的位置
C-x C-t	**transpose-lines**	交换两个文本行的位置
ESC c	**capitalize-word**	把单词的首字母改为大写
ESC u	**upcase-word**	把单词的字母全部改为大写
ESC l	**downcase-word**	把单词的字母全部改为小写

表 H-6: 查找与替换（参见第三章）

键盘操作	命令名称	动作
C-s	**isearch-forward**	向文件尾方向开始递增查找操作
C-r	**isearch-backward**	向文件头方向开始递增查找操作
RETURN	（无）	退出一次成功的查找操作
C-g	**keyboard-quit**	取消递增查找操作（可能需要连按两次这个组合键）
DEL	（无）	删除查找字符串中不正确的字符
C-s RETURN *Search → Search*	（无）	向文件尾方向开始非递增查找操作
C-r RETURN *Search → Search Backwards*	（无）	向文件头方向开始非递增查找操作
ESC % *Search → Query-Replace*	**query-replace**	进入查询－替换

表 H-7: 查询－替换操作中使用的命令（参见第三章）

键盘操作	动作
SPACE 或 **y**	替换并前进到下一个位置
DEL 或 **n**	不替换：前进到下一个位置
.	在当前位置做替换后退出操作
,	替换并暂停（按空格键或“**y**”继续）
!	对其余全部进行替换，不再要求询问
^	返回前一次进行了替换的位置
RETURN	退出查询－替换操作

表 H-8: 拼写和简写词汇（参见第三章）

键盘操作	命令名称	动作
ESC $ *Edit → Spell → Check Word*	**ispell-word**	检查光标位置上的单词或者光标后面的单词
（无） *Edit → Spell → Check Region*	**ispell-region**	检查文本块里的单词

表 H-8: 拼写和简写词汇（参见第三章）（续）

键盘操作	命令名称	动作
（无）	**spell-word**	检查光标位置上的单词或者光标后面的单词
（无）	**spell-buffer**	检查当前编辑缓冲区的拼写
（无）	**abbrev-mode**	进入（或退出）单词简写模式
C-x a - 或 **C-x a i g**	**inverse-add-global-abbrev**	输入全局性简写词之后，输入其短语定义
C-x a i l	**inverse-add-local-abbrev**	输入局部性简写词之后，输入其短语定义
（无）	**write-abbrev-file**	保存简写词汇文件
（无）	**list-abbrevs**	查看简写词汇表

表 H-9: 窗口（参见第四章）

键盘操作	命令名称	动作
C-x 2 *Files → Split Window*	**split-window-vertically**	把当前窗口分割为上、下排列的两个窗口
C-x o	**other-window**	移动到其他窗口；如果有多个窗口，按顺时针方向移动到下一窗口
C-x 0	**delete-window**	删除当前窗口
C-x 1 *Files → One Window*	**delete-other-windows**	删除所有窗口，只保留当前窗口
C-x ^	**enlarge-window**	加高当前窗口
（无）	**shrink-window**	压低当前窗口
ESC C-v	**scroll-other-window**	对其他窗口做卷屏操作
C-x 4 f	**find-file-other-window**	在其他窗口里查找并打开一个文件

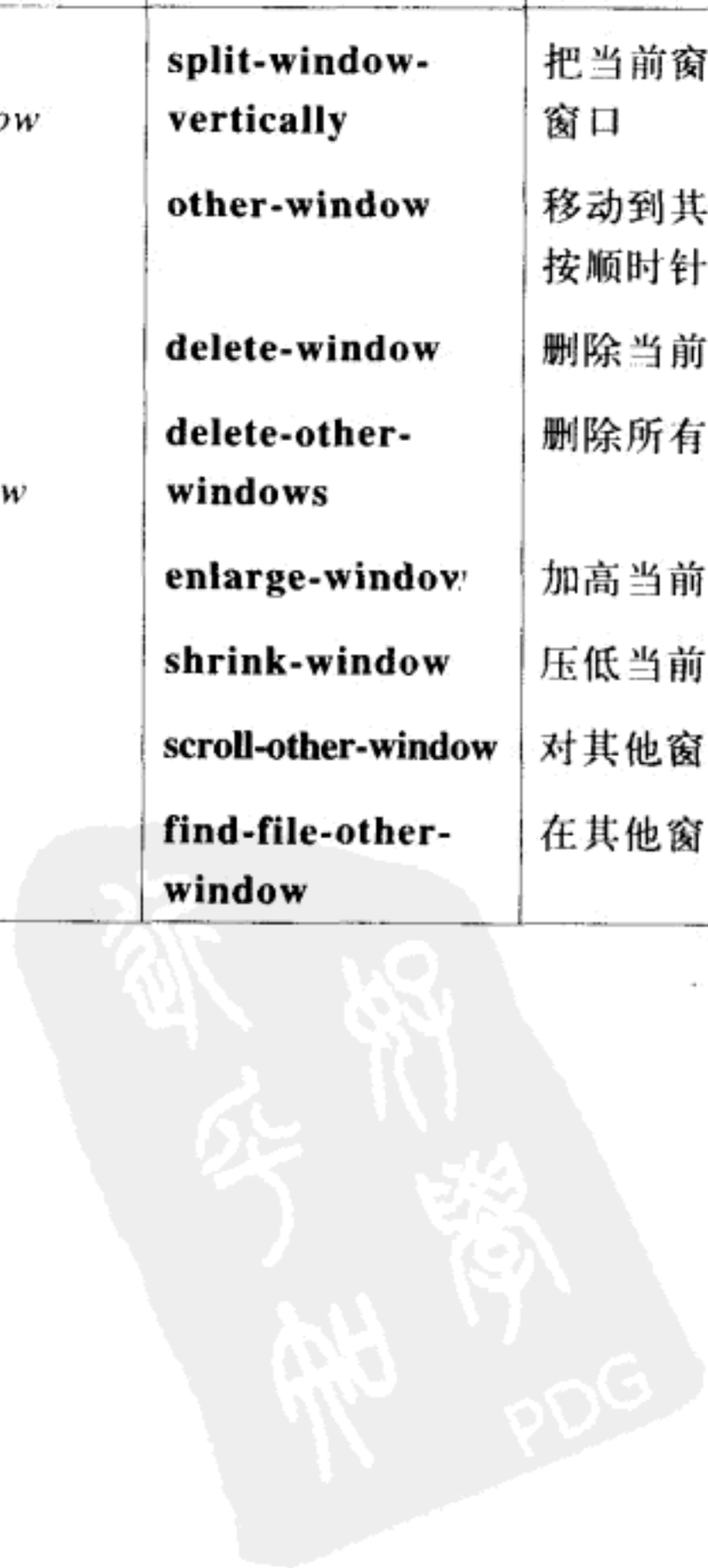


表 H-10：窗格（参见第四章）

键盘操作	命令名称	动作
C-x 5 o	**other-frame**	移动到其他窗格
C-x 5 2 *Files → Make New Frame*	**make-frame**	创建一个新窗格
C-x 5 0 *Files → Delete Frame*	**delete-frame**	删除当前窗格
C-x 5 f	**find-file-other-frame**	在一个新窗格里查找文件
C-x 5 b	**switch-to-buffer-other-frame**	创建新窗格并显示另一个编辑缓冲区

表 H-11：书签（参见第四章）

键盘操作	命令名称	动作
C-x r m *Search → Bookmarks → Set bookmark*	**bookmark-set**	在当前光标位置处设置一个书签
C-x r b *Search → Bookmarks → Jump to bookmark*	**bookmark-jump**	跳转到书签指示的位置

表 H-12：发送邮件（参见第六章）

键盘操作	命令名称	动作
C-x m	**mail**	打开“*mail*”编辑缓冲区
C-x 4 m	**mail-other-window**	在一个新窗口里打开“*mail*”编辑缓冲区
C-x 5 m	**mail-other-frame**	在一个新窗格里打开“*mail*”编辑缓冲区
C-c C-f C-t *Headers → To*	**mail-to**	移动到“To:”栏
C-c C-f C-c *Headers → Cc*	**mail-cc**	移动到“CC:”栏（如果没有就创建之）
C-c C-f C-s *Headers → Subject*	**mail-subject**	移动到“Subject:”栏

表 H-12: 发送邮件（参见第六章）（续）

键盘操作	命令名称	动作
C-c C-w *Mail → Insert Signature*	**mail-signature**	插入“`.signature`”文件的内容
C-c C-c *Mail → Send Message*	**mail-send-and-exit**	发送邮件并退出“`*mail*`”编辑缓冲区
（无）	**define-mail-alias**	为某个名字或邮件表定义一个缩写形式

表 H-13: 读取邮件（参见第六章）

键盘操作	命令名称	动作
（无） *Tools → Read Mail*	**rmail**	启动 RMAIL 读取邮件
SPACE	**scroll-up**	卷屏，查看此消息的下一个画面
DEL	**scroll-down**	卷屏，查看此消息的上一个画面
.	**rmail-beginning-of-message**	移动到此消息的开头
n *Move → Next*	**rmail-next-undeleted-message**	移动到下一条消息
p *Move → Previous*	**rmail-previous-undeleted-message**	移动到上一条消息
< *Move → First*	**rmail-first-message**	移动到第一条消息
> *Move → Last*	**rmail-last-message**	移动到最后一条消息
j	**rmail-show-message**	如果这个命令的前面有一个数字“*n*”，跳到第 *n* 条消息
d *Delete → Delete*	**rmail-delete-forward**	给邮件加上待删除标记，然后移动到下一个
C-d	**rmail-delete-backward**	给邮件加上待删除标记，然后移动到上一个

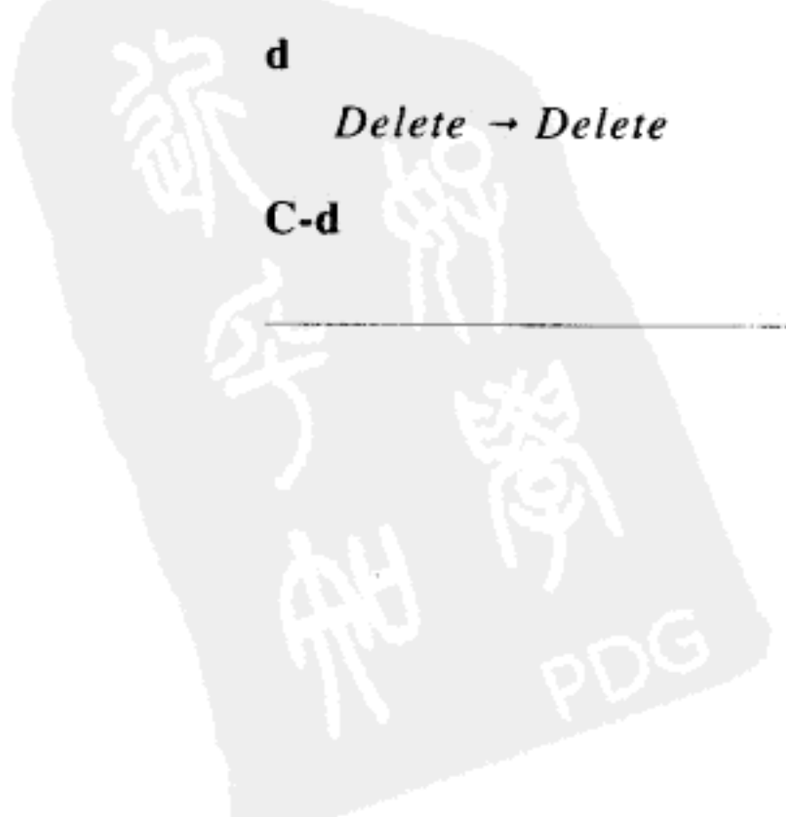


表 H-13: 读取邮件（参见第六章）（续）

键盘操作	命令名称	动作
u *Delete → Undelete*	**rmail-undelete-previous-message**	去掉邮件消息上的待删除标记
x *Delete → Expunge*	**rmail-expunge**	删除已经加有待删除标记的全部消息
o *filename* RETURN *Classify → Output (Rmail)*	**rmail-output-to-rmail-file**	把邮件消息保存为RMAIL 文件格式
C-o *filename* RETURN *Classify → Output (inbox)*	**rmail-output**	把邮件消息保存为 UNIX 邮件文件格式（一个标准的 ASCII 文本文件）

表 H-14: RMAIL 邮件清单（参见第六章）

键盘操作	命令名称	动作
h	**rmail-summary**	显示邮件清单
d	**rmail-summary-delete-forward**	给消息加上待删除标记（在消息序号前出现字母“D”标记）
u	**rmail-summary-undelete**	去掉当前消息上的待删除标记
n	**rmail-summary-next-msg**	移动到下一条消息并把它显示在 RMAIL 窗口里
p	**rmail-summary-previous-msg**	移动到上一条消息并把它显示在 RMAIL 窗口里
x	**rmail-summary-expunge**	删除所有加有待删除标记的消息
q	**rmail-summary-quit**	退出 RMAIL

表 H-15: shell 模式命令（参见第五章）

键盘操作	命令名称	动作
（无）	**shell**	进入 shell 模式
C-c C-c *Signals → Break*	**comint-interrupt-subjob**	中断当前作业；相当于 UNIX 的 shell 中的“**C-c**”组合键

PDG

表 H-15: shell 模式命令（参见第五章）（续）

键盘操作	命令名称	动作
C-d	**comint-delchar-or-maybe-eof**	如果是在编辑缓冲区的末尾，送出 EOF 字符；如果是在其他位置，删除一个字符
C-c C-d *Signals → EOF*	**comint-send-eof**	送出 EOF 字符
C-c C-u	**comint-kill-input**	删除当前行；相当于 UNIX 的 shell 中的“**C-u**”组合键
C-c C-z *Signals → STOP*	**comint-stop-subjob**	对非 X 用户，挂起或者停止一个作业；相当于 UNIX 的 shell 中的“**C-z**”组合键
ESC p *In/Out → Previous Input*	**comint-previous-input**	检索上一个命令（可以重复执行以找回更早的命令）
ESC n *In/Out → Next Input*	**comint-next-input**	检索下一个命令（可以重复执行以找回更近的命令）
RETURN	**comint-send-input**	送出当前行上的输入
TAB *Complete → Complete Before Point*	**comint-dynamic-complete**	自动补足当前命令、文件名或者变量名
C-c C-o *In/Out → Kill Current Output Group*	**comint-kill-output**	删除最后一条命令的输出
C-c C-e *In/Out → Show Maximum Output*	**comint-show-maximum-output**	把输出内容的最后一行移到窗口的底部

表 H-16: 用来创建正则表达式的字符（参见第三章）

字符	匹配情况
^	匹配行首
$	匹配行尾
.	匹配任意单个字符（类似于文件名中的问号 ?）

PDG

表 H-16: 用来创建正则表达式的字符（参见第三章）（续）

字符	匹配情况
.*	匹配任意（零个以上）个字符
\<	匹配单词的开头
\>	匹配单词的结尾
[]	匹配方括号中的任何一个字符；比如“[a-z]”表示匹配任意一个字母表字符

表 H-17: Dired 命令（参见第五章）

键盘操作	命令名称	动作
C-x d *Files → Open Directory*	**dired**	启动 Dired
C *Operate → Copy to*	**dired-do-copy**	复制文件
d *Mark → Flag*	**dired-flag-file-deletion**	给文件加上待删除标记
f *Immediate → Find This File*	**dired-find-file**	编辑文件
g *Files → Revert Buffer*	**revert-buffer**	从磁盘上重新读入目录
n *Subdir → Next Dirline*	**dired-next-line**	移动到下一行
q	**dired-quit**	退出 Dired
R *Operate → Rename to*	**dired-do-rename**	重新命名文件
u *Mark → Unmark*	**dired-unmark**	去掉待操作标记
x	**dired-do-flagged-delete**	删除加有待删除标记“D”的文件
Z *Operate → Compress*	**dired-do-compress**	对文件进行压缩或解压缩操作

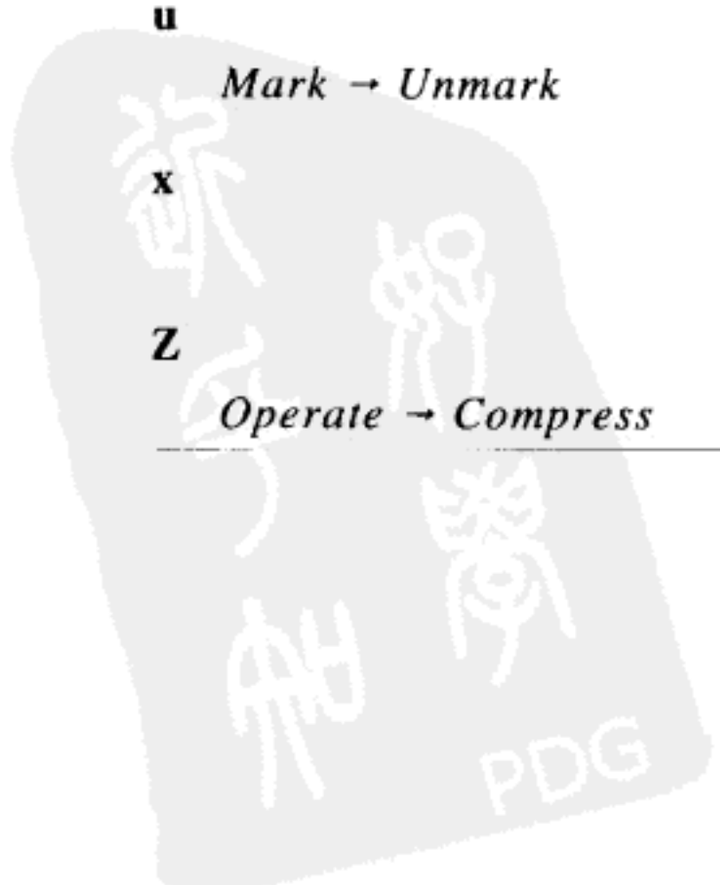


表 H-17: Dired 命令（参见第五章）（续）

键盘操作	命令名称	动作
ESC DEL *Mark → Unmark All*	**dired-unmark-all-files**	把所有文件上的各种待操作标记（不管它是什么）都去掉
> *Subdir → Next Dirline*	**dired-next-dirline**	移动到下一个目录
< *Subdir → Prev Dirline*	**dired-prev-dirline**	移动到上一个目录
s	**dired-sort-toggle-or-edit**	对 Dired 画面按日期或按文件名进行排序（在两者之间切换）

表 H-18: nroff 模式（参见第九章）

键盘操作	命令名称	功能
（无）	**nroff-mode**	进入 nroff 模式
ESC n	**forward-text-line**	把光标移动到下一个文本行
ESC p	**backward-text-line**	把光标移动到上一个文本行
ESC ?	**count-text-lines**	统计文本块中的文本行数
（无）	**electric-nroff-mode**	进入一个副编辑模式。在这个模式里，若输入必须配对出现的 **nroff** 命令组中的第一个，按“**C-j**”组合键，Emacs 就会自动插入该配对命令组中的第二个命令
C-j	**electric-nroff-newline**	只能用在 nroff 配对模式里，自动插入一组必须配对出现的 **troff** 宏定义标记中的第二个

表 H-19: TEX 模式（参见第九章）

键盘操作	命令名称	功能
（无）	**tex-mode**	根据文件内容进入 TEX 模式或 LATEX 模式
（无）	**plain-tex-mode**	进入 TEX 模式
（无）	**latex-mode**	进入 LATEX 模式

PDG

表 H-19: TEX模式（参见第九章）（续）

键盘操作	命令名称	功能
C-j	**tex-terminate-paragraph**	插入两个硬回车（标准的段落结束标志）并检查段落的语法
C-c {	**tex-insert-braces**	插入两个花括号，然后把光标放在它们中间
C-c }	**up-list**	如果光标在两个花括号中间，就把光标放到紧跟在右括号后面的位置上
（无） *TeX → Validate Buffer*	**validate-tex-buffer**	检查编辑缓冲区有无语法错误
C-c C-b *TeX → TeX Buffer*	**tex-buffer**	对编辑缓冲区进行TEX或LATEX排版处理
C-c C-l *TeX → TeX Recenter*	**tex-recenter-output-buffer**	把排版信息画面显示在屏幕上，显示（至少）最近一条出错信息
C-c C-r *TeX → TeX Region*	**tex-region**	对文本块进行TEX或LATEX排版处理
C-c C-k *TeX → TeX Kill*	**tex-kill-job**	中断 TEX 或 LATEX 的排版处理工作
C-c C-p *TeX → TeX Print*	**tex-print**	打印 TEX 或 LATEX 的排版输出
C-c C-q *TeX → Show Print Queue*	**tex-show-print-queue**	查看打印队列
C-c C-e	**tex-close-latex-block**	只能用在LATEX模式里，自动插入某个命令组中的第二个命令

表 H-20: 宏（参见第十章）

键盘操作	命令名称	动作
C-x (	**start-kbd-macro**	开始录制一个宏
C-x)	**end-kbd-macro**	结束宏的录制工作
C-x e	**call-last-kbd-macro**	执行最近一次录制的宏
C-u C-x (	**universal-argument; start-kbd-macro**	执行最近一次录制的宏，然后允许再增加新按键组合

表 H-20: 宏（参见第十章）（续）

键盘操作	命令名称	动作
C-x q	**kbd-macro-query**	在宏定义里插入一个查询
C-u C-x q	（无）	在宏定义里插入一个递归编辑
ESC C-c	**exit-recursive-edit**	退出递归编辑

表 H-21: Gnus 新闻阅读器“Group”编辑缓冲区操作命令（参见第六章）

键盘操作	命令名称	动作
（无）	**gnus**	启动 Gnus
SPACE	**gnus-group-read-group**	阅读光标位置处的新闻组里的文章
u	**gnus-group-unsubscribe-current-group**	订阅或者撤销订阅这个新闻组
j	**gnus-group-jump-to-group**	提示输入一个新闻组名称以转入其中（可以转到未曾订阅的新闻组）
l	**gnus-group-list-groups**	列出已订阅并且有新闻可读的新闻组
A k	**gnus-group-list-killed**	列出那些被“*.newsrc.eld*”文件里的语句所排除掉的新闻组
L	**gnus-group-list-all-groups**	列出这个服务器上全部可用的新闻组
g	**gnus-group-get-new-news**	取回启动 Gnus 以来新收到的新闻
a	**gnus-group-post-news**	为这个新闻组写一篇新文章
C-x C-t	**gnus-group-transpose-groups**	交换当前行和上一行的位置
q	**gnus-group-exit**	退出新闻功能并刷新“*.newsrc*”文件

表 H-22: Gnus 新闻阅读器“Summary”编辑缓冲区操作命令（参见第六章）

键盘操作	命令名称	动作
SPACE	**gnus-summary-next-page**	文章前卷
DEL	**gnus-summary-prev-page**	文章后卷
l	**gnus-summary-goto-last-article**	移动到刚读过的最后一篇文章（如果移动得太快，可以用这条命令返回）

表 H-22: Gnus 新闻阅读器“Summary”编辑缓冲区操作命令（参见第六章）（续）

键盘操作	命令名称	动作
H f	**gnus-summary-fetch-faq**	取回这个新闻组的常见问题答疑文件
n	**gnus-summary-next-unread-article**	移动到下一篇文章
p	**gnus-summary-prev-unread-article**	移动到上一篇文章
u	**gnus-summary-tick-article-forward**	给当前文章加上未阅读标记，在它旁边放上一个惊叹号以保留它供今后阅读。如果重复输入此命令，则保存下一篇文章
C-o	**gnus-summary-save-article-mail**	以 UNIX 格式保存当前文章
o	**gnus-summary-save-article**	以 RMAIL 格式保存当前文章
q	**gnus-summary-exit**	返回“Group”编辑缓冲区
d	**gnus-summary-mark-as-read-forward**	给当前文章加上已阅读标记，从当前行开始向下移动
C-c C-f	**gnus-summary-mail-forward**	把这篇文章的副本发送给某人
g	**gnus-summary-show-article**	显示当前文章（特别适用于扩展了“Summary”窗口或者需要在这个窗口里移动的情况）
x	**gnus-summary-remove-lines-marked-as-read**	把带有已阅读标记的文章全部删掉
=	**gnus-summary-expand-window**	扩展“Summary”窗口，让它充满整个屏幕

表 H-23: Telnet（参见第七章）

键盘操作	命令名称	动作
（无）	**telnet**	进入 Telnet 模式
C-d	**comint-delchar-or-maybe-eof**	根据上下文，发送 EOF 字符或删除光标位置下的字符
RETURN	**telnet-send-input**	送出 Telnet 输入
C-c C-c	**telnet-interrupt-subjob**	中断当前作业；相当于 *shell* 中的“**C-c**”组合键

PDG

表 H-23: Telnet（参见第七章）（续）

键盘操作	命令名称	动作
C-c C-q	**send-process-next-char**	发送紧随其后的控制字符；相当于 shell 中的“**C-q**”组合键
C-c C-d	**comint-send-eof**	发送 EOF（文件尾）字符
C-c C-r	**comint-show-output**	让输出内容的第一行显示在窗口的顶部
C-c C-e	**comint-show-maximum-output**	让输出内容的最后一行显示在窗口的底部
C-c C-o	**comint-kill-output**	删除上一个命令的输出内容
C-c C-z	**telnet-c-z**	挂起或暂停这个作业；相当于 shell 中的“**C-z**”组合键
C-c C-u	**comint-kill-input**	删除当前行；相当于 shell 中的“**C-u**”组合键
ESC n	**comint-next-input**	查看此后输入的命令（重复此操作可查看到更靠后的命令）
ESC p	**comint-previous-input**	查看此前输入的命令（重复此操作可查看到更靠前的命令）

表 H-24: C 模式（参见第十二章）

键盘操作	命令名称	动作
ESC C-a	**beginning-of-defun**	移动到当前函数的开头
ESC C-e	**end-of-defun**	移动到当前函数的结尾
ESC C-h	**mark-c-function**	把光标放到函数的开头，把文本块标记放到函数的结尾

表 H-25: LISP 模式（参见第十二章）

键盘操作	命令名称	动作
ESC C-b	**backward-sexp**	移动到上一个 S- 表达式
ESC C-f	**forward-sexp**	移动到下一个 S- 表达式
ESC C-t	**transpose-sexps**	交换光标前后的两个 S- 表达式的位置
ESC C-@	**mark-sexp**	把文本块标记设置在当前 S- 表达式的末尾，把光标设置在当前 S- 表达式的开头

表 H-25: LISP 模式（参见第十二章）（续）

键盘操作	命令名称	动作
ESC C-k	**kill-sexp**	删除光标后面的那个 S- 表达式
ESC C-DEL	**backward-kill-sexp**	删除光标前面的那个 S- 表达式
ESC C-n	**forward-list**	移动到上一个列表
ESC C-p	**backward-list**	移动到下一个列表
ESC C-d	**down-list**	向前移动，进入下一级括号层次
（无）	**up-list**	向前移动，退出当前的括号层次
ESC C-u	**backward-up-list**	向后移动，退出当前的括号层次
ESC C-a	**beginning-of-defun**	移动到当前函数的开头
ESC C-e	**end-of-defun**	移动到当前函数的结尾
ESC C-h	**mark-defun**	把光标放到函数的开头，把文本块标记放到函数的结尾

表 H-26: FORTRAN 模式（参见第十二章）

键盘操作	命令名称	动作
C-c C-n	**fortran-next-statement**	向前移动一个语句
C-c C-p	**fortran-previous-statement**	向后移动一个语句
ESC C-a	**beginning-of-fortran-subprogram**	移动到当前子程序的开头
ESC C-e	**end-of-fortran-subprogram**	移动到当前子程序的结尾
ESC C-h	**mark-fortran-subprogram**	把光标放到子程序的开头，把文本块标记放到子程序的结尾

表 H-27: Emacs 中的鼠标操作（参见第十四章）

鼠标动作	命令名称	对应的 Emacs 标准命令（和组合键）
左键	**x-mouse-set-point**	（把 Emacs 光标移动到鼠标光标位置）
中键	**x-paste-text**	yank（“**C-y**”）
右键	**x-cut-text**	copy-region-as-kill（“**ESC w**”）
C- 中键	**x-cut-and-wipe-text**	kill-region（“**C-w**”）

表 H-27: Emacs 中的鼠标操作（参见第十四章）（续）

鼠标动作	命令名称	对应的 Emacs 标准命令（和组合键）
C- 右键	**x-mouse-select-and-split**	split-window-vertically（“**C-x 2**”）
S- 中键	**x-cut-text**	copy-region-as-kill（“**ESC w**”）
S- 右键	**x-paste-text**	yank（“**C-y**”）
C-S- 右键	**x-mouse-keep-one-window**	delete-other-windows（“**C-x 1**”）

表 H-28: 在线帮助系统（参见第十六章）

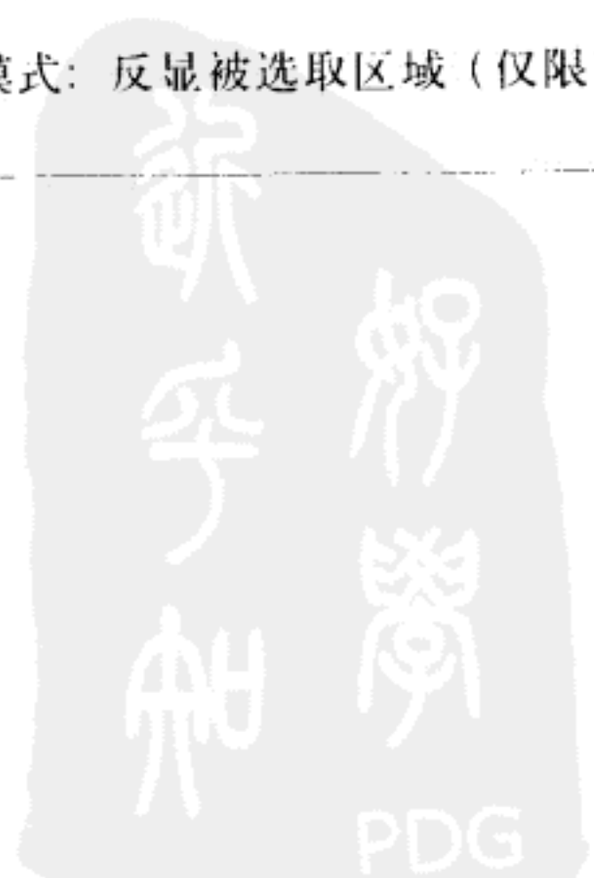
键盘操作	命令名称	回答的问题
C-h c	**describe-key-briefly**	这个按键组合将运行哪个命令
C-h k	**describe-key**	这个按键组合将运行哪个命令？这个命令的作用是什么
C-h l	**view-lossage**	最近输入的 100 个字符是什么
C-h w	**where-is**	这个命令的按键绑定是什么
C-h f	**describe-function**	这个函数的作用是什么
C-h v	**describe-variable**	这个变量的含义是什么？它有哪些可取值
C-h m	**describe-mode**	查看当前编辑缓冲区所在编辑模式的有关资料
C-h b	**describe-bindings**	这个编辑缓冲区都有哪些按键绑定
C-h a	**command-apropos**	这个概念都涉及到哪些命令
C-h t	**help-with-tutorial**	运行 Emacs 教程
C-h i	**info**	启动文档阅读器 **Info** 程序

表 H-29: Emacs 重要的编辑模式

模式	功能
基本模式（fundamental-mode）	默认模式，无特殊行为
文本模式（text-mode）	主编辑模式：书写文字材料（参见第二章）
缩进文本模式（indented-text-mode）	主编辑模式：自动缩进文本（参见第八章）
C 模式（c-mode）	主编辑模式：书写 C 语言程序（参见第十二章）

表 H-29: Emacs 重要的编辑模式（续）

模式	功能
FORTRAN 模式（fortran-mode）	主编辑模式：书写 FORTRAN 程序（参见第十二章）
LISP 模式（lisp-mode）	主编辑模式：书写 LISP 程序（参见第十二章）
RMAIL 模式（rmail-mode）	主编辑模式：阅读和组织电子邮件（参见第六章）
nroff 模式（nroff-mode）	主编辑模式：按 **nroff** 的要求对文件进行排版（参见第九章）
TEX 模式（tex-mode）	主编辑模式：按 TEX 的要求对文件进行排版（参见第九章）
LATEX 模式（latex-mode）	主编辑模式：按 LATEX 的要求对文件进行排版（参见第九章）
大纲模式（outline-mode）	主编辑模式：书写大纲（参见第八章）
shell 模式（shell-mode）	主编辑模式：在 Emacs 里运行一个 UNIX shell（参见第五章）
缩写模式（abbrev-mode）	副编辑模式：缩写单词（参见第二章）
自动换行模式（auto-fill-mode）	副编辑模式：激活字换行（word wrap）功能（参见第二章）
改写模式（overwrite-mode）	副编辑模式：打字时对字符进行改写而不是插入（参见第二章）
自动保存模式（auto-save-mode）	副编辑模式：按一定周期自动把文件保存到特殊的自动保存文件里（参见第二章）
行号模式（line-number-mode）	副编辑模式：在状态行上显示当前的行编号（参见第二章）
临时标记模式（transient-mark-mode）	副编辑模式：反显被选取区域（仅限 X 界面）（参见第二章）



词汇表

abbrev mode（简写词汇模式）

用来定义简写词汇的编辑模式，由用户定义的简写词汇在输入的时候被自动替换。可以为短语、长单词或容易拼写错误的单词定义简写词汇。Emacs的简写词汇功能（**vi**编辑器里也有类似功能）类似于Microsoft Word中的自动更正功能，但比后者出现得要早得多。关于简写词汇模式的详细讨论请参考第三章。

ange-ftp mode（ange-ftp模式）

一个很容易使用的FTP（File Transfer Protocol，文件传输协议）接口，作者是Andy Norman。用户可以用**find-file**命令打开被保存在因特网或者其他网络上的文件或目录，就像它们是在本地系统上一样。要想指定远程文件，请输入"*/user@systemname:/ pathtofile/filename*"。最开始的斜线字符和系统名与路径名之间的冒号很容易忘记；但缺少了它们，ange-ftp模式将不会工作。如果省略了路径名和文件名，Emacs将使用Dired来显示远程系统上的顶级目录。无须使用FTP命令来检索文件，它们完全能够用Dired命令来显示和复制。因此，ange-ftp模式很适合用来查看和下载远程文件。从Emacs 19.29开始，ange-ftp模式已经包括在Emacs的发行版本里了。

auto-fill mode（自动换行模式）

Emacs实现自动换行功能的一个副编辑模式。在自动换行模式下，当到达文本行尾的时候可以继续进行输入，Emacs会把换行符自动插入到适当位置。自动换行模式在缺省状态是关闭的。

auto-save file（自动保存文件，*#file#*）

Emacs会定期把用户的编辑缓冲区保存到一种叫做"自动保存（auto-save）"文件的临时文件里。在Emacs会话被非正常终止的时候，Emacs也会把文件保存到自动

保存文件里。比如，如果系统在用户正编辑着一个名为*budget*的文件时死机了，那么用户可以在系统恢复正常后找到一个名为*#budget#*的自动保存文件。关于自动保存文件的详细讨论请参考第二章。

backup file（**备份文件**，*file~*）

在Emacs保存文件的时候，它会先把当前版本复制到一个名字相同但后面多出一个波浪字符（~）的文件里。比如说，如果保存的文件是*budget*，Emacs会把前一个版本转移到文件*budget~*里。此后，如果用户决定不想保留自己所做的修改，就可以使用备份文件。关于备份文件的详细讨论请参考第二章。

body（**消息体/段落体**）

1. 邮件消息分为消息头（header，里面的信息用来指明谁是收信人和谁是发信人）和消息的内容两部分，后者就是消息体。2. 在大纲模式里，编辑缓冲区里的内容被分为标题（大纲的框架）和文本段落，后者构成了段落体。

bookmark（**书签**）

文件中一个有名字的位置。打开文件操作永远会把用户带到第一行；用书签打开文件，会把用户带到指定的位置。书签特别适合用在大纲文档等想在下次打开时直接回到上次离开位置处的文件里。关于书签的详细讨论请参考第四章。

browser（**浏览器**）

1. 一个允许用户访问WWW网的程序。它又分为图形化浏览器，比如Mosaic和Netscape，和文本浏览器，比如Lynx和W3模式。2. 查看某种格式的信息所使用的阅读器程序。Info模式就是一种用来阅读Info格式文档的浏览器。

buffer（**编辑缓冲区**）

屏幕上Emacs显示文件的一个副本或者空白区域以供用户进行输入的工作区。当用户编辑文件的时候，Emacs会把文件复制到一个同名的编辑缓冲区里；而磁盘上的文件则不会被修改。当保存编辑缓冲区的时候，Emacs会把用户做的修改复制到磁盘上文件里。除对应于文件的编辑缓冲区外，Emacs本身还会在阅读和书写电子邮件、配合UNIX shell工作以及使用日历等工作中创建专用的编辑缓冲区。

byte-compile（**字节编译**）

编译Emacs LISP文件（以“*.el*”为后缀名）的一种方法。对文件进行字节编译之后，它的后缀名字将变成“*.elc*”。经过字节编译的Emacs LISP函数加载和运行得更快。

calendar（**日历**）

输入“**ESC x calendar**”，Emacs就将在屏幕底部显示一个三个月的日历，并且会把光标放在当前日期上。在日历里，可以用日记条目标记日期和查看节假日。关于日历的详细讨论请参考第五章。

clipboard（**剪贴板**）

参见 kill ring（删除环）。

comment（**注释**）

程序设计语言或文本排版程序不进行处理的文本。Emacs 会在用户按下“**ESC ;**”时，根据用户所在的编辑模式自动插入注释语法。文本模式和基本模式等普通的编辑模式里没有定义注释语法。

completion（**自动补足功能**）

一种快捷输入方式：用户输入命令、变量或文件名的前几个字符再按下 **TAB** 键，Emacs就会自动补足它（如果它是惟一的）或者在一个“*Help*”窗口里列出候选。

copyleft

一种软件必须自由共享的共识，使接受方也能够按此方式共享这个软件。copyright（版权）限制了信息的使用，而根据附录四中自由软件基金会（Free Software Foundation，FSF）的条款，copyleft将保证软件能够继续被任何人自由共享。术语“copyleft”还被用来引述《GNU 通用公共许可证》。

cursor（**光标**）

光标指示了在编辑缓冲区里的特定位置。有时候，Emacs把光标也称为插入点。从技术上讲，插入点位于光标和前一个字符的中间。

customization（**定制**）

在缺省的情况下，Emacs有一套固定的行为方式。通过定制，可以把这种固定的行为方式改变为自己需要和喜欢的形式。对Emacs的定制是通过“*.emacs*”文件进行的。关于定制的详细讨论请参考第十一章。

cut（**剪切操作**）

删除文本，以便用于粘贴。

default（**缺省值**）

变量或选项的默认取值。比如说，在缺省的情况下，自动换行模式是关闭的。

default direction（**缺省绘制方向**）

在输入文本的时候，文本通常是从左向右出现的。在图形模式里，可移动的方向有8个。刚进入图形模式时，缺省绘制方向是右。如果指定了一个不同的方向——比如上、下、左上、右下等等，它就将成为缺省绘制方向。“**C-c C-f**”组合键的向前移动和“**C-c C-b**”组合键的向后移动都是相对缺省绘制方向而言的。

delete（**删除操作**）

删除文本且不把它保存在删除环里以供今后粘贴等操作使用的操作。

diary（**日记**）

Emacs允许为特定日期加注留言的功能。此后，Emacs将在特定日期到来时显示一

个提醒。日记条目可以设置在给定的日期、日期块或循环周期（比如每两星期一循环）上。关于日记的详细讨论请参考第五章。

digest（**邮件文摘**）

为求方便而单独做为一条消息发送的一组邮件消息。来自邮件表的消息经常是以邮件文摘的形式发送的。如果使用RMAIL阅读邮件，则命令“**ESC x undigestify-rmail-message**”将把邮件文摘分割为各个消息。在Gnus里，可以用“**C-d**”组合键把做为一个新闻组稿件的邮件文摘扩展开来。

Dired

Emacs的目录编辑器。可以通过Dired完成文件和目录上的各种操作，比如移动、压缩、删除、复制和字节编译等。关于Dired的详细讨论请参考第五章。

dot（**插入点**）

光标位置或插入点的同义词。在Emacs的在线帮助功能里会见到“dot”这个术语。

.emacs file（*.emacs*文件）

用来改变Emacs缺省行为的初始化文件。这个文件里的命令将在启动Emacs的时候运行。但Emacs没有这个文件也完全能够运行。我们在这本书里介绍了许多可以添加到“*.emacs*”文件里以改变Emacs某些行为的语句。

email（**电子邮件**）

电子化的邮件。Emacs自己带有一个名为RMAIL的邮件阅读程序和一个名为sendmail（不要把它与UNIX的sendmail命令混为一谈）的邮件发送程序。我们在第六章里对它们进行了讨论。Emacs还有面向邮件发送程序MH和vm的接口。对mh接口感兴趣的读者可以读一读Jerry Peek写的《MH & xmh: Email for Users and Programmers》。

file（**文件**）

磁盘上的一个存储区域。当用Emacs打开一个文件的时候，它会把文件复制到一个编辑缓冲区，即屏幕上的一个工作区域里。当保存修改的时候，Emacs又会把当前编辑缓冲区里的修改以改写方式复制回磁盘上的文件。

fill prefix（**缩进前缀**）

在文本模式和自动换行模式里，缩进前缀是定义并让Emacs插入到段落中的每一行开头去的一串字符。缩进前缀可以是一串空格（便于对文本进行缩进）或者是一个简单的大于号（>）；后者通常用来在电子邮件里区分引用性文字和上下段落。

flow control（**流控制**）

终端与主机进行通信的一种方式。有些计算机把“**C-s**”解释为停止接受输入、把“**C-q**”解释为重新开始接受输入。但“**C-s**”和“**C-q**”在Emacs里有其他的用途，如果将“**C-s**”用做流控制字符，终端就会在用户输入带“**C-s**”的Emacs命令的时

候被挂起来。可以通过“**ESC x enable-flow-control**”命令用“**C-**”和“**C-^**”分别来代替Emacs命令中的“**C-s**”和“**C-q**”。关于流控制的详细讨论请参考第十一章。

formfeed（换页符）

一个控制字符，通常显示为“^L”，它的作用是让打印机从新的一页开始打印。如果想在Emacs文件里插入一个formfeed换页符，请按下“**C-q C-l**”组合键。

frame（窗格）

显示一个Emacs编辑缓冲区的X窗口。在缺省的情况下，X窗口系统的用户只有一个窗格，但他们能够在一次Emacs会话里创建多个X窗口。在只有一个窗格的时候，系统名将显示为窗口的标题；在有多个窗格的时候，窗格的名字将是编辑缓冲区的名字。关于窗口和窗格的详细讨论请参考第四章。（字符终端上也能使用窗格，但它们在窗口化的操作界面里更有用）。

global abbreviations（全局简写词汇）

由用户定义的能够用在各种主编辑模式下的简写词汇。与此形成对比的是局部简写词汇，它们只能用在对它们做出定义的编辑模式里。因此，全局简写词汇适合用在想让Emacs进行自动替换的时候，而局部简写词汇适合用在只想在文本模式而不想用在C模式里的替换。关于简写词汇的详细讨论请参考第四章。

GNU

“GNU's Not UNIX（GNU不是UNIX）”的字头缩写。

Gnus

GNU Emacs 19自带的一个新闻阅读器。Gnus使Emacs用户能够在不退出Emacs的情况下阅读和投稿Usenet新闻。

header（消息头/控制头）

1. 在LaTeX等文件的开头对文档格式某些特征做出定义的部分。2. 在电子邮件里，在邮件消息的开头给出发信人、收信人、发信日期以及其他信息的标题部分。3. 嵌在文件里的版本控制信息，它们会在签出、签入和复原操作中自动刷新。

home directory（主目录）

个人目录。在一台UNIX系统上，它往往就是子目录*/home /username*。

home page（主页）

1. 启动WWW浏览器时看到的缺省页面。2. 在某给定WWW站点上看到的第一个页面。

hotlink（热链）

超链接的另一个术语。

hotlist（**收藏夹**）

WWW 上用户最喜欢访问的站点的超链表。

hyperlink（**超链接**）

文本或图形中指向某个本地或远程资源的链接。选择一个超链接将会进入指定的资源。

Hypertext Markup Language（HTML，**超文本标记语言**）

超文本标记语言，一种对文档按 WWW 显示要求进行排版的 ASCII 标记语言。HTML 文件也叫做 Web 页面，可以用一种人们称之为浏览器的查看器进行浏览。

incremental search（**递增查找**）

Emacs 中的一种查找方式，输入第一个字符时，查找工作就开始了。

initialization file（**初始化文件**）

参见“*.emacs*”文件。

kill（**删除操作**）

删除并保存到删除环。

kill file（**过滤文件**）

一个由新闻阅读器（比如 Gnus）读取的文件，它将根据用户设置的规则排除用户不想看到的消息。比如说，过滤文件可以把某特定用户投稿的消息或者关于某特定主题的消息过滤掉。

kill ring（**删除环**）

Emacs 存放被删除文本的地方。在缺省的情况下，删除环可以容纳最近 30 次删除或剪切下来的东西。但有些删除操作（比如字符删除命令“**C-d**”）不会把被删除的文本保存到删除环里。关于删除环的详细讨论请参考第二章中的讨论。

local abbreviations（**局部简写词汇**）

用户定义的仅能用在某个指定编辑模式里（即仅供编辑模式局部使用）的简写词汇。比如说，用户可能想让文本模式和 C 模式分别使用不同的简写词汇。关于局部简写词汇的详细讨论请参考第三章。

macros（**宏**）

一组被录制下来的按键序列或鼠标动作，目的是减少重复性的编辑工作。关于宏的详细讨论请参考第十章。

major mode（**主编辑模式**）

Emacs 用来使其行为适应手中主要任务的方式。文本模式是主要用于书写文字材料的主编辑模式；C 模式是主要用于书写 C 语言代码的主编辑模式。不同的主编辑模式有不同的编辑命令，这些命令只对该主编辑模式有意义；那些能用在一切编辑模式里的命令叫做全局编辑命令。

mark（文本块标记）

一个辅助性的指针，它与光标一起定义文本块的边界。文本块可以被删除、移动或者复制。GNU Emacs 里的文本块标记是不可见的。

markup（排版标记）

告诉文本排版器程序按某种特定格式打印或显示文本的排版指令代码。TEX和**troff**就是两个能够把内含排版标记的ASCII文件输出为类似于印刷排版效果的文本排版程序的例子。HTML 则通过一个能够在屏幕上对它们进行排版的浏览器显示出来。

minibuffer（辅助输入区）

屏幕底部一个供用户输入特定信息的区域。比如说，当你按下“**C-x C-w**”组合键准备保存一个文件时，Emacs将要求在辅助输入区里输入一个文件名。Emacs还会利用这个区域来显示各种提示性的消息。

minor mode（副编辑模式）

独立于主编辑模式而能够被单独启用或禁用的功能。启用自动换行功能的自动换行模式就是一个副编辑模式。

mode（编辑模式）

Emacs用来使其行为适应用户手中工作的定制方式，又分为主编辑模式和副编辑模式。主编辑模式负责定义手里的主要工作，而副编辑模式则是一些可以在主编辑模式里单独启用或禁用的功能。比如说，文本模式就是一种主编辑模式，而启用自动换行功能的自动换行模式就是一个副编辑模式。

mode line（状态行）

Emacs窗口的最后一行，通常呈反显状态或者显示为另外一种颜色。状态行的作用是告诉用户正在编辑的编辑缓冲区、启用的主编辑模式和副编辑模式、在编辑缓冲区里的当前位置等信息。根据具体情况，状态行也可以提供其他信息，比如时间和日期等。

Mosaic

第一个图像界面的Web浏览器，由伊利诺斯大学Urbana-Champaign分校的国家超级计算应用中心开发。很多商业化的浏览器都是以Mosaic为基础的。Netscape就是由许多原来的Mosaic开发者们创建的。

output groups（输出组）

在shell模式下，一个“输出组”是由一条shell命令和该命令的执行输出组成的。输出组使我们能够更方便地在shell模式里的命令之间移动，因为某个给定的命令说不定就有好几页长。

overwrite mode（改写模式）

一个副编辑模式，它允许新敲入的字符覆盖（擦除）现有的文本。启用改写模式的命令是“**ESC x overwrite- mode RETURN**”。

paragraph formatting（段落重排）

参见 auto-fill mode（自动换行模式）。

paste（粘贴）

把此前删除或者剪切下来的文本插入到光标当前位置。Emacs在提到粘贴操作的时候使用的是术语“yank”。

pause（等待输入）

一项与宏有关的功能，即临时停下宏的执行等待用户的输入。用户完成输入后再用“**ESC C-c**”组合键继续执行。有关详细讨论请参考第十章。

picture mode（图形模式）

设计目的是允许用户使用键盘字符绘制简单图形的主编辑模式。关于图形模式的详细讨论请参考第八章。

point（插入点）

即光标位置。精确地讲，插入点是位于光标所在字符和前一个字符的中间。在实际工作中，通常可以把光标和插入点看做是同义词。但知道插入点的准确位置能够帮助用户更好地掌握某些命令（比如粘贴或位置交换命令等）。此外，每一个编辑缓冲区都有一个插入点（一个当前位置），而光标则每次只能出现在一个编辑缓冲区里。

query（查询）

一项与宏有关的功能，即由用户决定是否需要继续执行某个宏

query-replace（查询－替换）

一项与查找与替换操作有关的功能，即由用户逐个地决定是否要进行给定的替换。

read-only（只读）

一个只可以查看不可以修改的文件或编辑缓冲区。但可以把一个只读编辑缓冲区里的文本复制并粘贴到另外一个编辑缓冲区里。

read-write（读写）

一个既可以查看（读）也可以修改（写）的编辑缓冲区。

rectangle editing（矩形编辑）

利用Emacs中的矩形概念，能够重新安排和编辑信息的列。关于矩形寄存区的详细讨论请参考第八章。

region（文本块）

光标（或者插入点）与文本块标记之间的区域。可以对文本块进行删除、移动或者复制操作。当前定义的文本块在某些显示器上呈反显状态。如果不是这样，可以按下“**C-x C-x**”组合键（命令名是**exchange-point-and-mark**）查看文本块的边界。

registers（**矩形寄存区**）

用来存放矩形或文本的区域。类似于删除环，但为了便于对信息进行（经常会重复进行的）检索，矩形寄存区的名字只能是单个字符。

regular expression（**正则表达式**）

完成复杂和变化的字符串匹配工作的一项UNIX功能。Emacs支持正则表达式查找操作和正则表达式替换操作。能够用来构造正则表达式的字符清单见第三章；对正则表达式更深入的讨论请参考第十三章。

RMAIL

Emacs的邮件阅读功能组件。

rot13

一个简单的文本加密程序，用来对可能引起别人反感的消息进行加密。它可以被任何也拥有rot13程序的人解密。Gnus能够对rot13文件进行加密和解密操作。

search and replace（**查找与替换操作**）

参见query-replace（查询－替换）。

search string（**查找字符串**）

在查找操作中，正在被寻找的文本。

setup file（**设置文件**）

参见“*.emacs*”文件。

shell buffer（**shell编辑缓冲区**）

Emacs创建的一个编辑缓冲区，它里面运行一个UNIX的shell；这样，当与UNIX操作系统互动的时候，就可以利用Emacs的编辑命令和功能，比如它的自动补足功能等。打开一个shell编辑缓冲区的命令是“**ESC x shell**”。

Telnet mode（**Telnet模式**）

一个面向Telnet的Emacs接口。Telnet是一个网络协议，用户可以通过它登录和使用本地网或因特网上的其他计算机。

uniform resource locator（URL，**统一资源定位器**）

通常简称为URL地址，它是某个因特网资源的地址。URL的格式一般为“*protocol://resourcename*”。比如说，WWW资源的URL是“*http://www.ora.com*”形式；FTP档案的URL则是“*ftp://ftp.ora.com*”形式。出现在斜线字符前面的是Web访问指定资源的网络协议（比如FTP）。跟在斜线后面的是被访问的资源的名字，它可以很宽泛（比如O'Reilly & Associates出版公司Web站点的总称*www.ora.com*），也可以很具体，比如像《ora.com杂志》主页的URL：“*http://www.ora.com/www/oracom*”。

variable（变量）

Emacs提供的上百种功能都可以由用户通过设置相应的变量进行定制。如果只是临时性地需要在某次会话对变量进行设置，可以输入“**ESC x set-variable**”命令；如果想做永久性的设置，就需要使用“*.emacs*”文件。关于变量的详细讨论请参考第十一章。

W3

William Perry 为 Emacs 开发的一个 WWW 浏览器。W3 并不是 Emacs 标准发行版本的组件，但它提供了一种把 WWW 访问功能与 Emacs 操作集成为一个整体的绝佳方法。W3 能够与 Mosaic 共享收藏夹和历史记录文件。

wastebasket（回收站）

参见 kill ring（删除环）。

window（窗口）

屏幕上显示一个编辑缓冲区的区域。在缺省的情况下，Emacs 只有一个窗口。但很多命令，包括帮助命令，都会自动把屏幕拆分为两个窗口。可以拆分出水平窗口（这类窗口最常用）和垂直（呈左右并排）窗口。可以在多个窗口里同时打开同一个编辑缓冲区。X 用户还可以创建出多个 X 窗口。为了避免概念混乱，多个 X 窗口在 Emacs 里称为窗格（frame）。

word search（单词查找）

一种忽略换行符和标点符号的查找操作。如果你知道自己的编辑缓冲区里确实存在着某些文本，但用其他查找操作（比如递增查找）却找不到，就可能是其中有一个换行符——递增查找会把换行符解释为一个字符。再用单词查找操作试试。

word wrap（字换行）

参见 auto-fill mode（自动换行模式）。

World Wide Web（万维网）

通常简称为 WWW，这是一种基于超文本的因特网信息服务；通过它，即使用户不了解某些特定因特网应用软件（比如 FTP 软件）的使用方法，也可以毫无困难地访问很多因特网资源。用户通过选择超链接（hyperlink）漫游 WWW。WWW 是由 Tim Berners-Lee 在瑞士日内瓦的 European Particle Physics laboratory（CERN，欧洲粒子物理实验室）创立的。

yank（粘贴）

把剪切下来的文本复制到当前编辑缓冲区里。