LaTeX 2ε用户手册

TeXGuru
texguru@263.net
©TeXGuru, August 15, 1999

©TEXGuru, August 15, 1999

目 录

第一章 简介 **1**
§1.1 文本格式 1
§1.2 TEX与其演化史 2
§1.2.1 TEX程序 2
§1.2.2 Plain TEX 2
§1.2.3 LATEX 3
§1.2.4 LATEX 2ε 3
§1.2.5 在不同类型计算机上的 TEX 4
§1.3 LATEX 2ε的新内容 4
§1.3.1 类与宏包 4
§1.3.2 字体管理 5
§1.3.3 浮动对象的安排 5
§1.3.4 扩充的语法 6
§1.3.5 与 LATEX2.09 的兼容性 6
§1.3.6 “我应该用哪一个版本呢？” 7
§1.3.7 更新 LATEX 2ε 7
§1.4 如何使用本书 7
§1.4.1 从 2.09 中区分开 LATEX 2ε 8
§1.5 LATEX文件的基础知识 9
§1.5.1 文本与命令 9
§1.5.2 LATEX文档的结构 9
§1.5.3 LATEX的处理模式 10
§1.5.4 生成 LATEX文档 11

第二章 命令与环境 **13**
§2.1 命令的名称与参数 13
§2.2 环境 14
§2.3 声明 15
§2.4 长度 16
§2.4.1 固定长度 16
§2.4.2 橡皮长度 16
§2.5 特殊字符 17
§2.5.1 空格与回车 17
§2.5.2 引号 17

§2.5.3 连字符与破折号 17
§2.5.4 得到命令字符 17
§2.5.5 特殊字符 §, †, ‡, ¶, ©, £ 18
§2.5.6 外文字母 18
§2.5.7 重音 18
§2.5.8 连写 18
§2.5.9 日期 19
§2.6 脆弱的命令 19
§2.7 练习 20

第三章 文档的布局与组织 23
§3.1 文档类 23
§3.1.1 标准的类选项 23
§3.1.2 利用宏包增加功能 26
§3.1.3 样式选项 27
§3.1.4 全局和局部选项 27
§3.2 页面样式 28
§3.2.1 生成页眉的声明 28
§3.2.2 页的编号 29
§3.2.3 与段落有关的距离 29
§3.2.4 页面格式 30
§3.2.5 单双列页面 32
§3.3 文档中的部分 32
§3.3.1 标题页 33
§3.3.2 摘要 35
§3.3.3 章节 35
§3.3.4 附录 37
§3.3.5 书的结构 37
§3.4 目录表 38
§3.4.1 自动条目 38
§3.4.2 显示目录表 38
§3.4.3 其它条目 39
§3.4.4 其它列表 39
§3.5 精调正文 40
§3.5.1 单词与字符的距离安排 40
§3.5.2 断行 43
§3.5.3 段落间距 44

§3.5.4 段落的缩进 44
§3.5.5 分页 45
§3.6 断词 47
§3.6.1 人工断词 47
§3.6.2 断法列表 48
§3.6.3 禁止断词 48
§3.6.4 行宽与断词 48
§3.6.5 其它与断词有关的内容 49
§3.6.6 多语言文本中的断词 49

第四章 显示文本 **51**
§4.1 改变字体 51
§4.1.1 强调 51
§4.1.2 字体尺寸的选择 52
§4.1.3 字体属性 53
§4.1.4 字体命令 54
§4.1.5 旧字体声体 55
§4.1.6 其它字体 55
§4.1.7 字符集与符号 56
§4.2 居中与缩进 56
§4.2.1 居中文本 56
§4.2.2 单边调整 56
§4.2.3 两边缩进 57
§4.2.4 诗歌缩进 57
§4.3 列表 58
§4.3.1 `itemize` 样例 58
§4.3.2 `enumerate` 样例 59
§4.3.3 `description` 样例 59
§4.3.4 嵌套列表 59
§4.3.5 改变标签样式 61
§4.3.6 参考文献 62
§4.4 广义列表 64
§4.4.1 标准标签 65
§4.4.2 列表样式参数 65
§4.4.3 用户制作列表的示例 67
§4.4.4 做为新环境的列表 68
§4.4.5 平凡列表 69

§4.4.6 嵌套列表 69
§4.5 定理型的声明 70
§4.6 制表位 71
§4.6.1 基础知识 71
§4.6.2 样本行 71
§4.6.3 制表位与左边界 72
§4.6.4 其它的制表命令 72
§4.6.5 关于制表的注解 73
§4.7 盒子 74
§4.7.1 LR 盒子 75
§4.7.2 LR 盒子的竖直移位 76
§4.7.3 子段盒子和小页 77
§4.7.4 竖直摆放的问题 78
§4.7.5 具有指定高度的段落盒子 79
§4.7.6 标尺盒子 80
§4.7.7 嵌套盒子 80
§4.7.8 盒子样式参数 81
§4.8 表格 82
§4.8.1 构造表格 82
§4.8.2 表格样式参数 85
§4.8.3 表格样例 85
§4.8.4 浮动表格 93
§4.9 显示源文本 94
§4.10 脚注和边注 95
§4.10.1 标准脚注 95
§4.10.2 非标准脚注 96
§4.10.3 禁止模式中的脚注 97
§4.10.4 小页环境中的脚注 98
§4.10.5 脚注样式参数 98
§4.10.6 边注 98
§4.10.7 边注的样式参数 100
§4.11 正文中的注释 100

第五章 数学公式 **103**
§5.1 数学环境 103
§5.2 数学公式的主要组成 104
§5.2.1 常量与变量 104

©TEXGuru, August 15, 1999

§5.2.2 指数和指标 104
§5.2.3 分数 105
§5.2.4 方根 105
§5.2.5 求和与积分 106
§5.2.6 连续点 — 省略号 106
§5.3 数学符号 107
§5.3.1 希腊字母 107
§5.3.2 花体字母 108
§5.3.3 二元运算符 108
§5.3.4 关系运算符及其否定 109
§5.3.5 箭头与指针 110
§5.3.6 其它各类符号 110
§5.3.7 具有两种尺寸的符号 111
§5.3.8 函数名 111
§5.3.9 数学重音 112
§5.4 其它要素 113
§5.4.1 括号符号的尺寸自动调整 114
§5.4.2 公式中的普通文本 115
§5.4.3 矩阵和域 116
§5.4.4 公式上下的直线 118
§5.4.5 堆积符号 119
§5.4.6 其它的 TEX数学命令 119
§5.4.7 多行公式 120
§5.4.8 有框公式和并列公式 123
§5.4.9 化学方程式和数学公式中的黑体 124
§5.5 数学公式的精调 125
§5.5.1 水平间距 126
§5.5.2 在公式中选择字体尺寸 126
§5.5.3 括号符号的手工尺寸调整 128
§5.5.4 数学样式参数 129
§5.5.5 更进一步的建议 130
§5.5.6 有框的显示公式 131
§5.5.7 补遗 132

第六章 图形 **135**
§6.1 图形的尺寸和位置 135
§6.2 图形环境 136

§6.3 定位命令 136
§6.4 基本画图命令 137
§6.4.1 图形中的文本 137
§6.4.2 图形中的盒子 — 矩形 138
§6.4.3 直线 140
§6.4.4 箭头 141
§6.4.5 圆 142
§6.4.6 卵形线与圆角 142
§6.4.7 竖直堆积文本 144
§6.4.8 有框文本 145
§6.4.9 曲线 146
§6.5 其它图形命令与示例 146
§6.5.1 直线粗细 146
§6.5.2 嵌套图形 147
§6.5.3 存贮部分图形 148
§6.5.4 图形环境的广义语法 151
§6.5.5 更多的例子 152
§6.5.6 扩展软件包 153
§6.5.7 一般性建议 154
§6.6 浮动表格和插图 155
§6.6.1 浮动安置 155
§6.6.2 延迟浮动 156
§6.6.3 浮动中的样式参数 157
§6.6.4 浮动对象中的说明 158
§6.6.5 浮动示例 159
§6.6.6 在正文中对插图和表格的引用 162

第七章 用户定制 LaTeX 163
§7.1 记数器 163
§7.1.1 LaTeX记数器 163
§7.1.2 用户自定义的记数器 163
§7.1.3 改变记数器的值 164
§7.1.4 显示记数器的值 164
§7.2 长度 165
§7.3 用户定义命令 166
§7.3.1 没有参数值的命令 167
§7.3.2 有参数值的命令 168

©TEXGuru, August 15, 1999

§7.3.3 有一个可省参数值的命令 170
§7.3.4 用户定义命令的其它样例 170
§7.3.5 条件文本 173
§7.4 用户定义的环境 176
§7.4.1 没有参数的环境 177
§7.4.2 有参数的环境 178
§7.4.3 有一个可省参数值的环境 180
§7.5 对用户定义结构的解释 181
§7.5.1 保存用户定义的结构 181
§7.5.2 缩写结构 181
§7.5.3 命令和记数器的名称相同 181
§7.5.4 用户定义的作用范围 182
§7.5.5 定义的顺序 182
§7.5.6 传递参数值 183
§7.5.7 嵌套定义 183
§7.5.8 不期望的空格 184
§7.5.9 最后两个例子 184

第八章 高级功能 189

§8.1 处理文档的各个部分 189
§8.1.1 \input 命令 189
§8.1.2 \include 命令 190
§8.1.3 终端输入和输出 192
§8.2 在 LaTeX中包含 TeX命令 193
§8.3 正文内的引用 194
§8.3.1 交叉引用 194
§8.3.2 参考文献的引用 196
§8.3.3 索引记录 197
§8.3.4 汇总 199
§8.4 MakeIndex— 关键词处理器 199
§8.5 新字体选择框架 (NFSS) 202
§8.5.1 在 NFSS中的字体属性 202
§8.5.2 简化的字体选择 205
§8.5.3 默认属性值 206
§8.5.4 定义字体命令 207
§8.5.5 数学字母表 207
§8.5.6 数学符号字体 208

§8.5.7 处理属性值 209
§8.5.8 定义 NFSS中的字体 210
§8.5.9 编码命令 212
§8.6 输入代码 213
§8.7 特殊符号的替代 214
§8.8 其它标准文件 215
§8.8.1 特殊文档 215
§8.8.2 其它类 216
§8.8.3 标准宏包 216
§8.8.4 附属软件 218
§8.8.5 捐献的宏包 218
§8.9 各种 LaTeX文件 219
§8.10 报告材料的准备 (SliTeX) 223
§8.10.1 `slides` 类 223
§8.10.2 制作幻灯片的环境 224
§8.10.3 其它功能 225

第九章 错误消息 231
§9.1 错误消息的基本结构 231
§9.1.1 TeX错误消息 231
§9.1.2 LaTeX的错误消息 233
§9.1.3 来自于 LaTeX2.09 中的错误消息 237
§9.1.4 来自于 TeX宏的错误消息 238
§9.2 一些错误样例 239
§9.2.1 错误的传播 239
§9.2.2 典型的严重错误 241
§9.2.3 数学错误 243
§9.2.4 来自于多文件的错误 244
§9.3 LaTeX错误消息清单 245
§9.3.1 一般 LaTeX错误消息 245
§9.3.2 LaTeX宏包错误 250
§9.3.3 LaTeX字体错误 251
§9.4 TeX错误消息 253
§9.5 警告 258
§9.5.1 普通 LaTeX警告 258
§9.5.2 LaTeX宏包警告 260
§9.5.3 LaTeX字体警告 261

§9.5.4 TEX警告消息 262
§9.6 搜索顽固错误 264

附录A 书信的编辑 265
§A.1 LATEX 的 letter 类 265
§A.2 局部范围的信件样式 269
§A.3 定制信件样式 272

附录B 参数文献数据库的处理 279
§B.1 BIBTEX程序 279
§B.2 创建参考文献数据库 280
§B.2.1 条目类型 282
§B.2.2 域 283
§B.2.3 域的交叉引用 285
§B.2.4 特殊的域格式 286
§B.2.5 缩写 287
§B.2.6 使用模板 288
§B.3 扩展 BIBTEX 289
§B.3.1 作者–年代 参考文献样式 289
§B.3.2 定制参考文献样式 291

附录C LATEX程序设计 293
§C.1 类与宏包文件 293
§C.1.1 LATEX2.09 中的样式文件 293
§C.1.2 LATEX 2ε的新概念 294
§C.1.3 命令的层次 294
§C.1.4 TEX命令 296
§C.2 LATEX 2ε程序设计语言 297
§C.2.1 文件识别 297
§C.2.2 上载其它类和宏包 298
§C.2.3 选项的处理 298
§C.2.4 延期处理 300
§C.2.5 牢固的命令 300
§C.2.6 有短参数值的命令 301
§C.2.7 给出错误和警告消息 301
§C.2.8 输入文件 302
§C.2.9 检测文件 303
§C.2.10 兼容模式 304

§C.2.11 有用的内部命令 305
§C.2.12 有用的 TEX命令 306
§C.3 宏包示例 307
§C.3.1 修改文本尺寸 308
§C.3.2 重新设计页眉和页脚 310
§C.3.3 为其它语言改编 LATEX 312
§C.3.4 作者–年代 引用 318

附录D 扩展 LATEX 325
§D.1 国际化的 LATEX 325
§D.1.1 german.sty 文件 326
§D.1.2 french.sty 文件 327
§D.1.3 多语言 LATEX—babel 系统 330
§D.1.4 使用 \language 命令 333
§D.2 LATEX与 PostScript 334
§D.2.1 调用 PostScript 字体 334
§D.3 其它的 LATEX附件 335
§D.3.1 DocStrip工具 335
§D.3.2 建立 LATEX代码的档案 338
§D.3.3 图形与彩色 345
§D.3.4 其它有用的宏包 349
§D.4 LATEX 2ε的安装 350
§D.4.1 更新 LATEX 351
§D.5 LATEX文件的来源 352
§D.5.1 TEX组织 353

附录E 计算机现代字体 355
§E.1 简介 355
§E.2 TEX基本字体的分类 356
§E.3 文本字体 357
§E.3.1 罗马字体族 357
§E.3.2 Sans serif 族 360
§E.3.3 打字机字体 362
§E.4 装饰性字体族 364
§E.5 数学与符号字体 365
§E.5.1 数学字体族 365
§E.5.2 其它字符字体 367
§E.6 字体中的字符对应 368

©TEXGuru, August 15, 1999

§E.7 Cyrillic 字体 373
§E.8 字体文件 374
§E.8.1 基本名 374
§E.8.2 字体放大 375
§E.8.3 像素代码 377
§E.8.4 压缩代码 378
§E.9 对 METAFONT的注释 379

©TEXGuru, August 15, 1999

表格目录

6.1 表格浮动示例之一 159
6.2 表格浮动示例之二 160

8.1 NFSS编码 框架 203
8.2 NFSS序列 属性 204
8.3 计算机现代字体的属性 205

D.1 psnfss 宏包及相应字体 335

©TEXGuru, August 15, 1999

插图目录

3.1 页面布局参数 31
3.2 标题页示例及其结果 33

4.1 `list` 参数 66

6.1 Left 160
6.2 Right 160

8.1 幻灯片文件样例 227

A.1 利用标准 `letter` 类生成的信件示例 268

D.1 CTAN 服务器上的部分目录树结构 353

第一章　简介

§1.1　文本格式

在当今时代，计算机的一个最通用的功能就是对文本的电子化处理，它主要由如下四步组成：

1. 文本输入到计算机里，存贮起来供以后修改、扩充和删减；
2. 格式化输入文本，使其以相同长度的行和特定尺寸的页显示出来；
3. 在计算机的监视器上显示格式化后的结果；
4. 把最终的输出送到打印机上打印出来。

有很多字处理系统可以在一个软件包中同时实现这四方面的功能，因此用户也就意识不到上面这几种划分。而且，第 3 ， 4 步实际上是一样的：都是把格式化后的结果送到一个输出设备上，只不过一种是监视器，另一种是打印机罢了。

而类似于 TEX这样的文本格式化程序，就只进行第 2 步的处理。任何一种文本编辑器都可以用来输入和修改源文本。如果你已经熟悉而且习惯于使用某一编辑器，那就不妨继续用它。而字处理程序就不一定满足这里的要求了，因为这种程序通常要加入很多不可见的控制字符。对字处理程序而言，所见即所得是一个非常好的功能。但是另一方面，你所得到的，也不必就是你所见到的。

用来作为格式化程序输入的在编辑器生成的文本中，应该包含一些特殊命令或指令，但这些指令是用可以看到的普通文本表示的。从某种意义上来，这种供格式化用的指令集很像一种装饰语言，它只是用来表示段落、章节等等从哪儿开始，而不是直接对文本进行格式化处理。而在格式化的过程中，这些指令是如何被解释的，则要看所选用的版面设计格式了。同样的文本，在不同的版面格式下可以形成完全不同的样式。

而格式化程序的功能远不至这些。实际上 TEX也是一种有丰富功能的编程语言，知识丰富的用户可以用它编写代码，来增加某一功能。 LATEX自身也就是一组复杂的宏的集合。而且任何用户都可以通过编程对 LATEX进行扩展，或者直接利用其它程序员已设计好的宏。 TEX和 LATEX的功能并不只是那些包含在基本软件包中的内容。

对于格式化软件而言，文本处理的最后一步是把结果送到输出设备上，即打印机或者计算机监视器上，甚至可以是一个文件。实现这种功能的程序称为驱动程序；它把格式化程序已编码好的输出翻译成用户可以使用的某一设备上的特定指令。这也就是说，对每种类型的打印机，也就必须有相应的驱动程序。

§1.2 TEX与其演化史

对于科技著作而言，在能排版出如同书籍一样漂亮的格式化程序中，功能最强的就是 Donald E. Knuth 所设计的 TEX程序了，其名字是由希腊字母 $\tau\epsilon\chi$ 的大写形式组成的。正是由于这个原因，其最后一个字母的发音并不是 x，而类似于苏格兰语单词 loch 或者德语单词 ach 中的 ch，也类似于西班牙语中的 j 或俄语中的 kh。这个名字强调指出了数学公式的印刷是该程序的不可分割的一部分，而不是额外附加上去的。除了 TEX外，Knuth 还设计了另一个软件 METAFONT，用来生成各种字符字体。在标准的 TEX软件包中有 75 种不同设计尺寸的字体，而且每种字体有八种不同的放缩比例。所有这些字体都是用 METAFONT程序生成的。为了满足其它应用的需要，还设计了其它字符字体，如古斯拉夫语或日语字母的字体，有这些字母的文本也可以用书籍质量排版出来。

§1.2.1 TEX程序

最基本的 TEX程序是由一些很基本的命令组成，它们可以完成简单的排版操作和程序设计功能。然而，TEX也允许用这些基本命令定义一些更复杂的高级命令。这样就可以利用低级的结构块，形成一个用户界面相当友好的环境。

当处理器运行 TEX时，该程序首先读取所谓的格式文件，格式文件中包含各种以基本语言写成的高级命令，也包含分割单词的连字号安排式样。接着处理程序就处理源文件，源文件由要处理的真正文本，以及在格式文件中已定义了的各种命令组成。

创建新格式也是一件需要由知识丰富的程序员来做的事情。把定义写到一个源文件中，这个文件接着被一个名叫 initex的特殊版本 TEX程序处理。它采用一种紧凑的方式存贮这些新格式，这样就可以被通常 TEX程序很快地读取。

虽然一般用户可能从来用不着编写这格的格式文件，但是提供给用户的有可能是需要用 initex来安装的格式源文件。例如，当我们要更新 LATEX 2_ε格式时，我们就要按照第 D.4 节所描述的方法进行操作。

§1.2.2 Plain TEX

Knuth 设计了一个名叫 Plain TEX的基本格式，以便与低层次的 TEX互应。这种格式是 TEX字处理的相当基本的部分，以致于我们有时候根本分不清到底哪是真正的 TEX处理程序，哪是这个特殊的格式。大多数声称只使用 TEX的人，实际上指的是只用 Plain TEX。

Plain TEX也是其它高极格式的基础，这些格式进一步加强了把 TEX和 Plain TEX认为是同一事物的印象。

©TEXGuru, August 15, 1999

§1.2.3 LATEX

Plain TEX的重点还只是停留在如何排版的层次上，而不是从一位作者的观点来看问题。当然对 TEX深层功能的进一步发掘，需要相当高超的编程技巧。因此它的应用就需要高级排版和程序设计人员。

正是由于这种原因，美国计算机学家 Leslie Lamport 开发了 LATEX格式，这种格式提供了一组生成复杂文档所需要的更高级命令。利用这种格式，即使使用者没有排版和程序设计的知识也可以充分发挥由 TEX所提供的强大功能，能在几天，甚至几小时内生成大量具有书籍印刷质量的结果。在生成复杂表格和数学公式方面，这一点表现得尤为突出。

LATEX相对于其基础 Plain TEX而言，更像一个包装语言。它可以在作者根本不知道所以然的情况下，自动给出标题，章节，表格目录，交叉索引，公式编号，文献引用，浮动图表。版面布局信息包含在类文件中，这些类文件并不是位于源文件中的。这些布局可以改动，也可以直接套用。

由于 LATEX是在二十世纪八十年代出现的，同其它软件一样，它也周期性地进行更新和修订。经过了很多年，版本号固定为 2.09 ，而修订只是用是日期来区分。较近的一次大修订发生在 1991 年 12 月 1 日，其后直到 1992 年 3 月 25 日为止，还有几次小的修订。

§1.2.4 LATEX 2ε

由于 LATEX相当普及，以及它在许多原本没有想像到的领域中的扩展，再加上计算机技术的日新月异，特别是价格低廉，但功能强大的激光打印机的出现，使得相当广泛的一类格式都冠以 LATEX的标签。为了尝试建立一个真正的改进标准，在 1989 年 Leslie Lamport, Frank Mittelbach, Chris Rowley 和 Rainer Schöpf 创立了 LATEX3 项目。他们的目标是建立一个最优的，有效的命令集合，这些命令是来自于各种软件包为了实现某一目的而设计的。

正如项目名称所表明的，它的目标就是得到 LATEX的一个新版本 3 。然而，由于这是一个长期目标，朝向这个目标迈进的第一步就是在 1994 年中发行了 LATEX 2ε并出版了 Lamport 基本手册第二版，同时还有一本新书，专门描述在新系统中许多可用的扩展软件包和 LATEX程序设计。 LATEX 2ε是在本世纪末左右出现的令人著目的 LATEX3 之前的现在标准版本。

实际上，在 LATEX 2ε出现之前，其处理字体安装和选择的一些部分已经以新字体选择框架 (或 NFSS) 的形式公开了，而且被许多组织或个人集成到其软件中。这种框架有两个版本，但不幸的是它们并不兼容，两个版本分别相应于 LATEX2.09 和 LATEX 2ε 。后来以一种完全与 2.09 版本兼容的方式对 NFSS进行了重新实现。

§1.2.5 在不同类型计算机上的 TEX

TEX和 LATEX原本是设计运行在中央大型机上的，但是随着工作站，甚至更强大的个人计算机 (即 PC 机) 的发展，包括 LATEX在内的 TEX软件包已经可以运行在强大的小型机上了。在本书中有关编写 LATEX文本文件的所有内容都同样地适用于大型机和个人计算机。

这样就对使用者提出了一个问题。在中央大型机上有操作员负责管理 TEX软件及附件的安装与维护。然而，工作站或 PC 机的用户也就需要同时客串管理员的角色，即他 (她) 必须比以前的 TEX用户要多知道一些关于如何设置 TEX和 LATEX的东西。或者更经常发生的情形是，他们必须知道是谁帮他们做这些事。这个人通常也就称为小团体里的 TEX专家 (TeXGuru) 。

对于工作站和 PC 机而言，相当实用的输出方式是屏幕预览，即把输出显示在监视器上。在进行实际打印之前，尤其是输出中有复杂的表格或数学公式时，这是一种相当经济的检查页面排版如何的方法。本书中练习的结果最好就只在监视器上查看一下就可以了。

在 PC 机或工作站上也有 Shell，它使用户可以只需按一下按钮或点击一下鼠标，就可以在编辑器和处理程序之间来回切换。对于这样的系统，TEX实际上也是所见即所得了。

§1.3 LATEX 2ε的新内容

本节内容主要针对于那些已相当熟悉 LATEX2.09 的读者。下面简要列出了 LATEX 2ε的新功能。

§1.3.1 类与宏包

在 LATEX2.09 与 LATEX 2ε之间的一个最本质的差别就是声明所有布局的第一条命令。这个差别实际上使得这两个版本可以兼容。

在 LATEX2.09 中，必须像下面这样来声明所需要的主样式，这个样式同时带有一些选项：

```
\documentstyle[ifthen,12pt,titlepage]{article}
```

这里的主样式是 article，它保存在一个叫 article.sty 的文件中，而同时用 12pt 作为基本字体大小，标题放在单独一页上。有上百个这样的附加文件可以用来做为样式选项 (也称为子样式)。

然而， LATEX 2ε对主样式的补充选项与真正的内部选项之间有一个明确的区分。现在主样式更名为类，而扩展文件称为宏包。上面那个初始化声明现在变为

```
\documentclass[12pt,titlepage]{article}
\usepackage{ifthen}
```

©TEXGuru, August 15, 1999

布局信息包含在文件 article.cls 中，它可以处理选项 12pt 和 titlepage 。而文件 ifthen.sty 还像以前那样读取；然而，这里的新功能是它也可以有自己的内部选项。不仅如此，那些列在 \documentclass 命令中的选项，由于被看作是全局选项，因此对所有宏包都有作用（见 3.1.2 节）。

初始化声明 \documentstyle 在 LaTeX 2ε中还可以使用，这时它切换到一个兼容模式，来模拟 LaTeX2.09 中的行为。

为了帮助那些坚韧不拔的 LaTeX程序员更好地进行程序设计，现在增加了许多新功能。对选项的处理做了改进，而且如上所述，也可以在宏包中加进选项。增加了一些安全机制，以保证版本号的匹配。在读其它文件时，现在有了更好的测试方法，以保证该文件不存在时，采取其它的方法。在附录 C 中对这些程序设计要素进行了描述。

§1.3.2 字体管理

在 LaTeX2.09 中， TeX的计算机现代字体 (Computer Modern Fonts) 被牢靠地固化在格式中。在人们喜欢用的字体也就那么几种的年代里，这不失为一种可行的方法。但到了今天，可用的字体数目繁多，特别是 PostScript 打印机的出现，更加要求一个有弹性的系统。新字体选择框架 (NFSS) 应运而生，并与 LaTeX 2ε完全结合为一体。要选择非计算机现代字体作为基本字体只是一些很简单的重定义（见 8.5 节）。

NFSS也改变了在文档内部引进字体的方法。 LaTeX2.09 继承了 TeX的字体命令，如 \bf（黑体）， \it（斜体）都是严格地选择一特定的字体。这些命令中只有字体大小维持不变。而在 NFSS中，字体是用某种属性来描述的，可以分别彼此独立地进行选择。因此就有可能先选择黑体，然后选斜体，从而得到黑斜体，而这在 LaTeX2.09 中是行不通的。

LaTeX 2ε鼓励使用字体选择命令，而不是用字体声明。例如，为了强调某一文字，命令 \emph{word} 就比声明 {\em word} 要好。这样的命令对初学者来说，更符合逻辑，虽然习惯于用后者的经验丰富的 LaTeX2.09 用户可能持相左的看法。

在数学模式中，文本的字体是通过用特殊的数学字母命令来选择的，而不是原来的字体声明。也就是说，在数学模式中不允许原先的 \rm, \bf 和 \cal 等声明，而代之以 \mathrm, \mathbf 和 \mathcal 这些有参数的命令。

§1.3.3 浮动对象的安排

在 LaTeX2.09 中一个令人头痛的问题就是如何安排浮动对象（图与表），使它出现在人们最希望看到的地方。而浮动对象的安排有一套相当复杂的规则，并不是所有的人都能很好掌握。 LaTeX 2ε提供了两种新的机制，以控制这一过程，其中一种是不鼓励对浮动对象的安排，另一种则鼓励这种安排。

通过使用命令 \suppressfloats 可以使当前页中没有浮动对象。作为选项，可以使用参数 t 或 b 来只禁止浮动对象不出现在当前页面的顶部或底部。

另一方面，有一个新的浮动位置指定符 !，它可以取消这个浮动对象所在页的关于可以出现的浮动对象数与文本总量的限制。这也克服了通常会出现的一种不满，而原来必须重定义 \texttraction 或其它浮动安排参数，并进行无数次的调试才能缓解。浮动位置指定符 ! 同其它参数一样使用：如，

```
\begin{figure}[!]
```

现在为了使浮动对象与文本更好的分离开，也可以在其顶部或底部画上定义好的标尺。在 6.6 节中对这些功能进行了介绍。

§1.3.4 扩充的语法

与 2.09 版本相比，在 LaTeX 2ε中对几条命令的语法进行了扩充。当然原来的语法仍然有效。

- \newcommand, \renewcommand, \newenvironment 和 \renewvironment 命令用来定义新的命令和环境，其除了几个必须的参数外，还可以包含一个可省略的参数（ 7.3 和 7.4 节）。
- 盒子命令 \makebox, \framebox 和 \savebox 在定义其实际尺寸时，可以引用其自然的尺度。也就是说，你可以用定义其宽度为自然宽度的两倍。见 4.7 节。
- 现在 \parbox 和 minipage 环境除了定义水平宽度外，还可以指定一个竖直高度。有一个新的内部安排参数来决定盒子中的文本是否需要被推向顶部，居中，或者放在底部，甚至伸展文本以填满整个盒子（ 4.7.5 节）。
- 以前命令 \settowidth 可以用来测量某些文本的宽度；现在又补充了 \settoheight 和 \settodepth（ 7.2 节）。

§1.3.5 与 LaTeX2.09 的兼容性

为了使 LaTeX 2ε与 LaTeX2.09 尽可能地保持兼容性，已做了各种各样的努力。这就是指根据原来版本标准写的文档，在新的版本下应得到相同的结果。同时也表明软件包中绝大多数的样式（或子样式）选项应具有与以前一样的功能，而不需任何修正。

这只是理论上应该如此。实际上，一定存在着不兼容之处。那么用户遇到这种情形的状况是怎样的呢？

- 如果只使用高级命令，在命令中不包含 @ 字符，那么兼容性是 100% 。
- 如果运用了内部命令，这就可能会导致问题，特别是如果其中包含盒子或者输出程序。
- 如果使用了内部字体控制命令，特别是那些来自于 lfonts.tex 文件的命令，就很有可能会出现问题。然而，应该指出是，即使这样，也比 NFSS的

第一个版本与第二个版本之间的不兼容性小。

据我们目前的经验，我们发现即使把盒子和输出程序算在内，也只有很少几个宏包不能在 LaTeX 2ε下运行。我们遇到的最糟糕情形是在 LaTeX2.09 下显式地调入一个支持文件，而这个文件在 LaTeX 2ε下却不保证存在。如果它被强行调入，会导致出现严重的错误信息。

§1.3.6 “我应该用哪一个版本呢？”

如果你不知道在你所用的系统中是否安装了 LaTeX 2ε，那么有许多方法可以检查这一点。在任一 LaTeX作业开始处，都会在监视器和抄本文件 (transcript file) 中列出格式的名字和日期，如：

若是 LaTeX2.09，则为 `LaTeX Version 2.09 <25 March 1992>`，

若是 LaTeX 2ε，则是 `\LaTeXe <1994/06/01>`

当然你的日期可以与这里的不同。

如果不喜欢这种方法，还可以尝试处理一个文件，第一行用的是

```
\documentclass.
```

如果你得到错误信息：

```
! Undefined command sequence
      \documentclass
```

那么你用的就不是 LaTeX 2ε。

§1.3.7 更新 LaTeX 2ε

按计划在每年的 6 月和 12 月份要对 LaTeX进行更新，不仅要改进内部编码，也增加一些额外的功能。这就是说随着时间的推移，仅仅只说一个文档是在 LaTeX 2ε下写的还是不够的，而是不但要指明必要的版本，还要加上发行的日期。

在本书的正文和附录 ?? 的命令汇总中，对于某一不是 1994 年 6 月 1 日所发行的正式版本中命令，都标上发行日期。

也可以在文档文件中声明一个最早的可以处理所用全部功能的版本。这可以在靠近开头的地方，加上一条鉴别命令（ C.2.1 节)。对于要用的第一个版本，应该用

```
NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

这里的日期必须是数字，格式要如上面的一样，即 / 年 / 月 / 日，中间有斜线和必要的零。如果有这一命令的文件被 LaTeX更早的版本处理，就会显示一条警告信息。

§1.4 如何使用本书

本指导是教科书与参考手册的混合体。它解释了 LaTeX所有的基本组成，

特别是那些新标准 LaTeX 2ε中的部分，而且尽可能地只限于那些在标准发行中的内容。与 Lamport 的书相比，这里讲得更详细些，给出了更多的例子和习题，描述了很多来自于作者经验的 “技巧”。与 The LaTeX Companion 一书不同，我们并没有对在标准发行版本以外的可用的许多宏包进行讨论，因此对普通读者而言，这些软件包可能并不是很容易得到。唯一的，也是必需的一个例外是附录 D，那里讲述了一些额外的功能。

本书适用于那些具有很少，甚至没有计算机使用经验的用户。但是它并没有包含与计算机系统有关的操作，如怎样注册用户，启动编辑器，或者使用编辑器等等。

本书的正文有相当多的重复，特别是前半部分，因此读者只要掌握了一些表达中的主要定义，并不必要一定精确在几页后给出的完整定义。另外，读者应习惯从 2.1–2.4 节开始出现的那种描述基本组成的方式。

作者已尽力避免使用计算机行话，虽然这不一定很成功。希望类似于文件和编辑器等这样再不可能用其它方式表达的术语，能广为人知。

在描述命令语法时，对那些必须精确照原样输入的部分用打字机字体表示，而那些可以改变的部分或者文本自身，则用斜体表示。例如，生成表格的命令就用如下方式陈述：

```
\begin{tabular}{ 列格式 }  文本行  \end{tabular}
```

用打字机字体表示的部分是不能省略的，而 *列格式* 表示必须在此处加上列格式的定义。可允许取的值及其组合在命令的描述中有详细的介绍。在上面的例子中，*文本行* 代表表格中的行元素，它是文本的一部分。

§1.4.1 从 2.09 中区分开 LaTeX 2ε

由于现在的标准是 LaTeX 2ε，所以这里给出的语法和功能描述都是针对于这一版本的。然而，我们相信还会有很多用户在未来的几年内仍然使用的是 2.09 版本，这一方面可能是由于他们没有机会更新，另一方面也可能因为他们本来就不想更新。因此本书也考虑到了一点。

由于这个原因，新出现在 LaTeX 2ε中的命令，或者已存在的命令，但进行了扩充，我们有记号 $\boxed{2\varepsilon}$表示。类似地，任一只适用于 LaTeX2.09 版本的文本，则标上 $\boxed{2.09}$。

由于 LaTeX 2ε是向下兼容于 LaTeX2.09 的，因此不必标出只属于原来版本的任一命令。不但如此，有些命令只所以还保留下来（如 `\bf` 和 `\rm`），主要是为了兼容，因此在 LaTeX 2ε中并不鼓励使用这些命令，而在 LaTeX2.09 中它们则是不可缺少的。

©TEXGuru, August 15, 1999

§1.5 LaTeX文件的基础知识

§1.5.1 文本与命令

每一文本都是由字符组成，这些字符放在一起组成单词。由多个单词组成句子，句子再组成段落。而段落可以做为更大单位（如章节）的一部分。

单词由一个或多个字符组成，由空白或回车终止。TeX把空白和回车都做为单词的结束符。而单词间的空白多少则无关紧要，多个空白对单词间的间隔没有更多的作用。

段落的分隔用的是一个或多个空行。同样地，段落间隔与空行的数目无关。TeX把段落中的所有单词当做一个单词长串来处理，选择其间隔，以使得尽可能得均匀，而且每一行都要向左或右调整。行的断开是自动的，与文本的输入基本无关。

行的间隔与所选的字体尺寸有关。段落是以首行的缩进或者增大的行间隔为标志的。在必要的时候，段和行的间隔都会发生细微的变化。TeX（和LaTeX）通过调整这些间隔，使得一页的顶部和底部位于需要之处。这个位置在文档内可以改变。当断行结束后，会自动地进行分页。

在最简单的情形中，一个文本文件中只有输入文本。TeX就会用标准的宽度和高度来处理这一文本；也就是说，把要排版的文本均匀分成行，段，页。

然而，每个LaTeX文档通常都不只是包含要处理的文本，还包含定义如何处理它的命令。因此就有必要给一种方法，区分开这些命令和文本。命令由不能做为文本字符的单个字符组成，或者由单词组成，它们前面都要加上一个特殊的字符，即反斜杠 \ 。

§1.5.2 LaTeX文档的结构

在每一个LaTeX文件中，都一定有导言 (preamble) 与正文 (body) 。

导言是一组命令的集合，它指定处理后面文本的全局参数，如页面格式，文本的高度和宽度，输出页的页码、页眉与页脚的组成。即使最简单的情形，导言也必须包含命令 `\documentclass` ，以指定文档的全局处理类型。这通常也是导言中的第一条命令。

如果在导言中再没有其它命令，LaTeX就会为行宽，页边，段落间隔，页面高度与宽度和其它东西选择标准值。在原始版本中，这些设置采取的是美国标准。对于欧洲用户所用的程序，存在内建的选项，使得文本高度和宽度为 A4 纸标准。而且还存在与语言有关的软件包，已把 'Chapter' 和 'Abstract' 的标题进行了翻译。

导言是用 `\begin{document}` 来表示结束的。紧接这条命令的每一样东西都被解释为正文。它由文本中混杂其它的命令组成。与导言的内容相比，

这些命令只有局部的作用，即它们只适用于一部分文本，如缩进，公式，对字体的暂时改变，等等。正文是用 \end{document} 来结束的。这通常也是文件的结束。

一个 LaTeX文件的通常语法如下：

2ε \documentclass[选项]{ 类 }

其它全局命令和定义

\begin{document}

文本与只有局部作用的命令的混合

\end{document}

可以出现在 \documentclass 命令中的所用可能的选项和类在 3.1.1 节介绍。

对于 LaTeX2.09 ，或者与它兼容，但出现在 LaTeX 2_ε之前版本，必须用 \documentsytle 来取代初始化命令 \documentclass 。在版本 2.09 中其一般性语法应该是：

2.09 \documentstyle[选项]{ 类 }

. . . . . .

\begin{document}

. . . . . .

\end{document}

在一个计算中心，所有可以使用的 LaTeX类和软件包的信息是在一本叫“局部指南” 的书中。其中也应有如何进行程序设计的介绍，以及安装了哪些输出设备，如打印机，缩影胶片，绘图仪，等等，同时这指南应该介绍了如何使用这些设备。设备驱动程序也可以识别选项，在生成输出时进行各种不连续的放缩。

§1.5.3 LaTeX的处理模式

在处理过程中， LaTeX总是处于下面三种模式之一：

1. 段落模式，
2. 数学模式，
3. 从左到右 (LR) 模式。

段落模式也就是正常的处理模式，这时 LaTeX把输入文本做为一队要被（自动）断开成行、段落与页的单词和句子。

当遇到特定命令，表示下面的文本代表公式时， LaTeX就会切换进入数学模式，在公式中空白被忽略。 is 和 i s 这两组文本都解释成 i 与 t 的乘积 it 。当遇到相应的表示公式结束的命令时， LaTeX就会切换回段落模式。

LR 模式类似于段落模式，这时 LaTeX从左到右处理输入文本，把它们做为一组不能被断行的单词来看待。例如，当普通文本被嵌入到公式中时，或

者用命令 \mbox{短文本} 来强迫 短文本 位于一行上时，LaTeX就位于 LR 模式。

理解与识别处理模式是相当重要的，因为有些命令只能用在特定的模式中，或者在不同的模式中有不同的作用。

今后我们将把段落模式和 LR 模式合称为文本模式，以表明它们的性质一致性。但要把它们与数学模式区分开，因为通常它们的差别是很大的。

§1.5.4 生成 LaTeX文档

从输入文本到在打印机上得到输出的 LaTeX文档结果，是一个三步的过程。首先利用计算机的编辑器创建（或修改）文本文件。这个文本文件由实际的文件混杂 LaTeX命令组成。

文本文件的全名由基本名加上扩展名 .tex 组成（如 sample.tex ）。计算机操作系统通常对文件名的选取有一些附加限制，如基本名和扩展名可以拥有的最多字符数，或者哪些字母不可以使用。例如，如果基本名只限于不超过 8 个字符，那么 finances.tex 就是允许的名字，而 financial.tex 则不可以做文件名。

然后文本文件由 LaTeX处理。在 1.2.1 节中已做了介绍，这时用 LaTeX格式执行 TeX程序。在大多数安装版本中，对此都有一个特定的缩略命令，叫 latex ，这样只要在它后面接下文本文件名字就可以了，不必带后缀 .tex 。

例如，如果文本文件的名字是 sample.tex ，那么应该用下面的调用来启动 LaTeX处理程序：

```
latex sample
```

在这一处理过程中，终端监视器会显示页号以及可能会有的错误和警告信息。第 9 章对这些错误信息及其后果做了介绍。当 LaTeX结束了这一处理过程后，它会生成一个新的文件，其基本名不变，后缀为 .dvi 。在上面这个例子中，它就是 sample.dvi 。

这个 .dvi（与设备无关）文件由格式化后的文本以及所需要的字符字体有关的信息组成，但是它与要使用的打印机的特征无关。这样的与设备无关的文件也叫做元文件 (metafile) 。

最后，在 .dvi 元文件中的信息要被转化成可以在选定打印机上输出的形式，这一过程是由一个叫打印机驱动程序来完成的，它与打印机是相关的。对驱动程序的调用，与调用 LaTeX程序一样，也只需使用基本名，即不需加扩展名 .dvi 。打印机驱动程序处理完后，就又会生成另一个相同基本名的文件，而扩展名与打印机有关。

例如，如果输出要关到激光打印机上，可以如下调用驱动程序：

```
dvilj sample
```

这样就会生成一个叫 sample.lj 的文件，或者输出直接就送到了打印机上。

除了 PostScript 打印机外，也存在其它高质量的点阵打印机。然而，本书的目的不是描述 DVI 驱动程序，因为在这一方面有许多可用的程序，而且有不同的名字和特征，例如可以只打印特定的页或者反序打印。在一个计算中心中，应该已安装了一些可以使用的重要的驱动程序。关于这些调用的汇总或进一步的信息可以查看任何用户都可以使用的 TEX或 LATEX局部指南。然而关于这些信息的最好来源，我们前面已提到过，那就是问那些专家了。

©TEXGuru, August 15, 1999

第二章　命令与环境

LaTeX处理程序所需要的文本必须用文本编辑器把它们输入到一个后缀为 .tex 的文件中，任何人都必须这样做。这个文件的内容完全是可以打印的字符，没有在屏幕不可见的字符或命令。然而，为了使得 LaTeX进行某种操作，就必须有一些命令。这些命令在输入文本中是可见的，在输出的结果中就不存在了。因此知道命令与要打印出来的文本的区别，以及命令的作用是相当重要的。

§2.1　命令的名称与参数

字符 # $ & ~_ ^ % { } 表示特殊的命令，它们的含义后面会讲到。如果要以文本的形式打印这些字符，在它们前面必须有 \ (反斜杠)。

大约有二十条命令是由 \ 后接另一个非字母字符的，即它共只由两个字符组成。在这种方式中，\$ 就表示暂时取消命令 $ 的含义，而以文本的方式打印出这个字符。

绝大多数的命令是由 \ 后接一个或多个字母组成的，第一个非字母字符表示命令名称的结束。许多命令可以有参数，或变量来扩展相应的功能。参数有可能是可以省略的，也就是说它们可以有，也可以没有；也有的参数是不能省略的，即至少必须给一个允许的值。这些命令的语法是

\命令名[可省参数]{不可省略的参数}

这里可省参数是放在中括号 [ ] 里，而不可省略参数放在大括号 { } 里。一条命令可以有几个可省略参数，它们就必须按指定顺序位于各自的中括号内。如果没有用可省略参数，方括号可以不写。在命令名称和参数之间可以有任意数目的空格。

对于可省略参数，有可能出现两个问题：如果没有使用任何可省略参数，而接下来的文本中第一个字符是 [，或者一个可选参数的文本中有] ，那么LaTeX就会误解参数。这可以通过使用大括号来遮住的中括号，如 {[} 和 {]} 。

有的命令有几个不可省略的参数。它们中的每一个都必须位于一对大括号内，而且顺序要同命令描述中的顺序一样。例如，

\rule[提升高度]{宽度}{高度}

就会生成一个大小为指定 宽度 和 高度 的黑矩形，并且在当前基线基础上向上提升一定的高度。一个宽 10mm ，高 3mm ，并且位于基线上的矩形，可以如此生成： \rule{10mm}{3mm} 。在这里并没有给出可省参数。参数必须按描述中的顺序出现，不可能交换。

有些命令有两种形式，一种是标准形式，而另一种就是所谓的 *- 形式。后者的标志是命令名称以 * 号结尾，这个 * 号位于成对的相应于可省参数和

不可省参数的 [ ] 和 { } 之前。标准形式与 *- 形式之间的区别，我们会在每条命令中加以解释。

命令名称是在第一个非字母字符之前结束的。如果命令后接可省参数或不可省参数，那么命令名称是在 [或 { 之前结束的，因为这两个字符就为非字母字符。然而有很多命令，其没有参数，只是由名称组成，如生成 LaTeX标志的命令 \LaTeX 。如果这样的命令后面接的是一个如逗号或句号的标点符号，那这显然表示命令的结束。如果它后接一个普通单词，那么命令名称与接下来单词间的空白当做命令结束符对待： \LaTeX logo 生成的结果是 LaTeXlogo ，也就是说，这里的空白只当做命令的结束符，而不当做两个单词的间隔。这是空白的一个特殊规则，在 2.5.1 中有介绍。

为了在只由名称组成的命令之后插入一个空格，那就需要在命令后使用一个空结构 {} 或空格命令（ \ 加空格 ）。因此生成 'The LaTeX logo' 的正确方法是输入 The \LaTeX{} logo 或 The \LaTeX\ logo 。如果不这样做，也可以把命令自身放在大括号内，如 The {\LaTeX} logo ，这样也会得到有空格的正确输出： 'The LaTeX logo' 。顺带说一下，用命令 \LaTeXe 可以得到 LaTeX 2_ε标志。

§2.2 环境

一个环境是用命令 \begin{ 环境 } 来初始化的，最后用 \end{ 环境 } 结束。对于环境的所允许取的值列在有关命令的章节中。

环境的作用是：位于其内的文本要根据环境参数进行不同的处理。有可能（暂时地）改变某一处理特征，如缩进，行宽，字样等等。这些改变只在该环境内有作用。例如，对于 quote 环境，

```
前面的文本
\begin{quote}
文本 1 \small 文本 2 \bfseries 文本 3
\end{quote}
后续文本
```

与前面和后续文本的左边和右边页边相比，环境中的页边界要增大。在这个例子中，对三部分文本： 文本1 、 文本2 和 文本3 的页边界要增大。在 文本1 后面是命令 \small ，这样就把后面文本变成小字样。在 文本2 后又有一命令 \bfseries ，它切换到黑体字样。而所有这些命令的作用到 \end{quote} 时也就结束了。

在 quote 环境中的三部分文本都相比于前面和后面的文本进行了缩进。 文本1 是以正常字样出现的，这同环境外是一样的。而 文本2 和 文本3 则是小号字样，而且 文本3**还是黑体** 。

©TEXGuru, August 15, 1999

当 quote 环境结束时，接下来的文本字样同以前的一样。

很多命令的名称与环境名称一样。在这种情况中，所用的命令名是没有前缀 \ 的。例如，命令 \em 切换成强调字样，通常用的是斜体，而相应的环境 \begin{em} 则把所有文本变为斜体，直到 \end{em} 为止。

一个没有名称的环境可以用一对大括号 {...} 来表示。其中所有命令的作用当遇到右大括号时也就结束了。

用户也可以创建另外的环境，见 7.4 节的描述。

§2.3 声明

声明 就是一种命令，它并不打印出任何文本，而是改变某一参数或命令的值或含义。声明一旦出现，其马上就起作用，直到遇到同一类型的另一个声明。然而，如果声明是位于一个环境或 {...} 中，那它的作用也就只能延续到相应的 \end 命令或右大括号 } 为止。在前一节中提到的命令 \em, \bfseries 和 \small 都是这样一种没有直接输出，而是改变当前字样的声明例子。

有一些声明与参数有关，如命令 \setlength 会给长度参数赋值（见 2.4 和 7.2 节）。

例：

`{\bfseries This text appears in bold face}` 这里的 \bfseries 声明改变了字样：**This text appears in bold face**. 这个声明的作用到遇到右大括号 } 时结束。

`\setlength{\parindent}{0.5cm}` 把段落中第一行的缩进设为 0.5cm 。当遇到下一个命令 \setlength{\parindent}，或者最近的一个结束当前环境的 \end 时，这个声明的作用就会终止。

`\pagenumbering{roman}` 用罗马数字来显示页码。

有些声明，如上面最后那个例子，它的作用是全局的，也就是说它并不局限于当前环境。下面的声明都具有这种性质，稍后给出它们的含义：

```
\newcounter     \pagenumbering      \newlength
\setcounter     \thispagestyle      \newsavebox
\addtocounter
```

用这些命令结出的声明，也是马上起作用，而且直到被同类型的一个新声明覆盖其含义为止。在上面最后那个例子中，只有当遇到 \pagenumbering{arabic} 这样的命令时，才不会以罗马数字显示页码。

§2.4 长度

§2.4.1 固定长度

长度是由前面可能有符号（ + 或 - ）的小数，后接一个必需的尺寸单位组成。下面是可允许的单位及缩写名称：

`cm` 厘米，
`mm` 毫米，
`in` 英寸（ 1in = 2.54cm ），
`pt` 点（ 1in = 72.27pt ），
`bp` 大点（ 1in = 72bp ），
`pc` pica（ 1pc = 12pt ），
`dd` didôt 点（ 1157dd = 1238pt ），
`cc` cicero（ 1cc = 12dd ），
`em` 与字体相关的尺寸，表示大写字母 M 的宽度，
`ex` 也是与字体相关的尺寸，表示字母 x 的高度。

在 TEX和 LATEX中的小数，既可以采用英国方式，也可以采取欧洲方式，即小数点可以句号或逗号，也就是说 `12.5cm` 和 `12,5cm` 都可以。

注意 `0` 不是一个合法的长度，因为其没有长度单位。要给出一个零长度，必须用 `0pt` 或 `0cm` 等等。

要想给长度参数赋值，可以用 LATEX命令 `\setlength` ，我们将在第 7.2 节中介绍它和所有其它处理长度的命令。它的语法是：

```
\setlength{\ 长度命令 }{ 已定义的长度 }
```

例如，一行的长度是由一个叫 `\textwidth` 的参数定义的，它通常根据样式和字体尺寸来取默认值。要想把行宽设为 12.5cm ，可以用：

```
\setlength{\textwidth}{12.5cm}
```

§2.4.2 橡皮长度

有些参数的值为橡皮长度，即这个长度可以伸展或收缩给定的量。橡皮长度的语法是：

```
正常值 plus 伸展值 minus 收缩值
```

这里的 `正常值` ， `伸展值` 和 `收缩值` 都是长度。例如：

```
\setlength{\parskip}{1ex plus0.5ex minus0.2ex}
```

其含义为：两段间的行距（称为 `\parskip` ）等于当前字体中 x 的高度，但是该行距可以伸长到 1.5 倍或收缩到 0.8 倍这个长度。

有一个特殊的橡皮长度是 `\fill` 。其正常长度是零，但可以伸展到任何长度。

§2.5 特殊字符

§2.5.1 空格与回车

空格或空白字符具有与通常字符不太一样的性质，我们在前面的 2.1 节中已提到了一些。在处理的过程中，空格是用橡皮长度（ 2.4.2 节）代替的，这样可以使得行恰好填满一行的宽度。因此，如果你不知道下面的规则，就可能出现一些古怪的现象：

- 一个空格同一千个空格是一样的，实际接受的只是第一个空格；
- 在一行开头的空格是被忽略不计的；
- 回车（开始新行）是当做空格对待的。

这些规则的一些结果就是可以在单词间或一行的开头插入随意多的空格（这样会使源文件更好看些），而且一个单词可以恰好位于一行的末尾，而它和后一个单词之间可以没有空格。为了强迫在一般情形中会消失的空格必须出现，那就要用命令 \␣（ \ 后接一个空格键，这里用符号 ␣ 表示）。

而有的时候又有必要避免由于开始一个新行而出现不必要的空格。在这种情形中，该行必须在回车键前插入注释符号 % 。

一行中有两个回车键得到一个空行，这也是开始一个段落的标志。而对于空白字符，一个与一千个是一样的。

§2.5.2 引号

在打字机上可以找到的引号 " 并不会用在书籍印刷中。取而代之的是在开始和结束用不同的符号，如 ‘单引号’ 和 “双引号” 。单引号是用 ‘ 和 ’ 生成的，而双引号是用两个相应的字符表示的： ‘‘ 表示 “， ’’ 表示” 。另外，打字机符号 " 会生成右双引号。

§2.5.3 连字符与破折号

在书籍印刷中，打字机上为 - 的符号有各种不同的长度： -, –, — 。其中最短的是连字符，用在诸如 father-in-law 这样的复合单词中，以及在一行结尾处的单词分割中；中间长短的是短破折号，用于数值范围，如第 33–36 页；而最长的那个是全破折号，用于复合句 — 通常也就称为破折号。可以用输入连字符一次、两次或三次来得到这三种不同长度的横线，即 - 结果为 - ， -- 结果为 – ， --- 结果为 — 。第四种类型的破折号是负号 – ，必须用 $-$ 这样的数学模式来输入（第 5 章）。

§2.5.4 得到命令字符

在 2.1 节中提到，字符 # $ ~ _ % { } 都被当做命令处理。如果要把它们做为文本来显示，必须在它们前面加上字符 \ ：

$ = \$　 &= \&　 %=\%　 #=\#　 _=_　 {=\{　 }=\}.

§2.5.5 特殊字符 §, †, ‡, ¶, ©, £

在计算机键盘上并没有这些特殊字符。可以用如下特殊命令来得到它们：

§=\S †=\dag ‡=\ddag ¶=\P ©=\copyright £=\pounds

在第 5 章中描述了如何生成希腊字母和其它数学符号。

§2.5.6 外文字母

在非英文的其它欧洲语言中，还存在一些特殊字符，它们也可以用 TEX得到。这些字符有：

œ={\oe} Œ={\OE} æ={\ae} Æ={\AE} å={\aa} Å={\AA}

¡=!‘ ø={\o} Ø ={\O} ł={\l} Ł ={\L} ß={\ss}

SS={\SS} ¿=?‘

Ångstrøm 可写为 {\AA}ngstr{\o}m, Karlstraße 可用输入 Karlstra{\ss}e 来得到。大写的 \SS 是 LATEX 2ε中的新命令。

§2.5.7 重音

在欧洲语言中，有很多发音记号或重音，用 TEX可以显示出大多数的重音符号：

ò=\‘{o} ó=\’{o} ô=\^{o} ö=\"{o} õ=\~{o}

ō=\={o} ȯ=\.{o} ŏ=\u{o} ǒ=\v{o} ő=\H{o}

o͡o=\t{oo} o̧=\c{o} ọ=\d{o} o̲=\b{o} o̊=\r{o}

（最后的命令 \r 是新出现在 LATEX 2ε中的。）上面的 *o* 只是一个示例，实际上可以用任何字母。而 i 和 j 应该专门提一下，在加上重音时必须去掉点，这只要在这两个字母前面加上 \ 就可以了。命令 \i 和 \j 就会得到 ı 和 ȷ。所以 ǐ 和 ǰ 是用 \u{\i} 和 \H{\j} 来得到的。

对于由非字母组成的重音命令，可以不使用大括号：

ò=\‘o ó=\’o ô=\^o ö=\"o õ=\~o ō=\=o ȯ=\.o

而由字母组成的重音命令就必须用大括号。在 TEX中也可以用其它的记号，但是这些记号很容易被混淆，由此这里也就不提及那些方法了。

§2.5.8 连写

在书籍印籍中，有些特殊组合的字母，并不是单独印出，而是用一个符号来显示，这称为 连写 。在 TEX中对于字母组合 ff, fi, fl, ffi 和 ffl 不是显示为

ff, fi, fl, ffi, ffl ， 而是 ff, fi, fl, ffi, ffl.

在 3.5.1 节中描述了如何强迫 TEX分开显示这些字符，而不是当作连写处理。对于排版象 shelfful 这样的单词，就必须这样做，因为当印刷时象通常那样处理为 ff 连写，那么单词 shelfful 就显得很古怪了。

©TEXGuru, August 15, 1999

§2.5.9 日期

在文本中的任何地方，都可以用命令 \today 来显示当前日期。日期的标准形式采用的是美国月，日，年（如 Novermber 15, 1995 ）形式。若要实现其它语言中的日期形式可以借助于 TEX命令 \day ， \month 和 \year ，这三条命令以数字的形式返回当前的参数值。在 C.3.3 节中有一个例子，演示了如何构造这样的一个新 \today 命令。

§2.6 脆弱的命令

有些命令的作用不只是局限于它所处的地方，而对文档的其它地方也有影响。例如，类似于 \chapter{标题} 这样的章节命令，不但只是在调用它的地方生成标题，而且也有可能在后续页面的顶部以不同的字样显示出来，甚到以第三种字样显示在目录中。类似于这样的可以显示在文档不同地方的参数，称为移动参数。

这样的参数从字面意义上看，是被移动重组的。（更精确地说，是未被完全解释的。）如果在移动参数中包含了某些命令，那么这些命令会发生变形，而未发挥其正常作用。有人形象地称之为分离组合。我们称这样的命令为脆弱的，而其它的不是这样易变的，能抵抗这种重组的，称为牢固的命令。

从本质上说，所有包含可省参数的命令，如 \begin 和 \end 都是脆弱的。脆弱命令可以包含在移动参数中，但要前缀命令 \protect ，这样就可以防止其被分离重组了。

在移动参数中未用 \protect 进行安全保护的脆弱命令，也不是一定要被重组。事实上，只有在极少数情形下才会发生分解。在本书的德语第一版中，只有章节标题中的元音变音（如 ä, ö 和 ü 这样的两点）需要 \protect\" ，以确保其正确地出现在目录中。（在 LATEX的后来版本中，重音命令不再是脆弱命令，而且在 LATEX 2ε中，它变得更牢固。）

对于脆弱命令， LATEX是不能正确处理的，因此会在屏幕上显示出一长串的错误信息。只要按一下回车键，用户可尝试继续处理下去，而不理睬对该命令的不正确对待。有可能还会出现更多的错误，但最终只要按足够多的回车， LATEX通常是能进行后面的处理的，除非那么被损坏的命令导致不可能进行下面的处理，而使程序停止运行。

只有下列命令包含移动参数：

- 所有传递文本信息给目录表的命令.
  它们是章节命令（ 3.3.3 节），\addtocontents, \addcontentsline（ 3.4.3 节）和 \caption （ 6.6.4 节）;
- \typein 和 \typeout 命令（ 8.1.3 节）;
- \markboth 和 \markright 命令（ 3.3.1 节）;

- 标题页上的 \thanks 命令（ 3.3.1 节);
- @- 表达式（ 4.8.1 节);
- \bibitem 的可省参数（ 4.3.6 节);
- 调用了 \makelabels 命令后的 \begin{letter} 命令（ A.1 节)。

只有当脆弱命令出现在上述命令的参数中时，才有必要用 \protect 命令来保护它们。在移动参数中，对于大多数命令而言，只要它是脆弱的，在其前面加上 \protect ，对处理没有什么影响。这个规则的例外就是长度命令（ 7.2 节) 和记数器命令（ 7.1.4 节)，如 \arabic 和 \value 。在这些命令前从来不要用 \protect 。

§2.7 练习

作为一部成功的自学教材的一部分，必须要用练习题。除了正文丰富的例子，读者应自己进行尝试外，在本书的教学中也应给出相当数量的进一步的练习，建议把它们当做必须完成的家庭作业。

所有的练习都是针对于 LaTeX 2$_\varepsilon$的，如果你用的还是 LaTeX2.09 ，那你就必须在开头用 \documentclass 的地方用 \documentstyle 。

练习 2.1本练习用一小段文本来测试一下对运行 LaTeX程序的基本步骤的掌握。同时也包含了几个简单的命令。用计算机的编辑器输入如下文本，并把它存到一个叫 exer.tex 的文件中：

```
\documentclass{article}
\begin{document}
Today (\today) the rate of exchange between the British
pound and American dollar is \pounds 1 =\$1.63, an
increase of 1\% over yesterday.
\end{document}
```

虽然对 LaTeX程序的执行在不同的计算机上可能不一样，但这里我们假定利用命令 latex 来调用它，那么可如下处理文件：

```
latex exer
```

注意：虽然文件名是 exer.tex ，但在调用 LaTeX时只需要给出基本名 exer 。

如果处理过程中没有出现任何错误信息，那么就会成功地生成 .dvi 文件 exer.dvi ，进而可以借助于打印机驱动程序对它进行转化。转化程序的名称与计算机系统有关。那么最后的打印结果应该如下 (日期应是当前日期)：

Today (August 15, 1999) the rate of exchange between the British pound and American dollar is £1 =$1.63, an increase of 1% over yesterday.

对这里所用的命令，用几点要注意：

- 在 \today 后面不必要加上空格，因为) 就完全可以终止它；

- 在 \pounds 后面的空格也可以省略，而且不会打印在结果中；
- 命令 \$ 和 \% 并不需要空格来终止它；如果用了空格，其将会打印在结果中。

练习 2.2: 找本书或期刊，摘录约有 3/4 页的文本，把它输入到 LaTeX文件中。注意用空行来分段。使用与练习 2.1 相同的命令集，即把文本放在命令 \begin{document}...\end{document} 之间，重复上述过程，以得到打印结果。

注意：所选文本中不应包含特殊结构，如缩进块，不同字样，居中文本，列举，数学公式，表格，等等。我们将在后续章节中介绍这些结构。

©TEXGuru, August 15, 1999

第三章　文档的布局与组织

§3.1　文档类

在一个 LaTeX文件导言中的第一条命令通常用来确定整篇文档的全局处理格式。该条命令的语法是：

2ε `\documentclass[ 选项 ]{ 类 }`

或

2.09 `\documentstyle[ 选项 ]{ 类 }`

其中后者只适用于 LaTeX2.09 。对于 LaTeX 2ε , 可以用命令 `\documentstyle` , 但这只是为了与老的源文件兼容为提供的。

而类的可能取值是： `book`, `report`, `article` 或 `letter` ， 只能从中选取一种。(在附录 A 中讲解 letter 类的性质。) 这些类之间的差异包含页面布局和组织方面的不同。一篇 article (论文) 可以有 parts (部分), sections (节), subsections （ 小节), 等等, 而一篇 report (报告) 还可以有 chapters (章)。一本 book （ 书) 也有 chapters (章), 同时对奇偶页区别对待; 而且它根据章节标题在每页上打印出当时的页眉。

§3.1.1　标准的类选项

用选项可以对格式进行各种不同的修改。它们可如下分组：

选择字体尺寸

可以用如下选项来选择基本的字体尺寸：

2ε `10pt`　`11pt`　`12pt`

这个选项就是在文档中普通文本所取的字体尺寸。默认值是 `10pt` , 这意味着如果没有指定尺寸选项时就用这个值。其它所有字体尺寸都是相对于这个标准尺寸而言的, 因此如果选定了不同的基本字体尺寸, 节的标题, 页脚等等就会相应地自动改变尺寸。(注意在 LaTeX2.09 中没有 `10pt` 选项; 如果想用 10pt 做为基本尺寸, 只要不指定尺寸选项就可以了。)

指定纸张大小

LaTeX是根据选定的字体尺寸和纸张大小来计算行宽和每页行数。同时它也会选择适当的页边, 以使文本水平和竖直居中。为了做到这一点, 就需要知道所用纸张的格式。下面的选项可以用来指定纸张大小, 其全部只存在于 LaTeX 2ε中：

2ε letterpaper(11×8.5in)　2ε a4paper(29.7×21cm)
2ε legalpaper(14×8.5in)　2ε a5paper(21×14.8cm)
2ε executivepaper(10.5×7.25in)　2ε b5paper(25×17.6cm)

默认值是 letterpaper，即美国信纸尺寸的纸张，大小为 11×8.5in 。

通常情况下，纸张格式中长的方向是竖直方向，即所谓的纵向模式。利用选项

2ε landscape

可以使短的方向成为竖直方向，即横向模式。

页面格式

在一页上的文本可以用下面的选项来使得以一列或两列形式分布：

2ε onecolumn　twocolumn

默认值是 onecolumn 。当用的是 twocolumn 选项时，两列间距以及列间可能存在的标尺的宽度用 \columnsep 和 \columnseprule 来指定，后面将会讲到它们。

要想奇偶页的页码打印方式不一样，可以用选项

2ε oneside　twoside

当用的是 oneside 时，所有页码的打印方式是一样的；然而，当用的是 twoside 时，在即时标题中若当前页码为奇数，页码出现在右边，若为偶数，页码出现在左边。注意它并不是强迫打印机双面打印。这里的想法是当以后真的双面印刷时，页码总是在每页的外侧，阅读时容易看到。这是 book 类的默认值。对于 article 和 report 类，默认值是 oneside 。

对于 book 类，每一章通常都是开始于右边，即从奇数页开始。选项

2ε openright　2ε openany

可以控制这个功能：用 openany 时，总是在下一页上开始新的一章，而用的是默认值 openright 时，必要时可以插上一空页。

通常 book 或 report 的标题是单独一页的，而在 article 中，标题则是与开头的文本在同一页上。利用选项

2ε notitlepage　titlepage

可以重定义这种标准行为。请参见 3.3.1 节和 3.3.2 节。

其它选项

还剩下的标准选项有：

leqno 显示公式中的公式编号出现在左边，而不是像通常那样放在右边（ 5.1 节 ）。

fleqn 显示公式左对齐，而不是居中（ 5.1 节 ）。可以参数 \mathindent(下面会讲到它) 来设置缩进大小。

openbib 参考文献的格式可以改变成每一片断位于一个新行上。默认方式下，每个条目的文本都是聚在一起的。

draft 如果 LaTeX的断行机制不能很好的发挥作用，总有些文本在右边界突出，那么这个选项就会用一个粗黑条来标识它，以使得突出易见。

2ε final 与 draft 相反，这是默认值。无论文本行有多宽，也不会加上标识。

如果同时给出了多个选项，那就要用逗号把它们分开，例如，

```
\documentclass[11pt,twoside,fleqn]{article}
```

选项顺序是无关紧要的。如果同时指定了两个矛盾的选项，如用了 oneside 和 twoside，无法确定哪个将发挥作用。这主要看在类文件中的具体定义了，因此最后避免出现这种情形。

与某些选项相关的参数

有些选项要使用一些已有确定默认值的参数：

\mathindent

当选择了 fleqn（ 5.1 节）时，指定公式相对于左边界的缩进量；

\columnsep

为 twocolumn 选项指定两列间距（见 ?? 页上的图 ?? ）；

\columnseprule

为 twocolumn 选项确定两列间竖线的宽度。默认值是宽度为零，即没以竖线（见图 ?? ）。

这些参数的标准值可以用 LaTeX命令 \setlength 来修改。例如，要把 \mathindent 改为 2.5cm，可以用

```
\setlength{\mathindent}{2.5cm}
```

对这些参数的修改，既可以在导言中进行，也可以在文档的其它任意地方进行。在导言中进行的修改，适用于整个文档，而在文本内的修改，其作用终止于下一次修改，或者它所在的环境结束（ 2.3 节）。在后一种情形中，先前的值就开始继续发挥作用。

练习 3.1: 打开在练习 2.2 中所用的文本文件，把初始化命令

```
\documentclass{article}
```

先后改为

```
\documentclass[11pt]{article}
```

和

```
\documentclass[12pt]{article}
```

，并分别打印出结果。把这两次输出以及练习 2.2 的输出放在一起，比较断行点的不同。

注意：如果有些单词的断开是不适当的，你可以用命令 \- 告诉 LaTeX哪些地方是恰当的断点，例如， man\-u\-script 。（这是 TeX英文单词断开算法不能很好处理的很少的几个单词之一。）在 3.6 节中对修改断开算法的其它方法进行了介绍。

如果在 LaTeX处理过程中，有类似于 Overfull \hbox ... 这样的警告，这说明 TeX不能很好的断开这一行。在输出的结果中，这些行将会越出右边界。一般的原因是 TeX不能断开某单词造成的，之所以不能断开某个单词，要么是因为它不可断开，或者 TeX单词断开程序不适用。此时在单词中插入建议断点，就可以解决问题。我们很快会给出其它的解决办法。

练习 3.2: 现在把那文本文件的初始化命令改为

```
\documentclass[twocolumn]{article}
```

。

如果这次你得到了许多 \Underfull \hbox ... 警告，那么这些行将会进行左右调整，从而在单词间会有很大的空档。亲自检查一下输出，看一下单词间隔是否还能令人接受。如果不行的话，就试着在下一行开头那个单词中加上建议断点。

注意：如果在前面的练习中你用的是 book 类或者 report 类，而不是 article 类，你会注意到结果实际上是没有区别的。这是因为这些类只对文档的最终结构有作用。一般地讲，对短文章（比如说 10–20 页），应该用 article 类，对长篇报告用 report，这样可以分章。而且章总是从新页开始。要写书就要用 book 类。

§3.1.2 利用宏包增加功能 [2ε]

虽然文档类确定了文档的全部一般性质，如页面步局和章节划分，但是也可以用一些更具体的宏包，来改变某些命令的行为，甚至定义一些全新的命令，以增加非 LaTeX标准的其它功能。在 LaTeX的发行中，有很多这样的宏包，甚至你也可以从 LaTeX使用者团体那里获取成千个可用的宏包（见附录 D），当然你也可以自己编写（附录 C）。

一个宏包就是一组 LaTeX（或 TeX）命令集，并且被保存在一个后缀为 .sty 的文件中，虽然有些特殊命令只能用在宏包中。要调用一个宏包，只需在导言中用

[2ε] \usepackage{ 宏包 }

这里的 宏包 就是文件的基本名。用一条 \usepackage 可以调用不只一个宏包。例如，在标准 LaTeX中有两个宏包，保存在文件 makeidx.sty（ 8.4 节）和 ifthen（ 7.3.5 节）中。那么可以用如下方式调用它们：

```
\usepackage{makeidx,ifthen}
```

一个宏包也可以具有与之相关的选项，它以与文档类选项相同的方式来选择，即要把选项名放在中括号内。其一般的语法是：

\usepackage[选项 1, 选项 2 ...]{宏包 1, 宏包 2, ...}

这里所列出的选项将适用于所有选择的宏包。如果有宏包不理解所给选项，将在监视器上给出一条警告信息。

至于有哪些选项可用，那就要看具体的宏包了。你必须去阅读随宏包一起的描述和 / 或文档，不但看看到底有哪些选项，还要看看这个宏包是做什么的，它定义了那些新命令。

§3.1.3 样式选项 2.09

在 LaTeX2.09 中并没有 \usepackage 命令；事实上，若 \documentclass 地方用的是 \documentstyle，该命令就根本没有作用。在 LaTeX2.09 中，宏包与样式选项以同样的方法处理，即其文件名要加到在 \documentstyle 命令中所列出的选项表中。正是由于这个原因，它们也称为 *样式选项文件*，或简称为 *样式文件*，这也就是这些文件后缀 .sty 的来源。在 LaTeX 2ε 中继续使用这个后缀，是为了保证原来的样式文件还可以作为宏包使用。

在真正的选项（在类或宏包文件中对其进行程序设计）和宏包（从与之同名的文件中读取）之间是有本质的区别的。而在 LaTeX2.09 中，这种差别是不明确的，经常导致混淆。

在原来版本的 LaTeX 中，要想向类（或 “主样式”） article 调入宏包 makeidx，同时带有选项 12pt 和 titlepage，就必须用

2.09 \documentstyle[makeidx,12pt,titlepage]{article}

由于这里是把宏包做为选项来处理，显然它们自己不可能再有选项了。

§3.1.4 全局和局部选项 2ε

用 \documentclass 命令指定的选项有一个有趣的特点，那就是它适用于所有后面的宏包。这就是说如果有几个宏包具有相同的选项，那就只需在 \documentclass 中声明一次。例如，你可以设计一个宏包，修改 article，以生成一个局部的特殊样式，使得当文本为单列或双列时做不同的事情；这个宏包可以用类选项 onecolumn 和 twocolumn 来实现这一点。或者对 draft 选项进行精细加工，以生成类似于手稿那样的双倍行距。之与相应的，几个宏包可能具有与语言有关的特征，用选项 french 或 german 来激活；那么只要在 \documentclass 命令中声明就可以适用于所有的选项。这样的选项称为全局的，因为它对后续的所有宏包都有作用。

全局选项并不一定只限于列在 3.1.1 节中的那些标准类选项。如果类和任一宏包都不理解所列出的一个或多个选项，只会显示出一条警告信息。相反地，由 \usepackage 命令指定的选项只适用于列于该条命令中的宏包；而且它应适用于其中每一宏包。如果那些宏包中的一个或多个不认识任一个局部选项，都会显示出一条警告信息。

§3.2 页面样式

基本的页面格式是由页面样式决定的。除了一种特殊情形外，该命令通常放在导言中。其形式为：

\pagestyle{ 样式 }

对不可省参数，有如下可取的值：

plain

页面的页眉是空的，页脚由居中的页码组成。当在导言中没有 \pagestyle 时，这是默认值。

empty

页眉和页脚都是空的；也不显示页码。

headings

页眉由页码及文档类所决定的标题信息（章节标题）组成；页脚为空。对每一章的第一页没有作用。

myheadings

同 headings 差不多，除了页眉的标题不是自动选取，而且由 \markright 或 \markboth 命令显式决定（见下面的解释）。

命令

\thispagestyle{ 样式 }

的作用同 \pagestyle 一样，只是它只对当前页起作用。例如，利用命令

\thispagestyle{empty}

可以取消当前页的页码。这实际上只是不打印页码，下一页的页码就与没有用该命令一样。

§3.2.1 生成页眉的声明

对于页面样式 headings 和 myheadings，出现在页眉中的信息可以用下面的声明来确定：

\markright{ 右边纸页眉 }

\markboth{ 左边纸页眉 }{ 右边纸页眉 }

\markboth 声明用于文档类选项为 \twoside，这时假定偶数页位于左边，奇数页右边。而且，对左边页，页码打印在页眉的左边，对右边页，页码打印在页眉的右边。

对于单面输出，认为所有页都是右手方向的。在这种情况下，应该用声明 \markright。但对双面输出，也可以用它来重新定义 \markboth 中的右边纸页眉。

对于页面样式 headings，位于页眉上的标准信息是章、节和小节的标题，具体要看文档和页面样式，对此有如下的图解：

样式		左边纸	右边纸
book, report	单面页	—	章
	双面页	章	节
article	单面页	—	节
	双面页	节	小节

如果在一页上有不只一个 \section 或 \subsection，就在页眉上显示最后那一个的内容。

§3.2.2 页的编号

定义页编号的声明形式如下：

\pagenumbering{ 数字形式 }

数字形式 可取如下值：

arabic 通常（阿拉伯）数字,

roman 小写罗马数字,

Roman 大写罗马数字,

alpha 小写字母

Alpha 大写字母。

标准值是 arabic。这个声明把当前页码重置为 1。为了用罗马数字显示前言的页码，而其余部分用罗马数字显示，而且第一章开始的页码为 1，那就必须在开始前言时用声明 \pagenumbering{roman}，然后紧接第一个 \chapter 命令，把其重设为 \pagenumbering{arabic}。（在 3.3.5 节中有另一种方法。）

如果想使页码不是从 1 开始编号，那么可以使用命令

\setcounter{page}{ 页码 }

这里的 页码 是前一页的编号。

练习 3.3: 向你练习用的文本文件中再填一些内容，使其长度不只一页，而且其导言为：

```
\documentclass{article}
\pagestyle{myheadings} \markright{Exercises}
\pagenumbering{Roman}
\begin{document}
```

§3.2.3 与段落有关的距离

下述参数对段落的形式有影响，可以用在第 7.2 节介绍的 \setlength 来给它们设定新值：

\parskip

两个段落之间的距离。以 ex 为单位，可以使它随着字体尺寸而自动改

变。其应是一个橡皮长度。

\parindent

段落中第一行的缩进量。

\baselinestretch

这是一个度量两条基线之间正常间距的数值，所谓基线就是字母所坐的位置，即忽略类似于 g 和 y 这样的字母下挂的部分。这个值初始化为 1，表示标准的线距。它可以用下面的命令改值：

```
\renewcommand{\baselinestretch}{ 因子 }
```

这里的 因子 是十进制小数，如 1.5 表示增加了 50% 。这样其就适用于所用的字体尺寸。如果这条命令放在导言外，那么要直到选择了另一条字体尺寸命令，其才会发挥作用（ 4.1.2 节）。

这些参数既可以放在导言中，也可以放在文档的其它任何地方。在后一种情况里，这种修改的作用持续到下一次修改为止，或者它所在的环境结束（ 2.3 节）。

真正的行间距是包含在 \baselineskip 这个长度参数中，只要声明了一个字体尺寸，就会自动设置其值。对每一种字体尺寸，都存在一个正常值，把它乘上 \baselinestretch ，就是 \baselineskip 。详情请见 4.1.2 节。

如果在一个段落中间修改 \baselineskip ，新值确定了整段的行间距。更精确地说，在一段结束时的值才有效。

练习 3.4: 在你的练习文件的导言中加入下列命令：

```
\setlength{\parindent}{0em}
\setlength{\parskip1.5ex plus0.5ex minus 0.5ex}
\renewcommand{\baselinestretch}{1.2}
```

在处理完这个练习后，把 \baselinestretch 的值改一下，如改为 1.5 ，重复这个练习，这样就会得到知道它的作用。做完这些练习后把上述命令从文件中去掉。

§3.2.4 页面格式

每一页都是由页眉、包含实际文本的正文，以及页脚组成的。所谓页样式的选择就是确定在页眉和页脚中应包含哪些信息。

对于页眉，正文与页脚之间的距离，上页边界和左页边界，文本行宽，以及页眉、正文和页脚的高度，LaTeX都有默认值。在图 3.1 中演示了这些格式化长度的意义。可以给这些长度声明一个新值，这些声明最好放在导言中，使用命令 \setlength（ 7.2 节）。例如，可以用

```
\setlength{\textwidth}{12.5cm}
```

来使文本行宽变为 12.5cm 。

©TEXGuru, August 15, 1999

\oddsidemargin
奇数页的左边界，
\evensidemargin
偶数页的左边界，
\topmargin
从上页边到页眉顶的距离，
\headheight
页眉的高度，
\headsep
页眉基线到正文顶部的距离，
\topskip
从正文顶部到正文第一行的基线之间的距离，
\textheight, \textwidth
主要正文的高度和宽度，
\footskip
从正文底部到页脚底部的距离，
2ε \paperwidth, \paperheight
由纸张大小选项所确定的总宽度和高度，包含所有的页边，
2.09 \footheight
已过时，在 LaTeX 2ε 中已被去掉。

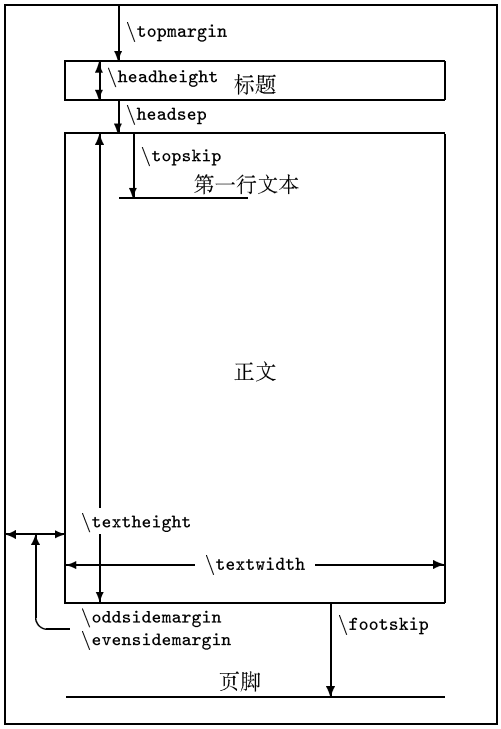


图 3.1: 页面布局参数

在 LaTeX2.09 中还定义了一个额外的参数 \footheight，但从来没有用到它。因此在 LaTeX 2ε 中就去掉它了这个参数。

对于一列和两列输出时的页面格式，在附录 ?? 结尾处的图 ?? 和 ?? 中有详细的图示。

为了能准确计算页面布局，我们必须知道 LaTeX 中的度量是从纸张上边一英寸的点和纸张左边一英寸的点开始的。因此整个左边界是 \oddsidemargin 加上一英寸。已经包含了这个额外一英寸的 LaTeX 2ε 参数 \paperwidth 和 \paperheight 是通过在 \documentclass 中的纸张大小选项赋值的；在内部用它来计算页边界，从而使文本居中。用户也可以使用这两个参数。

对于文档类 book 或者选项 twoside，每一页上正文的底边恰好处在相同的位置上。而对于其它的类或选项，这就会稍有点儿变化。在前面那两种情形中，常量底边是用内部命令 \flushbottom 来得到的，而变化的底边是用命令 \raggedbottom 生成的。用户也可以用这两个声明来随时改变底边的形式，从而使之与文档类和选项无关。

练习 3.5: 你也可以用上面的参数来改变文本的页面格式。在你练习用文本文件的导言中加入下述指令：

```
\setlength{\textwidth}{13cm}  \setlength{\textheight}{20.5cm}
```

这样输出中的上边界和左边界就会很小。给 \oddsidemargin 和 \topmargin 一新值，以改变这点。

注意：不要忘记了在左边和上边有一英寸的额外长度。当你选择参数值 \oddsidemargin 和 \topmargin 时必须考虑到这一点。

练习 3.6: 继续向所用的文件中填加内容，使其当用的是精简页面格式时也不只两个满页。在导言中加入 \flushbottom，那你就会看到所有页上最后一行是在相同的位置上。

练习 3.7: 从文件中去掉命令 \flushbottom，而选择文档类为

```
\documentclass[twoside]{article}
```

那么现在即使不用 \flushbottom, 最后一行也在相同的位置上。另一方面，奇数页的左边界与偶数页的右边界可能不一致。可以用 \evensidemargin 声明来校正。

§3.2.5 单双列页面

文档类选项 twocolumn 可以使整篇文档的每页都是双列形式显式。而默认值是每页为单列形式。如果要使某一单独页以双列或单列形式排版，可以用下面的声明来达此目的：

\twocolumn[文本]

它中止当前页，用两列形式排版后面的每一页。这里的不省略文本是用一列形式显示在页面顶部的，所用的是整页的宽度。

\onecolumn

中止当前的双列页面形式，用单列形式排版后续的每一页。

选项 twocolumn 会自动地改变某些相对于单列格式时的样式参数，例如缩进量。而对于命令 \twocolumn，就不会进行这种操作。如果你想要进行这种改变，那就必须用相应的 \setlength 声明来实现这一点。如果整篇文档都是双列格式，那么类选项不失为一种最好的方案。

还有一个页面样式参数 \columnwidth，它表示一列文本的宽度。对于单列文本，它与 \textwidth 是相同的，当选择的是 twocolumn 时，LaTeX就会根据 \textwidth 和 \columnsep 的值来计算出其值。文档作者不应该改变这个值，但可以利用它，如果画一个与列文本同宽的标尺。

§3.3 文档中的部分

每一篇文档都要被分成章、节、小节等等。在结尾处还可能有附录，开头处有标题页，目录和摘要等等。在LaTeX中有许多可用的命令，把用户从这种复杂的格式考虑中解脱出来。除此而外，还可以进行连续的标题编号和小编号。甚至目录也可以自动生成和显示出来。

©TEXGuru, August 15, 1999

```
\title{
   How to Write DVI Drivers}

\author{
   Helmut Kopka\thanks{Tel.
     05556--401--451 FRG}\\
   Max--Planck--Institut\\
   f\"ur Aeronomie
 \and
   Phillip G. Hardy
     \thanks{Tel.
     319--824--7134 USA}\\
   University\\of Iowa}

\maketitle
```

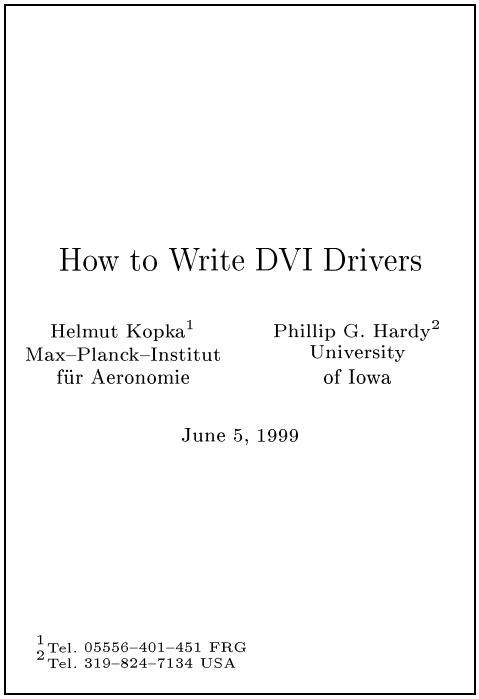


图 3.2: 标题页示例及其结果

某些章节命令的作用与所选用的文档类有关，并不是在所有的类中都可以用所有的命令。

§3.3.1 标题页

标题页可以用如下环境来生成没有格式的形式：

`\begin{titlepage}` 标题页文本 `\end{titlepage}`

或者用如下命令利用 LaTeX已预定义好的格式：

`\title{` 标题文本 `}`

`\author{` 作者名与地址 `}`

`\date{` 日期文本 `}`

在标题页的 LaTeX标准格式中，所有内容将来都是以居中的形式出现的。如果标题太长，也会自动断行。作者自己可以用命令 `\\` 来选择断行点，也就是说所用形式为 `\title{...\\...\\...}` 。

如果有几个作者，其姓名可以用 `\and` 彼此分开，如

`\author{G. Smith \and J. Jones}`

这些姓名将在会打印在一行上。而

`\author{` 作者 1`\\` 学术团体 1`\\` 地址 1

`\and` 作者 2`\\` 学术团体 2`\\` 地址 2`}`

就会把 作者 1，学术团体 1，地址 1 和 作者 2，学术团体 2，地址 2 分成两部分，分别一行行的居中排列。而且这相邻的两块在标题页上也是居中的。

如果不想把作者名左右相邻排版，也可以把它们上下排列，这时只要用 \\ 取代 \and 就可以了。此时，可以用紧接 \\ 后的可以省略的长度定义 [距离] 来调整竖直距离。

如果没有给出 \date 命令，就会自动在标题页的作者项后加上当前日期。另一方面，命令 \date{日期文本} 就会在出现日期的地方显示 日期文本 。这个地方可以插入任何文本，也可以用断行命令 \\ 以插入多行文本。

命令

\thanks{ 脚注文本 }

可以出现在 title, author 和 date 文本的任何地方。它会在命令出现的地方加上一个标志，而在标题页上以脚注的形式排版脚注文本。

要想以出现在 \title, \author, \date 和 \thanks 中的条目生成标题页，就必须用命令

\maketitle

标题页本身没有页码，后续文档第一页的页码为 1 。（对于 book，页码是由 3.3.5 节中的特殊命令控制的。）只有在 book 和 report 中，标题页才会单独占一页。而在 article 中，当使用了命令 \maketitle 时，就会以来自于 \title, \author 以及可能有的 \date 和 \thanks 中的条目在第一页上创建居中标题头。如果用了 titlepage 文档选项，那么即使在 article 类中也会让标题位于单独一页上。

图 3.2 是一个 LaTeX标准格式的标题页示例。注意由于在标题页的定义中没有出现 \date 命令，所以当前日期是自动加上的。可以用这条命令在出现日期的地方显示任何所需要的文本。

对于在 titlepage 环境中的没有格式的标题页，命令 \title 和 \author 是没有作用的，整个标题页的设计来自于作者在这个环境中的定义。此时我们可以利用第 4 章中的所有构造命令。在这种情形里，当结束 titlepage 标题页环境时，就会实现标题页的排版，因此也不需用命令 \maketitle 。

练习 3.8: 删掉练习 3.5–3.7 中的改变页面格式的声明。向练习用的文本文件中加入标题 'Exercises'，作者就是你自己的姓名和地址，以及日期条目为 '地方，日期' 形式的标题头。为此，在 \begin{document} 命令后输入如下命令：

\title{Exercises} \author{ 你的姓名 \\ 你的地址 }

\date{ 你所在的城市， \today} \maketitle

要确保你所用的文档类是 article 。当打印出文档后，把文档类命令改为

\documentclass[titlepage]{article}

这样就可以把标题信息显示在单独一页上，而不是原来那种标题头形式。

©TEXGuru, August 15, 1999

在这些命令的每一行前面加上注释符号 % 以取消它们。要这种方法你可以避免在下面的练习中还会得到标题页，当然可以把 % 去掉，从而很容易地重新激活这些命令。

§3.3.2 摘要

可以用下面的命令生成摘要：

\begin{abstract} 摘要文本 \end{abstract}

在文档类 report 中, 摘要是位于单独一页上的, 而且有页码; 在 article 中, 摘要是紧接标题头位于第一页上，除非选择了文档类选项 titlepage，在这种情形中，摘要也是单独占一页的。在文档类 book 中没有摘要。

§3.3.3 章节

下面的命令可以用来自动生成有序的章节：

```
\part     \chapter     \subsection      \paragraph
          \section     \subsubsection   \subparagraph
```

除 \part 外, 其它命令形成一个章节序列。在文档类 book 和 report 中, 最高的章节层次是 \chapter 。章用命令 \section 分为节，它又可以进一步用 \subsection 等等再细分。在文档类 article 中，序列是由 \section 开始的，因为这时 \chapter 是不可用的。

所有这些命令的语法是

\ 章节命令 [短标题]{ 标题 } 或者

\ 章节命令 *{ 标题 }

在第一种情形中, 把序列中下一个编号赋给章节, 并把这个编号同来自于 “标题” 的文本一起做为章节标题显示出来。 “短标题” 文本是用于目录（ 3.4 节 ）和页眉（假定已选择了 headings 页面样式 ）。如果没有这部分文本，就假定其与 “标题” 是一样的。除非这里的标题太长，不适用出现在上述地方，否则通常就不出现这部分文本。

在第二种（ *- 形式 ）情形中，不会打印出节号，而且不会在目录表中列出该项（见 3.4.3 节中的例外)。

章节标题的大小和编号的长度与章节命令在序列中的位置有关。对于文档类 article， \section 命令就生成单个数字（如 7 ），而 \subsection 命令生成两个数字，中间用句号分开这两部分（如 7.3 ），其它依次类推。

在文档类 book 和 report 中，用 \chapter 命令给出的章标题是单个数字为编号，而 \section 得到的则是两个数字，依次类推。而且， \chapter 命令总是开始新页，并在章节标题上面打印 **Chapter n**，这里的 **n** 表示当前章编号。在本书当前位置，我们是在 *Chapter3, Section 3.3, Subsection 3.3.3* 。

对于每个章节命令，都有一个内部记数器，每当调用一次命令，对应记

数器就会增一，而当调用下一个更高级的章节命令时，其就会被重置为 0 。这些记数器不受 *- 形式的影响，这个事实在如下情况中会造成一种困难：那就是当标准形式和 *- 形式混用，而 *- 形式命令在序列中的位置要比标准形式的高。然而，如果 *- 形式总是低于标准形式，这就不会产生什么问题。下面的序列

```
\section ... \subsection ... \subsubsection* ...
```

的结果就是 \section 和 \subsection 有编号，而 \subsubsection 没有编号。

章节命令 \part 是一个特殊情形，它对其它命令的编号没有影响。

章节的自动编号就意味着在写作的时候，可能不需要知道编号。作者没有必要按最终的顺序进行创作，也可以加进或去掉一些章节。如果作者在正文中想引用某一章节的编号，那么这就需要一种不是直接输入编号的机制。将在 8.3.1 节描述 LaTeX的交叉引用系统，它利用如下两条基本命令完成这个任务：

```
\label{ 名称 }      \ref{ 名称 }
```

前一个给章节编号一个名称或关键词，而后一个则是用在正文中，以引用章节的编号。关键词名称可以是字母、数字或符号的任意组合。例如，本书中在 8.3.1 节开头使用了命令 \label{sec:xref} ，因此上面这句话中就只需包含进命令 \ref{sec:xref} 。

还有一个引用命令，那就是 \pageref ，它显示对应的 \label 命令所在那页的页码。

引用命令还可以用在其它许多地方，前提条件只要那些项的编号是自动生成，如图，表和公式。

每一个章节命令都有一个层次号，如 \section 总是第 1 层，\subsection 为第 2 层，. . . ，\subparagraph 为第 5 层。在文档类 article 中，\part 为第 0 层，而在 book 和 report 类中，\part 为第 −1 层，\chapter 成为第 0 层。章节编号向下进行的层次由 secnumdepth 决定。对于 book 和 report ，其值为 2 ，而对于 article ，其值为 3 。这也就是说对于 book 和 report ，章节编号只进行到 \subsection 层次，而对于 article ，则是到 \subsubsection 。

为了拓展（或减小）章节编号的层次，那就必须改变 secnumdepth 的值。这可以用命令

```
\setcounter{secnumdepth}{ 数 }
```

来实现。（在 7.1 节中对 \setcounter 命令进行了解释。）在 article 中，secnumdepth 可以取 −1 到 5 之间的值，而对于 book 和 report ，可取 −2 到 5 之间的值。取最小值意味着禁止所有的章节编号。

在文档中也可以用如下命令来改变章节命令的初始值：

```
\setcounter{ 章节名称 }{ 数 }
```

这里的 *章节名称* 指的是没有前缀 \ 的章节命令名称。当要用 LaTeX处理某单个章节时，这个命令是相当有用的。例如，

```
\setcounter{chapter}{2}
```

就把 \chapter 记数器设为 2 。当下一次调用 \chapter 时，其值会增 1 ，即得到 **Chapter 3** 。

有时候需要改变章节标题的字体大小和样式。这可以用在 4.1.2–4.1.6 节中描述的用于章节标题文本的声明达此目的。例如，

```
\section*{\Large\slshape Larger Slanted Font}
```

就会以 \Large 尺寸和 \slshape 样式打印出节标题 'Larger Slanted Font' 。我们不推荐使用这种抛弃标准字体的做法，因为这些修改只对标题文本有作用，而对其前面的编号没有影响。

§3.3.4 附录

可以用命令 \appendix 来加进附录。这一命令具有重设 article 中的节记数器和 book 与 report 中章记数器的作用, 而且它把这些章节命令的编号形式从数字变为大写字母 A,B, ... 。另外, 还用单词 'Appendix' 代替 'Chapter' , 因此后续章节的标题前缀变为 'Appendix A', 'Appendix B', 等等。低层章节命令的编号中用字母取代了章节编号，如 A.2.1 。另外一种方法是，附录也可以包含在如下一个环境中：

```
\begin{appendix}
附录文本
\end{appendix}
```

§3.3.5 书的结构

为了简化书的结构，在 book 类中提供了命令

2ε `\frontmatter`

前言，目录

2ε `\mainmatter`

正文的主体

2ε `\backmatter`

参考文献，索引，版本记录

\frontmatter 命令把页码切换为罗马数字格式，并且不对章进行编号；另一条命令 \mainmatter 把页码重设为阿拉伯数字 1 ，并激活对章的编号；而在 \backmatter 中又再次停止对章的编号。

练习 3.9: 在你练习用的文本开头插入命令 \section{Title A} ，在靠近中间的某一适当的地方插入 \section{Title B} ，这里你要用相应的文本取代

Title A 和 Title B 。然后在适当的地方插入合适的 \subsection 命令。去掉练习 3.3 中加入的下列的命令：

```
\pagestyle{myheadings}  \markright{Exercises}
\pagenumbering{Roman}
```

打印生成结果。

练习 3.10: 在第一个 \section 命令前加上命令 \chapter{Chapter title}，这里的 Chapter title 可以取你自己喜欢的任何内容做为章的标题。把文档类命令改为 \documentclass[twoside]{report}，并在导言中调用页面样式命令 \pagestyle{headings} 。注意在标题和页眉中的章节命令的两面效果。把这里的结果与 3.2.1 节中的表格做比较。

练习 3.11: 把章命令改为

```
\chapter[Short form]{Chapter title}
```

并给出章标题的适当缩减形式。那么原来出现完整章标题的页眉，现在取代的是短标题。

§3.4 目录表

§3.4.1 自动条目

LaTeX可以自动为整篇文档准备和打印一个目录表。这个表包含章节号和相应的在章节命令标准形式中的标题，以及章节起始页码。出现在目录表中的章节深度可以在导言中用命令

```
\setcounter{tocdepth}{ 数 }
```

来设定。这里的 数 与上面描述的 secnumdepth 记数器中的意义相同，而自动分章节的最大层次是固定的。到哪一层次的章节可以包含在目录表中，与自动章节编号中的是一样的，也就是说，对 book 和 report 到 \subsection，对 article 到 \subsubsection 。

§3.4.2 显示目录表

目录表的生成和显示是用下面的命令实现的：

```
\tableofcontents
```

把这条命令放在希望目录所位于的地方，通常就放在标题页和摘要的后面。

这就导致了一个两难的处境，因为目录表的信息是显示在靠文档开头的地方，而这些信息直到文档结束才可能全知道。 LaTeX是如下解决这个问题的：当文录第一次被处理时，并没有包含进任何目录表信息，而是由 LaTeX打开一个与文档文件同名，而后缀为 .toc 的新文件；在后面的处理过程中，把目录表所需的条目写到这个文件中。

©TEXGuru, August 15, 1999

当下次对这个文档运行 LaTeX时，\tableofcontents 命令使得文件文档名 .toc 被读入，从而打印出目录表。当继续后面的处理时，如果在运行后前一次后做了很大改变，那么 .toc 文件会被更新。也就是说打印出来的目录表总是对应于前一次文档的。正是由于这个原因，对于最后的文档，应该运行两次 LaTeX 。

§3.4.3 其它条目

*- 形式的章节命令不会自动加入到目录表中。为了把它或其它条目插入到目录表中，可以用命令：

\addcontentsline{toc}{ 章节名称 }{ 条目文本 }

\addtocontents{toc}{ 条目文本 }

用第一条命令，就会形成目录表格式的相应条目，其中 section 标题头要比 chapter 的向里缩进一些，但比 subsection 缩进的少。这是由章节名称参数决定的，取值为章节命令没有了前缀 \ 字符（如 section ）。条目文本与页码一起插入到目录表中。如果想同标准形式那样也有章节编号，那么条目文本必须具有形式 \protect\numberline{ 章节名称 }{ 文本 } 。

\addtocontents 命令把任何所希望的命令或文本加入到 .toc 文件中。它可以是一条格式化命令，如 \protect\newpage ，当显示目录表时这些命令会发挥作用。

§3.4.4 其它列表

除了目录表，图与表的清单列表也可以由 LaTeX自动生成和显示出来。生成这些列表的命令为：

\listoffigures	读和 / 或生成文件 .lof
\listoftables	读和 / 或生成文件 .lot

在这些列表中的条目是根据 figure 和 table 环境中的 \caption 命令自动生成的（见 6.6.4 节）。其它条目可以用与目录表相同的命令生成，它们的一般形式为：

\addcontentsline{ 类型 }{ 格式 }{ 条目 }

\addtocontents{ 类型 }{ 条目 }

这里的 类型 代表 toc（目录表）、 lof（图的列表）和 lot（表格的列表）三种类型中的一种。 格式 参数就是上面描述的目录表中的章节名称，或者图列表中的 figure ，以及表格列表中的 table 。 条目 参数表示要插入相应文件中的文本。

练习 3.12: 在练习文件中，在抑制了标题页命令后插入命令

\pagenumbering{roman}

\tableofcontents \newpage

```
\pagenumbering{arabic}
```

用 LaTeX处理两次练习文件，并打印出第二次的结果。在做下面的练习之前，用 % 符号去掉这些命令。

注意：如果章节标题中有命令，那么应该前缀 `\protect` 命令，以保证它们被安全地传送到相应文件中。

§3.5 精调正文

§3.5.1 单词与字符的距离安排

在单词和字符之间的距离，通常都是由 TeX自动安排的，它不但利用字符的自然宽度，而且还考虑特定字符组合的变化。例如，如果 A 后接 V，结果并不是 AV，而是 AV；也就是说为了好看，它们彼此稍移近了一点。在一行中单词之间的距离是一致的，而且距离的选择保证左右边界与页边是对齐的。这也称为左右调整。 TeX也尽力使不同行中单词间的距离尽可能一样。

由标点符号结尾的单词，其后要给出一个额外的空档，具体大小要看到底是哪种符号了：接在句号 '.' 或惊叹号 '!' 后面的距离就要比接在逗号后面的大。这相应于英文的排版规则：在句子之间应有多一点的空档。在某些情形中，自动安排的结果可能不会令人满意，这样就可以按下面几节中的方法进行重新定义。

句子结束与句号

TeX把一个接在小写字母后的句号认为是句子的结束，这时就会插入额外的单词间距。这样就会与缩写产生混淆，如 i. e., Prof. Jones 或 Phys. Rev.，此时需要的是正常单词间距。这时可以用字符 ~ 或 \␣ 取代通常的空格。（字符 ␣ 只是为了表示一个空格，否则根本看不到它。）这两种方法都是插入通常的单词间距；而且 ~ 禁止在这一点断行。所以上面的例子应该输入 `i.~e.`，`Prof.~Jones` 和 `Phys.\ Rev.`，以得到输出 i. e., Prof. Jones 和 Phys. Rev.，这时的间距是很正常的，而且必要时也强迫其中的某些相邻字符必须在一行上。在第三种情形中，把 Phys. 和 Rev. 放在两行上并没有什么不妥。

紧接在大写字母后的句号并不认为是句子的结束，而只认为是一个缩写。可如果它真的是结束一个句子，那就需要在句子前面加上 `\@`，以得到额外的间距。例如，句子 'this sentence ends with NASA.' 就应该输入为 `this is sentence ends with NASA\@.`

法语间距

可以用命令 `\frenchspacing` 来告诉 TeX不必增加某些情形中的额外单词间距，当再用 `\nonfrenchspacing` 命令时，其又会恢复到原来的方式。对

前一种情形，命令 `\@` 是被忽略的，可以不用。

字符组合“‘和’”

用命令 `\,` 可以生成一个很小距离。这一点是有用的，例如，当双引号“和”与单引号‘和’在一起时，可以用 `\,` 把它们分开。例如，文本 ``‘\,‘Beginning’ and ‘End’\,’’`` 的输出为“‘Beginning’ and ‘End’”。

倾斜校正

当字样从斜体（意大利斜体或 slanted）变到竖直字样时，由于最后一个字符可以斜穿进接下来的空档中，从而使得这个距离显得比原来要小。因此必须在这儿加上根据不同字符而不同的额外间距（d 的要比 a 的大）。TEX用命令 `\/` 来加进这个额外间距，称之为倾斜校正。当一个倾斜字母后接一个竖直字母时，总要用这种校正，例如 `{\slshape slanted\/} spacing` 可以得到 *slanted* spacing。当倾斜字母后接逗号或句号时，这种校正可以不加。

LATEX 2ε的字体命令 `\textsl` 和 `\textit`（4.1.4 节）会自动加上倾斜校正，这也是它比字体声明 `\slshape` 和 `\itshape` 好用的另一个优点。例如，不要用 `{\slshape slanted\/}`，而应该用 `\textsl{slanted}`。然而，有时候我们可能希望没有这种校正，那么可以在倾斜字母后面用命令 `\nocorr` 以达此目标。

取消连写

`\/` 命令也可以用来禁止连写，所谓连写，就是特定的字母组合作为一个字符印刷。文本 `shelf\/ful` 的结果是 shelfful，而不是 shelfful。在 2.5.8 节中已提到，TEX把字母组合 `ff`, `fi`, `fl`, `ffi` 和 `ffl` 处理成连写。输入 `f\/f\/i`，可以得到 ffi，而不是 ffi 连写。

`\/` 命令也可以用来取消特定字母组合之间的距离减少，如 AV 和 Te：

`A\/V` AV　`T\/e` Te　而不是 AV　Te。

插入任意距离

可以用下面的命令在文本间插入任意大小的距离：

`\hspace{` *距离* `}`

`\hspace*{` *距离* `}`

这里的 *距离* 就是所要指定的间隔长度，例如 `1.5cm` 或 `3em`。（注意一个 `em` 就是当前字样中字母 M 的宽度。）

这两条命令在调用点和接下来要显示的对象之间插入宽度等于上述距离的空白。用标准形式（没有 *），可以使得若间隔位于两行之间时，就去掉这个的空档，这就如同在一行的开头，空格要去掉一样。而另一方面，*- 形式不管任何情况，都会插入空白。

这里定义的距离也可以是负数，这样该命令就如同退格，在另一个字符上打印新的字符。

在这条命令前后的空格仍然有作用：

`This is\hspace{1cm}1cm`	This is	1cm
`This is \hspace{1cm}1cm`	This is	1cm
`This is \hspace{1cm} 1cm`	This is	1cm

命令 `\hfill` 就是 `\hspace{\fill}` 的缩略形式（见 2.4.2 节）。它会在其两边的文本之间插入足够大的空白，从而使它们分别靠近左右页边。`Left \hfill Right` 的输出就是

Left Right

在一行上有多个 `\hfill`，那个它们每个都会插入相同宽度的空白，从而使这一行变得左右协调。例如，输入 `Left \hfill Center \hfill Right` 的结果为

Left Center Right

如果 `\hfill` 位于一行的开头，根据 `\hspace` 标准形式的行为准则，这个空白是被去掉的。如果确实需要在一行的开头或结尾加上一个橡皮空白，那就必须用 `\hspace*{\hfill}`。然而，LaTeX还提供许多命令和环境，以简化许多这种情形的应用（见 4.2.2 节）。

也可以用许多其它的长度固定的水平距离：

`\quad` 和 `\qquad`

命令 `\quad` 插入一个长度等于当前字样尺寸的空白，即若当前为 10pt 字样，则插入 10pt，而 `\qquad` 是其两倍。

插入长度可变的 …… 和 ——— 序列

还有两条命令，其用法同 `\hfill` 完全一样：

`\dotfill` 和 `\hrulefill`

当不想插入空白时，可以用这些命令如下插入点或直线：

```
Start \dotfill\ Finish\\
Left \hrulefill\ Center \hrulefill\ Right\\
```

的结果为

Start ..... Finish

Left ——— Center ——— Right

可以在一行上出现 `\hfill`，`\dotfill` 和 `\hrulefill` 的任意组合。如果在同一地方多次使用这种命令，那么相应的填充就单个时的要大很多。

```
Departure \dotfill\dotfill\dotfill\ 8:30 \hfill\hfill
Arrival \hrulefill\ 11:45\\
```

©TEXGuru, August 15, 1999

结果为

Departure ..... 8:30 Arrival ——— 11:45

§3.5.2 断行

把文本断成行的操作是由 TEX和 LATEX自动进行的。然而，也有这样的时刻，你想强迫或希望在某地方断行，或不想在某地方断行。

命令 \\

可以用命令 \\ 来得到一个有或没有额外行距的新行。该命令的语法为

\\[距离]

*[距离]

这里的可省参数 距离 是一个长度，它用来定义增加多少额外的行间距。如果此时需要开始一个新页，那就不加这个额外间距，新页开头为下面的文本行。 *- 形式禁止在两行之间分页。

用 *[10cm] ，当前行就会结束，在这行与后面文本加上一个 10cm 的竖直距离，而且这两行文本必须在同一页上。如果需要一个分页，那就会在当前行前面进行，这样当前行， 10cm 的竖直距离，以及后面的文本就位于新页上。

命令 \newline 与没有可省参数的 \\ 命令是一样的。也就是说，新行之前没有额外的空白，而且此处也可以分页。

这两条命令都可以用在段落中间，除此而外，它们再没有其它意义。

其它的断行命令

命令 \linebreak 用来鼓励或强迫在文本中某点断行。它的形式为

\linebreak[数]

这里的 数 为可省参数，它是一个介于 0 到 4 之间的数，用来表示断行的重要性。这条命令鼓励断行，越大的数，鼓励的力度就越大。取 0 值表示允许在本来不会断行的点处（如一个单词的中间）可以断行，而 4 意味着必须断行，它等同于没有参数的 \linebreak 。这条命令与 \\ 或 \newline 的区别在于当前行要进行左右调整，增加单词间距，以使文本充满该行。而对于 \\ 和 \newline ，当前行用空白填充最后一个单词后的空白，单词间距保持正常值不变。

与之相反的命令为

\nolinebreak[数]

它不鼓励在给定点断行，这里的 数 表示不鼓励的力度。而且没有参数的命令 \nolinebreak 同 \nolinebreak[4] 有一样的效果，即在这里绝对不可能进行断行。

另外一种使文本呆在一行上的方法是用命令 \mbox{文本 }。对于类似于 'Voyager-1' 这样的表达式，如果想在连字符处禁止断行，用这个命令就是非常方便的。

§3.5.3 段落间距

段落间的正常间距是由长度 \parskip（ 3.2.3 节）设置的，可以用命令 \setlength 来改变其初始值：

\setlength{\parskip}{ 距离 }

也可以用如下命令在某一特定段落间加上额外的竖直距离：

\vspace{ 距离 }

\vspace*{ 距离 }

即使在此处开始一个新页，或者命令位于新页的开头， *- 形式也要加上这个额外空档，而同样对这两种情形，标准形式禁止加上额外竖直距离。

如果在一个段落中间用这两条命令，那就在当前行后面加上额外间距，而且按能常那样对当前行进行左右调整。

距离 参数可以为负数，这样可以抬高后面文本的相应于正常打印的位置。

命令 \vfill 是 \vspace{\fill}（见 2.4.2 节）的缩略形式。其等价于水平间距的 \hfill ，它会插入足够的竖直空白，以使得文本的顶部和底部与页面的顶部和底部对齐。对于同时出现多个 \hfill 的解释，也同样适用于 \vfill 。如果这条命令位于一页的开头，它没有任何作用，这同标准形式 \vspace{\fill} 一样。如果想在页面底部加上一个橡皮间距，那就必须用 *- 形式 \vspace*{\fill} 了。

其它的用于增加段落间距的命令是

\bigskip \medskip \smallskip

这些命令根据在文档类中声明的字体尺寸来确定增加的竖直距离。

如果不用空行，也可以用命令 \par 来表明一个段落的结束。

§3.5.4 段落的缩进

段落缩进的大小是由长度 \parindent（见 3.2.3 节）决定的，可以用命令 \setlength 改变它的值：

\setlength{\parindent}{ 距离 }

这样每段的第一行左边都有一个宽度为上面 距离 参数指定的空白。如果想取消某一段落的缩进，或者如果没有缩进（如一节的第一段）时想强迫出现缩进时，可以用下面的命令做到：

\noindent 和 \indent

要把它们放在段落的开始，这样它们才会发挥作用。

§3.5.5 分页

在 TEX和 LATEX中，同自动断行一样，把文本分成页的操作也是自动进行的。同样的，也可以影响程序确定何处为分页点。

正常分页

命令

```
\pagebreak[ 数 ]
\nopagebreak[ 数 ]
```

等价于断行时的 \linebreak 和 \nolinebreak 。如果 \pagebreak 位于两段之间，那就会在此处进行分页。如果它位于一段的中间，那么就会在当前行完成后进行分页。同样要对该行进行左右调整。

命令 \nopagebreak 具有相反的作用：在两段之间，它禁止在这儿分页；在一段中间，那它禁止可能在当前行后面出现的分页。

界于 0 到 4 之间的可省略参数表达了鼓励或不鼓励分页的力度。而且 \linebreak 要做更多的事：除了把断开的行增加单词间距，进行左右调整，还要增加行间距，以使页面顶部和底部分布一致。

要在一页中间结束该页，用空白填充余下空间，再开始一个新页的命令是

```
\newpage
```

它等价于分页中的 \newline 命令。

练习 3.13: 在练习文件中适当的一个地方或两个地方插入 \newpage 命令。在输出中，将会看到页面下面是空的。现在去掉这些命令，在一个自然分页点后面一行或两行处插入一个 \pagebreak 命令。处理完后，把命令移到正常分页点前面几行处。看看这几种修改时的结果.

有图表时的分页

如果文本中有表格，图画或者为插图预留的空间，它们会被插入在与相应于命令出现的地方，前提条件是在当前页上要有足够的空间。如果没有足够的空间，就会继续正文，而把图表保留到下一页上。

命令

```
\clearpage
```

同 \newpage 命令一样结束当前页，而且用一页或多页来输出没有打印的图表。（请见 6.6 节。）

两列模式时的分页

如果选择了文档类选项 twocolumn，或者用了命令 \twocolumn，那么命令 \pagebreak 和 \newpage 就会结束当前列，开始一个新列，即把列当做页

处理。另一方面， \clearpage 和 \cleardoublepage （见下面）就是中止当前页，如果必要的话，插入一个空列。

两面模式时的分页

当选择了文档选项 twoside 时，还可以用一个分页命令：

\cleardoublepages

它的作用同 \clearpage 一样（当前页被中止，所有未打印的图表输出在后面的页上），而且接下来的页应是奇数页。如果必要的话，由此会得到一个具有偶数页码的空页。

有限制的分页

声明 \samepage 可以使得分页只会发生在段落之间，显示公式或文本的前后，一个章节标题的前后，或者列表环境中的 \item（第一个除外）之间。如果想在正文其它地方进行分页，那就必须用命令 \pagebreak 了。

实际上这一功能的作用并不会同期望的那样好，正是由于这个原因，LaTeX 2ε提供了一些新方法，以控制不必要的分页。只是为了与原来文档兼容，才继续保留了 \samepage 命令。

控制分页

LaTeX 2ε用如下命令提供了稍微增加当前页高度的可能性：

2ε \enlargethispage{ 尺寸 }

2ε \enlargethispage*{ 尺寸 }

这两命令把定义中的 尺寸 只加到当前页的 \textheight 上。有时候只需要稍微一点儿的调整，就可以避免一个糟糕的分页。命令的 *- 形式在当前页上无论如何也要调整行距，以最大化文本高度。

分页的其它方法

在 3.2.4 节中已提到，在文档类 book 或选项 twoside 中，在每一页上，页面底边的位置是一样的，而在其它情形中则不然。这意味着在这两种情形里，将需要在适当的位置（如落段之间）插入行间距。

这种情形是由于在 book 和 twoside 中，有声明 \flushbottom，而在所有其它情形中，则是 \raggedbottom 声明造成的。它们最初是在内部类和选项的定义中设置的，但用户也可以在导言中或正文的其它地方用这些命令。除非遇到相反的命令，或当前环境结束，否则该命令一致有效。

通常是不可能用选择页面长度 \textheight（ 3.2.4 节）来在一页上放更多的文本。然而，如果确实需要多挤进一行或两行文本，那就必须为它们清除竖直间距。这可以用放在适当点的带负长度的 \vspace 或 \\ 命令来做到。最佳地方是在显示，枚举，列表，表格或插图的前后（见第 4 章）。

©TEXGuru, August 15, 1999

我们在这里不会去详细讲解分页的内部规则，只提及下面一点，对那些不鼓励在此分页的点，TEX要加上所谓的罚点（对应于命令 \penalty ），而负的罚点则意味着鼓励分页。段落中的每行都对应着一个罚点，比如说是 10 个罚点（ \linepenalty=10 ），这样就可以使得在段落中间分页比段落之间分页要难。

段落的第一行和最后一行给定 150 个罚点（也就是指 \clubpenalty=150 和 \widowpenalty=150 ），这样以不鼓励（但没有禁止）在一页上只留下一段的一行。

因此，做为一种手段，我们也可以通过修改罚点值，以获取所需的分页。例如，若取 \clubpenalty=450 ，那么在段落中第一行后分页就要比平常时的难上三倍。把罚点值取为 10000 或更多，就会使得绝对不会在此处分页。然而对罚点的修改，只应该用在其它方法都行不通的时候，因为这破坏了与其它部分文档之间的平衡，从而可能引起更多古怪的分页。

§3.6 断词

当要对一行进行左右调整时，有时候会出现无法在单词之间断行的情形，因为这样做要么会使单词挤得很紧，要么彼此之间离得很远。这样就需要分开一个单词。这一重大的任务是由 TEX（即 LATEX的本质基础）来完成的，它用的断词算法对英语（绝大多数作者用的都是这种语言）文本（绝大多数情形下）处理的很好，然而，它也有出错的时候，这就需要人来干预它了。

如果通常的 TEX/LATEX用于其它语言，或者在英文中出现了外文单词，也很有可能出现不正确的断词。（在 3.6.5, 3.6.6 节和 D.1 节有关于 LATEX与其它语言共用的更多讨论。）在这种情况下，那也必须用如下的方法重定义 TEX的断词算法。

§3.6.1 人工断词

最简单地纠正一个被错误断开的单词的方法就是在这个单词中间的恰当位置上插入命令 \- 。例如对于单词 manuscript ，根本不可能把它断开，因此如果在断行时它造成了一些问题，那可把它写成 \man\-u\-script 。这就告诉 TEX在必要时可以把它断成 man-uscript 或 manu-script ，而不用顾及通常的规则。

\- 只是使得在指定位置有可能断开；而没有强迫在此处断开。如果作者坚持要在某一点断开某个单词，比如说在 manuscript 的 u 和 s 之间，那可以输入 manu-\linebreak script 来做到。但是，我们并不推荐这种粗暴的做法，因为不管以后对正文有没有修改，这个地方总是有个断行点。

对于英文，单词的拼写即使断开也是不变的，而对于其它语言，这可就

不一定了。例如在德语中，就有一条规则，如果 ck 被分开，它就变成 k-k 。在 TEX中可以用一般断开命令来做到这一点：

```
\discretionary{ 前 }{ 后 }{ 无 }
```

这里的 *前* 与 *后* 都表示字母 (有连字符)，它们指的是如果要断开的话，在断点两边的字母，而 *无* 则是没有断开时的正常写法。因此 Boris Becker 的姓名应该写成

```
Boris Be\discretionary{k-}{k}{ck}er
```

只有在特殊情形下才需要这种输入。顺带提一下， `\-` 命令的定义就是

```
\discretionary{-}{}{}
```

。

§3.6.2 断法列表

断开不正确，但是又经常在正文中出现的单词可以在导言中放进一个异常列表，以避免每次费事地插入 `\-` ：

```
\hyphenation{ 列表 }
```

这里的 *列表* 就是一组单词，用空格或换行分开，而且用连字符表示可允许的断点。例如，

```
\hyphenation{man-u-script com-pu-ter gym-na-sium
             coun-try-man re-sus-ci-tate ...}
```

这个列表中可以包含通常的从 a 到 z 的所有字母，但不能出现特殊字殊或重音。

§3.6.3 禁止断词

另外一种避免糟糕断词的做法就是为至少一两段关闭断词。实际上命令

```
\begin{sloppypar} 段落文本 \end{sloppypar}
```

并没有禁止断词，只是允许更大的单词间隔，而不会给出警告信息。这就意味着实际上所有行都是在单词间断开的。也可以在导言中或当前环境中加入命令 `\sloppy` 来减少整篇文档或当前环境中的被断开的单词数目。当行相当窄时，推荐使用这种方法。

当命令 `\sloppy` 起作用时，那么可以用如下环境来暂时重新打开断词：

```
\begin{fussypar} 段落文本 \end{fussypar}
```

在环境中用命令 `\fussy` 也可以得到相同的效果。

§3.6.4 行宽与断词

现在有必要在这里补充一些关于行宽与断词的注解。

当 TEX处理到一段末尾时，它就尝试在考虑 3.5.1 节中提到的间距要求的前提下，通过在单词间断行，把文本组织成长度相等的行；这也说是，不会出现断词。如果这种尝试行不通，那它就开始尝试在单词中间断开。当行宽很大，或者单词的平均长度很小时，在单词之间断行是相对比较容易的。

而且，对于给定的行宽，如果字体尺寸是小的，那么被断开的单词数目也会少些。

若行宽很窄,即使可以在单词中断开,可能左右调整也很困难,这样 TEX就需要插入比正常情况下所允许的要多的单词间距。要做到这一点，可以用上面给出的 sloppypar 环境或 \sloppy 命令。这样可以进行调整，但是代价是在单词间加进了不可接受的间距。在这两种情形中， TEX都会打印出一条警告信息,要么是 Overfull \hbox(不可能左右对齐, 一个单词将向右边突出) 或 Underfull \hbox (单词间距太大)。如果无法容忍这两种情形，那就必须对正文进行调整，利用命令 \linebreak 和 \hfill 可能会强迫出现适当的断词点。

§3.6.5 其它与断词有关的内容

TEX需要用来进行断词的信息保存在一个 hyphen.tex 的内部文件中，其中包含一组供特殊断词算法所用的字母组合。这个列表是基于所考虑语言的统计研究结果的。这个文件中也包含一些例外，算法不能正确地对这些单词进行断开。 TEX首先在这个例外列表中搜索，根据在其中可以找到的模式断词；如果在其中没有找到，那它就用它的算法来断词。对于其它语言，那就必须给出不同的字母组合列表和例外列表了。

在文件 hyphen.tex 的尾部可以找到例外列表,其由前面给出的命令 \hyphenation 组成。这个列表中的每一项都可由文本编辑器展开。在一个大型计算中心，只有被授权的 TEX用户才可以对它进行修改。而在 PC 机上，用户必须查阅 TEX安装手册，看看 hyphen.tex 文件在哪儿。

每次对断词文件进行改变，对就必须生成一个新的格式文件。这是一种针对于 LATEX的主要宏定义文件，它以可快速输入的紧致方式存贮。关于安装 LATEX 2ε的更多信息见 D.4 节。

顺便提一下，你可以用下面的命令在屏幕上查看 TEX断词算法的结果：

```
\showhyphens{ 单词表 }
```

这会打印出 TEX对 单词表 中所有单词的断开结果。例如，给出：

```
\showhyphens{interrelations penumbra summation}
```

那在屏幕上就会显示出

```
in-ter-re-la-tions penum-bra sum-ma-tion
```

§3.6.6 多语言文本中的断词

在 TEX2.99 或更低的版本中，在一个 plain.fmt 或 lplain.fmt 中只允许有一组用于断词的字母组合。这些版本的 LATEX程充可适用于英文或者另一种语言。在一个文档中不可能把多种语言的断词算法混用。

从 TEX的 3.0 版本起，在格式中可以包含多个断开列表，这样就可以在一

个文档中切换到不同的断开框架中,所使用的就是新的 TEX命令 \language 。这一命令也可以用于与语言相应的改编中，把某些特定的英文单词翻译输出（如 'Contents'），以简化重音和标点符号，以及修改日期命令 \today 的定义。在 C.3.3 和 D.1.4 节给出了多语言和 \language 命令的更多信息。

©TEXGuru, August 15, 1999

第四章　显示文本

有许多方法可用来显示或强调文本：改变字体样式或字体尺寸，居中，缩进，标志段落，等等。LaTeX为我们提供了这些最常见显示形式的命令。

§4.1　改变字体

在排版印刷中，一组特定大小和样式的字母、数字及符号的组合称为一种字体。在LaTeX中用于文本主体的标准字体是一种直立的罗马字体，其由文档开头处的 `\documentclass` 或 `\documentstyle` 声明来定义其平均权重。可用的基本尺寸是 10, 11 或 12pt，分别相应于尺寸选项 `10pt`（默认值），`11pt` 或 `12pt`。（注意一英寸大约等于 72.27 点，而一厘米大约等于 28.45 点。）小括号 () 就占据了等于字体尺寸的高度和深度。

三种标准尺寸的视觉效果相应于这三个数字的比率而言是相当大的：

This is an example of the 10 pt font. ()

And this is the 11pt font for comparison. ()

And finally this is a sample of 12 pt font. ()

§4.1.1　强调

在用打字机打出来的手稿中，强调文本的最简单方法就是用 下划线。在印刷时，制版工人通常要把用下划线的文本转化为斜体。在LaTeX中要从标准文本切换为强调形式，只需要用声明 `\em` 或命令 `\emph`[2ε]。

声明 `\em` 的作用同下面将要讲解的其它字体声明一样：改变当前字体，直到被其它相应的声明取消（它可以是 `\em` 自身），或者当前环境的结束（2.2 节）。也可以只用一对大括号 `{...}` 来创建一个环境。这是一种最简单的强调一小段文本的方法，请见示例：

```
This is the easiest way to {\em emphasize} short ...
```

而在LaTeX 2ε中提供了命令 `\emph`，这是一种强调一两个单词的更合逻辑的方法：

```
A more logical method of \emph{emphasizing} a word ...
```

注意它们的差别，声明的作用是当局部环境用右大括号结束时而终止，而命令只对其包含在大括号内的参数有作用。另外一种更精细的差别就是命令 `\emph` 自动插入倾斜校正（见 3.5.1 节），而在声明 `\em` 中则必须人工加入这一校正。

声明和命令都切换进入一种强调字体。这也就是说如果当前字体是直立的，那就切换为斜体，而如果当前字体是 slanted，则就切换为一种直立字体。

可以嵌套强调，而且其非常易懂：

```
The {\em first}, second, and {\em third font switch}
```

```
The {\em first, {\em second, and {\em third font switch}}}
```

的结果都是 'The *first*, second, and *third font switch*' 。

§4.1.2 字体尺寸的选择

在 LaTeX中可以用下面的声明来改变字体尺寸：

`\tiny`	smallest		
`\scriptsize`	very small	`\Large`	larger
`\footnotesize`	smaller	`\LARGE`	even larger
`\small`	small	`\huge`	still larger
`\normalsize`	normal	`\Huge`	largest
`\large`	large		

上面所有的声明都是相对于在文档类选项中的标准尺寸而言的。在本书中，标准尺寸为 10pt ，这可以用 `\normalsize` 来选择该尺寸。

字体尺寸声明与所有其它声明的行为是一样的：它立即发挥作用，直到被另一个相反的声明所取消，或者当前环境结束。如果在大括号 `{...}` 内调用，声明的作用就只局限于大括号内，这就如同一个无名环境：

```
normal {\large large \Large larger} normal again
```

normal large larger normal again

在 LaTeX2.09 中字体尺寸声明也同时把所有其它字体属性（ 4.1.3 节）重设为默认值，也就是直立罗马平均权重。与之相反，在 LaTeX 2_ε中，这些属性是不变的。试做一比较：

2.09 `\sl slanted {\Large larger}` *slanted* larger

2_ε `\sl slanted {\Large larger}` *slanted larger*

如果在 `\documentclass` 的地方用的是 `\documentstyle` ，那么这些尺寸声明的作用同 LaTeX2.09 中的一样，这就是为了与原来文档兼容才如此设计的。

用上面的命令来改变字体尺寸，也同时自动改变了行间距。对于每一种字体尺寸，都有一个对应的自然行间距 `\baselineskip` 。可以在任何时候改变其值。例如，如果自然行距是 12pt ，命令 `\setlength{\baselineskip}{15pt}` 就会把其增为 15pt 。

在一段结束时有作用的 `\baselineskip` 命令被用来包装整段。这就意味着如果在一段中对 `\baselineskip` 进行了几次修改，那么只会考虑最后那次修改。

每次修改字体尺寸， `\baselineskip` 都会被重设为相应的自然行距。由 `\setlength` 所做的设置也就无效了。

为了创建一个对所有字体尺寸都适用的行间距修改，那就必须用因子 `\baselinestretch` ，其正常值为 1 。实际上真正的行间距是

```
\baselinestretch×\baselineskip
```

这样对所有的字体尺寸就会保持相应的间距。用户可以随时改变其值：

```
\renewcommand{\baselinestretch}{ 因子 }
```

这里的 因子 是一个浮点数。取值 1.5 就意味着把行间距（基线与基线之间的距离）从相应于所有尺寸的自然间距增加了 50% 。

然而， `\baselinestretch` 的新值只有当又进行了一次字体尺寸改变时才会发挥作用。为了在当前字体尺寸中实现一个新值，那就必须切换到一个新的字体尺寸，然后马上切换回来。如果当前字体尺寸为 `\normalsize` ，那么序列

```
\small\normalsize
```

就可以达到所需的效果。可以用任何命令来取代 `\small` 。

并不是在所有的尺寸中都可以用所有的字体样式。如果选择了一种不可能的字体尺寸和样式组合，那么 LaTeX就会发出一个警告信息，并在打印时告诉你用何种字体取代了该字样。

§4.1.3 字体属性 2ε

字体的尺寸只是用来描述这种字体的几种属性中的一个。做为 LaTeX 2ε内部集成的一部分，在新字体选择框架 (NFSS) 中，可以与 8.5 节所描述的那样，只用这些属性来选择字体。然而，对于普通用户，就需要用声明和相应的命令来简化这一过程。

对于由 TeX和 LaTeX提供的计算机现代字体，具有下列的属性和及其对应值：

Family（族）指一般的综合样式。传统印刷中的族有诸如 baskerville, Bodoni, Times Roman, Helvetica 等等的名称。标准 LaTeX 2ε的安装中有如下的声明提供了三种族：

`\rmfamily` 切换（回）到一种罗马字体；

`\ttfamily` 切换到一个打字机 (`typewriter`) 字体；

`\sffamily` 切换到一个 sans serif 字体。

Shape（形状）指的是字体的构成。在标准安装中可以使用的形状声明为

`\upshape` 切换（回）到一种直立字体；

`\itshape` 切换到一种斜体；

`\slshape` 选择一种叫 *slanted* 的字体；

`\scshape` 切换到小形大写字母 (Small Caps) 。

Series（序列）指的是字体的宽度和 / 或权重（黑度）。可用的声体为

`\mdseries` 切换（回）到平均权重；

`\bfseries` 切换到黑体字样字体。

上述并没有穷尽所有的字体属性，但其涵盖了最标准的字体，尤其是计算机现代字体。对于其它字体，特别是 PostScript 字体，还存在其它的属性。详情

请见 8.5 节。

这些声明的用法同其它的一样，通常包含在一对大括号 {...} 内，如 {\scshape Romeo and Juliet}，结果为 Romeo and Juliet 。对于很长一段文本，可以用相应的环境：

\begin{ 字体样式 }... 新字体中的文本 ...\end{ 字体样式 }

这样的开始结束切换是非常明了的。其中的 字体样式 ，可以上面的字体命令，只要去掉 \ 就可以了。

改变字体的一种属性，而其它的保持不变，这样就可以得到许多种组合。（然而，这并非意味着对每种可能的组合，都存在着一种字体；如果不存在的话，那就会用另一种来替代。）如果我们首先用 \bfseries 选择一种黑体序列，接着用 \slshape 改为 slanted 形状，那么我们就得到一种黑色的 slanted 字体：

```
normal and {\bfseries bold and
  {\slshape slanted} and back} again.
```

的结果为： normal and **bold and *slanted* and back** again.

最后要指出一点，声明 \normalfont 要把所有的属性（除了尺寸）重设成其默认值，即罗马字体，直立和平均权重。有时为了使字体有效，调用这条命令是非常有用的。

§4.1.4 字体命令 2ε

对于上面所列的每一种字体声明，都有相应的字体命令，它们给其参数中的文本取指定属性的字体。

Family:	\textrm{ 文本 }	\texttt{ 文本 }	\textsf{ 文本 }
Shape:	\textup{ 文本 }	\textit{ 文本 }	\textsl{ 文本 }
	\textsc{ 文本 }		
Series:	\textmd{ 文本 }	\textbf{ 文本 }	
默认值：	\textnormal{ 文本 }		
强调：	\emph{ 文本 }		

注意这里也包含了 \emph 命令，其相应的声明是 \em 。 \textnormal 的参数被设置成 \normalfont 所选择的标准字体。

这些改变字体的命令可以有来作用于一小段文本或单个单词，这样就比在一个环境中放一个声明要合理得多。前面那个例子现在变为：

```
normal and \textbf{bold and \textsl{slanted}
   and back} again.
```

结果仍然为： normal and **bold and *slanted* and back** again.

同 \emph 命令一样，当在直立和斜体 /slanted 字体之间切换时，这些字体命令会自动地加进必要的倾斜校正（ 3.5.1 节）。当会出现这种校正时，可

以加入命令 \nocorr 来取消它，如

```
\textit{some italics\nocorr} withouot correction
{\slshape italics \nocorr\textup{without} correction}
```

请记住：字体属性声明和命令只适用于 LaTeX 2_ε，而不适用于 2.09 版本。

§4.1.5 旧字体声体 2.09

为了与 LaTeX2.09 兼容，仍旧保留了两字母的字体声明（它们很早就是TeX的一部分）。

\rm	Roman	\it	*Italic*	\sc	Small Caps
\bf	**Bold face**	\sl	*Slanted*	\sf	Sans Serif
\tt	Typewriter				

这些命令与新的相应命令有不同的作用方式：它们是严格地选择一种新字体，而不是只改变某一属性，但是不会改变当前尺寸。这就好像同时调用了 \normalfont 声明一样。

```
{\bf text} 等于 {\normalfont\bfseries text}
```

注意：在 LaTeX 2_ε之前的两个 NFSS版本中，这些两字母声明被定义成新的长声明。为了处理用那些系统编写的文档，需要在导言中加进软件包 newfont（8.8.3 节）。

§4.1.6 其它字体

很可能你所在的计算中心或者所用的 PC TeX软件包具有不只上面所列的那些字体形状和尺寸。可以查查局部指南或者软件包指令手册。如果确实如此，那么在 LaTeX文档中可以使用它们，方法是要么直接引用它们的名称，要么如果它是建立在 NFSS之上的，可以使用它们的属性。

要用名称上载一种字体，可以用命令

```
\newfont{\fnt}{ 名称 scaled 因子 } 或
\newfont{\fnt 字体 }{ 名称 at 尺寸 }
```

它把字体赋给新的字体命令名 \fnt 。在前一种情形中， 因子 是一个数，其 1000 倍于放缩因子，这个因子用来对字体的基本或设计尺寸进行放大或缩小。在第二种情形中，字体被放缩到指定的 尺寸 。要安装一种 slanted, sans serif 字体，尺寸为 20.74pt，名称为 \sss，那么我们可以用命令

```
\newfont{\sss}{cmssi17 at 20.74pt}
```

上载 cmssi17 at 20.74pt 。现在声明 \sss 就同 \rm 和 \it 改变字体一样把字体切换为上述字体，只是基线分隔没有改变。

与之相应的，也可以用属性来进行新字体声明（见 8.5.4 节）：

```
\DeclareFixedFont{\sss}{OT1}{cmss}{m}{sl}{20.74}
```

实际上，如果不知道当前所想用的字体编码和 \sffamily，或者不想那么精

确地声明尺寸，那么也可以如下给出：

```
\DeclareFixedFont{\sss}{\encodingdefault}{\sfdefault}
      {m}{sl}{20}
```

（默认值在 8.5.3 节解释。）

§4.1.7 字符集与符号

TEX和 LATEX显示所用的字符集是由程序实现的。对于不同的版本，约有 400 到 800 个的字体文件，分别相应于不同的字样、尺寸和放缩比例，每个文件由 128（将来会是 256 ）个字符或符号组成。

每个字符集分别存贮在它们自己的文件中。在附录 E 中列出了 75 个标准字符文件的名称，其中很多就是打印出来的。

在字符集中的每个符号都是通过一个介于 0 到 127（或者 255 ）之间的数来编址的。命令

```
\symbol{ 数 }
```

会生成当前字体中内部标识号等于 数 的符号。在当前字体中符号 ¿ 的内部编号为 62 ，因此可以用命令 `\symbol{62}` 打印出它来。标识号也可以用八进制（前缀 ’ ）或十六进制（前缀 " ）数给出。因此符号命令 `\symbol{28}` ，`\symbol{’34}` 与 `\symbol{"1C}` 是一样的，都会生成 ø 。

`\symbol` 命令也可以用来生成还没有为它定义其它命令的符号，例如，`{\tt\symbol{’40}\symbol{’42}\symbol{’134}}` 得到 ␣"\ 。

E.6 节给出了不同符号族中标识号与字符的对应关系。

§4.2 居中与缩进

§4.2.1 居中文本

环境

```
\begin{center} 第一行 \\ 第二行 \\... 第 n 行 \end{center}
```

把由 `\\` 命令分开的各节文本居中排列。（可以用 `\\[ 长度]` 来插入可以省略的额外行间距。）如果文本的长度超过一行，那就会用一致的单词间距把它分成几行，除了最后一行外，尽可能地使它充满每一行。不会有断词现象出现。

在一个环境内部，可以用命令 `\centering` 来居中后接文本，同样用 `\\` 做为行分隔符。声明的作用到该环境结束时为止。

单独一行文本可以把它做为 TEX命令 `\centerline{文本 }` 的参数来居中排列。

§4.2.2 单边调整

环境

\begin{flushleft} 第一行 \\ 第二行 \\... 第 n 行 \end{flushleft}

\begin{flushright} 第一行 \\ 第二行 \\... 第 n 行 \end{flushright}

使得文本向左（ flushleft ）或向右（ flushright ）对齐。如果这一段文本在一行中放不下，那就把它断成相同单词间距的几行，这同 center 环境一样。而且也不会出现断词。

也可以在环境内部用如下的声明达到同样的效果：

\raggedleft 代替 flushleft 环境，而

\raggedright 代替 flushright 环境。

§4.2.3 两边缩进

可以用下面的环境来使一段文本的两边都缩进一定的长度：

\begin{quote} 文本 \end{quote}

\begin{quotation} 文本 \end{quotation}

为了与通常的文本区分开，在这段文本的上方和下方插入了额外的间距。

要显示的文本可具有任意的长度；它可以是一句话，也可以是一个段落，甚至于几个段落。

段落与通常一样用空行表示分段，但是在显示文本的开始和结束并不需要空行，因为无论如何这些地方总要插入竖直间距。

上面两种引用环境的区别在于：

在 quotation 环境中，段落是用第一行具有缩进来标识的，而在 quote 环境中则是用更多的竖直间距来表示的。

当前文本就是用 quotation 环境生成的，而上面的文本则是用 quote 环境生成的。

只有当通常的文本段落是有第一行进行缩进来标识时 quotation 环境才具有意义。

§4.2.4 诗歌缩进

为了排版诗歌、韵文，需要两边都进行缩进，这就环境

\begin{verse} 诗 \end{verse}

就比较合适了。例如：

节与节之间用空行分开

而每一节中用 \\ 分开不同的行。

如果一行太长，以致于超过行宽，那就会对它进行左右调整，并在

下一行上继续，而且进一步向里缩进。

上述的缩进框架可以嵌套使用。在一个 quote 环境中可以有其它的 quote, quotation 或 verse 环境。每嵌套一次，文本的两边都会加上额外的缩进，

而且其上下都有竖直间距；然而随着嵌套层数的增加，这些量是减小的。最多允许嵌套六层。

练习 4.1: 在练习文件中使用 quote 和 quotation 环境，并插入一些文本，即把一些文本包围在如下结构中：

\begin{quote}. . . . . \end{quote} 或

\begin{quotation}. . . . . \end{quotation}

练习 4.2: 生成一个叫 poem.tex 的新文件，在 verse 环境中输入一首你钟爱的诗。选择 12pt 做为标准的字体尺寸，斜体字样。在 verse 环境之前用稍大的黑体字样（如 \Large\bfseries ）写出诗的题目。把诗的题目右对齐。

注意：记住可以在一个环境中包含改变字体样式或尺寸的声明，其作用到环境结束时为止。

练习 4.3: 再生成另一个叫 title.tex 的文件。还记得可以用 titlepage 环境以生成一个单独的标题页吧？如果不记得了，就回头看看 3.3.1 节。用该环境创建一个标题页，自己选择字体尺寸和样式，所有项都要居中。

注意: 在 titlepage 环境中, 你也能用 center 环境, 但是只用 \centering 声明也就足够了，因为当 titlepage 环境结束时，其作用也就终止了。

用命令 \\[长度] 选择单独一行的间距，这里要指定一个适当的长度。记住文本第一行之前的竖直间距必须用 *- 形式命令 \vspace*[长度] 才能得到（见 3.5.3 节）。

尝试在标题页上不同部分（如题目、作者姓名，地址）用不同的字体尺寸和样式，一直到结果令你满意为止。

把你自己设计的标题页与练习 3.8 中的结果做一对比。如果你更喜欢你自己的作品，不妨用它取代标准练习文件 titlepage 环境中的命令 \title, \author 和 \maketitle 。

§4.3 列表

要生成有格式的列表，有三种可以使用的环境：

\begin{itemize} 列表文本 \end{itemize}

\begin{enumerate} 列表文本 \end{enumerate}

\begin{description} 列表文本 \end{description}

在这些环境中每一个的 列表文本 都要相对于左边界进行缩进，并且具有一个标签或记号。标签的类型就与所用的环境是什么有关了。生成标签的命令是 \item 。

§4.3.1 itemize 样例

- 每一项前面有一个黑点，即所谓的 *bullet* ，做为标签。

- 每一项中的文本可以具有任意长度。标签位于文本的第一行开头处。
- 相邻项之间具有额外的竖直间距分开。

上述结果是用如下输入生成的：

```
\begin{itemize}
\item 每一项前面有一个黑点，即所谓的 \emph{bullet}，做为标签。
\item 每一项中的文本可以具有任意长度。标签位于文本的第一行开头处。
\item 相邻项之间具有额外的竖直间距分开。
\end{itemize}
```

§4.3.2 enumerate 样例

1. 标签由相邻的数字组成。
2. 每调用一次 enumerate 环境，标签都是从 1 开始编号。

上面的例子是由如下输入的文件生成的：

```
\begin{enumerate}
\item 标签由相邻的数字组成。
\item 每调用一次 \texttt{enumerate} 环境，标签都是从 1 开始编号。
\end{enumerate}
```

§4.3.3 description 样例

目的 这个环境适用于定义一组单词或表达式。
例子 关键词做为标签，每一项由分类或解释组成。
其它用途 也可以用于参考文献中的作者列表。

上面的样例是如下生成的：

```
\begin{description}
\item[ 目的 ] 这个环境适用于定义一组单词或表达式。
\item[ 例子 ] 关键词做为标签，每一项由分类或解释组成。
\item[ 其它用途 ] 也可以用于参考文献中的作者列表。
\end{description}
```

\item[选项] 命令中包含一个可省略的参数，它以黑体形式成为标签。

§4.3.4 嵌套列表

上面给出的列表环境也可以彼此嵌套，深度可达四层。所用标签要视嵌套层数而定。缩进总是相对于包围它的列表左边界来确定的。下面就是一个四重的 itemize 环境：

- 第一层的标签是一个黑点，即 bullet 。
  - 第二层的标签就是一条长破折号。
    - 第三层的是一个星号。
      - 而第四层的标签就是单个点了。

- 同时，随着嵌套层数的增加，竖直间距在减少。
  - 返回第三层。
- 返回第二层。

- 现在我们又回到 itemize 环境的第一层了。

enumerate 环境是类似的，随着嵌套层数的增加，标签的样式也在改变：

1. 第一层的标签是阿拉伯数字后跟一个句号。
   (a) 在第二层，它就是包在小括号 () 内的小写字母。
      i. 第三层就用小写罗马数字后跟句号来标识。
         A. 而在第四层，就用大写字母。
         B. 可以用后面一节的方法来改变标签样式。
      ii. 返回到第三层了。
   (b) 返回到第二层了。
2. 现在又回到 enumerate 环境的第一层了。

下面是一个混合类型的嵌套例子：

- 第一层的 itemize 标签是 bullet 。
  1. 这里的标签是阿拉伯数字，因为这是 enumerate 环境的第一层。
     - 这是嵌套的第三层，但却是第二层的 itemize 。
       (a) 这是全部嵌套的第四层，但只是第二层 enumerate 环境。
       (b) 因此标签是包在小括号 () 内的小写字母。
     - 在这一层的标签是一长破折号。
  2. 每个列表都至少应该有两项。
- 在 \item 命令前的空行是没有用的。

上面这个混合列表是用如下输入得到的：

```
\begin{itemize}
  \item 第一层的 \texttt{itemize} 标签是 bullet 。
  \begin{enumerate}
    \item  这里的标签是阿拉伯数字，因为这是...
   \begin{itemize}
      \item  这是嵌套的第三层，但却是...
      \begin{enumerate}
        \item  这是全部嵌套的第四层，但只是...
        \item  因此标签是包在小括号 () 内的小写字母。
      \end{enumerate}
      \item  在这一层的标签是一长破折号。
    \end{itemize}
    \item 每个列表都至少应该有两项。
  \end{enumerate}
```

```
  \item 在 \item 命令前的...
\end{itemize}
```

练习 4.4: 利用 itemize 和 enumerate 环境生成同上面例子那样的嵌套列表，只是命令的调用顺序做一些改变。

练习 4.5: 利用 description 环境，制造一份会议参加人员及其地址的名单，这里要把参加者的姓名作为 \item 命令的参数。

注意: 对于这三种列表的每一个而言，若第一条 \item 命令前面有文本，在处理时都会生成一个错误信息。

§4.3.5 改变标签样式

在 itemize 和 enumerate 环境中的标签可以很容易地改变，方法就是利用 \item 的可省参数。利用 \item[+] , 那么标签就变为 + , 而 \item[2.1:] , 标签就成为 2.1: 。这里可省参数要取代标准标签。对于 enumerate 环境，这就意味着对应的记数器并没有自动增 1 ，用户必须手工改变它。

可省标签是在专为标签保留的区域中右对齐。该区域的宽度是该层次的缩进总长减去标签与正文的间距；这意味着标签区域的左边界与包围它的外层左边界是对齐的。

可以改变文档中的全部或部分标准标签。在LaTeX中是用如下内部命令给 itemize 生成标签的：

\labelitemi, \lableitemii, \lableitemiii, \labelitemiv

而相应于 enumerate 的是：

\labelenumi, \labelenumii, \labelenumiii, \labelenumiv

这里的 i, ii, iii 和 iv 分别指的是四个可能的层次。

可以用 \renewcommand 来改变这些命令。例如，要把第三层的 itemize 标签从 * 改为 + ，可以用

```
\renewcommand{\lableitemiii}{+}
```

同样， enumerate 环境中的标准标签也可以改变。然而，这里还要复杂一些，因为每个 enumerate 层次，都还有一个记数器，名字叫 enumi, enumii, enumiii 和 enumiv 。一个记数器的值可以用命令 \arabic, \roman, \Roman, \alph 或 \Alph 来显示出来，这些命令的意义从名字上就很容易知道了。即 \Roman{xyz} 用大写罗马数字显示出记数器 xyz 的当前值，而 \alph{xyz} 则用的是小写字母（这样 a 对应于 1 ， z 对应于 26 ）。

这些记数器，同记数器样式命令一起，可以用来重定义标签命令。例如，要把第二层的标签改为阿拉伯数字后加 '.)' ，那就需要用

```
\renewcommand{\labelenumii}{\arabic{enumii}.)}
```

这样就把 \labelenumii 重定义为用阿拉伯数字显示的记数器 enumii 的值，后跟 '.)' 。用这种方法，所有的层次的编号都可以改变。而且甚至可以包含不至一个记数器：

```
\renewcommand{\labelenumii}{\Alph{enumi}.\arabic{enumii}}
```

这样就会使得每当在第二层调用 \item ，那就会用作为大写字母的记数器 enumi 中的值，后跟数字形式的 enumii 中的值，即形式为： A.1, A.2, ..., B.1, B.2, ..., 等等。

如果要把新的标准标签适用于整个文档，那就应当把重定义命令放在导言中。否则，它就只对其所出现的那个环境内有效。

练习 4.6: 把第一层的 itemize 标准标签改为长破折号 — (写作 ---)，而第二层的为 – (--)，第三层只是一个连字符 - 。

练习 4.7: 修改 enumerate 环境的标准标签，第一层为 (I), (II), ...，而第二层的形式为 I-1:, I-2:, ...，即第一层的大写罗马数字形式后跟第二层的数字形式的编号。

上面的练习表明 itemize 和 enumerate 环境中的标签可以改变为任何所期望的样式。然而，这种改变是不应给以鼓励的，因为从排版的角度来看，由 Leslie Lamport 所选定的标准标签是很难再做什么改进的。如果确实需要其它的设置，那应该整个文档中都应用这个设置。

§4.3.6 参考文献

科技出版物中通常都会包含一长串引用，或者称为参考文献，它由其它工作的名称组成，在正文中通过其活动编号对其进行引用。通常正文没有结束，参考文献也就不会完成。

每当向参考文献中加入一项，如果需要通读所有的正文，以改变所有的引用编号，那就是一件非常令人头痛的事。因此 LaTeX提供一种新的机制，它不但对参考文献进行格式化，而且跟踪对它进行的修改和填加，以在正文中自动改变引用。

参考文献可以用如下环境来生成：

```
\begin{thebibliography}{ 标签样本 }
所有的参考文献项
\end{thebibliography}
```

在参考文献中和每一项都是用如下命令开始的：

```
\bibitem[ 标签 ]{ 关键词 } 文本条目
```

如果没有可省参数 标签 ， \bibitem 就会生成一个位于中括号内的活动编号，以供在正文中引用。如果想有 标签 ，那你可以用任何东西做标签，如作者姓名的缩写，或者一个随意的引用编号。不可省参数 关键词 是不会出现

在参考文献中的引用关键词，而且将来在正文要被标签取代。关键词可以是字母，数字和除了逗号外的符号组成。

真正的参考文献信息是包含在 文本条目 中的，其形式可能为 '作者，题目，出版社，年代，版本，页码'，而且不同部分字样可能不同。第一行后的文本要进行缩进，宽度等于 标签样本 的宽度，因此标签样本应是参考文献中的最长标签。对于活动编号的标准应用， 标签样本 应该是一个没有意义的数字，但其位数等于最大标签的位数（例如如果有超过 10 个，但不足 100 个标签，可以用 99 ）。

在正文中的引用由如下命令生成：

```
\cite{ 关键词 }
```

这里的 关键词 就是出现在 \bibitem 命令中的引用关键词。例如，如果参考文献中包含：

```
\begin{thebibliography}{99}
  \bibitem{lamport} Leslie Lamport. \textsl{\LaTeX\ -- A Documnet
    Preparation System}. Addison--Wesley Co., Inc.,
    Reading, MA, 1985
  . . . . . .
  \bibitem{knuth} Donald E. Knuth. \textsl{Computers and
    Typesetting Vol.\ A--E}. Addison--Wesley Co., Inc.,
    Reading, MA, 1984--1986
  \bibitem[6a]{knuth:a} Vol A: \textsl{The \TeX book}, 1984
  \bibitem[6b]{knuth:b} Vol B: \textsl{\TeX: The Program.}, 1986
  . . . . . .
\end{thebibliography}
```

那么文本

```
For additional information about \LaTeX\ and \TeX\ see
\cite{lamport} and \cite{knuth, knuth:a}.
```

的结果为： For additional information about LaTeX and TeX see [1] and [6,6a].

这里 lamport, knuth 和 knuth:a 被选作关键词。用 99 做标签样本，因为对于这里的 \bibitem 标准形式，两位数的标签就生成足够的缩进。关键词为 knuth 的项是列表中的第六项，因此自动得到标签 [6] ；为了使其子项 knuth:a . . . 的编号为 [6a]. . . [6e] ，因此需要把可省的标签参数设置为 6a, . . ., 6e 。

thebibliography 环境的结果是以参考文献的形式列于文档尾部。对于文档类 book 和 report , 其开头会显示一个单词 **Bibliography** 做为章标题，而对于 article ， 则会显示 **References** 做为节标题。上面的那个示例参考文献结果为：

[1] Leslie Lamport. *LaTeX – A Documnet Preparation System*. Addison–Wesley

Co., Inc., Reading, MA, 1985

. . . . . .

[6] Donald E. Knuth. *Computers and Typesetting Vol. A–E.* Addison–Wesley Co., Inc., Reading, MA, 1984–1986

[6a] Vol A: *The TEXbook*, 1984

[6b] Vol B: *TEX: The Program.*, 1986

. . . . . .

本书实际所用的参考文献样式来自于LATEX标准：在正文中的引用用的是作者姓名与出版日期。列于参考文献中所有项都没有标签。在 C.3.4 节中描述了如何生成这种样式的参考文献。

练习 4.8: 利用 `thebibliography` 环境生成一个参考文献，其标签由作者姓名前三个字母后接出版年代后两位数字组成。如果同一位作者在同一年里有不只一篇作品，那就增加一个活动字母以示区别，如 knu86c ， knu86d 。对于这样的标签，其同时也适合于做关键词。而缩进宽度应为 1.5cm 。

注意：缩进宽度是由 `thebibliography` 环境中的标签样本参数确定的。这通常只是一个虚文本，所要的只是其宽度。这个宽度可由 `\hspace{` *宽度* `}` 精确给出。

练习 4.9: 把上面练习中的 `thebibliography` 环境复制到你一直在用的练习文件尾部（但要在 `\end{document}` 命令前面）。在正文中插入 `\cite` 命令来引用参考文献中的项。注意要保证 `\cite` 命令中的关键词与 `thebibliography` 环境 `\bibitem` 命令中的关键词完全一样。

通常LATEX安装中也包含了程序 BIBTEX，它可以从 `\cite`（和 `\nocite`）命令，通过引用一个或多个参考文献数据库来自动生成参考文献。在 8.3.2 节和附录 B 中对此有详细介绍。后者也介绍了许多用户所用的参考文献数据库是如何构成的。

§4.4 广义列表

类似于三种环境 `itemize`, `enumerate` 和 `description` 这样的列表可以从相当一般的形式构造而来。标签的类型与宽度，缩进的宽度，段落和标签之间的距离等等，都可以由用户通过 `list` 环境来全部或部分地进行设置：

`\begin{list}{` *标准标签* `}{` *列表声明* `}` *列表中的项* `\end{list}`

这里的 *列表中的项* 由列表的各项文本组成，每一项由 `\item` 命令开头，它要生成相应的标签。

标准标签 包含内容是当 `\item` 命令没有参数时生成的标签（见后）。

在 4.4.2 节中描述的列表参数可以由用户通过 *列表声明* 设为任何所期望的值。

§4.4.1 标准标签

在 list 环境中的第一个参数是 标准标签，当 \item 命令没有参数时，要用它生成标签。对于无变化的标签，如 itemize 环境中的标签，这只要是所期望的符号就可以了。如果希望它是一个数字符号，那就必须以 $ 符号名 $ 形式给出，即包含在 $ 符号内。例如，要用 ⇒ 做标准标签， 标准标签 就必须定义为 $\Rightarrow$ 。

然而，通常标签中需要包含一列数字。为此，就必须用 \newcounter{ 名称} 命令生成一个新的记数器，这里的 名称 就是它的标号。这条命令必须位于 list 环境中对这个记数器第一次应用之前。假设已为此而定义了一个记数器 marker ，那么参数 标准标签 就可以用 4.3.5 节中所给出的任一命令来显示记数器：例如， \arabic{marker} 就生成一个活动的阿拉伯数字编号。

即使再复杂的标签也可以用这种方法得到。如果接连的编号为 A–I, A–II, ..., 标准标签 就要设为 A--\Roman{marker} 。

一个记数器在 标准标签 里可以正常使用之前，必须通过在 列表声明 中包含命令 \usecounter{记数器} 来把该记数器与列表关联起来，这里的 记数器 就是赋给记数器的名称（上面例子中的 marker ）。

标准标签实际上是由 \item 命令调用 \makelabel{标签} 得到的。用户可以借助于 \renewcommand 命令在列表中重定义 \makelabel ：

\renewcommand{\makelabel}{ 新的定义 }

如果标准标签是以这种方式定义的，在 list 环境中相应的项左边是空的。这是因为 \makelabel 是更一般的命令，它覆盖了其它定义。在 7.5.9 节中有例子。

§4.4.2 列表样式参数

有很多样式参数，以格式化列表，LaTeX把它们都设置为特定的标准值。在特定的列表中，用户可以在 列表声明 中改变这些值。用 \setlength 命令可以像通常那样赋值。然而，如果这种赋值是在 list 环境外进行的，在大多数情形中，就简单地被忽略了。这是因为在每一层每一个参数都被预设置为默认值，只有 列表声明 才可以覆盖它。

下面列出了样式参数，同时在图 4.1 中进行了标识，它们是基于 Lamport 的选择：

\topsep

是一个竖直距离，其要与 \parskip 一起插入到列表与包围它们的上下文本之间。其默认值在每一个列表层次中进行设置，不能在 列表声明 外面进行全局重定义。

\partopsep

当第一个 \item 项前面有空行，或者最后一个 \item 项后面有空行时，

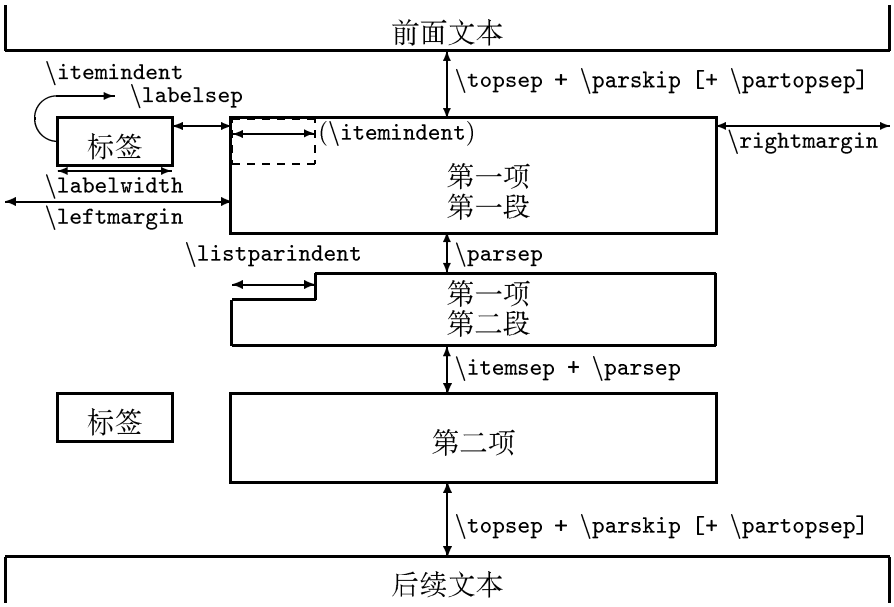


图 4.1: list 参数

要把这个量与 \topsep + \parskip 一起插入到列表的上面和下面。

\parsep

在一个 \item 项中段落之间的竖直距离。在每一个层次上设置其默认值，同 \topsep 一样。

\itemsep

是一个竖直距离，其与 \parsep 一起插入到两个 \item 项之间。同 \topsep 和 \parsep 一样，其默认值也是在每一个层次上设置，不能进行全局改变。

\leftmargin

是当前环境的左边界到列表文本左边界的距离。其在每个层次上都有默认值，可以如 4.4.6 节中描述的那样进行全局重定义。

\rightmargin

是当前环境右边界到列表文本右边界的距离。其标准值为 0pt，只可以在 *列表声明* 中对其进行改变。

\listparindent

是一个 \item 项中一段第一行相对于列表文本左边界的缩进宽度。通常设为 0pt，因此没有缩进。只可以在 *列表声明* 中改变其值。

\labelwidth

是为标签而预留的盒子宽度。在这个空间中，标签是右对齐的。可以为其设置一个全局的默认值，以适用于所用的列表层次。

\labelsep

是标签盒子与列表文本之间的距离。可以全局性地赋给其一个新值，但它只对第一层有作用。

\itemindent

是一个 \item 项中标签及文本第一行相对于右边界的缩进宽度。通常取为 0pt ，因此就没有这种效果。只可以在 列表声明 中改变其值。

把当竖直距离从其标准值变为其它值，最好使用橡皮长度（ 2.4.2 节 ）。

由 \item 命令生成的标签通常在一个宽度为 \labelwidth 的盒子中是右对齐的。也可以如下面参数列表那样，使其为左对齐的，即在标准标签的定义或者 \makelabel 命令尾部加上 \hfill 。

§4.4.3 用户制作列表的示例

一个关于图的列表为：

Figure 1: *Page format with head, body, and foot, showing the meaning of the various elements involved.*

Figure 2: *Format of a general list showing its elements.*

Figure 3: *A demonstration of some of the possibilities for drawing pictures with LaTeX.*

这个列表是用如下输入生成的：

```
\newcounter{fig}
\begin{list}{\bfseries\upshape Figure \arabic{fig}:}
  {\usecounter{fig}
  \setlength{\labelwidth}{2cm}\setlength{\leftmargin}{2.6cm}
  \setlength{\labelsep}{0.5cm}\setlength{\rightmargin}{1cm}
  \setlength{\parsep}{0.5ex plus0.2ex minus0.1ex}
  \setlength{\itemsep}{0ex plus0.2ex} \slshape}
  \item Page format with head, body, and foot, showing the
        meaning of the various elements involved.
  \item Format of a general list showing its elements.
  \item A demonstration of some of the possibilities for
        drawing pictures with \LaTeX.
\end{list}
```

命令 \newcounter{fig} 建立了一个新的记数器 fig 。标准标签设置为直立的黑体单词 **Figure** ，后接活动的阿拉伯数字，由：结尾。每个 \item 命令都会显示出一个这样的标签。

列表声明把 \usecounter{fig} 做为其第一条命令，这样就使得记数器 fig 在列表中可用。标签盒子的宽度(\labelwidth)设为 2.0cm , 列表文本的

左边界(\leftmargin)为 2.6cm，在标签和文本之间的距离(\labelsep)为 0.5cm，列表的右边界(\rightmargin)离包围它的文本距离设为 1cm。

在一项中段落之间的竖直距离(\parsep)为 0.5ex，但还可以伸展 0.2ex，或收缩 0.1ex。项与项之间的额外距离(\itemsep)为 0ex，可以伸展 0.2ex。

对于所有其它的列表参数，用的都是标准值。在列表声明中最后一条命令是 \slshape，这把列表文本设为 *slanted* 字样。

注意：在 LaTeX 2ε 中，如果在标签定义中没有使用 \upshape，每一 \item 的文本就是 slanted 字样的，如 ***Figure1:*** 。在 LaTeX2.09 中，就必须用 \bf 和 \sl 声明，但是不可能得到一个黑体的 slanted 字样。

§4.4.4 做为新环境的列表

如果在一个文档中要几次使用同一类型的列表，那么在列表环境中输入同样的 标准标签 和 列表声明 是一件非常麻烦的事。LaTeX提供了一种方法，可以把给定列表定义为具有自己名称的环境。这是用命令 \newenvironment 来做到的。

例如，可以用名称 figlist 保存上面例子中的列表，以后可通过该名称调用它：

```
\newenvironment{figlist}{\begin{list}
  {\bfseries\upshape Figure \arabic{fig}:}
  {\usecounter{fig} ... {0ex plus0.2ex}\slshape}}
  {\end{list}}
```

这样就可以如下调用它了：

```
\begin{figlist} 列表项 end{figlist}
```

其行为同预定义的列表一样。

练习 4.10: 用名称 sample 定义一个新的环境，它生成一个列表，每当调用 \item 时，生成标签为 Sample A, Sample B, 等等。标签在一个宽度为 20mm 的盒子中左对齐，标签盒子与列表文本之间的距离为 2mm，总的左边界为 22mm。文本的右边界相对于包围文本向内移了 5mm。两项之间的额外竖直距离除了通常的段间距外还有 1ex plus0.5ex minus0.4ex。在一项中第二段以后的段落具有 1em 的缩进。通常的段落间距为 0ex，可伸展到 0.5ex。

LaTeX自身就经常用列表环境来定义许多其它的结构。例如，quote 环境的定义为

```
\newenvironment{quote}{\begin{list}{}
    {\setlength{\rightmargin}{\leftmargin}}
    \item[]}{\end{list}}
```

因此该环境也就是一个列表，其 \rightmargin 的值被设置为 \leftmargin 的当前值，其默认值为 2.5em。列表本身只由一个 \item 项组成，它的标签

为空白，在 quote 的定义中自动调用了 \item[] 以达此目的。

用同样的方式，LaTeX在内部以特殊 list 形式定义了 quotation 和 verse 环境。左边界以及与包围文本的竖直距离都取的是 list 环境的标准值，因此只有当标准值改变时，其值才会变化。

最后，我们给出一个可能的自定义特殊列表做为例子：

```
\newenvironment{lquote}{\begin{list}{}{}\item[]}{\end{list}}
```

这生成一个 lquote 环境，其除了进行相对于包围文本进行 \leftmargin 的缩进，右边界与通常文本对齐（因为 \rightmargin 的标准值为 0pt ）外，再没有别的变化。

§4.4.5 平凡列表

在 LaTeX中还包含一个 trivlist 环境，其语法为

\begin{trivlist} 内部文本 \end{trivlist}

在这里没有 标准标签 和 列表声明 参数。它相当于一个列表，其标签是空的，\leftmargin, \labelwidth 和 \itemindent 都赋值 0pt ，而 \listparindent 等于 \parindent ， \parsep 等于 \parskip 。

LaTeX用这一环境生成其它环境。例如，当调用 center 环境时，内部实际上调用的就是

\begin{trivlist} \centering \item[] 内部文本 \end{trivlist}

环境 flushleft 和 flushright 也是类似定义的。

§4.4.6 嵌套列表

列表也可以与其它的列表环境 itemize, enumerate 和 \description 之间相互嵌套，最大深度为 6 。在每一层，都要相对于前面高一级那层进行总量为 \leftmargin 的缩进。

前面已提到过，只有在导言中才可能利用声明改变 list 参数的有限几个标准值。但有一个例外，那就是不同嵌套层数中左边界的缩进量。在内部它是用参数 \leftmargin$n$ 来设置的，这里的 n 表示 i, ii, iii, iv, v 或 vi 。用户可以改变这些值；例如，用声明 \setlength{\leftmarginiv}{12mm} 可以使得第四层的左边界相对于第三层向右移了 12mm 。这些声明必须放在 list 环境外面，也不能放在 列表声明 里。

在嵌套列表的每一层中，都要调用内部宏 \@list$n$（ n 从 i 到 vi ）。它使 \leftmargin 的值等于相应的 \leftmargin$n$ ，除非 \leftmargin 在 list 环境中显式地进行了声明。也就是说，在外部并不存在单个的 \leftmargin 标准值，而是 6 个不同的值。只有在 list 环境内部，参数 \leftmargin 才有意义。

§4.5 定理型的声明

在科技文献中，经常需要如下结构：

Theorem 1 (Balzano–Weierstrass) *Every infinite set of bounded points process at least one maximum point.*

或者

Axiom 4.1 *The natural numbers form a set S of distinct elemetns. For any two elements a, b, they are either identical, $a = b$, or different from one another, $a \neq b$.*

同样的结构除了这里的名称 Theorem 或 Axiom 外，有时还会有名称 Definition, Corollary, Declaration 或 Lemma 。它们的共同特点就是都有一个关键词和一个活动编号，而且它们以黑体字样显示，而其它相应的文本是斜体。

当然，这些结果用户可以通过显式给出样式和编号来得到，但是如果在中间插入那种类型的一个新结构，那么用户就需要对后面的结构进行麻烦的重新编号。利用命令

`\newtheorem{` *结构类型* `}{` *结构标题* `}[` *其它记数器* `]`

LaTeX就会自动跟踪编号的变化。这里的 *结构类型* 是用户给该结构指定的任意一个名称，而 *结构标题* 就是那个以黑体字样与活动编号列在一起的单词（如 **Theorem** ）。如果没有可省参数 *其它记数器* , 在整篇文档中是连续编号的。然而, 如果在 *其它记数器* 中有一个已存在的记数器的名称, 如 `chapter` , 每当该记数器增加时, 都会对这里的结构重新编号, 而且它们可以列在一起, 如上面的 **Axiom 4.1** 。

预定义的结构可以用如下命令来调用：

`\begin{` *结构类型* `}[` *附加标题* `]` *文本* `\end{` *结构类型* `}`

这里会给必要的记数器增 1 ，以生成恰当的编号。上面的例子就是用如下输入得到的：

```
\newtheorem{theorem}{Theorem} \newtheorem{axiom}{Axiom}[chapter]
 . . . . . .
\begin{theorem}[Balzano--Weierstrass] Every .... \end{theorem}
\begin{axiom} The natural numbers form ......... \end{theorem}
```

可以省略的 *附加标题* 紧接在活动编号后以黑体字样显示在小括号 () 内。

有时候一个结构的编号并不是与自己有关，它会与其它结构一起编号。为此可以在定义中包含另一个可省参数：

`\newtheorem{` *结构类型* `}[` *编号来源* `]{` *结构名称* `}`

这里的 *编号来源* 是一个已存在的定理结构名称，它们共享同一个记数器。因此通过定义 `\newthoerem{subthrm}[theorem]{Sub-Theorem}` , `theorem` 和 `subthrm` 这两个结构的编号是同一个序列： **Theorem 1**, **Sub-Theorem 2**,

Sub-Theorem 3, **Theorem 4** ，依次类推。

§4.6 制表位

§4.6.1 基础知识

在打字机上可以在一行的任何位置设置制表位；这样当按一些 tab 键，打字头或者回车键就会跳到下一个制表位处。

在 LaTeX的 `tabbing` 环境中也具有类似的情形：

`\begin{tabbing}` 文本行 `\end{tabbing}`

可以认为 tab 停留点按从左到右具有顺序编号。在 `tabbing` 环境开始，除了左边界（它称为第零个制表位）外，没有其它的任留点。在一行中可以用命令 `\=` 在任何地方设置制表位，而一行是用命令 `\\` 结束的：

```
Here is the \=first tab stop, followed by\= the second\\
```

这里把第一个制表位设在紧接单词 the 空白后面，而第二个是在单词 by 的后面。

当以这种方法设置好了制表位后，就可以在后续的文本行中从左边界开始，用命令 `\>` 跳到任一个制表位处。新行还是用通常的命令 `\\` 开始。

例如：

Type	Quality	Color	Price
Paper	med.	white	low
Leather	good	brown	high
Card	bad	gray	med.

```
\begin{tabbing}
Type\qquad\= Quality\quad\=
Color\quad\= Price\\[0.8ex]
Paper   \> med. \> white \> low \\
Leather \> good \> brown \> high\\
Card    \> bad  \> gray  \> med.
\end{tabbing}
```

§4.6.2 样本行

有时设置制表位比较好的，或者必须的方法是利用不必显示出来的样本行。例如，样本行可以由后面出现的各列中最宽项组成，或者由制表位之间最小列间距组成。在样本行中也可以包含 `\hspace` 命令，这样可以保证制表位之间的距离大于预定义的长度。

为了不显示样本行，那么该行文本不是用 `\\` 结束，而用的是 `\kill` 。

```
\hspace*{3cm}\=sample column \=\hspace{4cm}\= \kill
```

除左边界外，上例共设置了三个制表位：

在样本行开头的 `\hspace` 命令必须是 *- 形式，否则就会去掉在行边界所插入的间距。

§4.6.3 制表位与左边界

在 tabbing 环境中，每行的左边界首先要与包围该环境的文本左边界一致，并标记为第零个制表位。通过在一行的开头加上 *制表键* \>，可以使文本在第一个制表位处开始。然而，放在第一行开头处的命令 \= 会使得后续各行得到同样的效果。用 \+\+ 开始或结束一行，会使得后面的行都是开始于两个制表位后。只要一行中可以用多少个制表位，就可以使用多少个 \+ 命令。

而命令 \- 具有相反的效果：它把后续各行的左边界向左移动一个制表位。但用这条命令不能把边界设在第零个制表位的左边。

在每一行中要覆盖 \+ 命令的效果，可以采用在要被去掉的制表位前面加上 \< 的方法。该行就拥有到左边界那么多的制表位。对于下一个 \\ 命令，新行就在由 \+ 和 \- 命令总数所确定的左边界处开始。

§4.6.4 其它的制表命令

在任一行上都可以重设或增加制表位。如果具有足够的 \> 命令跳到下一个制表位，用 \= 命令就会插入一个制表位，否则它会重设下一个制表位。例如：

Old column 1 Old column 2
Left column Middle col Extra col
New col 1 New col 2 Old col 3
Column 1 Column 2 Column 3

```
\begin{tabbing}
Old column 1 \= Old column 2\\
Left column \> Middle col
  \= Extra col\\
New col 1 \= New col 2 \>
  Old col 3\\
Column 1\> Column 2 \> Column 3
\end{tabbing}
```

而有时候，在重设了制表位后又希望使用原来的。命令 \pushtabs 可以用来完成这一任务，它把当前制表位保存起来，并把它们从当前行上去掉。这样所有的制表位都可以重新设置。可以用命令 \poptabs 来激活保存的制表位。 \pushtabs 可以应需要而使用任意次，但在同一个 tabbing 环境中必须有与它相同数目的 \poptabs 命令。

可以用 *左边文本* \' *右边文本* 来在一个制表位处定位文本，这里的 *左边文本* 指的是恰好位于当前制表位前面（或者左边界）且具有很小的距离；而 *右边文本* 恰好在制表位处开始。在 *左边文本* 与制表位之间的距离由制表参数 \tabbingsep 确定。用户可以用命令 \setlength 如通常那样来修改其值。

可以用命令 \‘*文本* 来使文本相对于右边界对齐。在这一行中余下部分必须再没有 \> 或 \= 命令。

\=、 \‘ 和 ’ 命令在 tabbing 环境外是重音命令（ 2.5.7 节）。如果在 tabbing 环境内需要这些重音，那就必须用 \a=、\a‘ 和 \a’ 来代替。例如，要在 tabbing 环境内生成 ó、 ò 或 ō，就必须用 \a’o、 \a‘o 或 \a=o。 \-

命令在 tabbing 环境外也具有另外一种意义（单词的建议断点），但由于在这个环境中不会自动断行，因此不需要有替代形式。

下面是一个演示所有制表命令的例子：

Apples: consumed by: people
horses
and sheep
reasonably juicy
Grapefruits: a delicacy
(see also: melons
pumpkins)
Horses feed on apples

```
\begin{tabbing}
Graefruit: \= \kill
Apples: \> consumed by: \= people\+\+\\
horses \\
and \’ sheep\-\\
reasonably juicy\-\\
Grapefruits: \> a delicacy\\
\pushtabs
(see also: \= melons\\
 \> pumpkins)\\
\poptabs
Horses \> feed on \> apples
\end{tabbing}
```

§4.6.5 关于制表的注解

TEX如通常处理段落那样处理 tabbing 环境，如果在环境中有必要的话，会在两行中断开。然而，不允许在其中使用 \newpage 和 \clearpage 命令，而忽略 \pagebreak 命令。如果用户要强迫在 tabbing 环境中分页，那么他（或她）可以使用一个小技巧：在希望分页的那行末尾定义一个相当大的行间距（如 \\[10cm] ）。这就会强迫在此处分而，而且在下一页上所定义的空白是不存在的。

在一对 { } 中的文本行仍然有效，因此在某一行上的尺寸或字体声明只对该行有作用。文本不需显式地放在一对大括号内。

不可能在一个 tabbing 环境中嵌套另一个 tabbing 环境。

注意：制表移动命令 \> 总是移到下一个逻辑制表位处。如果前面的文本长度超过两个制表位之间可用的间距时，这实际上是一个回移。这与打字机上的制表机制不同。

在 tabbing 环境中没有自动的断行。在没有遇到 \\ 命令之前，一行是不会断开的。这样文本就有可能延伸超过右边界。这就需要依靠用户来决定到底该怎么样办了。

在 tabbing 环境中 \hfill 、 \hrulefill 和 \dotfill 命令是没有作用的，因为这里不会发生伸展。

练习 4.11: 利用 tabbing 环境生成下面这个表格。

Project: Total Requirements = $900 000.00
of which 1995 = $450 000.00
1996 = $350 000.00
1997 = $100 000.00
1995 approved: $350 000.00 Deficiency: $100 000.00

	1996	$300 000.00	$150 000.00
	1997	$250 000.00 Surplus:	$150 000.00
	tentative	1996 = $100 000.00 for deficiency 1995	
		1997 = $ 50 000.00 1996	
		+ $100 000.00 excess for 1995 in 1996	
Commitments		1995 = $100 000.00	
		1996 =$1 50 000.00	signed: H. André

提示：在 tabbing 环境中的第一行应该是

```
Project: \=Total Requirements\= = \$900\,000.00 \+\\
```

该行末尾的 \+ 有什么作用呢？为了使第二行到第四行上 1995 、1996 和 1997 年都开始于第二个制表位，该如何操作呢？在第二行的 \\ 命令前应用哪个命令呢？

虽然第八行有额外的制表位，但第一行到第四行以及第八行到第十二行的制表位是一致的。利用 \$1\=00\,000.00 ，可以使得第九行上的 \$\>50\,000.00 项与上面所有行中的小数点对齐。

第 5–7 行具有自己的制表位。这里要使用制表位的保存与恢复功能，进行制表位重设，并在结束时返回原状态。第 5–7 行的左边界对应于前面的第一个制表位。那么为此在第四行末尾应该用哪条命令呢？如何重设第二个到最后一个制表位呢？

最后一行的 'signed: H. André' 是右对齐的。在 tabbing 环境中是如何做到这一点的呢？要小心处理在 tabbing 环境中的重音 é 。

§4.7 盒子

所谓 盒子 ，就是一部分文本，TEX把它做一个整体，如同单个字符那样进行处理。一个盒子（连同其中的文本）可以上下左右移动。由于盒子是一个整体，TEX再不能把它分开，即使最初它是由更小的不可分的盒子组成。然而，如果乐意的话，可以把更小的盒子放在一起，以构造一个更大的盒子。

这实际上也就是TEX内部进行格式化时的方法：单个字符构成字符盒子，然后把它放到水平的行盒子中，在单词间插入橡皮长度。行盒子竖直堆积，构成段落盒子，在行之间也要插入橡皮长度。然后把它们放到页面主体盒子中，其连同页眉和页脚盒子一起构成页面盒子。

在 LATEX中，用户可以选择三种类型的盒子： LR 盒子，段落盒子和标尺盒子。 LR（左 - 右）盒子是由水平的从左到右的有序材料组成。段落盒子则是由竖直堆积的行组成。标尺盒子是一个用黑色填充的实心矩形，通常用来画水平或竖直线。

©TEXGuru, August 15, 1999

§4.7.1 LR 盒子

要创建一个由 LR 模式中文本组成的盒子，可以用命令

`\mbox{` 文本 `}` 和 `\makebox[` 宽度 `][` 位置 `]{` 文本 `}`

`\fbox{` 文本 `}` 和 `\framebox[` 宽度 `][` 位置 `]{` 文本 `}`

左边两条命令生成一个 LR 盒子，其宽度恰好是在 `{ }` 中所给 文本 的宽度。`\fbox` 命令同 `\mbox` 命令一样，只是 文本 要用方框包围起来。

而对于右边两条命令，宽度是由可省的长度参数 宽度 来预先确定的。另一个可省参数 位置 定义 文本 在盒子的位置。如果不给任何值，文本 是居中排列的。位置 参数可以取如下值：

`l` 文本左对齐，

`r` 文本右对齐的，

2ε `s` 伸展文本，以达到所定义的宽度。

因此 `\makebox[3.5cm]{centered text}` 生成一个宽度为 3.5cm 的盒子，其中的文本是居中排列的，即 centered text ，多余地方是用空白填充的，而 `\framebox[3.5cm][r]{right justified}` 的结果是文本在一个宽度为 3.5cm 的盒子中右对齐：right justified 。对于 LaTeX 2ε ，也可以用

`\framebox[3.5cm][s]{stretched\dotfill text}`

以生成一个填满的盒子：stretched.....text ，在这种情形中需要利用橡皮长度或者其它的填充物（ 42 页 ）以填充伸展出现的空白。

如果 文本 的自然宽度要比所定义的 宽度 大，那就会视 位置 参数的不同值，而在盒子的左边、右边或两边突出。例如，

`\framebox[2mm]{centered}` 结果为 centered

上面这种用法对于 `\framebox`，确实显得有点儿愚笨，但是对于 `\makebox` 却可能非常有用。在 `picture` 环境的图示中，可以利用 `\makebox` 定义一个宽度为 0pt 的盒子，这样就可以生成一个居中的，或左（右）对齐的文本（见第 6 章的例子）。用它也可以使得两部分文本重叠，例如 `\makebox[0pt][l]{/}S` 的结果为一条斜线穿过 S ，即 $\not S$ 。

注意：长度的定义中必须包含长度单位，即使所定义的长度是零。因此这里定义零宽度必须用 `0pt` ，而不能只用 `0` 。

在 LaTeX 2ε 中也可以相对于一个 LR 盒子的自然宽度（即只用命令 `\mbox` 时的宽度）来定义其 宽度 ：

2ε `\width` 表示盒子的自然宽度，

2ε `\height` 表示从基线到顶部的距离，

2ε `\depth` 表示从基线到底部的距离，

2ε `\totalheight` 就是 `\height` 加上 `\depth` 。

要生成一个宽度为其中包含文本的总高度六倍的有框盒子，可以用命令：

`\framebox[6\totalheight]{Text}` Text

注意：这些特殊的长度参数只在一个 LR 盒子的 *宽度* 定义或者下面将要介绍的段落盒子的 *高度* 定义中才有意义。而出现在其它情形中，就会产生错误信息。

如果一部分文本要出现在文档的几个地方，那么可以把它们保存起来，首先用如下命令：

`newsavebox{\boxname}`

这样就可以创建一个叫 '\boxname' 的盒子。该名称要符合 LaTeX命令语法（只能有字母），并且前面有 \ 。其也不能与任何已存在的 LaTeX命令同名。一旦已经如此初始化了一个盒子，那么命令：

`\sbox{\boxname}{`*文本*`}` 或者

`\savebox{\boxname}[`*宽度*`][`*位置*`]{`*文本*`}`

就会把 *文本* 的内容保存起来，以备后面使用。这里的可以省略参数 *宽度* 和 *位置* 与 `\makebox` 和 `\framebox` 中的意义一样。接着就可以在需要时用命令

`\usebox{\boxname}`

来把保存的文本当做一个整体插入文档中。

LR 盒子的内容也可以用如下环境来保存：

2ε
```
\begin{lrbox}{boxname}
    文本
    \end{lrbox}
```

其等价于 `\sbox{\boxname}{`*文本*`}` 。其好处在于它可以保存位于用户定义的环境（ 7.4 节）中的 *文本* ，以备将来用 `\usebox` 插入。

§4.7.2 LR 盒子的竖直移位

命令

`\raisebox{`*上移量*`}[`*高度*`][`*深度*`]{`*文本*`}`

生成一个内容为 *文本* 的 `\mbox` ，它相对于当前基线向上移动了 *上移量* 。可省参数告诉 LaTeX该盒子在基线上方伸展的 *高度* ，基线以下的 *深度* 。若没有这些参数值，盒子具有由 *文本* 和 *上移量* 确定的自然尺寸。注意这里的 *上移量* 、 *高度* 和 *深度* 都是长度（ 2.4.1 节）。如果 *上移量* 为负数，那么盒子就相对于基线向下移动。

例如：

```
Baseline \raisebox{1ex}{high} and \raisebox{-1ex}{low}
and back again
```

结果为： Baseline ^{high} and $_{\text{low}}$ and back again 。

高度 和 *深度* 的值可以与 *文本* 的实际值完全不同。其结果是根据同一行上所有其它盒子（字符也是盒子）的高度和深度，来确定当前文本行与前后

文本行的距离。当提升了一个盒子，而 高度 却是正常的字符大小，那么被提升的盒子就会与上面的文本行重叠，同样当降低一个盒子时 深度 也具有同样的效果。

§4.7.3 子段盒子和小页

整个段落可以用如下命令来放到单独的竖直盒子（或者套用 LaTeX术语称为 子段盒子 ）中：

\parbox[位置]{ 宽度 }{ 文本 }

也可以用环境实现同样的效果：

\begin{minipage}[位置]{ 宽度 } 文本 \end{minipage}

其都生成一个给定 宽度 的竖直盒子，其中的文本行如通常段落模式一样彼此堆积。

可省的参数 位置 有可取如下值：

- b 盒子的底边与当前基线对齐，
- t 顶行文本与当前基线对齐。

当没有任何 位置 参数时，子段盒子相对于外部文本的基线竖直居中排列。

位置 参数只有当 \parbox 命令或者 minipage 环境位于一个段落内部时才有意义，否则当前行与基线都没有意义。如果子段盒子前面紧接一个空行，这就是要开始一个新段。在这种情形中，子段盒子的竖直位置是由段落中后面的元素确定的。而后面的元素也可以还是子段盒子。如果整个段落只是由一个子段盒子或者小页组成， 位置 参数也没有意义，是无效的。

例：

```
\parbox{3.5cm}{\sloppy This is a 3.5 cm wide parbox. It is
    vertically centered on the}
\hfill CURRENT LINE \hfill
\parbox{5.5cm}{Narrow pages are hard to format. They usually
produce many warning messages on the terminal. The command
{\tt\symbol{92}sloppy} can stop this.}
```

This is a 3.5 cm wide parbox. It is vertically centered on the CURRENT LINE Narrow pages are hard to format. They usually produce many warning messages on the terminal. The command \sloppy can stop this.

```
\begin{minipage}[b]{4.3cm}
  The minipage environment creates a vertical box like the parbox
  command. The bottom line of this minipage is aligned with the
\end{minipage}\hfill
\parbox{3.0cm}{middle of this narrow parbox, which in turn is
  aligned with}
\hfill
\begin{minipage}[t]{3.8cm}
  the top line of the right hand minipage. It is recommended that
  the user experiment with the positioning arguments to get used
  to their effects.
\end{minipage}
```

The minipage environment creates a vertical box like the parbox command. The bottom line of this minipage is aligned with the middle of this narrow parbox, which in turn is aligned with the top line of the right hand minipage. It is recommended that the user experiment with the positioning arguments to get used to their effects.

在 4.7.7 节中我们演示了如何按所希望的那样把子段盒子彼此竖直堆积起来。

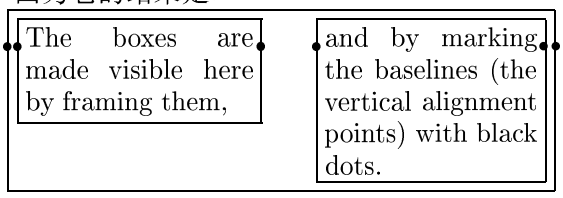
\parbox 命令生成一个由 文本 组成的竖直盒子，这同 minipage 环境是一样的。但是后者更具有一般性。在 \parbox 中的 文本 不可以包含在 4.2–4.5 节中所讲解的居中、列表或者其它环境。而另一方面，在 minipage 环境中是完全可以包含这些环境的。也就是说，一个 minipage 可以包含居中或缩进文本，也可以有列表和制表。

§4.7.4 竖直摆放的问题

对小页或者子段盒子的竖直定位有时会产生意想不到的结果，我们可以用图形来形象地说明 LaTeX是如何处理盒子的。假设我们想把两个高度不同的盒子并排摆放，并且对齐第一行，而它们的底部与当前文本行对齐。‘最明显的’做法就是

```
\begin{minipage}[b]{..}
   \parbox[t]{..}{..} \hfill  \parbox[t]{..}{..}
\end{minipage}
```

但这行不通，因为它的结果是：

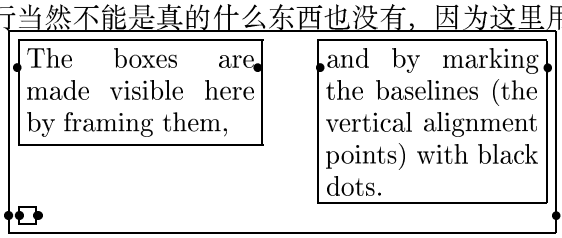
Current line 

of text.

之所以如此的原因在于 LaTeX内部把每个子段盒子或小页都当做一个具有自己相对于基线上下的高度和深度的字符来对待。当要考虑最外层的小页时，它只是在同一行上有两个‘字符’，而且该行同时是顶行和底行。因此最外面的小页的底行确实与文本行对齐，但它同时也是顶行。解决的办法是在最外层盒子中加一个没有内容的第二行：

```
\parbox[t]{..}{..} \hfill \parbox[t]{..}{..} \\ mbox{}
```

没有内容的行当然不能是真的什么东西也没有，因为这里用了 \mbox 。



如果要对齐两个盒子的底行，而顶部与当前行对齐，也会有同样的问题出现。下面是两种加入没有内容的第一行的方法：

```
\mbox{} \\                      恰好顶部对齐，或者
\mbox{} \\[-\baselineskip]      第一行文本对齐。
```

在 161 页上有第一种情形的一个例子。在第 81 页的练习 4.12 中也需要用到没有内容的行。

§4.7.5 具有指定高度的段落盒子 2ε

在 LaTeX 2ε中， \parbox 命令和 minipage 环境的语法已进行了推广，可以还有两个可省参数：

2ε `\parbox[ 位置 ][ 高度 ][ 内部位置 ]{ 宽度 }{ 文本 }`

2ε
```
\begin{minipage}[ 位置 ][ 高度 ][ 内部位置 ]{ 宽度 }
    文本
\end{minipage}
```

在这两种情形中， 高度 定义盒子的高度；在 高度 参数中，可以同 \makebox 和 \framebox（ 75 页）中的 宽度 参数一样，使用 \height, \width, \depth 和 \totalheight 参数。

可省参数 内部位置 说明文本在内部是如何定位的，其只有当给定了 高度 时才有意义。可能的取值为

- t 要文本靠盒子的顶部，
- b 把文本推向盒子的底部，
- c 竖直居中，
- s 伸展文本以填满整个盒子。

在最后那种情形中，当出现竖直伸展时，要用到橡皮长度（ 2.4.2 节）。

注意外部定位参数 位置 与内部定位参数 内部位置 的区别：前者说的是盒子如何与包围它的文本对齐，而后者决定在盒子中的内容是如何安排的。

例：

```
\begin{minipage}[t][2cm][t]{3cm}
  This is a minipage of height 2~cm with the text
  at the top.
\end{minipage}\hrulefill
\parbox[t][2cm][c]{3cm}{In this parbox, the text
```

```
  is centered on the same height.}\hrulefill
\begin{minipage}[t][2cm][b]{3cm}
  In this third paragraph box, the text is at the bottom.
\end{minipage}
```

This is a minipage of height 2 cm with the text at the top.

In this parbox, the text is centered on the same height.

In this third paragraph box, the text is at the bottom.

盒子间的 \hrulefill 命令显示了基线的位置。这三个盒子的尺寸是一样的，区别只是 *内部位置* 的值不一样。

§4.7.6 标尺盒子

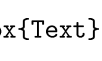
所谓 *标尺盒子* 就是完全用黑色填充的矩形。其相应命令的一般语法为

\rule[*提升*]{ *宽度* }{ *高度* }

其生成一个具有给定 *宽度*， *高度* 的实心盒子，并且相对于基线向上做 *提升*。因此 \rule{8mm}{3mm} 的结果是  。若没有可省参数 *提升*，矩形就放在当前文本行的基线上。

参数 *提升*, *宽度* 和 *高度* 都是长度（ 2.4.1 节）。如果 *提升* 为负数，则矩形位于基线以下。

也可以生成一个宽度为零的标尺。这样就得到一个不可见的具有给定高度的盒子。如此的构造称为一个 *支撑*，这样可以强迫其所在的水平盒子具有不同于其所包含内容需要的高度和深度。对于这种情形，用 \vspace 就行不通，因为它只是增加已经存在的竖直间距。

例如， \fbox{Text} 的结果为  。为了得到 ，那就必须告诉 TEX，盒子中的内容要相对于基线向上和向下延伸一定的长度。这可以用 \fbox{\rule[-2mm]{0cm}{6mm}Text} 来实现。这也就是说要用方框包围起来的文本是 ‘一个不可见的柱子，其底位于基线下 2mm，高有 6mm，它后面接单词 Text’。这个竖直条虽然看不到，但它却可以确定方框的顶边和底边。

标尺盒子的高度也可以是零，这就是一个不可见的具有给定宽度的水平线；然而，这种构造应是没有什么实际作用的，因为可以用 \hspace 命令很容易得到水平位移。

§4.7.7 嵌套盒子

上面描述的盒子可以相互嵌套任意次。要把一个 LR 盒子包含在一个子段盒子或者小页中没有任何概念上明显困难。相反，把一个子段盒子放在 LR 盒子中也是可能的，只要头脑中具有这样的想法：每个盒子都是一个整体，TEX把它们当做具有相应尺寸的单个字符一样来处理，那么一切就很清楚了。

把子段盒子放在 \fbox 命令内部，可以使得整个子段盒子都用方框包围起来。当前的结构就是用
\fbox{\fbox{ \parbox{10cm}{把子段盒子... }}}
得到的一个在方框盒子内的宽度 10cm 的子段盒子，而这个方框盒子又位于另一个方框盒子的内部，这样可以得到双层方框的效果。

把子段盒子包围在 \raisebox 内，可以得到任意希望的竖直位移。这里的两个盒子都具有位置参数 [b]，而右边的那个是如下生成的：

```
\raisebox{0.5cm}{\begin{minipage}[b]{2.5cm}
                  a b c d e ... x y z\\
                  \underline{baseline}
                \end{minipage} }
```

a b c d e f g h i
j k l m n o p q r
s t u v w x y z
baseline

它向上位移了 1cm 。 baseline

一个很有用的结构就是在一个 minipage 环境中，两个 minipage 环境相互定位。外层 minipage 的位置参数可以用来把其内容做为一个整体，与相邻文本或者盒子对齐。在练习 4.12 中给出了如此结构的一个例子。

最后提一下，类似 \parbox 命令和 minipage 环境这样的竖直盒子也可以做为 \sbox 或者 \savebox 命令中的 文本，从而保存起来，以后可以通过 \usebox 调用它们，具体与 4.7.1 节中的描述一样。

§4.7.8 盒子样式参数

对于有框盒子 \fbox 和 \framebox，有两个用户可以重设的样式参数：

\fboxrule 确定方框线的粗细，
\fboxsep 设置方框与被包围文本之间的距离大小。

与通常的 LaTeX方式一样，用 \setlength 命令可给这些长度参数赋新值：利用命令 \setlength{\fboxrule}{0.5mm}，可以使得所有后面的 \framebox 和 \fbox 命令中的线粗都为 0.5mm 。

这些设置的适用范围也遵从通常的规则：如果其位于导言中，那么其适用于整个文档；如果它位于一个环境中，那么其有效性持续到环境结束。

这些参数对于用在 picture 环境（ 6.4.2 节）中的 \framebox 命令没有作用。因为那里面的 \framebox 命令的语法和作用要比通常时候的更加广泛。

练习 4.12: 如何生成下面这个嵌套结构？（注意：字体尺寸是 \footnotesize ）

The first line of this 3.5cm wide minipage or parbox is aligned with the first line of the neighboring minipage or parbox.

This 4.5cm wide minipage or parbox is positioned so that its top line is at the same level as that of the box on the left, while its bottom line is even with that of the box on the right. The naive notion that this arrangement may be achived with the positioning arguments set to `t`, `t`, and `b` is incorrect. Why? What should this selection really produce?

The true solution involves the nesting of two of the three structures in an enclosing minipage, which is then separately aligned with the third one.

注意：由于有可能左边和中间的结构首先包围在小页中，或者是中间与右边的结构包围在小页，因此可以有两种不同的解法。试给出这两种解法。顺带提一下，第三个小页的宽度为 3cm 。

注意：这里又出现了两个盒子并排放在一起，如何与同行文本对齐的问题，（ 4.7.4 节 ）。为了使得正确地竖直对齐，需要加入一个没有内容的行。

练习 4.13: 生成下面给出的那个有框结构，并把它用命令 `\sbox{\warning}{`structure`}` 保存起来。为此你需要先用 `\newsavebox{\warning}` 创建一个叫 `\warning` 的盒子。这样以后你就可以在练习文件中任何需要打印这条警告信息的地方用 `\usebox{\warning}` 。

Vertical placement of minipage and parboxes can lead to surprising results which may be corrected by the use of dummy lines.

注意：这个子段盒子的宽度是 10cm 。如果依照前面有双层框的例子一样做，要生成这个有框文本，应该没有什么困难。要确保在 `\sbox{\warning}{结构}` 的最后输入了正确数目的右大括号。

然后，改变方框线粗细（ `\fboxrule` ）和方框间距（ `\fboxsep` ）的值，并再次打印出结果。

§4.8 表格

利用前面几节中的 盒子 结构和 `tabbing` 环境，可以生成各种类型的有框或无框表格。然而， TEX还提供了更方便的建立如此复杂结构的方法。

§4.8.1 构造表格

环境 `tabular`, `tabular*` 和 `array` 是生成表格和矩阵的基本工具。这些环境的语法如下：

```
\begin{array}[位置]{列}                  行      \end{array}
\begin{tabular}[位置]{列}                行      \end{tabular}}
\begin{tabular*}{宽度 }[位置]{列}        行      \end{tabular*}
```

array 环境只能用在 数学模式（见第 5 章）中。只所以在这里提及它，就是因为它的语法和参数意义与 tabular 环境中的完全一样。这三种环境都创建一个小页。参数意义如下：

位置 竖直定位参数（也可以看 4.7.3 节子段盒子中同名参数的意义）。它可以取下列值：

t 表格顶部与当前外部文本行的基线对齐；

b 表格底部与外部基线对齐；

当没有定位参数时，表格相对于外部基线居中摆放。

宽度 该参数只能出现在 tabular* 环境中，其确定它的整体宽度。在这种情形中， 列 参数必须在第一项后面某个地方包含 @- 表达式（细节见下） @{\extracolsep{\fill}} 。对于其它两种环境，整体宽度是由其文本内容确定的。

列 列格式参数。除了可能存在的相应于表格左右边界和列间距的额外项外，每列都必须在其中有一个相应的项。

可能的列格式符号有

l 列内容是左对齐的；

r 列内容是右对齐的；

c 列内容是居中的；

p{宽} 在该列的文本设置成具有给定 宽 的行，顶行与其它列对齐。实际上文本是用命令 \parbox[t]{宽 }{列文本} 放在一个子段盒子中的；

*{数}{列 } 包含在 列 中的列格式被复制了 数 份，因此 *{5}{|c}| 的结果与 |c|c|c|c|c| 相同。

相应于左右边界和列间距的可用 格式化符号 有：

| 画一条竖直线；

|| 画两条紧相邻的竖线；

@{文本} 这一条也叫做 @- 表达式，它在自己出现的两列中间每一行上插入 文本 。

@- 表达式去掉了原本自动加在两列之间的空白。如果在插入文本和后面的列之间需要有空白，那么需要在 @- 表达式的 文本 中包含 \hspace{ } 命令。如果想使某两个特定列之间的距离与其它的标准间距有所不同，那么可以通过在格式参数中对应于这两列的地方放上 @{\hspace{宽 }} ，这样就可能很容易地达到目标了。这里给定 宽 度的空白取代了标准列间距。

在 @- 表达式中的 \extracolsep{宽 } 会使得所有后面的列间距都增加给定 宽 度的额外间距，其作用持续到下一个 \extracolsep 为止。与标准间距不同，后面的 @- 表达式并不会去掉这个额外空白。在

tabular* 环境中，在列格式中的某处必须有 @{\extracolsep\fill} 命令，以使得后面所有列间距可以伸展到预定义的表格宽度。

如果表格的左右边界并没有竖线，那么就会在该处加入等于通常列间距一半的空白。如果不希望加入这个空白，可以在列格式的开始或结尾处包含一个空的 @- 表达式 @{}，以删掉这种空白。

行 由表格的实际条目组成，每一水平行都由 \\ 结束。这些行是由一组彼此之间用 & 符号分开的列条目组成。因此每一行应具有与在列定义 列 中相同数目的列条目。可以有些条目是空白的。 TEX把每个列条目当做就好像用大括号 { } 包围起来一样，因此对于类型样式或尺寸的修改，将被局限在这个列中。

\hline 这条命令只能位于第一行前面，或者紧接在行结束符 \\ 后面。它在刚结束的那行下面画一条水平直线，或者如果其位于开头时，在表格顶部画一横线，横线的宽度与表格的宽度相同。

放在一起的两条 \hline 命令就会画出两条间隔很小的水平线。

\cline$n$–m} 该命令从第 n 列的左边开始，画一条到第 m 列右边结束的水平线。与 \hline 一样，它也只能位于行结束符 \\ 的后面，而且可以同时有多次。命令 \cline{1-3} \cline{5-7} 就会在刚结束的行下面画两条水平线，一条是从第 1 列到第 3 列，另一条是从第 5 列到第 7 列。在每种情形下，用的都是完全列宽。

\multicolumn{数}{ 列}{文本} 该命令把接下来的 数 列组合成单个列，其宽度等于总宽度加上列间距。参数 列 由一个定位参数 l, r 或 c，以及可能有的 @- 表达式和竖线 — 组成。通过把 数 的值取 1，可以改变特定行中某一列的定位参数。

在这种情况下，‘列’是由定位符号 l, r 或者 c 开始的，包含所有接下来的内容，直到遇到另一个定位符号为止。第一列也可以包含在第一个定位符号之前的内容。因此 |c@{}rl| 就包含三列：第一列是 |c@{}，第二列是 r，第三列是 r| 。

\multicom 命令只能位于一行的开始或者一个列分隔符 & 后面。

\vline 该命令画一条竖直线，其高度等于其所位于地方的行高。用这种方法，可以得到那些不是贯穿整个表格的竖直线。

对于 tabular 和 array 环境，也可以用 LATEX 2_ε命令 \tabularnewline 结束一行。这是一条明确的行结束符，而 \\ 可以在一个列条目中结束一个文本行。（该命令是在 1994 年 12 月 1 日引进的。）

由于表格是与子段盒子和小页一样的竖直盒子，因此它也可以与其它的盒子或者文本水平定位（见 4.7.3 节的例子）。特别要指出的是，为了使表格

在页面上居中表格必须包围在

\begin{center} 表格 \end{center}

§4.8.2 表格样式参数

在表格的生成中，LaTeX要利用许多样式参数，来设置其标准值。用户也可以改变这些值，既可以在导言中进行全局性改动，也可以只局限于一个环境中。但不能在表格内部对其进行改动。

\tabcolsep 是插入在 tabular 和 tabular* 环境中两列间距离的一半；

\arraycolsep 是在 array 环境中相应于列间距的一半；

\arrayrulewidth 是表格中水平线与竖直线的粗细；

\doublerulesep 是双直线时两线之间的距离。

可以用 \setlength 命令像通常那样改变这些参数的值。例如，利用命令 \setlength{\arrayrulewidth}{0.5mm} 可使直线粗变为 0.5mm 。除了上面的参数外，还有参数

\arraystrecth 可以用来修改表格中的行间距。这是一个放缩因子，标准值为 1 。取值 1.5 就意味着行间距增大 50% 。可以用如下命令来重定义该参数：

\renewcommand{\arraystrecth}{ 因子 }

§4.8.3 表格样例

实际上表格的创建要比上面所列的复杂格式简单得多。有几个例子可以很好地演示这一点。

Position	Club	Games	W	T	L	Goals	Points
1	Amesville Rockets	33	19	13	1	66:31	51:15
2	Borden Comets	33	18	9	6	65:37	45:21
3	Clarkson Chargers	33	17	7	9	70:44	41:25
4	Daysdon Bombers	33	14	10	9	66:50	38:28
5	Edbartown Devils	33	16	6	11	63:53	38:28
6	Freeburg Fighters	33	15	7	11	64:47	37:29
7	Gadsby Tigers	33	15	7	11	52:37	37:29
8	Harrisville Hotshots	33	12	11	10	62:58	35:31
9	Idleton Shovers	33	13	9	11	49:51	35:31
10	Jamestown Hornets	33	11	11	11	48:47	33:33
11	Kingston Cowboys	33	13	6	14	54:45	32:34
12	Lonsdale Stompers	33	12	8	13	50:57	32:34
13	Marsdon Heroes	33	9	13	11	50:42	31:35
14	Norburg Flames	33	10	8	15	50:68	28:38
15	Ollison Champions	33	8	9	16	42:49	25:41
16	Petersville Lancers	33	6	8	19	31:77	20:46
17	Quincy Giants	33	7	5	21	40:89	19:47
18	Ralston Regulars	33	3	11	19	37:74	17:49

最简单的表格就是由行列组成的文本条目，其每一项要么居中，要么向一边对齐。列宽、列间距以及相应的表格宽度都是自动计算出来的。

上面这个表格有八列，第一列是右对齐，第二列是左对齐，第三列居中，接下来的三列又是右对齐，最后两列居中。因此在 tabular 环境中的列格式参数是

```
{rlcrrrcc}
```

生成这个表格的输入为：

```
\begin{tabular}{rlcrrrcc}
Position & Club & Games   & W & T & L & Goals & Points\\[0.5ex]
 1  & Amesville Rockets    & 33 & 19 & 13 &  1 & 66:31 & 51:15 \\
 2  & Borden Comets        & 33 & 18 &  9 &  6 & 65:37 & 45:21 \\
... & ......               & .. & .. & .. & .. &  ...  &  ...  \\
17  & Quincy Giants        & 33 &  7 &  5 & 21 & 40:89 & 19:47 \\
18  & Ralston Regulars     & 33 &  3 & 11 & 19 & 37:74 & 17:49
\end{tabular}
```

在每一行中，各个列条目之间是用符号 & 分开的，行本身是用 \\ 结束的。在第一行后面的 [0.5ex] 是增加前两行的竖直间距。最后一行不必用结束符号，因为当遇到 \end{tabular} 命令时会自动结束该行的。

在列格式参数中可以包含符号 |，以使得列之间用竖线分开。把第一行改为

```
\begin{tabular}{r|l||c|rrr|c|c}
```

那么结果为：

Position	Club	Games	W	T	L	Goals	Points
1	Amesville Rockets	33	19	13	1	66:31	51:15
2	Borden Comets	33	18	9	6	65:37	45:21
⋮	⋮						⋮
17	Quincy Giants	33	7	5	21	40:89	19:47
18	Ralston Regulars	33	3	11	19	37:74	17:49

在第一个列格式之前或者最后一个列格式之后的符号 — 都会在表格的外边生成一条竖线。两个符号 —— 生成双竖线。可以用命令 \hline 生成与表格宽度相同的水平线。这一命令只能出现在一个行结束符 \\ 后面或者第一行的开始。如此的两个命令 \hline\hline 会绘制双重水平线。

```
\begin{tabular}{|r|l||c|rrr|c|c|} \hline
Position & Club & Games   & W & T & L & Goals & Points\\
\hline\hline
 1  & Amesville Rockets    & 33 & 19 & 13 &  1 & 66:31 & 51:15 \\
 \hline
```

```
 . . . . . .
18  & Ralston Regulars     & 33 &  3 & 11 & 19 & 37:74 & 17:49 \\
\hline
\end{tabular}
```

那么现在的表格是：

Position	Club	Games	W	T	L	Goals	Points
1	Amesville Rockets	33	19	13	1	66:31	51:15
2	Borden Comets	33	18	9	6	65:37	45:21
⋮	⋮						⋮
17	Quincy Giants	33	7	5	21	40:89	19:47
18	Ralston Regulars	33	3	11	19	37:74	17:49

在这种情形中，由于在表格的最后有 `\hline`，因此必须给出行结束符 `\\`。

在这个例子中，所有行的第三列元素都是一样的，即 33。这样的公共条目可以利用列格式中的 `@{文本}` 形式的 @- 表达式自动插入，它把 `文本` 插入到相邻两列中。在我们的例子中，可以把列格式改成

`{rl@{ 33 }rrrcc}` 或者 `{|r|l||@{ 33 }|rrr|c|c|}`

这样文本 '33' 就会连同空白一起出现在每行的第二列和第三列之间。这样得到了行条目稍有点儿不同的同样表格：例如，第四行现在的输入应为

```
 4 & Daysdon Bombers    & 14 & 10 &  9 & 66:50 & 38:28 \\
```

列格式现在只是由 7 个列定义组成的：`rlrrrcc`。原来对应第三列的 `c` 现在已去掉了，从而现在每行应少一个列分隔符 `&`。现在的第三列，即取胜的局数，是由第二个 `&` 开始的，而且其与俱乐部名称之间填充进 `@{ 33 }` 的内容，它不用额外的 `&` 符号就能自动插入进来。

后面两列以居中的形式给出了得失球与分数的关系。这里的冒号 ':' 只是由于巧合上下位置一致，因为在所有行中冒号两边都是一个两位数。如果某一项为 `9:101`，那么由于该项要居中摆放，冒号就会向左稍微偏一点儿了。

也可以实现不依赖于数字位数，而仍然让 ':' 上下对齐，这就要利用在列格式中 `r@{:}l` 形式的 @- 表达式。其意义为在每一行的一个右对齐和左对齐列之间放入冒号。那么例子中现在的列格式参数变为：

`{rl@{ 33 }rrrr@{:}lr@{:}l}`

或者

`{|r|l||@{ 33 }|rrr|r@{:}l|r@{:}l|}`

而行条目变为：

```
 4 & Daysdon Bombers    & 14 & 10 &  9 & 66 & 50 & 38 & 28 \\
```

先前的一个 `c` 列，现在被 `r@{:}l` 形式的两列所取代。@- 表达式就是在

相邻两列间插入文本，而且去掉通常情况下应该存在的列间距。因此 `r` 列是紧靠 ':' 右对齐的，而 `l` 列是紧靠它左对齐的。

当一列是由具有小数点的长度不定的数字组成时，也可以用同样的方法达到小数点对齐的目的。

原来表示进球数和得分关系的条目现在由关于 ':' 符号定位的两列组成。对于输入得失球或者得分数字时，这没有任何问题。然而，列标题是 'Goals' 和 'Points'，其每个占两列的空间，中间没有冒号。可以用命令 `\multicolumn` 来解决这一问题，它把特定行中所选定的两列合并，并重新定义列格式。无方框的表格中第一行应该是：

```
Position & Club & W & T & L & \multicolumn{2}{c}{Goals}
                          & \multicolumn{2}{c}{Points}\\[0.5ex]
```

这里的 `\multicolumn{2}{c}{Goals}` 意味着接下来的两列被组合成一个居中的列，其中包含着文本 'Goals'。对于有框表格，在 `\multicolumn` 命令中新的格式参数必须是 `{c|}`，因为当原来的列被合并时，竖线符号 — 也同时被去掉了。要想知道哪些东西属于给定的列，只要记住规则：一列拥有所有的直至下一个 `r`, `l` 或 `c` 为止的内容。

那么我们现在 1994/95 年橄榄球联赛表格有如下的标题：

1st Regional Soccer League — Final Results 1994/95							
	Club	*W*	*T*	*L*	*Goals*	*Points*	*Remarks*
1	Amesville Rockets	19	13	1	66:31	51:15	League Champs
2	Borden Comets	18	9	6	65:37	45:21	Trophy Winners
3	Clarkson Chargers	17	7	9	70:44	41:25	Candidates
4	Daysdon Bombers	14	10	9	66:50	38:28	for
5	Edbartown Devils	16	6	11	63:53	38:28	National
6	Freeburg Fighters	15	7	11	64:47	37:29	League
7	Gadsby Tigers	15	7	11	52:37	37:29	
8	Harrisville Hotshots	12	11	10	62:58	35:31	
9	Idleton Shovers	13	9	11	49:51	35:31	
10	Jamestown Hornets	11	11	11	48:47	33:33	
11	Kingston Cowboys	13	6	14	54:45	32:34	Medium Teams
12	Lonsdale Stompers	12	8	13	50:57	32:34	
13	Marsdon Heroes	9	13	11	50:42	31:35	
14	Norburg Flames	10	8	15	50:68	28:38	
15	Ollison Champions	8	9	16	42:49	25:41	
16	Petersville Lancers	6	8	19	31:77	20:46	Disbanding
17	Quincy Giants	7	5	21	40:89	19:47	Demoted
18	Ralston Regulars	3	11	19	37:74	17:49	

输入为：

```
\begin{tabular}{|r|l||rrr|r@{:}l|r@{:}l||c|}\hline
  \multicolumn{10}{|c|}{\bfseries 1st Regional Soccer League ---
  Final Results 1994/95} \\ \hline
```

```
&\itshape Club &\itshape W &\itshape T &\itshape L &
  \multicolumn{2}{c|}{\itshape Goals}
  & \multicolumn{2}{c||}{\itshape Points}
  & \itshape Remarks \\ \hline\hline
  . . . . . .
```

从 3–5， 7–14 和 17 位的水平线是用命令 `\cline{1-9}` 生成的，而其它地方的则是用 `\hline` 生成的：

```
 11 & Kingston Cowboys     & 13 &  6 & 14 & 54&45 & 32&34 &
      Medium Teams \\ \cline{1-9}
```

对最后两行需要特别说明一下。备注 'Demoted' 在竖直方向恰好位于两行的中间。这是用如下输入来得到的：

```
 18 & Ralston Regulars     &  3 & 11 & 19 & 37&74 & 17&49
    & \raisebox{2.3ex}[0pt]{Demoted}\\ \hline
```

`\raisebox` 命令把文本 'Demoted' 向上提升 2.3ex 。这里如果没有可省参数 `[0pt]` 。那么这里对盒子的提升会导致最后一行的总高度也增加了 1.5ex 。这就会使得在第 17 行下面的水平线与第 18 行文本之间的竖直距离增大。我们就是用可省参数 height = `[0pt]` 来抑制这一额外间距。（参见 4.7.2 节中关于 `\raisebox` 命令的详细描述。）

有时我们需要增大水平直线与包围文本之间的竖直距离。如果前面那个表格的标题为如下形式时，就会更好看些：

1st Regional Soccer League — Final Results 1994/95						
	Club	*W*	*T*	*L*	*Goals*	*Points*	*Remarks*

这可以通过在标题文本中插入一个不可见的竖直标尺，即支撑（ 4.7.6 节）来做到这一点：

```
\multicolumn{10}{|c|}{\rule[-3mm]{0mm}{8mm}\bfseries 1st
 Regional Soccer League ---  Final Results 1994/95} \\ \hline
```

这个插入进来的标尺宽度为 0 mm，所以它是不可见的，它向基线下面伸展了 3 mm，有 8mm 高。因此它向基线上面伸展了 5mm（ 8 − 3 ）。因此它非常有效地把两个方向的水平线推离了标题。如果一行中不只一列，那么只要在一列中包含支撑就可以了，因为整行的尺寸是由最大列决定的。

练习 4.14: 用与上面排版橄榄球比赛结果一样的方式，生成一个你自己钟爱的比赛项目结果表。注意确保正确对齐得失球与得分中 ':' 的关系。

练习 4.15: 生成下面这个时间表，其中条目 'Day' 和 'Subj.' 等提升的高度与前面表格中 'Demoted' 的一样。为了简化使用，可以引进如下用户定义的命令：

```
\newcommand{\rb}[1]{\rasiebox{2.3ex}[0pt]{#1}}
```

（见 7.3.2 节），这样 `\rb{条目}` 的作用就和 `{\rasiebox{1.5ex}[0pt]{条目}}` 一样。这条命令也同样可以用于其它需要提升的 条目，如 `\rb{ Mon. }` 或者 `\rb{UNIX}` 等等。

Day	6.15–7.15pm		7.20–8.20pm		8.30–9.30pm	
	Subj.	Teacher Room	Subj.	Teacher Room	Subj.	Teacher Room
Mon.	UNIX	Dr. Smith Comp. Ctr	Fortran	Ms. Clarke Hall A	Math.	Mr. Mills Hall A
Tues.	LaTeX	Miss Baker Conf. Room	Fortran	Ms. Clarke Conf. Room	Math.	Mr. Mills Hall A
Wed.	UNIX	Dr. Smith Comp. Ctr	C	Dr. Jones Hall B	ComSci.	Dr. Jones Hall B
Fri.	LaTeX	Miss Baker Conf. Room	C++	Ms. Clarke Conf. Room	canceled	

在上面所有的例子中，在每个列中的条目都只有一行。而有的表格中某些列包含多行文本。

Model	Description	**Price**
GXT 1	**The PC Compatible**: Intel 80386, 1 MB main memory, color graphic card, multi–I/O card, 2 drives 1.2 MB, 17” monitor, keyboard, MS-DOS 6.2, GW Basis	883.70
GXT 20	**The XT Compatible**: Data as for GXT, but with Hercules compatible card, 1 drive 1.2 MByte, 1 hard disk 120 MByte unformatted	1376.40
GAT	**The AT Compatible**: Intel 80486, 1 drive 1.2 MB/720 KB, 8 MByte main memory, 17” monitor, keyboard, Hercules compatible card, hard disk 240 MByte unformatted, Serial/Parallel card, MS-DOS 6.2, GW Basic	2356.00

上面这个表格由三列组成，第一列左对齐，第三列右对齐。中间的那列包含着多行文本，每行宽度为 8.0cm 。这是用列格式符号 `p{宽度}` 来做到的。在这个表格中全部的列格式参数是 `{lp{8.0cm}r}` 。

```
\begin{tabular}{lp{8.0cm}r}
 \bfseries Model & Description & \bfseries Price \\[1ex]
 GXT 1 &\small{\bfSeries The PC Compatible}: Intel
 80386, 1 MB main memory, color graphic card,
 multi--I/O card, 2 drives 1.2 MB, 17'' monitor,
 keyboard, MS-DOS~6.2, GW Basis & 883.70\\
```

```
. . . . . .
MS-DOS~6.2, GW Basic & 2356.00
\end{tabular}
```

中间那列的文本只需简单输入就可以了，自动被断成 8.0cm 宽的行。该列就像通常那样用 & 符号与其它列分开。

警告：在 p 列中不能用行结束符 \\ ，因为它会被解释为表格行的结束。而可以用断行命令 \newline 和 \linebreak 。然而，如果某一行确实需要用 \\ 来断开，那么整个列就要放在 \parbox 中，其宽度与 p 列的一样。列条目可以由几段组成，空行表示分段。

练习 4.16: 生成下面这个表格。

Course and Date	Brief Description	Prerequisites
Introduction to LSEDIT March 14–16	Logging on — explanation of the VMS file system — explanation and intensive application of the VMS editor LSEDIT — user modifications	none
Introduction to LaTeX March 21–25	Word processors and formatting program — text and commands — environments — document and page styles — displayed text — math equations — simple user-defined structures	LSEDIT

最后这个例子描述了一个有框表格的空白表。这儿的困难在于设置空白盒子的高度和宽度，因为通常这些量是由文本条目自动确定的。这个例子说明了如何借助于支撑和 \hspace 命令来做到这一点。

Budget Plan 1995–1997

Project	Nr.		Name										
Year	1995		1996		1997								
	(GB £)	US $	(GB £)	US $	(GB £)	US $							
Investment Costs													
Operating Costs													
Industrial Contract													
Signature					Authorization								

```
\newsavebox{\kk}\newsavebox{\kkk}
\sbox{\kk}{\framebox[4mm]{\rule{0mm}{3mm}}}
\sbox{\kkk}{\usebox{\kk}\usebox{\kk}\usebox{\kk}}
\begin{tabular} {|l|c|c|c|}\hline
```

```
  \multicolumn{4}{|c|}{\rule[-0.3cm]{0mm}{0.8cm}\bfseries
      Budget Plan 1995--1997}\\
  \hline\hline
  \rule[-0.4cm]{0mm}{1cm}Project
  & \multicolumn{3}{l|}{Nr. \usebox{\kkk}\hspace{0.5cm}
 \vline\hspace{0.5cm}Name\usebox{\kkk}\usebox{\kkk}\usebox{\kkk}
   \usebox{\kkk}}\\ \hline
  \multicolumn{1}{|r|}{Year} & 1995 & 1996 & 1997 \\
  \cline{2-4}
  & (GB \pounds) \vline\ US \$ & (GB \pounds) \vline\ US \$
  & (GB \pounds) \vline\ US \$ \\ \hline
Investment & \hspace{2.5cm} & \hspace{2.5cm} & \hspace{2.5cm} \\
Costs    & & & \\ \hline
Operating & & & \\
Costs    & & & \\ \hline
Industrial& & & \\
Contract & & & \\ \hline
\multicolumn{4}{|l|}{\rule[-1.2cm]{0mm}{1.5cm}Signature
    \hspace{5cm}\vline~Authorization} \\ \hline
\end{tabular}
```

前三行只是直接与表格的结构有关。当使用了命令 `\usebox{\kkk}` 时就会画出三个空白框 （见 4.7.1 节）。

我们用 `\hspace{2.5cm}` 命令来设置后三列的宽度，用 `\vline` 在一列中画一条竖直线，除这两点外，这个例子同前面的例子相比，再没有什么新的地方。这里只需要对最后一行加以解释：

命令 `\multicolumn{4}{|l|}` 把全部四列合并为一列，文本被设置成左对齐。其文本中首先是一个支撑 `\rule[-12mm]{0mm}{15mm}`，这也就是说最后这行的高度开始于基线下面 12mm，总高度为 15mm。接着，从左页边开始，显示单词 `Signature`，间隔 5cm 后是 `\vline`，即一条竖线。单词 `Authorization` 与竖线之间用一个空白 (`~`) 隔开。

上面的例子清楚地说明了表格中列宽度和行高度是如何自动确定的。然而这些尺寸也会受支撑和 `\hspace` 命令的影响。而且在 4.8.2 节中描述的命令除了可以改变线粗细外，还可以增加列间距和行间距。例如，

```
  \setlength{\tabcolsep}{5mm}
```

就会在每列的前后插入 5mm 的间距；即得到了 10mm 的列间距。 4.8.2 节中有关于如何使用这些表格样式参数的说明。

©TEXGuru, August 15, 1999

§4.8.4 浮动表格

tabular 环境是在其所出现的地方生成一个表格，紧接其前面的文本，后面就是跟着它的其它内容。当在页面上表格与周围文本匹配得很好时，这没有什么问题，而且这通常也说是我们想要的。然而，如果表格很长，从它定义的地方开始，在一页中是放不下来的，那么就会结束该页，下一页开头就是这个表格，后面再接后续文本。这样就会导致当前页的格式并不是希望看到的样子。

如果在当前页上表格出现的地方有足够空间放下这个表格，那么就把它马上放下，否则就继续排版文本，保留表格到当有足够空间时，如下一页的开头，再排版表格，这种方式就是很好的。因于表格通常有标题和 / 或题目，这些项也自然应该随着它一起移动。

LaTeX提供了浮动表格（还有插图）的功能，其附加文本也以同样方式移动。这可以用如下环境来实现此功能：

```
\begin{table} 上部文本  表格  下方文本 \end{table}
```

这里的 *表格* 代表诸如 tabular 环境定义的整个表格，*上部文本* 表示出现在 *表格* 上方的文本，而 *下方文本* 表示出现在 *表格* 下方的文本。与表格相关联文本的宽度、间距和位置都要由用户来安排。

出现在 \begin{table} 和 \end{table} 中的所有内容是与包围它的外部文本无关，它们通常都放在当前页的开头。如果已有一个表格占用了这个位置，那就会尝试是否可能有足够的空间，把它放在当前页的底部。否则，就会把它放在下一页上，这样就有可能积累很多表格。包围表格的文本就如同没有表格那样进行排版。关于浮动的一般性细节，以及自动有序编号，请见 6.6 节。

在本页顶部的表格就是由放在此处的浮动表格生成的，其输入文本为（这里没有列出脚注的生成，我们将在 4.10.4 节中介绍）：

```
\begin{table}{\bfseries Primary Energy Consumption}\\[1ex]
  \begin{tabular*}{118mm}{@{}ll...rr@{}}
  . . . . . .
  \end{tabular*}\\[0.5ex]
  \emph{Source:} Energy Balance Study Group, . . .
\end{table}
```

有许多格式参数可以与 table 环境给合使用，我们将把它们与插图相应的参数一起在 6.6 节中介绍。

练习 4.17: 补全上面的生成浮动表格的文本（不要脚注）。注意下面的问题（必要时可查看在 4.8.1 节中关于 @- 表达式的解释）：

1. 在格式定义中开头和结尾的 @{} 有什么作用？

Primary Energy Consumption

Energy Source		1975	1980	1986
Total Consumption				
(in million tons of BCU[a])		347.7	390.2	385.0
of which	(percentages)			
	pertroleum	52.1	47.6	43.2
	bituminous coal	19.1	19.8	20.0
	brown coal	9.9	10.0	8.6
	natural gas	14.2	16.5	15.1
	nuclear energy	2.0	3.7	10.1
	other[b]	2.7	2.3	3.0

[a]BCU– Bituminous Coal Unit(1 ton BCU corresponds to the heating equivalent of 1 ton of bituminous coal = 8140 kwh

[b]Wind, water, solar energy, etc.

Source: Energy Balance Study Group, Essen 1987.

2. `tabular*` 环境生成一个给定宽度的表格，这里是 118mm 。在格式定义开头处的 `\@{\extracolsep{\fill}}` 的作用是什么呢？
3. 为了得到这里所显示的表格样式， `@{\extracolsep{\fill}}` 和相反的命令 `@{\hspace{1em}}@{\extracolsep{1em}}` 应该出现在格式定义的什么地方呢？如果只有 `@{\extracolsep{1em}}` 做为相反命令，那么表格会是什么样子呢？

§4.9 显示源文本

有时我们会希望某些文本直接按输入的样子显示,即不进行格式化处理。这可以用如下命令来实现：

```
\begin{verbatim}        源文本        \end{verbatim}
\begin{verbatim}*       源文本        \end{verbatim*}
```

这样 源文本 就会同输入时的一样，用打字机字样显示出来，同时包含所有的空格和断行。通常情况下被当做命令的特殊字符也做为字符串直接显示出来。唯一的例外就是标志着该环境结束的命令 `\end{verbatim}` 本身。这里的显示是在新的一行上开始，然后再在新一行上继续 `verbatim` 环境后的正常文本。

这个环境的标准形式与 *- 形式的差别在于，对后一种情形，空格是用符号 ␣ 表示的，以使得空格更容易看到。

例如，在 97 页上就有一些源文本被直接显示出来，以演示在禁止模式中脚注的应用。这可以用如下方法得到：

```
\begin{verbatim}
  \addtocounter{footnote}{-1}\footnotetext{Small insect}
  \stepcounter{footnote}\footnotetext{Large mammals}
\end{verbatim}
```

源文本也可以用如下命令在一行里显示出来：

\verb$c$ 源文本 c

\verb*c 源文本 c

这里的 c 为不出现于 源文本 中的任一字符。它是用来终止这种原样显示模式，返回如通常那样进行正常的文本格式。在 \verb 或者 \verb* 命令与 c 之间不能用空格，显然，对于普通形式， c 不能是 * 。

要显示普通文本中的命令名称，可以用下面这种方法。例如，为了得到 '\xyz{ }'，输入应该是 \verb=\xyz{ }= 。这里的等号 = 就用来做为定界符 c 。 *- 形式的作用类似，只是同 verbatim* 环境一样，要把空格显示为 ␣ ：\xyz{␣} 。

标准的 LaTeX 2ε宏包 shorvrb（第 217 页）提供了 \verb 命令的一种简凑而方便的版本。

同 verbatim 环境相比，在这里的 源文本 中不能有断行。如果其中的文本长度超过一行，断行点就会被当做空白对待（ LaTeX2.09 ）或者给出一个错误信息（ LaTeX 2ε ）。这个错误信息就会提示你可能忘记用结束的 c 符号了。

重要：无论是 verbatim 环境，还是 \verb 命令都不能用作其它任何命令的参数。

练习 4.18: 从本书中按源文本复制一些文本行。

§4.10 脚注和边注

§4.10.1 标准脚注

脚注是用如下命令生成的：

\footnote{ 脚注文本 }

该命令紧接在需要在脚注中进行解释的单词后面。 脚注文本 用较小的字样以脚注的形式显示在当前页的底部。脚注的第一行要进行缩进，并给出与正文插入点处相同的脚注标记。在一页上的第一个脚注是用一条短水平线与正文分开的。

标准脚注标记是一个小的偏上的数字 [1] ，它是顺序编号的。

[1]在一个用打字机打印的草稿中，通常是用 *, ** 等等来标记脚注，这里也可以使用这种方法；然而，由于当输入文本时并不知道将在哪儿分页，因此这里就存在着如何避免在一页上有相同标记的问题。

... 小的偏上的数字 \footnote{ 在一个用打字机打印的草稿中，通常是用... 有相同标记的问题。 }，它是顺...

在 article 类中，整个文档统一对脚注进行编号，而在 report 和 book 类中，每当开始新的一章时其都会重置为 1 。

\footnote 脚注只可以位于通常的段落模式中，不能位于数学模式或 LR 模式（ 1.5.3 节）中。实际上，这就意味着它不能位于一个 LR 盒子（ 4.7.1 节）或者子段盒子（ 4.7.3 节）中。然而，它却可以用在 minipage 环境中，此时脚注文本显示在小页的下面，而不是实际页面的底部. [2]

\footnote 命令必须紧接在接受这个注释的单词后面，中间不必有任何空格或间隔。一句话的脚注可以在句号后面给出。

§4.10.2 非标准脚注

如果用户希望在 article 类中每当开始新的一节时，脚注编号重置为 1，那可以用如下命令：

```
\setcounter{footnote}{0}
```

就把这条命令放在 \section 命令的前面或后面。

内部脚注记数器的名称为 footnote 。每次调用 \footnote 都会使这个记数器值增 1 ，并以阿拉伯数字形式显示出新值做为脚注的标记。可以用如下命令来实现不同样式的标记：

```
\renewcommand{\thefootnote}{\ 数字样式 {footnote}}
```

这里的 数字样式 就是一个在 4.3.5 节中所描述的记数器显示命令： \arabic, \roman, \Roman, \alph 或 \Alph 。然而，对于 footnote 记数器，还有另一个记数器显示命令可以使用，它就是 \fnsymbol ，它把从 1 到 9 的记数器值显示为如下 9 个符号：

* † ‡ § ¶ ‖ ** †† ‡‡

这里需要用户在调用第十个 \footnote 命令之前把脚注记数器重值为零。

在 \footnote 命令中还可以有一个可省参数：

```
\footnote[ 数 ]{ 脚注文本 }
```

这里的 数 是一个正数，要用它来取代显示标记的脚注记数器的值。在这种情形中，脚注记数器的值并没有增加。例如 **,

```
\renewcommand{\thefootnote}{\fnsymbol{footnote}}
例如 \footnote[7]{ 第七个标记符号。 },
\renewcommand{\thefootnote}{\arabic{footnote}}
```

[2]对于嵌套的小页，脚注会在遇到第一个 \end{minipage} 命令时显示出来，这样会导致出现在错误的地方。

**第七个标记符号。

这里的最后一行是为了把脚注标记样式恢复成标准形式。否则，后面所有的脚注都以符号为标记，而不是用数字。

§4.10.3 禁止模式中的脚注

可以用如下模式在文本中插入一个脚注标记：

```
\footnotemark[ 数 ]
```

即使这里的文本中不允许脚注，例如 LR 盒子，表格和数学模式中。这里的标记就是相应于可以省略的参数 *数* 或者当这个参数省略时脚注记数器增加后的值。脚注本身并没有生成。它必须在禁止模式外面用如下命令生成：

```
\footnotetext[ 数 ]{ 脚注文本 }
```

如果在脚注标记中已用了可省参数，那么在文本命令中也必须用相同的 *数* 做选项。类似地，如果在标记中没有用选项，在文本中也不能出现选项。这儿的脚注生成时将要利用 *数* 的值或者脚注记数器的值。

当 `\footnotemark` 命令没有可省参数时，调用一次会使记数器增 1 。而相应的 `\footnotetext` 命令并不改变记数器的值。

如果在接下来的 `\footnotetext` 命令前面有许多 `\footnotemark` 命令，其都没有带可省参数，那么就需要用如下命令调整记数器的值：

```
\addtocounter{footnote}{ 差 }
```

这里的 *差* 是一个负数，它表示记数器必须被减少多少。因此在每次调用命令 `\footnotetext` 之前，记数器必须增 1 。这就可以用 *差* = 1 的 `\addtocounter` 命令或者

```
\stepcounter{footnote}
```

命令来给记数器增 1 。

For example: mosquitoes[3] and elephants[4]

```
For example: \fbox{mosquitoes\footnotemark\ and
  elephants\footnotemark}
```

生成脚注标记 [3] 和 [4] 。那么现在记数器的值为 4 。为了使有框盒子外面的第一个 `\footnotetext` 相应于正确的记数器值，现在要把它减 1 。这样两个脚注文本就可以如下生成：

```
\addtocounter{footnote}{-1} \footnotetext{Small insects}
\stepcounter{footnote}\footnotetext{Large mammals}
```

把它们就放在 `\fbox{}` 命令的后面。现在脚注记数器的值与它离开 `\fbox` 时的相同。

[3] Small insects
[4] Large mammals

§4.10.4 小页环境中的脚注

在 4.10.1 节中已提到，在小页环境中可以用脚注命令。然而脚注是显示在小页的下面，而不是当前页的底部。

在小页环境 [a] 中的脚注命令有不同的标记样式。脚注会在接下来第一个的 \end{minipage} 命令时显示出来。 [b] 小页环境中的脚注与正文所用的脚注使用不同的记数器，它的记数器称为 mpfootnote，它的记数与 footnote 无关。

\begin{minipage}{6cm}
在小页环境 \footnote{ 标记是偏上的小写字母。 } 中的脚注命令有不同的...

[a] 标记是偏上的小写字母。

[b] 当小页环境嵌套时这有可能导致麻烦。

在表格即 tabular 环境中的脚注通常只能用上面所描述的命令得到：\footnotemark 在表格内部，\footnotetext 在环境外面。然而如果 tabular 环境是在一个小页环境内部，通常的 \footnote 命令也可以在表格内部使用。脚注显示在小页结束时的表格下方。

练习 4.19: 在标准练习文件中的适当地方插入一些脚注，并选择适当的脚注文本。

练习 4.20: 重定义命令 \thefootnote，使得脚注标记变为在 4.10.2 节中所描述的符号。把这个定义加到标准练习文件的导言中。

练习 4.21: 完成练习 4.17，使得在 93 页上的表格中有脚注 [a] 和 [b]。

§4.10.5 脚注样式参数

有两个脚注样式参数可以在需要时进行修改，这种修改既可以在导言中进行，也可以在一个环境中进行。

\footnotesep

两个脚注间的竖直距离。可以用 \setlength 命令修改这个长度。

\footnoterule

这条命令在正文与脚注之间画一条水平线。它并不会增加任何竖直间距。可以修改它的定义，例如，

```
\renewcommand{\footnoterule}
     {\rule{ 宽度 }{ 高度 }\vspace{- 高度 }}
```

当 高度 的值为零时就会生成一个零高度的不可见直线。

§4.10.6 边注

在页边上的注释可以用下面的命令生成：

\marginpar{ 注释文本}

它把 *注释文本* 放在右面的页边上，开始点与命令所在行相平。这里所出现的边注 就是用如下输入得到的：

```
... 这里所出现的边注 \marginpar{ 这是 \\ 一个 \\ 边注 }} 就是 ...
```

这里的 *注释文本* 通常放在一个宽度为 1.9cm(0.75in) 的子段盒子中。这样窄的盒子通常会使得断行非常困难，这也就是上面为什么要用 \\ 命令人为断行的缘故。因此在边注中，这样的盒子对于只有一个符号就是非常合适的，如这里所显示的箭头。

这是
一个
边注
⟸

另一种使用需要使用边注的地方就是在要引起注意的文本旁边画一条竖线。这有时用来表示相对于以前的版本，这部分文本进行了改动。本段中的标记样例就是在第一行中用如下输入得到的：

```
\begin{marginpar}{\rule[-17.5mm]{1mm}{20mm}}
```

边注的宽度可以用下一节所描述的样式参数进行改动。用户必须保证总宽度不能变得超过打印机所接受的限度。

在默认状态下，边注出现在一页的右边，或者当选定了 `twoside` 选项时的外页边。这里的 '外' 指的是奇数页的右页边，偶数页的左页边。当用了 `twocolumn` 选项时，它们就被放在外面的边界上：左列的左边，右列的右边。

但这样做有时会使类似上面给出的箭头这样的页边记号出现问题。如果该页为偶数页，它就必须指向相反的方向。事实上，它的方向是与它位于页的哪一边有关，也就是依赖于页码和列。由于在书写（或者后来对它进行修改）时并不知道它指向它哪个方向。这可以用 `\marginpar` 命令的扩充语法来实现这一点：

```
\marginpar[ 左文本 ]{ 右文本 }
```

这种形式的命令包含页边文本的两个版本， *左文本* 会出现在左边， *右文本* 会在出现在右边，具体视需要哪个而定。因此为了全面起见，排版上面那个箭头的边注命令应该是

```
\marginpar[\hfill$\Longrightarrow$]{$\Longleftarrow$}
```

（箭头命令是数学符号，在 5.3.5 节对它们进行详细介绍。）

在上面的 `\marginpar` 例子中若没有用 `\hfill` 命令，那么位于左边界上的箭头就会显示在其所在段的左边，从而离正文很远。之所以会这样，是因为 `\marginpar` 命令把它的内容左对齐处理，在下一页上的一个有框边注就很好地说明了这一点：靠左边对齐正文的方式只适用于边注位于右边界的情形；如果它位于左边界，就会离正文太远。`\hfill` 命令就是为了使得边注内容向右边对齐，从而使得它处在相对于正文比较恰当的位置上。

对于这一节第一个边注，也需要用同样的方法来处理。实际生成它的命令是

```
\marginpar[\flushright 这是 \\ 一个 \\ 边注 ]{ 这是 \\ 一个 \\ 边注
}
```

这里的 \flushright（ 4.2.2 节）就等价于在每一行上放一个 \hfill 。

边注的标准位置可以用 \reversemarginpar 命令来进行变换。一旦使用了这个命令，边注就会位于左边，或者在 \twoside 选项时出现在内边界。这个命令的作用直到出现相反命令 \normalmarginpar 为止。对于 twocolumn 命令，这两个命令没有作用。

在边注中不能进行分页。如果一个边注太靠近页面底部，从而没有足够空间放下它，那边注就会延伸到最后一行的下面。在这种情形中，\marginpar 命令中的文本开头应包含一个 \vspace 命令，以使得边注向上移动一点儿，或者就用两条 \marginpar 命令把它分成两部分，分放在不同的页上。这种手工调整只有在整篇文档完成后才能进行，因为以后对文档进行改动就会使情形发生变化。

§4.10.7 边注的样式参数

可以改动下面的样式参数，以重定义边注的显示方式：

\marginparwidth
: 定义边注盒子的宽度；

\marginparsep
: 设置边注盒子与正文边界之间的距离；

\marginparpush
: 两个边注盒子之间的最小竖直距离。

这些参数都是长度，可以用 \setlength 命令像通常那样赋予一个新值。

§4.11 正文中的注释

在正文中若有注释、解释、提示等等的东西，对于告诉作者或者其它后来由于某一目的进行某种构造的用户而言，是非常有用的。这些注释不与其它的正文一样被格式化。

在 TEX中，单个字符命令 % 就引进了注释。当这个字符出现在正文中，它自己本身以及该行后面的其它文本都要被忽略。如果一个注释有几行长，那么每行都必须前缀 % 。

注释符号 % 对暂时抑制某些命令也是很有用的。把一个 % 放在这样一条命令的前面，那么就会使该行后面部分都被忽略。这称为 *注释抑制行* 。类似地，在源文件中可能有不同的替换文本，因此可以注释掉一些行，这样作者可以决定要用哪一版本。

最后提一下，% 符号对去掉每行结尾的那个隐含空格有相当重要的作用。在用户定义中当希望在两个参数间不要由于分行而引进不必要的空格时，它是特别有用的。

©TEXGuru, August 15, 1999

练习 4.22: 注释掉在练习 4.20 中对导言做的改动。以后可以通过去掉 % 符号来重新激活这些命令。

©TEXGuru, August 15, 1999

第五章　数学公式

数学是 TEX的灵魂。就是由于排版数学公式是那么得复杂，在通常的打字机上根本无法进行， Donald Knuth 才开发这个文本格式化系统。另一方面，LATEX的灵魂是文档设计。不但如此，所有 TEX的强大数学排版功能 LATEX也都具备，而且提供了相当好的组合。我们在本章所讲的绝大部分（并不是全部）内容都适用于 TEX ，因此要把它们两者区分开，有时是很困难的。我们因此经常指出命令和环境是属于 LATEX的，即使它也同样适用于 TEX 。

数学公式是通过输入特殊的描述性文本来生成的。这就意味着必须告诉LATEX要把下面的文本解释成一个数学公式，而且也能告诉它数学公式已结束，返回正常的文本状态。数学文本的处理是通过切换进入数学模式（ 1.5.3 节）实现的。数学环境就是为了这个目标而引进的。

§5.1　数学环境

数学公式可以出现在一个文本行中，如 $(a+b)^2 = a^2 + 2ab + b^2$ ，也可以与正文分开，如

$$\int_0^\infty f(x)\,dx \approx \sum_{i=1}^{n} w_i e^{x_i} f(x_i)$$

这两种形式分别被称为 *正文公式* 和 *显示公式* 。

正文公式或者方程是用如下环境生成的：

`\begin{math}` *公式文本* `\end{math}`

由于正文公式通常都很短，有的时候只有一个字符，因此也可以用一个更方便的版本，即 `\(` *公式文本* `\)` 。如果还嫌它长，那可以用更短的形式 `$` *公式文本* `$` 。所有这三种形式是等价的，虽然内部具有某些差别，如 `\(` 是脆弱的，而 `$` 就比较牢靠。

公式的内容 *公式文本* 是由数学构造组成，我们将在下面几节中介绍这些构造。

显示公式或方程是用下面的环境生成的：

`\begin{displaymath}` *公式文本* `\end{displaymath}`

`\begin{equation}` *公式文本* `\end{equation}`

这两种环境的差别在于 `equation` 环境会自动给公式加上一个顺序的公式编号。 `displaymath` 环境也可以用方便形式 `\[`*公式文本*`\]` 给出。

在默认方式下，显示公式是水平居中的，而且如果有公式编号的话，编号会显示在右页边。通过选择文档类选项 `fleqn`（ 3.1.1 节），公式就会左对齐，而且可以具有一个能调整的缩进。这个选项在整篇文档中有效，而缩进量可以用命令 `\setlength{\mathindent}{`*缩进量*`}` 来改变，这里的 *缩进量* 就

是所要定义的长度。除此之外，文档类选项 leqno 会使得整篇文档中公式编号显示在左边界。

最后提一下，可以用如下环境创建多行公式：

```
\begin{eqnarray}  公式文本 \end{eqnarray}
\begin{eqnarray*} 公式文本 \end{eqnarray*}
```

这里的标准形式会给每行公式都加上一个顺序的公式编号，而 *- 形式则没有公式编号。

§5.2 数学公式的主要组成

§5.2.1 常量与变量

出现在公式中的数字称为常量，而简单变量只由一个字母表示。在绝大多数的数学排版中是用罗马字样显示常量，用斜体显示变量。LaTeX在数学模式中也是自动遵守这个规则。在源文本中为了使作者容易读而加的空格都被忽略。在常量、变量和类似于 $+, -, =$ 这样的动算符之间的距离是由 LaTeX自动处理的。例如 `$z=2a+3y$` 和 `z = 2 a + 3 y $` 都生成 $z = 2a + 3y$ 。

在键盘上存在对应字符的数学符号有：

```
+ - = < > / : ! ’ | [ ] ( )
```

它们都可以直接用在数学公式中。大括号 { } 用来表示公式的逻辑组合，因此不能作为可直接显示的字符。为了在公式中显示大括号，就必须如通常文本中那样用命令 \{ 和 \} 。

$M(s) < M(t) < |M| = m$ `$M(s)<M(t)<|M| = m$`

$y'' = c\{f[y', y(x)] + g(x)\}$ `$y''=c\{f[y',y(x)] + g(x)\}$`

准备工作：创建一个新的 LaTeX文件，名称为 math.tex ，它只是由 \documentclass{article} ， \begin{document} 和 \end{document} 命令组成。

练习 5.1: 用你自己的练习文件生成下面的文本：‘The derivative of the indirect function $f[g(x)]$ is $\{f[g(x)]\}' = f'[g(x)]g'(x)$. For the second derivate of the product of $f(x)$ and $g(x)$ one has $[f(x)g(x)]'' = f''(x)g(x) + 2f'(x)g'(x) + f(x)g''(x)$.’

注意：高阶导数是用多个 ’ 符号生成的： `$y'''$` 的结果为 y''' 。

§5.2.2 指数和指标

在数学公式中经常可以见到指数和指标，即把字符相对于公式的主要文本所在行进行提升或下降，并以小号字体显示出来。虽然它们的数学意义可能不同，上标和下标在印刷上分别与指数和指标是完全一样的。甚至指数本

身还可以有指数或指标，依此类推。它们只是提升或下降操作的多次运用而已。

LaTeX和 TeX以一种简单的方式，提供了以适当字体尺寸创建任意指数和指标组合的方法：字符命令 ^ 把紧接下来的字符做为指数（提升），而字符命令 _ 把紧接下来的字符做为指标（下降）。

x^2 `x^2` a_n `a_n` x_i^n `x^n_i`

当指数和指标一起出现时，它们的顺序是无关紧要的。上面最后那个例子也可以用 `x_i^n` 来生成。

如果指数或指标的内容不只一个字符，那么就必须用大括号 { } 把这组符号包围起来：

x^{2n} `x^{2n}` x_{2y} `x_{2y}` $A_{i,j,k}^{-n!2}$ `A_{i,j,k}^{-n!2}`

也可以在指数和指标中多次进行提升或下降操作：

x^{y^2} `x^{y^2}` x^{y_1} `x^{y_1}`

$A_{j^{2n}_{n,m}}^{x_i^2}$ `A^{x_i^2}_{j^{2n}_{n,m}}`

注意：提升和下降命令 ^ 和 _ 只能用在数学模式中。

§5.2.3 分数

比较小的分数，尤其是正文公式中的分数最好用斜杠字符 / 表示，如 `$(n+m)/2$` 表示 $(n+m)/2$。对于较复杂的分数，命令

`\frac{`*分子*`}{`*分母*`}`

可以用来把 *分子* 放在 *分母* 上面，而且中间有一适当长度的水平线。

$$\frac{1}{x+y}$$ `\[ \frac{1}{x+y} \]`

$$\frac{a^2-b^2}{a+b}=a-b$$ `\[ \frac{a^2 - b^2} {a+b} = a-b \]`

分数也可以嵌套至任何层次：

$$\frac{\frac{a}{x-y}+\frac{b}{x+y}}{1+\frac{a-b}{a+b}}$$

```
\[ \frac{\frac{a}{x-y} +
\frac{b}{x+y}}
{1+ \frac{a-b}{a+b}} \]
```

LaTeX把分数中的分数有较小的字样显示出来。在 5.5.2 节中介绍了当LaTeX的选择不很好时，如何修改这种自动字样尺寸。

§5.2.4 方根

方根是用如下命令显示的：

`\sqrt[`*开方数*`]{`*参数*`}`

例如： `$\sqrt[3]{8} = 2$` 生成 $\sqrt[3]{8}=2$。如果不写可省参数 *开方数*，就会生成平方根： `$\sqrt{a}$` 得到 $\sqrt{a}$。

方根符号的尺寸和长度是自动与 *参数* 大小匹配的：

`$\sqrt{x^2 + y^2 + 2xy} = x+y$` $\sqrt{x^2+y^2+2xy}=x+y$，或者

$\sqrt[n]{\frac{x^n - y^n}{1 + u^{2n}}}$ `\[ \sqrt[n]{\frac{x^n - y^n}{1 + u^{2n}}} \]`

方根也可以嵌套至任何层次：

$\sqrt[3]{-q + \sqrt{q^2 + p^3}}$ `\[ \sqrt[3]{-q + \sqrt{q^2 + p^3}} \]`

§5.2.5 求和与积分

求和与积分符号是用命令 `\sum` 与 `\int` 生成的，根据其所在的是正文公式还是显示公式，它们可以有两个不同的大小。

求和与积分通常都有上下限。这可以用指数和指标命令 `^` 和 `-` 来得到。上下限的位置也要看其所在的是正文公式还是显示公式而定。

在一个正文公式中 `$\sum_{i=1}^n$` 和 `$\int_a^b$` 的结果是 $\sum_{i=1}^n$ 和 $\int_a^b$，而在显示公式中它们的形状如下面左边所示：

$$\sum_{i=1}^{n} \int_{a}^{b}$$

而有些作者喜欢把积分的上下限也放在积分号的上方和下方，就如同求和符号中那样。这可以在积分符号后面紧接 `\limits` 命令来得到：`\int\limits_{x=0}^{x=1}`

$$\int\limits_{x=0}^{x=1}$$

在求和与积分符号前后的其它公式文本会与它们适当对齐的。

$$2\sum_{i=1}^{n} a_i \int_{a}^{b} f_i(x)g_i(x)\,dx$$

```
\[ 2\sum_{i=1}^n a_i \int^b_a
                    f_i(x)g_i(x)\,dx \]
```

在类似于 $\int y\,dx$ 和 $\int f(z)\,dx$ 这样的积分中，微分运算符 dx 和 dz 应与前面的被积函数之间有很小的空档。只是输入一个空格，并不能实现这个目标，因为在数学模式中空格都被忽略：`$\int y dx$` 的结果为 $\int ydx$。我们可以用在 3.5.1 节中提到的小间距命令 `\,` 来做到这一点。因此 `$\int y\,dx$` 和 `$\int f(z)\,dz$` 会得到所希望的结果 $\int y\,dx$ 和 $\int f(z)\,dz$。在 5.5.1 节中给出了更多的可用于数学公式中的间距命令。

§5.2.6 连续点 — 省略号

公式中有时会包含一串点 ...，表示 等等。若只是简单地在一行中输入三个句号会得到不是想要的结果：...，即点靠得太近了。因此 LaTeX提供了几条命令：

`\ldots` $\ldots$ 偏下的点 `\cdots` $\cdots$ 中间点

`\vdots` $\vdots$ 竖直点 `\ddots` $\ddots$ 对角点

以生成正确间距的点。最好地说明前两条命令差别的例子是：$a_0, a_1, \ldots, a_n$ 和 $a_0 + a_1 + \cdots + a_n$，它们分别是用 `$a_0, a_1, \ldots, a_n$` 和 `$a_0 + a_1 + \cdots + a_n$` 生成的。

`\ldots` 命令也可以用在普通的文本模式中，而其余三条命令则只能用在数学模式中。在文本模式中，`\dots` 命令可以代替 `\ldots`，它们的作用一

样。

练习 5.2: 生成下面的结果：

The reduced cubic equation $y^3+3py+2q=0$ has one real and two complex solution when $D=q^2+p^3>0$. These are given by Cardan's formula as

$$y_1=u+v,\quad y_2=-\frac{u+v}{2}+\frac{i}{2}\sqrt{3}(u-v),\quad y_3=-\frac{u+v}{2}-\frac{i}{2}\sqrt{3}(u-v)$$

where

$$u=\sqrt[3]{-q+\sqrt{q^2+p^3}},\qquad v=\sqrt[3]{-q-\sqrt{q^2+p^3}}$$

注意：在显示公式中不同部分之间的空档是用空档命令 `\quad` 和 `\qquad` 得到的。

练习 5.3: 选择文档类选项 `fleqn` , 并把定义 `\setlength{\mathindent}{2cm}` 放在导言中。重新排版上面的三个 y 公式，每个都利用 `equation` 环境单独做为一个显示公式，而不是用这里的 `displaymath` 或 `\[...\]` 括号。

练习 5.4: 生成下列的文本：

Each of the measurements $x_1<x_2<\cdots<x_r$ occurs $p_1,p_2,\ldots,p_r$ times. The mean value and standard deviation are then

$$x=\frac{1}{n}\sum_{i=1}^{r}p_ix_i,\qquad s=\sqrt{\frac{1}{n}\sum_{i=1}^{r}p_i(x_i-x)^2}$$

where $n=p_1+p_2+\cdots+p_r$.

练习 5.5: 虽然下面这个公式看起来非常复杂，但得到它不会有任何困难：

$$\int\frac{\sqrt{(ax+b)^3}}{x}dx=\frac{2\sqrt{(ax+b)^3}}{3}+2b\sqrt{ax+b}+b^2\int\frac{dx}{x\sqrt{ax+b}}$$

同样生成公式 $\int_{-1}^{8}(dx/\sqrt[3]{x})=\frac{3}{2}(8^{2/3}+1^{2/3})=15/2$ 。

§5.3 数学符号

在数学文本中有相当多的符号，其中只有很少一部分可以直接从键盘上输入得到。LaTeX提供了通常使用的几乎所有可以想像得到的数学符号。可以用符号名称前缀命令字符 \ 来得到。而它们的名称就是来自于其数学含义。

§5.3.1 希腊字母

小写字母

α	\alpha	θ	\theta	o	o	τ	\tau
β	\beta	ϑ	\vartheta	π	\pi	υ	\upsilon
γ	\gamma	ι	\iota	ϖ	\varpi	ϕ	\phi
δ	\delta	κ	\kappa	ρ	\rho	φ	\varphi
ϵ	\epsilon	λ	\lambda	ϱ	\varrho	χ	\chi
ε	\varepsilon	μ	\mu	σ	\sigma	ψ	\psi
ζ	\zeta	ν	\nu	ς	\varsigma	ω	\omega
η	\eta	ξ	\xi				

大写字母

Γ	\Gamma	Λ	\Lambda	Σ	\Sigma	Ψ	\Psi
Δ	\Delta	Ξ	\Xi	Υ	\Upsilon	Ω	\Omega
Θ	\Theta	Π	\Pi	Φ	\Phi		

要得到希腊字母，只需要在字母名称前面加上命令字符 \ 就可以了。大写字母的区别在于名称的第一个字母是大写的。没有列在上面清单的希腊字母一定是与某一个拉丁字母一样。例如，大写的 ρ 与拉丁字母的 P 一样，因此就没有特殊符号。

在数学公式中的大写希腊字母通常用的是罗马（直立）字样。如果需要斜体字样，可以用数学字体样式命令 \mathnormal[2ε]来得到：

`$\mathnormal{\Gamma\Pi\Phi}$` 生成 $\mathnormal{\Gamma\Pi\Phi}$。

（这条命令取代了用在 LaTeX2.09 中的数学字样声明 \mit[2.09]，原来的用法是 `$\mit \Gamma\Pi\Phi$`。）

希腊字母只能用在数学模式中。如果要用在普通文本中，那么必须把命令用 `$...$` 括起来。

§5.3.2 花体字母

在数学公式中也可以用 26 个花体字母：

$\mathcal{A},\mathcal{B},\mathcal{C},\mathcal{D},\mathcal{E},\mathcal{F},\mathcal{G},\mathcal{H},\mathcal{I},\mathcal{J},\mathcal{K},\mathcal{L},\mathcal{M},\mathcal{N},\mathcal{O},\mathcal{P},\mathcal{Q},\mathcal{R},\mathcal{S},\mathcal{T},\mathcal{U},\mathcal{V},\mathcal{W},\mathcal{X},\mathcal{Y},\mathcal{Z}$

调用方法是用数学字体样式命令 \mathcal[2ε]：

```
$\mathcal{A,B,C,...,Z}$
```

（也可以用等价的声明 \cal[2.09]）。

§5.3.3 二元运算符

数学中通常把两个量用一个符号组合起来，生成一个新的量，这个符号称为二元运算符。可以用做二元运算的符号有：

±	\pm	∩	\cap	∘	\circ	◯	\bigcirc
∓	\mp	∪	\cup	•	\bullet	□	\Box
×	\times	⊎	\uplus	⋄	\diamond	◇	\Diamond
÷	\div	⊓	\sqcap	◁	\lhd	△	\bigtriangleup
·	\cdot	⊔	\sqcup	▷	\rhd	▽	\bigtriangledown
∗	\ast	∨	\vee	⊴	\unlhd	◃	\triangleleft
⋆	\star	∧	\wedge	⊵	\unrhd	▷	\triangleright
†	\dagger	⊕	\oplus	⊘	\oslash	\	\setminus
‡	\ddagger	⊖	\ominus	⊙	\odot	≀	\wr
⨿	\amalg	⊗	\otimes				

注意：上面以及后面列表中名称加下划线的符号只能用在 LaTeX 2ε中，而且需要调用软件包 latexsym（ 8.8.3 节）。

§5.3.4 关系运算符及其否定

当要比较两个数学量时，就要用关系运算符把它们连接起来。用于各种比较中不同类型关系运算符有：

≤	\le	\leq	≥	\ge	\geq	≠	\neq	∼	\sim
≪	\ll		≫	\gg		≐	\doteq	≃	\simeq
⊂	\subset		⊃	\supset		≈	\approx	≍	\asymp
⊆	\subseteq		⊇	\supseteq		≅	\cong	⌣	\smile
⊏	\sqsubset		⊐	\sqsupset		≡	\equiv	⌢	\frown
⊑	\sqsubseteq		⊒	\sqsupseteq		∝	\propto	⋈	\bowtie
∈	\in		∋	\ni		≺	\prec	≻	\succ
⊢	\vdash		⊣	\dashv		⪯	\preceq	⪰	\succeq
⊨	\models		⊥	\perp		∥	\parallel \\|	∣	\mid |

在上面的符号中有几个可以不只用一个名称调用。例如， ≤ 可以用 \le 或 \leq 生成。

而关系运算符的相反或者否定在数学中是用一个斜杠贯穿符号而得到的：如 = 和 ≠ 表示等于和不等于。而 ≠ 有一个特殊的命令 \neq 。但是我们可以在上面的符号名称前面用 \not，使斜线贯穿该符号。因此 \not\in 得到 ∉ 。这同样也适用于键盘字符，如 \not=, \not> 和 \not< 的结果为 ≠, ≯ 和 ≮ 。

这样下面的符号可以如此否定。注意最后两个符号 \not\in 和 \notin 并不是一样的： ∉ 和 ∉ 。后者比前者更好看些。

$\not<$	\not<	$\not>$	\not>	$\neq$	\not=
$\not\leq$	\not\le	$\not\geq$	\not\ge	$\not\equiv$	\not\equiv
$\not\prec$	\not\prec	$\not\succ$	\not\succ	$\not\sim$	\not\sim
$\not\preceq$	\not\preceq	$\not\succeq$	\not\succeq	$\not\simeq$	\not\simeq
$\not\subset$	\not\subset	$\not\supset$	\not\supset	$\not\approx$	\not\approx
$\not\subseteq$	\not\subseteq	$\not\supseteq$	\not\supseteq	$\not\cong$	\not\cong
$\not\sqsubseteq$	\not\sqsubseteq	$\not\sqsupseteq$	\not\sqsupseteq	$\not\asymp$	\not\asymp
$\notin$	\not\in	$\notin$	\notin		

§5.3.5 箭头与指针

在数学文稿中通常会有箭头符号，它也称为指针。可用的箭头符号有下面这些：

$\leftarrow$	\leftarrow	\gets	$\longleftarrow$	\longleftarrow	$\uparrow$	\uparrow
$\Leftarrow$	\Leftarrow		$\Longleftarrow$	\Longleftarrow	$\Uparrow$	\Uparrow
$\rightarrow$	\rightarrow	\to	$\longrightarrow$	\longrightarrow	$\downarrow$	\downarrow
$\Rightarrow$	\Rightarrow		$\Longrightarrow$	\Longrightarrow	$\Downarrow$	\Downarrow
$\leftrightarrow$	\leftrightarrow		$\longleftrightarrow$	\longleftrightarrow	$\updownarrow$	\updownarrow
$\Leftrightarrow$	\Leftrightarrow		$\Longleftrightarrow$	\Longleftrightarrow	$\Updownarrow$	\Updownarrow
$\mapsto$	\mapsto		$\longmapsto$	\longmapsto	$\nearrow$	\nearrow
$\hookleftarrow$	\hookleftarrow		$\hookrightarrow$	\hookrightarrow	$\searrow$	\searrow
$\leftharpoonup$	\leftharpoonup		$\rightharpoonup$	\rightharpoonup	$\swarrow$	\swarrow
$\leftharpoondown$	\leftharpoondown		$\rightharpoondown$	\rightharpoondown	$\nwarrow$	\nwarrow
$\rightleftharpoons$	\rightleftharpoons		$\leadsto$	\leadsto		

这里符号 $\rightarrow$ 和 $\leftarrow$ 也可用名称 \to 和 \gets 来调用。而 \Longleftrightarrow 命令也可以用 \iff 来代替，但是后者 ($\Longleftrightarrow$) 与前者 ($\Longleftrightarrow$) 相比，在两边要多一点空档。

§5.3.6 其它各类符号

在 LaTeX中无疑包括了在数学文本中所有可能出现的符号。然而，不但如此，LaTeX还提供了下面这些符号。(这里与前面的表格有些重复，之所以如此，就是为了使相关符号列在一起。)

$\aleph$	\aleph	$\prime$	\prime	$\forall$	\forall	$\Box$	\Box
$\hbar$	\hbar	$\emptyset$	\emptyset	$\exists$	\exists	$\Diamond$	\ Diamond
$\imath$	\imath	∇	\nabla	$\neg$	\neq	$\triangle$	\triangle
$\jmath$	\jmath	$\surd$	\surd	$\flat$	\flat	$\clubsuit$	\clubsuit
ℓ	\ell	∂	\partial	$\natural$	\natural	$\diamondsuit$	\diamondsuit
$\wp$	\wp	$\top$	\top	$\sharp$	\sharp	$\heartsuit$	\heartsuit
$\Re$	\Re	$\bot$	\bot	$\|$	\\|	$\spadesuit$	\spadesuit
$\Im$	\Im	$\vdash$	\vdash	$\angle$	\angle	$\bowtie$	\Join
$\mho$	\mho	$\dashv$	\dashv	\	\backslash	∞	\infty

©TEXGuru, August 15, 1999

§5.3.7 具有两种尺寸的符号

根据所处为正文公式还是显示公式，下列符号会以不同的大小显示出来：

$\sum$	$\sum$	\sum	$\cap$	$\bigcap$	\bigcap	$\odot$	$\bigodot$	\bigodot
$\int$	$\int$	\int	$\cup$	$\bigcup$	\bigcup	$\otimes$	$\bigotimes$	\bigotimes
$\oint$	$\oint$	\oint	$\sqcup$	$\bigsqcup$	\bigsqcup	$\oplus$	$\bigoplus$	\bigoplus
$\prod$	$\prod$	\prod	$\vee$	$\bigvee$	\bigvee	$\uplus$	$\biguplus$	\biguplus
$\coprod$	$\coprod$	\coprod	$\wedge$	$\bigwedge$	\bigwedge			

我们在 5.2.5 节中已介绍了符号 \sum 和 \int 。在那里我们演示了如何给这两个符号加上下限；同样，用移位命令 ^ 和 _ 也可以给上面所有符号加上下限。有些符号的上下限位置会视所处为正文公式还是显示公式而发生变化。正如在 5.2.5 节中指出的那样，如果上下限只是放在符号旁边时，可以用命令 \limits 可强迫上下限放在符号的上方和下方。类似地，当上下限的标准位置是上方和下方时，可以用相反命令 \nolimits 使得上下限只是位于符号的旁边。

$\oint_0^\infty$ $\oint\limits_0^\infty$ `\[ \oint^\infty_0 \oint\limits^\infty_0 \]`

$\prod_{\nu=0}^{n}$ $\prod\nolimits_{\nu=0}^{n}$ `\[ \prod^n_{\nu=0} \prod\nolimits^n_{\nu=0} \]`

§5.3.8 函数名

在数学公式中普遍使用的标准是把用斜体显示变量，而用罗马字体显示函数名。如果我们在数学模式中只是简单地写出函数名 *sin* 或 *log*， LaTeX就会把认为它们是变量 s i n 和 l o g，从而显示为 *sin* 和 *log* 。为了告诉LaTeX我们需要的是一个函数名，那就需要在函数名前面加上命令字符 \ 。LaTeX接受下面这些函数：

```
\arccos  \cosh  \det  \inf     \limsup  \Pr    \tan
\arcsin  \cot   \dim  \ker     \ln      \sec   \tanh
\arctan  \coth  \exp  \lg      \log     \sin
\arg     \csc   \gcd  \lim     \max     \sinh
\cos     \deg   \hom  \liminf  \min     \sup
```

在这些函数中有几个也可以在显示时带上(下)限。这只要在函数名后面接指标命令就可以了：如 \lim_{x \to \infty } 在正文模式中是 $\lim_{x\to\infty}$，而在显示模式中是 $\lim\limits_{x\to\infty}$ 。

下面这些函数名可以用指标命令 _ 加上一个下限：

```
\det   \gcd   \inf   \lim  \liminf  \limsup  \max  \min
```

```
\Pr     \sup
```

最后要提一点，函数命令 \bmod 和 \pmod{参数} 都生成函数 mod，但却是两种形式：

```
$ a \bmod b $          a \bmod b    或者
$ y \pmod{a+b} $       y  (\mod a+b) 。
```

§5.3.9 数学重音

在数学模式中可以用下面这些数学重音：

$\hat{a}$	hat\{a}	$\breve{a}$	\breve{a}	$\grave{a}$	\grave{a}	$\bar{a}$	\bar{a}
$\check{a}$	\check{a}	$\acute{a}$	\acute{a}	$\tilde{a}$	\tilde{a}	$\vec{a}$	\vec{a}
$\dot{a}$	\dot{a}	$\ddot{a}$	\{a}				

当要给字母 i 和 j 要重音时，应该去掉它们的点。为此，需要用符号 \imath 和 \jmath 代替直接字母输入，如

```
$\vec{\imath} + \tilde{\jmath}$:   \vec{\imath} + \tilde{\jmath}
```

对于 \hat 和 \tilde，还存在一种宽的版本，名称为分别是 \widehat 和 \widetilde 。这两种符号可以被放在一个公式上：

```
$\widehat{1-x}=\widehat{-y}$   1-x=-y
$\widetilde{xyz}$              xyz
```

练习 5.6: 两个集合 $\mathcal{A}$ 和 $\mathcal{B}$ 的并就是所有至少在其中一个集合中的元素全体，并 $\mathcal{A}\cup\mathcal{B}$ 表示。这种操作是可交换的，即 $\mathcal{A}\cup\mathcal{B}=\mathcal{B}\cup\mathcal{A}$，也是可结合的，即 $(\mathcal{A}\cup\mathcal{B})\cup\mathcal{C}=\mathcal{A}\cup(\mathcal{B}\cup\mathcal{C})$ 。如果 $\mathcal{A}\subseteq\mathcal{B}$，那么 $\mathcal{A}\cup\mathcal{B}=\mathcal{B}$ 。而且有 $\mathcal{A}\cup\mathcal{A}=\mathcal{A}, \mathcal{A}\cup\{\emptyset\}=\mathcal{A}$ 和 $\mathcal{J}\cup\mathcal{A}=\mathcal{J}$ 。

练习 5.7: 应用 l'Hôpital 法则，我们有：

$$\lim_{x\to 0}\frac{\ln\sin\pi x}{\ln\sin x}=\lim_{x\to 0}\frac{\pi\frac{\cos\pi x}{\sin\pi x}}{\frac{\cos x}{\sin x}}=\lim_{x\to 0}\frac{\pi\tan x}{\tan\pi x}=\lim_{x\to 0}\frac{\pi/\cos^2 x}{\pi/\cos^2\pi x}=\lim_{x\to 0}\frac{\cos^2\pi x}{\cos^2 x}=1$$

练习 5.8: Gamma 函数 $\Gamma(x)$ 的定义为

$$\Gamma(x)\equiv\lim_{n\to\infty}\prod_{\nu=0}^{n-1}\frac{n!n^{x-1}}{x+\nu}=\lim_{n\to\infty}\frac{n!n^{x-1}}{x(x+1)(x+2)\cdots(x+n-1)}\equiv\int_0^\infty e^{-t}t^{x-1}dt$$

这里的积分只有当 $x>0$ 时才有意义 (第二 Euler 积分)。

练习 5.9: 从练习 5.3 中的文档类选项中去掉 fleqn 选项，重新得到输出。

练习 5.10:

$$\alpha\vec{x}=\vec{x}\alpha,\quad \alpha\beta\vec{x}=\beta\alpha\vec{x},\quad (\alpha+\beta)\vec{x}=\alpha\vec{x}+\beta\vec{x},\quad \alpha(\vec{x}+\vec{y})=\alpha\vec{x}+\alpha\vec{y},$$

$\vec{x}\vec{y}=\vec{y}\vec{x}$, 但是 $\vec{x}\times\vec{y}=-\vec{y}\times\vec{x}$, $\vec{x}\perp\vec{y}$ 时 $\vec{x}\vec{y}=0$, $\vec{x}\|\vec{y}$ 时 $\vec{x}\times\vec{y}=0$ 。

练习 5.11: 生成下节中的公式 5.1 和 5.2 。

§5.4 其它要素

在前几节中描述的数学要素对于构造下面这样复杂的数学公式已是绰绰有余了：

$$\lim_{x\to 0}\frac{\sqrt{1+x}-1}{x}=\lim_{x\to 0}\frac{(\sqrt{1+x}-1)(\sqrt{1+x}+1)}{x(\sqrt{1+x}+1)}=\lim_{x\to 0}\frac{1}{\sqrt{1+x}+1}=\frac{1}{2} \tag{5.1}$$

$$\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}=0 \implies U_M=\frac{1}{4\pi}\oint\limits_{\Sigma}\frac{1}{r}\frac{\partial U}{\partial n}ds-\frac{1}{4\pi}\oint\limits_{\Sigma}\frac{\partial\frac{1}{r}}{\partial n}U\,ds \tag{5.2}$$

$$I(z)=\sin(\frac{\pi}{2}z^2)\sum_{n=0}^{\infty}\frac{(-1)^n\pi^{2n}}{1\cdot 3\cdots(4n+1)}z^{4n+1}-\cos(\frac{\pi}{2}z^2)\sum_{n=0}^{\infty}\frac{(-1)^n\pi^{2n+1}}{1\cdot 3\cdots(4n+3)}z^{4n+3} \tag{5.3}$$

按照从左到右的顺序看公式，那么构造出生成它们的文本应该是没有什么困难的。例如，最后一个公式就是用如下输入生成的：

```
\begin{equation}
I(z) = \sin( \frac{\pi}{2} z^2 ) \sum_{n=0}^\infty
  \frac{ (-1)^n \pi^{2n} }{ 1 \cdot 3 \cdots (4n+1) } z^{4n+1}
 -\cos( \frac{\pi}{2} z^2) \sum_{n=0}^\infty
  \frac{ (-1)^n \pi^{2n+1} }{ 1 \cdot 3 \cdots (4n+3) } z^{4n+3}
\end{equation}
```

上面的公式不是用 `displaymath` 环境或者其简写形式 `\[...\]` 生成，而用的是 `equation` 环境，这样它就会自动给公式加上编号。在文档类 `book` 和 `report` 中，公式是在章内顺序编号的，如上所示，编号前面有章号，并且把它们一起放在小括号 () 内。而对于文档类 `article`，公式是相对于整篇文档编号的。

在默认状态下，公式编号右对齐，而且相对于公式竖直居中。如果在公式行上没有足够空间显示它，那就把它放在公式下一行的右边。如果选择了文档类选项 `leqno`，那么整篇文档的公式编号是左对齐的。

公式的自动编号就意味着作者在写作时并不一定知道编号到底是多少。在 8.3.1 节中介绍的 LaTeX交叉索引系统讲解了如何引用章节编号(8.3.3 节)，这也同样适用于公式编号。通过在 `equation` 环境中包含一条命令 `\label{`*引用名*`}`，我们可以在正文中用命令 `\ref{`*引用名*`}` 来显示还不知道的公式编号，这里的 *引用名* 是一个关键词，它是字母、数字和符号的任意组合。

再仔细地看一下公式 5.3，你会发现在 cos() 和 sin() 中的括号 () 应该再大点。而且这个公式长度恰好等于行宽，如果它再长一点儿，那就需要在某个恰当的地方把它断开，后面那部分要相对于前面一行适当地定位。到现在为止我们所学的数学要素还没有提供这方面的功能。

即使是类似于如何在公式中包含普通文本这样简单的事情，我们到现在也没讲。这一节的其它部分就是为了解决这些问题的。

最后要提一下，有时候 TEX选择的尺寸并不总能令人满意，如公式 5.2 的后一个积分中，如果显示的是 $\partial\frac{1}{r}$，就要比 $\partial\frac{1}{r}$ 好看得多。我们在 5.5 节中会讲解这一点和其它的格式辅助工具，如怎样调整公式两部分之间的水平距离。

§5.4.1 括号符号的尺寸自动调整

在数学中经常包含括号符号，通常是成对出现的，用来包围公式的某部分。当显示的时候，这些括号应该与被包围公式有相同的尺寸。LATEX提供了一对命令

`\left` *左括号* *部分公式* `\right` *右括号*

用来做到这一点。把命令 `\left` 就放在 *左括号* 符号的前面，而 `\right` 就放在 *右括号* 符号的前面。

```
\[ \left[ \int + \int \right]_{x=0}^{x=1} \]
```

$$\left[\int+\int\right]_{x=0}^{x=1}$$

这里的一对括号 `[ ]` 会根据被包围公式自动调整其尺寸，从而导致指数和指标也相应地长高或除低。

`\left` 和 `\right` 命令必须成对出现。每一个 `\left` 命令必须在后面某处有一个相应的 `\right` 命令。这种匹配也可以嵌套。第一个 `\left` 是与最后一个 `\right` 配对的；接下来的 `\left` 与倒数第二个 `\right` 配对，依次类推。在一个嵌套中必须有相同数目的 `\right` 和 `\left` 。

相对应的 *左括号* 和 *右括号* 符号可以任意组合，并不一定要是逻辑上的一对：

这种括号的配对当然是不寻常的，但是可以接受：

```
\[ \vec{x} + \vec{y} + \vec{z} =
    \letf( ... \right[ \]
```

$$\vec{x}+\vec{y}+\vec{z}=\left(\begin{array}{c} a \\ b \end{array}\right[$$

有的时候公式中只有左括号或右括号，而没有相配对的部分。然而即使是这种情形，`\left ... \right` 命令也必须成对出现，只是用‘.’来表示那个看不见的括号符号：

```
\[ y = \left\{ \begin{array}
        {r@{\quad:\quad}l}
    -1 & x<0 \\ 0 & x=0 \\ +1 & x>0
        \end{array} \right.   \]
```

$$y=\left\{\begin{array}{r@{\quad:\quad}l} -1 & x<0 \\ 0 & x=0 \\ +1 & x>0 \end{array}\right.$$

上面例子中的 `array` 环境已在 4.8.1 节中介绍了，它生成数学模式中的表格。

`\left ... \right` 命令可以应用于下面这 22 种不同的符号上。它们是

(	(	)	)	⌊	\lfloor	⌋	\rfloor
[	[	]	]	⌈	\lceil	⌉	\rceil
{	\{	}	\}	⟨	\langle	⟩	\rangle
∣	|	∥	\\|	↑	uparrow	⇑	\Uparrow
/	/	\	\backslash	↓	downarrow	⇓	\Downarrow
				↕	\updownarrow	⇕	\Updownarrow

例如， \left| ... \right| 就会生成两条根据所包围公式文本自动调整高度的竖线。

练习 5.12: 在公式 5.3 中，用 $\cos\left(\frac{\pi}{2}z^2\right)$, $\sin\left(\frac{\pi}{2}z^2\right)$ 代替 $\cos(\frac{\pi}{2}z^2)$, $\sin(\frac{\pi}{2}z^2)$ 。

§5.4.2 公式中的普通文本

有时候需要在公式中包含一些普通文本，例如单个的单词 and, or, if 等等。在这种情形中，虽然处于数学模式中，但我们需要切换回 LR 模式（ 1.5.3 节和 4.7.1 节 ）。这可以通过在公式中执行命令 \mbox{普通文本} ，并结合水平间距命令如 \quad 和 \hspace 等等，来做到这一点。例如：

$$X_n = X_k \qquad \text{if and only if} \qquad Y_n = Y_k \quad \text{and} \quad Z_n = Z_k$$

```
\[ X_n = X_k \qquad\mbox{ if and only if} \qquad
   Y_n = Y_k \quad\mbox{and}\quad Z_n = Z_k    \]
```

为了得到如上面例子中的那样在显示公式中的一串长普通文本，最好的方法是把公式和文本分别放在自己的子段盒子或小页中，并把它们用适当的竖直定位并排摆放。

另一方面，如果要把文本字体的字母做为数学符号，那就需要用 LaTeX 2ε 的数学字母命令来输入：

\mathrm[2ε] \mathtt[2ε] \mathbf[2ε]
\mathsf[2ε] \mathit[2ε] \mathcal[2ε]

我们在 5.3.2 节中已经用 \mathcal 命令显示花体字母。所有这些命令的作用方式是一样的：给它们的参数取相应的字体。

$\mathbf{B}^0(x)$ T^i_j `$\mathbf{B}^0(x)$ \quad $\mathsf{T}^i_j$`

在 5.3.1 节中的 \mathnormal 命令也属于这个组。它与 \mathit 的差别在于它把自己的参数设成正常的数学斜体，而后者则设成普通文本斜体。字母是一样的，而间距则不一样。

`$\mathnormal{differ} \ne \mathit{differ}$`

$differ \neq \mathit{differ}$

所有的数学字母命令只是设置数学模式中的文本，因此这就意味着空格还是要像通常那样被忽略。而对于在 \mbox 中的文本则不会如此。

在 LaTeX2.09 中，即使在数学模式中，也可以通过在一个无名环境中利用两字母字体声明来达到相同的效果。上例中的前一个可以用 ${\bf B}^0(x)$ 来得到。

§5.4.3 矩阵和域

$$\begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{matrix}$$

类似于左边这样的结构是矩阵，行列式，方程组等等的基础。这里把它们统称为 域 。

域是用 array 环境生成的，我们已在 4.8.1 节关于表格的内容中给出了它的语法和构造。 array 环境生成数学模式中的表格，即把列条目看做公式文本。例如：

$$\begin{array}{*{3}{c@{\:+\:}}c@{\;=\;}c} a_{11}x_1 & a_{12}x_2 & \cdots & a_{1n}x_n & b_1 \\ a_{21}x_1 & a_{22}x_2 & \cdots & a_{2n}x_n & b_2 \\ \multicolumn{5}{c}{\dotfill} & \\ a_{n1}x_2 & a_{n2}x_2 & \cdots & a_{nn}x_n & b_n \end{array}$$

```
\[ \begin{array}{*{3}{c@{\:+\:}}c@{\;=\;}c}
     a_{11}x_1 & a_{12}x_2 & \cdots & a_{1n}x_n & b_1 \\
     a_{21}x_1 & a_{22}x_2 & \cdots & a_{2n}x_n & b_2 \\
       \multicolumn{5}{c}{\dotfill}                  \\
     a_{n1}x_2 & a_{n2}x_2 & \cdots & a_{nn}x_n & b_n
   \end{array}  \]
```

这里重新复习一下表格的构造要素（ 4.8.1 节）： @{文本} 在相邻两列间插入 文本 的内容。在上面的例子中，这就是 \:+\: 和 \;=\; 。 \: 和 \; 命令现在还没有讲，但它们是用来生成数学模式中小的水平间距（ 5.5.1 节）。 *{3}{c@{\:+\:}} 是列定义 c@{\:+\:} 的三次重复。 c 规定了列文本是居中排列的。 \multicolumn{5}{c} 把五列进行了合并，用一个居中条目代替。 \dotfill 用点充满该列。上面的方程组也可以用下面这样的格式更简单地得到：

```
\begin{array}{c@{\:+\:}c@{\:+\cdots+\;}c@{\;=\;}c}
```

也可以把 array 环境嵌套任意层次：

$$\left(\begin{array}{cc|c} x_{11} & x_{12} & x \\ x_{21} & x_{22} & \\ \multicolumn{2}{c|}{y} & \end{array} \right)$$

```
\[ \left( \begin{array}{c}
     \left| \begin{array}{cc}
   x_{11} & x_{12} \\ x_{21} & x_{22}
     \end{array} \right|
   x \\y \end{array} \right)   \]
```

绝大多数域中的列条目都是居中排列(c)的。在上面的列中第一个条目还是一个域，它具有两个居中的列。其左右用可自动调整高度的竖线包围。

array 环境在结构上与竖直盒子是一样的。这就是说在包围它的环境中，它只是被做为单个字符一样处理，因此它也可以同其它符号和结构一起出现：

$$\sum_{p_1<p_2<\cdots<p_{n-k}}^{(1,2,\ldots,n)} \Delta^{p_1p_2\cdots p_{n-k}}_{p_1p_2\cdots p_{n-k}} \sum_{q_1<q_2<\cdots<q_k} \begin{vmatrix} a_{q_1q_1} & a_{q_1q_2} & \cdots & a_{q_1q_k} \\ a_{q_2q_1} & a_{q_2q_2} & \cdots & a_{q_2q_k} \\ \cdots & \cdots & \cdots & \cdots \\ a_{q_kq_1} & a_{q_kq_2} & \cdots & a_{q_kq_k} \end{vmatrix}$$

```
\[ \sum_{p_1<p_2<\cdots<p_{n-k}}^{(1,2,\ldots,n)}
   \Delta_{\begin{array}{l}
            p_1p_2\cdots p_{n-k} \\ p_1p_2\cdots p_{n-k}
           \end{array}}
   \sum_{q_1<q_2<\cdots<q_k} \left| \begin{array}{llcl}
              a_{q_1q_1} & a_{q_1q_2} & \cdots & a_{q_1q_k} \\
              a_{q_2q_1} & a_{q_2q_2} & \cdots & a_{q_2q_k} \\
                 \multicolumn{4}{c}\dotfill \\
              a_{q_kq_1} & a_{q_kq_2} & \cdots & a_{q_kq_k}
                                  \end{array} \right|   \]
```

在这个例子中， array 环境用在 Δ 的指标。然而，这个指标相对于其它部分公式显得太大了。在 5.4.6 节中提供了更好地解决这种域指标的方法。

同所有的表格环境一样，也可以在 array 环境中包含可省的竖直定位参数 b 或 t 。在 4.7.3 和 4.8.1 节中已描述了其语法和结果。只有当域不需要竖直居中，而是相对于它的顶行或底行竖直定位时才需要这个参数值。

$$x - \begin{array}{c} a_1 \\ \vdots \\ a_n \end{array} \begin{array}{cc} & \\ - u - v & 10 \\ & 12 \\ u + v & -120 \end{array}$$

```
\[ x - \begin{arrray}{c}
     a_1 \\ \vdots \\ a_n \end{array}
   - \begin{array}[t]{cl}
       u - v & 10 \\
       u + v & \begin{array}[b]{r}
               12\\-120  \end{array}
     \end{array}                    \]
```

我们建议读者借助于右边的生成文本，尝试推断如何构造各种不同的域。

练习 5.13: 方程组的解

$$F(x,y)=0 \quad \text{and} \quad \begin{vmatrix} F''_{xx} & F''_{xy} & F'_x \\ F''_{yx} & F''_{yy} & F'_y \\ F'_x & F'_y & 0 \end{vmatrix} = 0$$

生成 $F(x,y)=0$ 所有可能拐点的坐标。

注意：上面的显示公式是由两个小公式组成的，中间的单词 and 要在其前后加上 `\quad` 以插入额外的间距。可以不用 `\left|...\right|` 确定的自动调整高度的竖线包围 `array` 环境，而代之以格式参数值 `{|...|}`（ 4.8.1 节）也能达同样的目的。这样的结构在数学上称为行列式。

练习 5.14: 两方程为

$$\frac{x-x_1}{l_1}=\frac{y-y_1}{m_1}=\frac{z-z_1}{n_1}\quad\text{and}\quad\frac{x-x_2}{l_2}=\frac{y-y_2}{m_2}=\frac{z-z_2}{n_2}$$

的直线之间的最短距离是如下表达式：

$$\frac{\pm\begin{vmatrix}x_1-x_2 & y_1-y_2 & z_1-z_2\\ l_1 & m_1 & n_1\\ l_2 & m_2 & n_2\end{vmatrix}}{\sqrt{\begin{vmatrix}l_1 & m_1\\ l_2 & m_2\end{vmatrix}^2+\begin{vmatrix}m_1 & n_1\\ m_2 & n_2\end{vmatrix}^2+\begin{vmatrix}n_1 & l_1\\ n_2 & l_2\end{vmatrix}^2}}$$

如果分母为零，那么这两条直线必相交。

注意：在上面表达式分母中根号下的三个行列式并不推荐使用 `{|cc|}` 形式的格式化参数值。这里应该用 `\left|...\right|` 。试试这两种可能，并把结果做一对比。

练习 5.15: Laurent 展开：令 $c_n=\frac{1}{2\pi i}\oint(\zeta-a)^{-n-1}f(\zeta)\,d\zeta$ ，那么对任何函数 $f(z)$ 都有下面的表达式（ $n=0,\pm1,\pm2,\ldots$ ）：

$$f(z)=\sum_{n=-\infty}^{+\infty}c_n(z-a)^n=\begin{cases}c_0+c_1(z-a)+c_2(z-a)^2+\cdots+c_n(z-a)^n+\cdots\\ \qquad\qquad +c_{-1}(z-a)^{-1}+c_{-2}(z-a)^{-2}+\cdots\\ \qquad\qquad +c_{-n}(z-a)^{-n}+\cdots\end{cases}$$

提示：公式的右边可以用只有一列的 `array` 环境生成。那么它的格式参数值是什么呢？

§5.4.4 公式上下的直线

命令

`\overline{` 部分公式 `}` 和 `\underline{` 部分公式 `}`

可以用来在公式或其一部分的上面或下面画一直线。而且它们可以嵌套任意次：

$$\overline{a^2+\underline{xy}+\overline{\overline{z}}}$$

```
\[ \overline{\overline{a}^2 + \underline{xy}
                  +\overline{\overline{z}}}      \]
```

命令 `\underline` 也可用在普通文本模式中以生成下划线，而 `\overline` 只能用在数学模式中。

©TEXGuru, August 15, 1999

与这两条命令完全类似地还有：

`\overbrace{ 部分公式 }` 和 `\underbrace{部分公式 }`

它们在 部分公式 的上方或下方放上一个水平的大括号：

$\overbrace{a+\underbrace{b+c}+d}$ `\overbrace{a + \underbrace{b+c} + d}`

在显示公式中，这些命令也可以有指数或指标。（升高的）指数被放在上括号的上方，而（降低的）指标放在下括号的下方。

$$\underbrace{a+\overbrace{b+\cdots+y}^{123}+z}_{\alpha\beta\gamma}$$

```
\[ \underbrace{a + \overbrace{b + \cdots +
     y}^{123} + z}_{\alpha\beta\gamma} \]
```

练习 5.16: 从 n 个元素中取 m 个的排列总数（用符号 P_n^m 表示）为

$$P_n^m=\prod_{i=0}^{m-1}(n-i)=\underbrace{n(n-1)(n-2)\cdots(n-m+1)}_{\text{total of } m \text{ factors}}=\frac{n!}{(n-m)!}$$

§5.4.5 堆积符号

命令

`\stackrel{ 上部符号 }{ 下部符号 }`

把 上部符号 居中放在 下部符号 的上方，这里放在上方的符号要以较小的字样显示：

$\vec{x}\stackrel{\mathrm{def}}{=}(x_1,\ldots,x_n)$ `$\vec{x} \stackrel{\mathrm{def}}{=} (x_1,...$`

$A\xrightarrow{\alpha'}B\xrightarrow{\beta'}C$ `$ A \stackrel{\alpha'}{\longrightarrow} B ...$`

利用数学字体尺寸命令（5.5.2 节），用这个命令可以构造新的符号。例如，有些作者喜欢符号 `\le` 是 $\stackrel{\leqslant}{=}$ 样子，而不是 $\leq$，那么这可以利用 $<$ 和 $=$ 的组合 `$\stackrel{\textstyle<}{=}$` 得到。如果这里不用命令 `\textstyle`，那么结果是 $\stackrel{<}{=}$。

§5.4.6 其它的 TEX数学命令

TEX数学命令 `\atop` 和 `\choose` 也是非常有用的的，可以应用于任一LATEX文档中。（事实上，所有 TEX数学命令中除了 `\eqalign`, `\eqalignno` 和 `\eqaligno` 外，都可以用于 LATEX文稿中。）它们的语法是

`{ 顶部公式 \atop 底部公式 }`

`{ 顶部公式 \choose 底部公式 }`

这两条命令都生成一个类似于没有分数线的分数结构。对于 `\choose` 命令，该结构被包围在小括号内（在数学中称之为二项式系数）。

$$\binom{n+1}{k}=\binom{n}{k}+\binom{n}{k-1}$$

```
\[ {n+1 \choose k} =
   {n \choose k} + {n \choose k-1} \]
```

$$\prod_{j\ge0}\left(\sum_{k\ge0} a_{jk}z^k\right)=\sum_{n\ge0} z^n\left(\sum_{\substack{k_0,k_1,\ldots\ge0\\ k_0+k_1+\cdots=0}} a_{0k_0}a_{1k_1}\cdots\right)$$

```
\[ \prod_{j\ge0}\left( \sum_{k\ge0} a_{jk}z^k \right) =
   \sum_{n\ge0} z^n \left(\sum_{k_0,k_1,\ldots\ge0 \atop
      k_0+k_1+\cdots=0} a_{0k_0} a_{1k_1}\ldots \right)   \]
```

利用 LaTeX环境也可以生成类似的结构：

```
\begin{array}{c} 顶部行 \\ 底部行 \end{array}          (atop)
\left(\begin{array}{c} 顶部行 \\ 底部行 \end{array}\right)   (choose)
```

在 array 结构与相应 TeX命令之间的差别在于前者总是以普通正文公式的样式和尺寸显示，而后者会视所处公式的部分而改变尺寸。

比较如下：

$\Delta_{\substack{p_1p_2\cdots p_{n-k}\\ p_1p_2\cdots p_{n-k}}}$ 这里的指标域是用 \atop 生成的

$\Delta_{\substack{p_1p_2\cdots p_{n-k}\\ p_1p_2\cdots p_{n-k}}}$ 这里的指标是用 array 环境生成的

上面的 TeX命令也可以用来生成正文公式中的小矩阵，如 $\left(\begin{smallmatrix} a & b & c\\ l & m & n\end{smallmatrix}\right)$ 或者 $\binom{1\,0}{0\,1}$。这里的第一个矩阵是用

```
${1\,0 \choose0\,1}$
```

生成的，而第二个是用

```
$\left({a \atop l}{b\atop m}{c\atop n}\right)$
```

生成的。

§5.4.7 多行公式

所谓多行公式，就是一个公式长达几行，其中每行的关系符号（例如 = 或 ≤ ）是彼此竖直对齐的。为此，环境

```
\begin{eqnarray}  第一行 \\ ... \\ 第 n 行 \end{eqnarray}
\begin{eqnarray*} 第一行 \\ ... \\ 第 n 行 \end{eqnarray*}
```

被用来排版显示模式中的长达几行的公式或方程。方程或公式中的行与行之间用 \\ 分开。每行的形式为

```
左边公式 & 中间公式 & 右边公式 \\
```

在显示的时候， 左边公式 在左边列中右对齐，而 右边公式 在右边列中左对齐， 中间公式 在中间列居中。列分隔符号 & 标志着公式的不同部分。通常 中间公式 就是单个的数学符号，即上面所说的关系运算符。因此每一行公式的样子如同 \begin{array}{rcl} ...\end{array} 环境中的行。

©TeXGuru, August 15, 1999

在 array 环境和 eqnarray 环境之间的差别是：后者排版为显示公式。这也就是说对于那些列在 5.3.7 节中的符号，会取它们的大尺寸形式，分数的分子和分母也是正常尺寸。另一方面，对于 array 环境而言，列条目被设为正文公式，会选取那些符号的小尺寸形式，分数的两部分也是以较小尺寸显示的。

在 eqnarray 环境的标准形式中，是会自动给公式加上有顺序的编号的，而在 *- 形式中则没有编号。为了不给标准形式中某行公式加编号，可以在该行公式的结束符 \\ 前面插入命令 \nonumber 。

公式的编号可以在正文中用 \ref{引用名} 命令引用，这里的 引用名 就是要用 \label{引用名} 命令赋给某一行公式的关键词。详情请见 8.3.1 节。

例：

$$\begin{aligned}(x+y)(x-y) &= x^2-xy+xy-y^2\\ &= x^2-y^2 \qquad (5.4)\\ (x+y)^2 &= x^2+2xy+y^2 \qquad (5.5)\end{aligned}$$

```
\begin{eqnarray}
    (x+y)(x-y) & = & x^2-xy+xy-y^2 \nonumber\\
               & = & x^2 - y^2 \\
    (x+y)^2    & = & x^2+2xy+y^2
\end{eqnarray}
```

$$\begin{aligned}x_nu_1+\cdots+x_{n+t-1}u_t &= x_nu_1+(ax_n+c)u_2+\cdots\\ &\quad +\left(a^{t-1}x_n+c(a^{t-2}+\cdots+1)\right)u_t\\ &= (u_1+au_2+\cdots+a^{t-1}u_t)x_n+h(u_1,\ldots,u_t)\end{aligned}$$

```
\begin{eqnarray*}
x_nu_1 + \cdots + x_{n+t-1}u_t & = & x_nu_1 + (ax_n +c)u_2 +
                                                    \cdots\\
 &   & + \left(a^{t-1}x_n + c(a^{t-2} + \cdots+1)\right)u_t\\
 & = & (u_1 + au_2 + \cdots + a^{t-1}u_t)x_n + h(u_1,\ldots,u_t)
\end{eqnarray*}
```

我们需要对后面这个例子做一些解释。在第二行上有一个自动调整尺寸的 \left(... \right) 对。这样的对只能位于一行中；即不能把用行结束符 \\ 把它们分开！如果多行公式中有自动调整尺寸的括号，只可能出现在单行中。

如果配对括号必须位于不同行中，那么可以尝试用 \left(... \right. \\ \left. ... right) 结构。在第一行，\left(与看不见的括号 \right.

配对，而第二行则由 `\left.` 开始，它与闭括号 `\right)` 配对。然而，这种方法只有当两部分公式的高度大致相当时才会使两个自动调整尺寸的结果差不多。在 5.5.3 节中描述了当自动调整尺寸失败时如何手工选择括号的尺寸。

第二行开头的 + 也需要加以解释。在数学中 + 和 − 有两种意义：在两个数学量之间，扮演一种结合的角色（二元运算符），但只是在一个数学符号前，那它就是一个符号标志（正号或负号）。LaTeX通过插入不同的间距来强调这种不同（例如，请看 $+b$ 和 $a+b$ 的差别）。 n

$$\begin{aligned} y &= a+b+c+d \\ &\quad +e+f+g \\ &\quad +h+i+j \end{aligned}$$

如果一个很长的公式被分成几行，其中有的行以 + 或 − 开头，LaTeX会认为它是一个标志符号，从而把它向下一个字符移近。

解决的方法是在该行开头处引进一个零宽度的看不见字符。这可以是一个空结构 `{}` 。请比较上面公式中 `&& +e+f+g` 和 `&&{}+h+i+j` 的结果。

有的时候对于多行公式最好采用下列这种断行的方法：

$$\begin{aligned} w+x+y+z= & \\ & a+b+c+d+e+f+ \\ & g+h+i+j+k+l \end{aligned}$$

即第二行及其后续各行并不是相对于第一行的等号对齐，而是相对于第一行缩进一点后左对齐。

```
\begin{eqnarray*}
\lefteqn{w+x+y+z = } \\
   & & a+b+c+d+e+f+  \\
   & & g+h+i+j+k+l
\end{eqnarray*}
```

在第一行的命令 `\lefteqn{w+x+y+z =} \\` 可以使得参数值的内容被显示出来，但是LaTeX认为它的宽度为零。因此左边列只有列间距，由它生成其余各行的左面缩进。

可以通过在 `\lefteqn{...}` 和行结束符之间插入 `\hspace{深度}` 命令来改变缩进深度。正的 *深度* 会增加缩进，而负值则减少缩进。

练习 5.17: 下面的公式的分行为：

$$\begin{aligned} \arcsin x &= -\arcsin(-x) = \frac{\pi}{2} - \arccos x = \left[\arccos\sqrt{1-x^2}\right] \\ &= \arctan\frac{x}{\sqrt{1-x^2}} = \left[\operatorname{arccot}\frac{\sqrt{1-x^2}}{x}\right] \end{aligned} \tag{5.6}$$

$$\begin{aligned} f(x+h,y+k) &= f(x,y) + \left\{\frac{\partial f(x,y)}{\partial x}h + \frac{\partial f(x,y)}{\partial y}k\right\} \\ &\quad + \frac{1}{2}\left\{\frac{\partial^2 f(x,y)}{\partial x^2}h^2 + +2\frac{\partial^2 f(x,y)}{\partial x\partial y}kh + \frac{\partial^2 f(x,y)}{\partial y^2}k^2\right\} \\ &\quad + \frac{1}{6}\{\cdots\} + \cdots + \frac{1}{n!}\{\cdots\} + R_n \end{aligned} \tag{5.7}$$

关于可能出现的错误信息的注释：

对那些有很多组逻辑括号的长公式，尤其是嵌套很多次的情形，刚开始排版时总会有错误。原因就在于括号的顺序不对或者遗漏了配对。

如果在公式处理过程中 LaTeX报告有错误，初学者经常无法理解错误信息（在第 9 章中详细介绍了各种错误信息），这时候就应该仔细地检查公式文本中括号配对。有些文本编辑器可以帮助有户搜索配对括号，这样可以简化这种操作。

如果这样还没有找到错误所在，那么用户就可以尝试当出现错误时按回车键，以继续处理下去。从这样所得的打印结果中就会找到错误所在。

练习 5.18: 在 `eqnarray` 环境中会在列分隔符号 `&` 位于的地方插入额外间距。当公式是用一个长求和表达中的 + 或 − 分行对齐时，这种间距就是不必要的。例如：

多项式展开 $y = f(x) = ax + bx^2 + cx^3 + dx^4 + ex^5 + fx^6 + \cdots\ (a \neq 0)$ 的逆函数开头几项为

$$\begin{aligned} x = \varphi(y) = \frac{1}{a}y &- \frac{b}{a^3}y^2 + \frac{1}{a^5}(2b^2 - ac)y^3 \\ &+ \frac{1}{a^7}(5abc - c^2d - fb^3)y^4 \\ &+ \frac{1}{a^9}(6a^2bd + 3a^2c^2 + 14b^4 - a^3e - 21ab^2c)y^5 \\ &+ \frac{1}{a^{11}}(7a^3be + 7a^3cd + 84ab^3c - a^4f - \\ &\quad 28a^2b^2d - 28a^2bc^2 - 43b^5)y^6 + \cdots \end{aligned}$$

选择 `\arraycolsep`（ 4.8.2 节）的一个适当值，使得 + 或 − 与分点之间的距离相对于其它部分的公式尽可能得小。

§5.4.8 有框公式和并列公式

显示公式或方程也可以放在适当宽度的竖直盒子中，即 `\parbox` 命令或 `minipage` 环境中。在竖直盒子中，公式是水平居中的，或者根据所选择的文档类选项进行 `\mathindent` 缩进后左对齐。

竖直盒子可以如同单个字符一样，相对于另一个竖直盒子定位（ 4.7.3 和 4.7.7 节）。通过这种方法，我们可以使得显示公式或方程并列摆放。

$$\begin{aligned} \alpha &= f(z) \quad (5.8) \\ \beta &= f(z^2) \quad (5.9) \\ \gamma &= f(z^3) \quad (5.10) \end{aligned} \qquad \begin{aligned} x &= \alpha^2 - \beta^2 \\ y &= 2\alpha\beta \end{aligned}$$

左面的公式放在宽度为 4cm 的 `\parbox` 中，而右边那个放在宽度为 2.5cm 的 `\parbox` 中，而这段文本则是在宽度为 4.5cm 的 `minipage` 中。

```
\parbox{4cm}{\begin{eqnarray} \alpha &=& f(z) ... \end{eqnarray}}
\hfill \parbox{2.5cm}{\begin{eqnarray*}
  x &=& \alpha^2 - \beta^2 \\  y &=& 2\alpha\beta \end{eqnarray*}}
\hfill \begin{minipage}{4.5cm} 左面的公式 ...\end{minipage}
```

竖直盒子也可以用来解决公式编号位置不恰当的问题。在 eqnarray 环境中是为每行生成一个公式编号，不要编号的话则用命令 \nonumber 。而为了给一组公式加上竖直居中的编号，如：

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \\ P(-x) &= a_0 - a_1 x + a_2 x^2 - \cdots + (-1)^n a_n x^n \end{aligned} \tag{5.11}$$

就可以用如下输入来得到：

```
\parbox{10cm}{\begin{eqnarray*} ... \end{eqnarray*}} \hfill
\parbox{1cm}{\begin{eqnarray}\end{eqnarray}}
```

这里真正的方程是用 eqnarray* 环境生成的，放在宽度为 10cm 的竖直盒子中，其后接一个空的 eqnarray 环境，它是一个宽度为 1cm 的盒子，由它生成公式编号。这两个盒子都是沿着它们自己的中间竖直对齐的。

如果要用方框强调某公式，也不需要什么新的结构要素。只要把它们放在 \fbox（ 4.7.7 节）中就可以了。正文公式 $\boxed{a+b}$ 要加上方框，只要输入 \fbox{$a+b$} 就可以了。

要给显示公式加上方框，那就首先需要把它放在一个宽度足够的 \parbox 或 minipage 环境中，然后把它们放在一个 \fbox 中。（另外的方法见 5.5.6 节。）

$$\boxed{\int_0^\infty f(x)\,dx \approx \sum_{i=1}^{n} w_i e^{x_i} f(x_i)}$$

就是用如下输入生成的：

```
\fbox{\parbox{5cm}{\[ \int^\infty_0 f(x)\, dx \approx ... \] }}
```

§5.4.9 化学方程式和数学公式中的黑体

数学中有时需要把单个字符或者部分公式设成黑体。要做到这一点，只需要利用在 5.4.2 节中提到的数学字样命令 \mathbf[2ε]就可以了：

```
$\mathbf{S^{-1}TS = db(\omega_1, \ldots, \omega_n) = \Lambda} $
```

可以得到 $\mathbf{S^{-1}TS = db(\omega_1, \ldots, \omega_n) = \Lambda}$ 。

在这个例子中，所有公式都被设为 \mathbf 的参数值，这样全部都是黑体。实际上只有数字、小写和大写的拉丁字母以及大写的希腊字母被 \mathbf 设成黑体。小写的希腊字母和其它数学符号仍然是普通数学字体。

如果只有一部分公式被设成黑体，那么就必须把这部分做为 \mathbf 的参数值。

```
$\mathbf{2\sqrt{x}/y} =z$
```
$\mathbf{2\sqrt{x}/y} =z$

数学字体命令 \boldmath 会把所有字符设成黑体，只有下列符号例外：

- 上升或下降的符号（指数和指标）
- 字符 + : ; ! ? ( ) [ ]
- 有两种尺寸的符号（ 5.3.7 节）

而 \boldmath 命令不能位于数学模式中。必须在切换进入数学模式之前或者在子段盒子及小页环境中使用。相反的命令 \unboldmath 把数学字体重设回正常的字样。

```
\boldmath \[ \oint\limits_C V\,d\tau =
  \oint\limits_\Sigma\nabla\times
  V\,d\sigma \] \unboldmath
```
$$\boldmath \oint\limits_C V\,d\tau = \oint\limits_\Sigma\nabla\times V\,d\sigma \unboldmath$$

即使在数学模式外面已经使用了 \boldmath，也可以在内部用如下命令暂时性地关闭这种设置：\mbox{\unboldmath$...$}，以把这部分的公式取成正常数学字体。

```
\boldmath\( P = \mbox{\unboldmath$m$}b\) \unboldmath
```
的结果为：\boldmath\(P = \mbox{\unboldmath$m$}b\) \unboldmath 。类似地，在数学模式中也可用 \mbox{\boldmath$...$} 结构暂时性打开 \boldmath 设置：\(W_r = \int\mbox{\boldmath$M\,d\varphi$}= ... \)

```
\( W_r = \int\mbox{\boldmath$M\,d\varphi$}= ... \)
```

化学公式通常采用的是罗马字样，而不是数学公式中的斜体。通过把公式作为字体命令 \mathrm[2ε] 的参数值可以实现这一效果：

```
$\mathrm{Fe_2^{2+}Cr_2O_4}$
```
$\mathrm{Fe_2^{2+}Cr_2O_4}$

如果仔细观察，你会发现 Cr 和 O 的指标不像 Fe 的那么低。 7.3.4 节在 172 页上的例 5 中描述了如何简单地生成化学方程式，而且不会有这种缺陷。

在 LaTeX2.09 中，没有字体命令 \mathbf 和 \mathrm，而是用如 5.4.2 节所示那样，用 \bf 和 \rm 取代。

§5.5 数学公式的精调

至此我们给出的数学要素，几乎可以排版普通文稿中所可能出现的各种公式，前提条件是作者要遵守印刷数学公式时某种一般性的广为接受的规则。那些到现在为止还在用打字机打印文稿的作者可能会觉得由 TeX选择的字符间距太窄了。事实上，关于数学排版， TeX要比绝大多数作者知道得都多。在采用下面这节中的格式化工具进行修改公式排版时，作者应该首先去查看一些出版文献中类似公式的排版。否则，就有可能这个 LaTeX文档中手工修改的公式最后还是被专业制版工人重新改回原来标准 TeX所排版出来的样子。

§5.5.1 水平间距

虽然 TEX对数学排版的规则有深入的了解，但是我们不能期望它理解数学公式的意义。例如 $y\,dx$ 通常意味着变量 y 与微分算子 dx 的组合，这种组合的标志是两者之间有一个小间距。然而， TEX会去掉 `y dx` 中的空格，从而显示为 ydx ，即 y, d 和 x 三个量的乘积。对于这种情形， LATEX就需要一些精调辅助。

在数学模式中可以用下面的命令插入小量水平间距：

`\,` 小间距 $=3/18$ of a quad
`\:` 中间距 $=4/18$ of a quad
`\;` 大间距 $=5/18$ of a quad
`\!` 负间距 $=-3/18$ of a quad

在下面的例子中，第三列就是没有插入水平间距命令的结果：

`$\sqrt{2}\,x$`	$\sqrt{2}\,x$	$\sqrt{2}x$
`$\sqrt{\,\log x}$`	$\sqrt{\log x}$	$\sqrt{\log x}$
`$O\left(1/\sqrt{n}\,\right)$`	$O\left(1/\sqrt{n}\,\right)$	$O\left(1/\sqrt{n}\right)$
`$[\,0,1)$`	$[\,0,1)$	$[0,1)$
`$\log n\,(\log\log n)^2$`	$\log n\,(\log\log n)^2$	$\log n(\log\log n)^2$
`$x^2\!/2$`	$x^2\!/2$	$x^2/2$
`$n/\!\log n$`	$n/\!\log n$	$n/\log n$
`$\Gamma_{\!2}+\Delta^{\!2}$`	$\Gamma_{\!2}+\Delta^{\!2}$	$\Gamma_2+\Delta^2$
`$R_i{}^j{}_{\!kl}$`	$R_i{}^j{}_{\!kl}$	${R_i}^j{}_{kl}$
`$\int_0^x\!\int_0^y dF(u,v)$`	$\int_0^x\!\int_0^y dF(u,v)$	$\int_0^x\int_0^y dF(u,v)$
`\[ \int\!\!\!\int_D dx\,dy \]`	$\iint_D dx\,dy$	$\int\int_D dx\,dy$

注意：在倒数第三个例子中的 `R_i{}^j` ，在 R_i 的指标之后构造一个零宽度的不可见字符，用这个空字符接受后面的指数。这样结果就是 ${R_i}^j$ ，而不是 `R_i^j` 的 R_i^j 。

对于数学间距命令，并没有一个严格的普遍适用的规则。可以考虑的就是前面提到的微分算子，正文公式中的小根号后接一个变量，除号 / ，以及多重积分号。上面的例子演示了许多适合的情形。

§5.5.2 在公式中选择字体尺寸

可以把各部分公式改变成 TEX所选择的字体尺寸。首先我们要解释一下在数学模式中有哪些尺寸可以使用， TEX的选择准则是什么。

在数学模式中可以选择四种字体尺寸，它们的实际尺寸是相对于文档类的基础字体尺寸的：

`\displaystyle`	D	显示公式的标准尺寸
`\textstyle`	T	正文公式的标准尺寸
`\scriptstyle`	S	上下标的标准尺寸
`\scriptscriptstyle`	SS	更低层上下标的标准尺寸

从现在开始我们就采用这里的符号缩写 D, T, S 和 SS 。当切换到数学模式时，被激活的字体是 D（显示公式）或 T（正文公式）。它们的差别就在于那些有两种尺寸的符号，以及那些上下标相应的样子（ 5.3.7 节）。大符号属于 D ，小符号属于 T 。

从这两个基础尺寸开始，各部分数学要素被设置为其它尺寸。一旦为某个要素选定了另一种尺寸，那么这个尺寸就在这个要素中起作用。

下面的表格说明的是选择规则：

激活尺寸	分数上	分数下	上、下标
D	T	T	S
T	S	S	S
S	SS	SS	SS
SS	SS	SS	SS

如果被激活的尺寸是 D ，那么就为分数的两部分选择 T 。也就是说对于分子和分母，被激活的尺寸是 T 。如果它们还包括分数，那么它们的两部分就是 S 。如果被激活尺寸是 DS 或 T ，上标和下标（指数和指标）就是 S ；在上下标中 S 是激活尺寸，其中的分数或移位就是 SS 。

TEX命令 `{ \atop }` 和 `{ \choose }` 当做分数对待。

在 `array` 环境中激活的字体尺寸是 T 。

最小的可应用数学字体尺寸是 SS 。一旦已达到了这个尺寸，就不会再更小了，因此所有更低层的上标和下标都是 SS 。

从上面的表格很容易看出

```
\[ a_0 + \frac{1}{a_1 + \frac{1}{a_2
       + \frac{1}{a_3 + \frac{1}{a_4}}}} \]
```

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4}}}}$$

的结果一定是右边那样的形式。

这样的数学结构，称为连分数，通常是如下排版的：

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{a_4}}}}$$

```
\[ a_0 + \frac{1}{\displaystyle a_1
       + \frac{1}{\displaystyle a_2
       + \frac{1}{\displaystyle a_3
       + \frac{1}{a_4}}}}                \]
```

通过在每部分中显式地给出字体尺寸，我们可以确定被激活的尺寸，而不是内部所选择的尺寸。在这个例子中，每个分母选择的都是 D 。因此后面的分数就以最大尺寸显示出来。最后那个分数中的 `\displaystyle` 命令可以忽略。（为什么呢？）

上面的尺寸选择表格使得我们可以精确地预见到所处部分公式的数学字

体尺寸，这样就可以确定是否有必要显式定义字体尺寸。如此选择的效果也可以从后续公式要素中计算出来。

下面的例子中在右边列出的是没有显式定义数学字体尺寸的结果。

$$\frac{\displaystyle\frac{a}{x-y}+\frac{b}{x+y}}{\displaystyle 1+\frac{a-b}{a+b}}$$

```
\[ \frac{\displaystyle\frac{a}{x-y}
                    +\frac{b}{x+y}}
   {\displaystyle 1+\frac{a-b}{a+b}} \]
```

$$\frac{\frac{a}{x-y}+\frac{b}{x+y}}{1+\frac{a-b}{a+b}}$$

$$e^{-\frac{x_i-x_j}{n^i+n^j}}$$

```
\[  e^{\textstyle -
    \frac{x_i-x_j}{n^i+n^j}}  \]
```

$$e^{-\frac{x_i-x_j}{n^i+n^j}}$$

$$\begin{pmatrix} \binom{ab}{cd} & \displaystyle\frac{e+f}{g-h} \\ 0 & \displaystyle\left|\begin{matrix} i & j \\ k & l \end{matrix}\right| \end{pmatrix}$$

```
\[ \left(\begin{array}{cc}
 \displaystyle{ab\choose cd} &
 \displaystyle\frac{e+f}{g-h}\\
    0 & \displaystyle \left|
        {ij\atop kl} \right|
  \end{array}\right)        \]
```

$$\begin{pmatrix} \binom{ab}{cd} & \frac{e+f}{g-h} \\ 0 & \left|\begin{matrix} i & j \\ k & l \end{matrix}\right| \end{pmatrix}$$

如果在文档中经常需要显示定义尺寸 — 例如在 array 环境中的每个条目 — 那么就可以通过在导言中加入下列指令以避免繁琐输入：

```
\newcommand{\D}{\displaystyle}\newcommand{\T}{\textstyle}...
```

用这种方法，当需要改变尺寸时，只要简单地输入 \D 或 \T 等等就可以了。

这一节中余下内容对实际应用显得不是很重要。上面所给出的数学字体选择规则只是一种简化后的结果。如果读者想知道所有的细节，我们下面就给出完整的描述。

四种数学字体尺寸 D, T, S 和 SS 中每一种都有一个修正版本 D', T', S' 和 SS' 。差别就在于 D, T, S 和 SS 的上标(指数)要比 D', T', S' 和 SS' 的稍高一点点。仔细比较一下分数线上下指数 2 的位置：$\frac{x^2}{x^2}$ 。在其它情形中有撇和无撇的字体尺寸是一样的。右面给出了真正的选择规则。

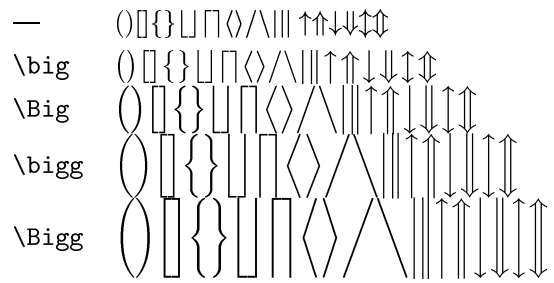
激活尺寸	分数 上	分数 下	上标	下标
D	T	T'	S	S'
D'	T'	T'	S'	S'
T	S	S'	S	S'
T'	S'	S'	S'	S'
S, SS	SS	SS'	SS	SS'
S', SS'	SS'	SS'	SS'	SS'

当在分子或者上标中显式地指定字体尺寸时，选取的是无撇形式；而在分母和下标中则选取的是有撇字体。

§5.5.3 括号符号的手工尺寸调整

在 5.4.1 节中列出的 22 个括号符号当前缀 \left ... \right 命令时会自动根据被包围公式文本的高度调整其自身尺寸。然而，也可以在括号命令

前面放上命令 \big, \Big, \bigg 或 \Bigg 来显式选择一个尺寸。



与 \left ... \right 对不同，显式括号尺寸命令并不需要包含在多行公式的一行中。开与闭括号可以处在不同的行上。这条规则也同样适用于下面段落中描述的命令。

也存在着另外两组名称为 \bigl ... \Biggl 和 \bigr ... \Biggr 的括号尺寸命令。这两组命令把后接的括号符号当做开或闭的括号处理。这些命令与标准命令之间的实际区别对于 LaTeX而言没有多大意义。
更进一步的括号尺寸命令是 \bigm...\Biggm 。对于这些命令，后接的括号符号如同关系运算符，在其前后与相邻公式之间插入较大的水平间距。

```
\[ \big[ (a+b) \big| (c+d) \big] \]
\[ \bigl[ (a+b) \bigl| (c+d) \bigl] \]
\[ \bigm[ (a+b) \bigm| (c+d) \bigm] \]
```

$$\big[(a+b)\big|(c+d)\big]$$

$$\bigl[(a+b)\bigl|(c+d)\bigr]$$

$$\bigm[\,(a+b)\,\bigm|\,(c+d)\,\bigm]$$

§5.5.4 数学样式参数

下面列出的数学样式参数都由 LaTeX赋予了标准值。用户可随时用命令 \setlength 按通常的方式改变它们的值。

\arraycolsep
: array 环境中列间距宽度的一半（也可见 4.8.2 节）。

\jot
: 在 eqnarray 和 eqnarray* 环境中插入多行公式两行之间的额外竖直距离。

\mathindent
: 当选择了文档类选项 fleqn 时数学公式的缩进量。

\abovedisplayskip
: 当显示公式的左边与左页边界的距离比前面文本行结尾到左页边界的距离还要小时插入在显示公式上方的竖直距离。这样的公式标记为太长。

\belowdisplayskip
: 插入在太长显示公式下方的竖直距离。

\abovedisplayshortskip

插入在太短显示公式上方的竖直距离。所谓太短公式就是公式的左边在前面文本行结尾的右边。

\belowdisplayshortskip

插入在太短显示公式下方的竖直距离。

\topsep

在选择文档类选项 fleqn 时并不应用上述四种距离，这里要在显示公式的上方和下方插入 \topsep（见 4.4.2 节）。

上面除 \jot 外的所有参数都应该是橡皮长度（ 2.4.2 节）。

§5.5.5 更进一步的建议

有时候作者会希望实现用到现在为止描述的方法无法得到的公式水平或竖直对齐。这时可以考虑把公式放在水平或竖直盒子中，这样就可以随心所欲地定位了。

类似地，在 array 环境中结合显示尺寸声明以及来自于 4.8.1 节和 4.8.2 节中的表格构造和样式要素可以实现任意希望的水平或竖直对齐。

练习 5.19: 生成右边的连公数。
注意：与 127 页上的例子相比，这里的分子 1 都是左对齐的。
提示：还记得命令 \hfill 吗？

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{1}{a_4}}}}$$

练习 5.20: 利用 array 环境生成下面这个方程组。

$$\begin{aligned}
\sin 2\alpha &= 2\sin\alpha\cos\alpha, & \cos 2\alpha &= \cos^2\alpha-\sin^2\alpha,\\
\sin 3\alpha &= 3\sin\alpha-4\sin^3\alpha, & \cos 3\alpha &= 3\cos^3\alpha-3\cos\alpha,\\
\sin 4\alpha &= 8\cos^3\alpha\sin\alpha-4\cos\alpha\sin\alpha, & \cos 4\alpha &= 8\cos^4\alpha-8\cos^2\alpha+1.
\end{aligned}$$

提示：注意在 array 环境格式域中的 @{...} 可以用来插入两列间的水平间距和 / 或数学文本（见 5.4.3 节的第一个例子）。

练习 5.21: 用 array 环境得到下面的输出。

注意：如果能成功地排版这个数学表格的话，那么在定位公式或者其部分时不会再有任何困难了。

下面给出一些提示：

1. 定义一些缩写，如 \D 表示 \displaystyle， \bm 表示 \boldmath，甚至可以有 \ba 和 \ea，分别表示 \begin{array} 和 \end{array}。
2. 分步建立这个表格。首先处理标题，只有它样子可以时再继续下去。注意在 array 环境中的普通文本必须放在 \mbox 里。
3. 然后处理第一个公式行。这里第二列第一个元素是一个 array 环境，它与其它行的对齐方式是由定位参数值 [t] 确定的。在这个结构中不要忘

记在必要时用 \D 激活字体尺寸。插入必要的支撑（ 4.7.6 节）以使得它离标题有适当的距离。而与第二行的距离可以用行结束命令 \\[..] 来定义。

4. 当成功地排版出这行时，下面的就没有什么困难了。
5. 在第三行公式中，第一列以及第二列的左边公式都是由 array 环境组成的。第三列由三个分数组成，其分母可以用 LaTeX的 array 环境或者 TeX命令 {...\atop...} 生成；而分数本身用另一个 array 环境分放在在两行上。
6. 最后一行公式的第二列和第三列也是 array 环境。有些部分的公式采用了 \boldmath 。注意这个命令只能用在文本模式中，即在 \mbox 中。
7. 最后一行把最外层 array 环境的三列合并为一，其中的文本被放在一个适当宽度的子段盒子中：

```
\multicolumn{3}{|c|}{\parbox{..}{......}}
```

Equations for the tangential plane and surface normal

Equation for the surface	Tangential plane	Surface normal
$F(x,y,z)=0$	$\dfrac{\partial F}{\partial x}(X-x)+\dfrac{\partial F}{\partial y}(Y-y)+\dfrac{\partial F}{\partial z}(Z-z)=0$	$\dfrac{X-x}{\frac{\partial F}{\partial x}}=\dfrac{Y-y}{\frac{\partial F}{\partial y}}=\dfrac{Z-z}{\frac{\partial F}{\partial z}}$
$z=f(x,y)$	$Z-z=p(X-x)+q(Y-y)$	$\dfrac{X-x}{p}=\dfrac{Y-y}{q}=\dfrac{Z-z}{-1}$
$x=x(u,v)$ $y=y(u,v)$ $z=z(u,v)$	$\begin{vmatrix} X-x & Y-y & Z-z \\ \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \end{vmatrix}=0$	$\dfrac{X-x}{\left\lvert\begin{smallmatrix} \frac{\partial y}{\partial u} & \frac{\partial z}{\partial u} \\ \frac{\partial y}{\partial v} & \frac{\partial z}{\partial v} \end{smallmatrix}\right\rvert}=\dfrac{Y-y}{\left\lvert\begin{smallmatrix} \frac{\partial z}{\partial u} & \frac{\partial x}{\partial u} \\ \frac{\partial z}{\partial v} & \frac{\partial x}{\partial v} \end{smallmatrix}\right\rvert}=\dfrac{Z-z}{\left\lvert\begin{smallmatrix} \frac{\partial x}{\partial u} & \frac{\partial y}{\partial u} \\ \frac{\partial x}{\partial v} & \frac{\partial y}{\partial v} \end{smallmatrix}\right\rvert}$
$\boldsymbol{r}=\boldsymbol{r}(u,v)$	$(\boldsymbol{R}-\boldsymbol{r})(\boldsymbol{r}_1\times\boldsymbol{r}_2)=0$ or $(\boldsymbol{R}-\boldsymbol{r})\boldsymbol{N}=0$	$\boldsymbol{R}=\boldsymbol{r}+\boldsymbol{\lambda}(\boldsymbol{r}_1\times\boldsymbol{r}_2)$ or $\boldsymbol{R}=\boldsymbol{r}+\boldsymbol{\lambda}\boldsymbol{N}$

In this Table x, y, z and $\boldsymbol{r}$ are the coordinates and the radius vector of a fixed point M on the curve; X, Y, Z and $\boldsymbol{R}$ are the coordinates and radius vector of a point on the tangential plane or surface normal with reference to M; furthermore $p=\frac{\partial z}{\partial x}$, $q=\frac{\partial z}{\partial y}$ and $\boldsymbol{r_1}=\partial\boldsymbol{r}/\partial u$, $\boldsymbol{r_2}=\partial\boldsymbol{r}/\partial v$.

§5.5.6 有框的显示公式

在 5.4.8 节中我们给出了一种给显示公式加上方框的方法，即首先把它放在一个适当宽度的 \parbox 或 minipage 中，然后再把它们放在 \fbox 中以生成方框。这里的问题在于需要经常多次尝试才可能找到合理的宽度。

然而，利用 5.5.2 节的数学字体尺寸命令还可以有另一种解决方法，它不需要定义有框显示公式的宽度（该方法是由德国 Kiel 大学的 Günter Green 提出的）：

```
\begin{displaymath}    或    \begin{equation}
      \fbox{$ \displaystyle 公式文本 $}
\end{displaymath}      或    \end{equation}
```

在 124 面上的有框方程例子现在的样子为：

$$\boxed{\int_0^\infty f(x)\,dx \approx \sum_{i=1}^{n} w_i e^{x_i} f(x_i)} \tag{5.12}$$

生成文本为：

```
\begin{equation}
  \fbox{$ \displaystyle \int^\infty_0 f(x)\, dx \approx ... $}
\end{equation}
```

由于这里用的是 equation 环境，因此在最右边给公式自动加上了编号，而方框只是围住了真正的公式。而用以前的方法，公式编号也要放在一个盒子中。与 124 页上的例子相比，现在方框与方程距离更近了。用户可以通过修改参数 \fboxsep（ 4.7.8 节）来改变这一距离。类似地，方框的线粗也可以用 \fboxrule 来设置。

采取同样的方法，可以利用 array 环境，得到有框的多行公式或方程组：

```
\begin{displaymath}    或    \begin{equation}
\fbox{$ \begin{array}{rcl} 公式文本 \end{array} $}
\end{displaymath}      或    \end{equation}
```

由于在这一应用中数学字体尺寸命令被相当频繁地使用，因此我们还是建议采用 128 页上的方法，重定义命令名称。

§5.5.7 补遗

在这一章中列出的数学结构要素以及格式调整（精调），对于满足在数学写作方面即使是离奇需要也是足够了。在 5.5.5 节中给出的建议更使得所期望的定位和公式构造成为可能：余下的问题就是如何积累丰富的经验，充分利用所提供的各种功能了。

如果所需要的某一符号，LaTeX并没有提供，那么我们可以尝试从已有的符号，利用叠印、提升或降低等操作来构造出来。通过利用命令 \newcommand，可以简化对这种组合符号的使用，另外为了一般用途，可以把它的定义存贮在一个文件中。

如果某个构造利用 LaTeX是无法达到的，那么借助于 TeX总是可以的。

然而，我们要指出的是，超出 LaTeX的任何 TeX结构要素都需要对 TeX的工作机制有相当的了解，而大多数用户是做不到这一点的。

对于这种要求具有足够弹性的设计，一种比较合适，但是更乏味的解决方法就是利用下一章中关于构造图画的命令集合。

如果作者坚持使用他自己的公式格式，那么当他把自己的文稿送到科技出版社时，他自己的格式有可能与国际标准相差太大，这样编辑们就可能把它重设成与 LaTeX类似的形式。

另一方面，对公式编号样式改变的要求通常就是如何对齐。在 7.3.4 节中给出了一个例子，说明如何做到这一点。可以把它做为一个样板。

©TEXGuru, August 15, 1999

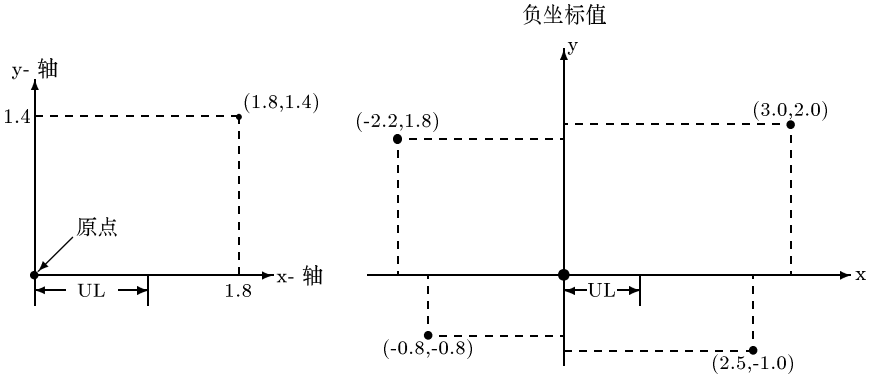
第六章　图形

利用 LaTeX可以生成简单的图形和插图。生成图形的构造模块是文本、各种斜率的直线、箭头、圆、卵形线和线段等等，用户可以把这些模块放在任何所希望的地方。

§6.1　图形的尺寸和位置

只有为图形建立了一个坐标系，其构造模块才有可能被安置。而坐标系由参考点 (或原点) 和两条互相垂直的坐标轴组成，坐标轴上要有单位长度以确定坐标。这里所取的原点就是图形的左下角，坐标轴就是底边与左边。这些边称为 x- 轴（底边）和 y- 轴（左边）。

一旦定义了单位长度 (UL)，图形中每点都唯一被两个数确定：第一个是沿 x- 轴的长度，第二个是沿 y- 轴的长度。



单位长度的选择是用如下命令进行的：

`\setlength{\unitlength}{` 长度 `}`

在上图中左边那个例子，单位长度用命令 `\setlength {\unitlength}{1.5cm}` 设置成 1.5cm 。那么点 $(1.8, 1.4)$ 即是指位于原点右边 1.8 倍单位长度（ $=$ 2.7cm ），原点上边 1.4 倍单位长度（ $=$ 2.1cm ）。

坐标值一般都是正数，这就是说点位于参考点的右上方向。由于参考点是图形的左下角，因此所有点都应该在它的右上方向。然而，坐标也可以取负值。负的 x 值（坐标对中第一个数为负数）定义的点在原点的左边，而负的 y 值（坐标对中第二个数为负数）定义的点在原点的下边。

上图右边的例子说明了这种情形。这里的单位长度为 1cm ，因此坐标值就是两个方向上离原点，以 cm 为单位的距离。

单位长度通常就设为方便的尺寸，如 1cm, 1mm 或 1in ，然后相应地建立我们的图形。当整幅图形完成后，只需修改单位长度的定义，就可以放缩

图形。一幅原始设计时 \unitlength 取 1cm，可以通过把单位长度重定义为 1.2cm 来放在为原来的 1.2 倍。

§6.2 图形环境

图形是用 picture 环境构造的，其语法为

\begin{picture}(x 尺寸， y 尺寸)
　　画图命令
\end{picture}

这里的 (x尺寸, y尺寸) 是一组数，它定义了图形在 x- 方向 (水平) 和 y- 方向 (竖直) 的尺寸 (范围)。这对数是用小括号围起来的！单位长度就是在此之前所定义的 \unitlength 。

```
\setlength{\unitlength}{1.5cm}
\begin{picture}(4,5) ... ... ...  \end{picture}
```

的结果为一幅 4 倍单位长度宽， 5 倍单位长度高的图形。而由于已把单位长度设为 1.5cm ，因此图形的实际尺寸为宽 6cm ，高 7.5cm 。

而 画图命令 就是下面将要介绍的用以生成和定位各个图形要素的命令。这些命令再加上字体样式和尺寸声明 (4.1 节) 以及线粗命令 \thincklines 与 \thinlines 就是可以位于 picture 环境中的所有命令。后面这两条命令用来确定两种可用线粗中的哪一种被用来绘制当前直线：我们按自己的意愿在两者之间来回切换。刚开始时激活的是细线。

参数 \unitlength 的值一定不能在 picture 环境内改变，因为对它而言，单位长度必须维持不变。当然可以在两幅图形中间改变它的值。

如果 \unitlength 的定义随同 picture 环境一起被包围在另一个类似于 \begin{center} ... \end{center} 这样的环境内，那么 \unitlength 值的作用持续到这个环境结束。一个前面没有 \unitlength 定义的 picture 环境，就会采用它的标准值 1pt 。

§6.3 定位命令

图形中的元素是用两条命令 \put 和 \multiput 来创建和定位的，它们的语法为：

\put(x- 坐标， y- 坐标){ 图形元素 }

\multiput(x- 坐标， y- 坐标)(x- 增量， y- 增量){ 数 }{ 图形元素 }

这里的 图形元素 就是下一节要讲述的画形基本命令。参数值 (x-坐标, y-坐标) 是安置坐标，规定了元素在画形坐标系中的位置，它是以 \unitlength 为

单位。如果单位长度为 1cm，那么 (2.5, 3.6) 就意味着元素定位在相对于图形左下角向右 2.5cm，向上 3.6cm 的地方。

命令 \multiput 是把同样的图形元素生成 数 次，每次移动 (x-增量, y-增量)。因此元素被重复地画在

(x- 坐标， y- 坐标), (x- 坐标 +x- 增量， y- 坐标 +y- 增量),

(x- 坐标 +2x- 增量， y- 坐标 +2y- 增量), ..., 一直到

(x- 坐标 +[数 -1]x- 增量， y- 增量 +[数 -1]y- 增量)

每画一次， (x-坐标, y-坐标) 都要增加 (x-增量, y-增量)。这里被增加的量可以是正数，也可以是负数。

因此 \multiput(2.5,3.6)(0.5,-0.6){5}{图形元素} 就会画五次 图形元素，第一个的位置在 (2.5, 3.6)，然后是 (3.0, 3.0), (3.5, 2.4), (4.0, 1.8)，最后是 (4.5, 1.2)。

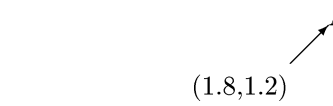
注意坐标和增量数对都是放在小括号（，）内，其中的两个数是用逗号分开的。而 数 和 图形元素 项则是像通常那样放在大括号内。

警告：由于这里是用逗号分开两个数，因此不能再用它表示小数点。对于坐标项，小数点必须是句号，而不能是逗号。

§6.4 基本画图命令

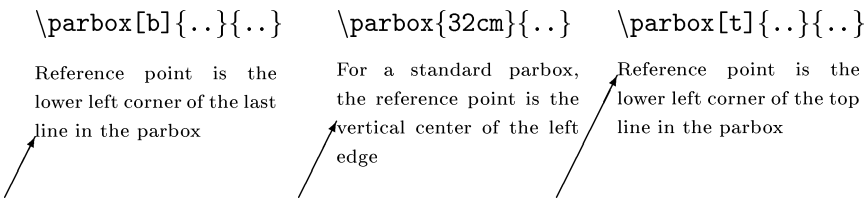
§6.4.1 图形中的文本

所有画形元素中最简单的就是一段文本，定位在图形中需要的位置。只要把文本放在 \put 或 \multiput 命令中的 图形元素 所在地方就可以了。



箭头所指的位置为 (1.8, 1.2)。
利用命令 \put(1.8,1.2){An arrow} 就会插入文本 'An arrow'，其左下角就是所指的位置。

做为图形中元素的文本也可以包装进一个 \parbox 或 minipage 环境中，这时在 \put 命令中坐标项的参考点与竖直盒子的定位参数值有关：



练习 6.1: 生成一幅宽 100mm，高 50mm 的图形，这里 \unitlength 取值 1mm。把给定文本放在下列位置： (0,0) 'The First Picture', (90, 47) 'upper left', (70,40) 'somewhere upper right'，并且放一个宽为 60mm 的子段盒子

在 (25,25) 点处，内容是 'A separate exercise file with the name `picture.tex` should be created for the exercise in this Chapter.'

练习 6.2: 给 `\unitlength` 取值 1.5mm，再重复上述的绘制操作，并且在子段盒子中让定位参数值分别取 `t` 和 `b`。

§6.4.2 图形中的盒子 — 矩形

在 `picture` 环境中也可用盒子命令 `\framebox`, `\makebox` 和 `\savebox`（4.7.1 节），但是其语法已做了推广。而且，还有一个盒子命令 `\dashbox`：

`\makebox(`x- 尺寸， y- 尺寸`)[`位置`]{`文本`}`

`\framebox(`x- 尺寸， y- 尺寸`)[`pos`]{`text`}`

`\dashbox{`虚线尺寸`}(`x- 尺寸， y- 尺寸`)[`位置`]{`文本`}`

尺寸数对 (x-尺寸, y-尺寸) 定义了矩形的宽度和高度，它们以 `\unitlength` 为单位。定位参数值 *位置* 定义了文本在盒子中的位置。它可以取如下值：

`[t]` top– 输入文本水平居中地位于盒子顶边的下面。
`[b]` bottom– 输入文本水平居中地位于盒子底边的上面。
`[l]` left– 输入文本竖直居中地位于盒子的左边。
`[r]` right– 输入文本竖直居中地位于盒子的右边。
2ε `[s]` stretch– 输入文本竖直居中，但要水平伸展以充满整个盒子。

如果没有可省参数 *位置*，那么输入文本是水平竖直居中放置在盒子中。

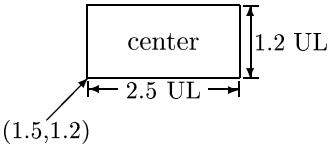
也可以一次这些参数中的两个组合起来使用：

`[tl]` top left– 输入文本位于左上角。
`[tr]` top right– 输入文本位于右上角。
`[bl]` bottom left– 输入文本位于左下角。
`[br]` bottom right– 输入文本位于右下角。

这里值的顺序是无关紧要的， `tl` 与 `lt` 的效果相同。

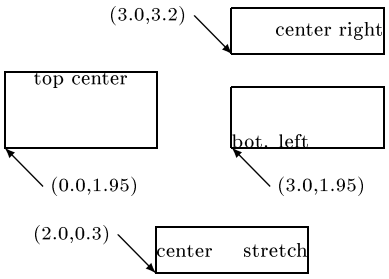
这些命令就是用在 `\put` 和 `\multiput` 命令中 *图形元素* 处。盒子的放置方式是其左下角所处位置就是放置命令中的坐标对。

```
\put(1.5,1.2){\framebox(2.5,1.2){center}}
```



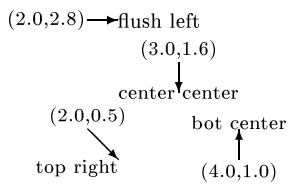
箭头所指位置为 (1.5,1.2)，这就是宽 2.5 单位，高 1.2 单位的矩形左下角所在的地方。文本 'center' 是水平和竖直居中的。UL = 0.8cm。

下面这个例子更好地说明了文本定位参数值的效果（ UL = 1cm ）：




```
\put(0.0,1.95){\framebox(2,1.0)
              [t]{top center}}
\put(3.0,1.95){\framebox(2,0.8)
              [lb]{bot. left}}
\put(3.0,3.2){\framebox(2,0.6)
              [r]{center right}}
\put(2.0,0.3){\framebox(2,0.6)
     [s]{center\hfill stretch}}
```

图形元素 \makebox 同 \framebox 命令完全一样，只是它没有矩形框而已。对它而言，经常把范围对取为 (0,0)，这样可以把文本放在所希望的地方。（关于零宽度盒子对被包围文本的效果请见 4.7.1 节。）

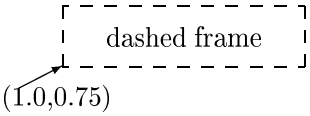



```
\put(3,1.6){\makebox(0,0){center center}}
\put(2,0.5){\makebox(0,0)[tr]{top right}}
\put(4,1.0){\makebox(0,0)[b]{bot center}}
\put(2,2.8){\makebox(0,0)[l]{flush left}}
```

[lb] 组合定位的文本就与不用盒子，直接把文本做为图形元素时的效果一样，见 6.4.1 节。

图形元素 \dashbox 也生成有框盒子，不过其框线为虚线。参数值 虚线尺寸 就是用来定义短线长度。

```
\put(1.0,0.75){\dashbox{0.2}(4,1){dashed frame}}
```



当盒子的宽度和高度都是短线长度的倍数时，虚线框要好看一些。

即使在上面这些图形盒子命令中，也可以把文本放在竖直盒子 (\parbox 或 minipage) 中。由于竖直盒子自身具有可省的定位参数值 b 或 t，它一定不能与图形盒子的定位参数值冲突，因此要遵守下面的规则：

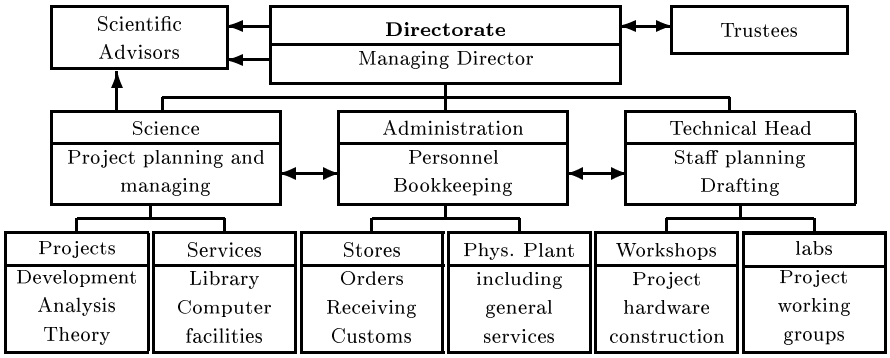
> 如果图形盒子中包含了定位参数值 b 或 t，那么被包围的竖直盒子中也必须有相同的定位参数值。如果图形盒子中没有定位参数值，或者只是 r 或 l，那么竖直盒子必须是标准（无参数值）形式。

图形盒子中的定位参数值对于被包围竖直盒子的作用同它对一行文本的作用一样，都是把它们当做一个整体对待。

练习 6.3: 复制下面这个机构表格，要求包含文本，但现在先不生成水平和竖直直线及箭头，它们是后面要做的练习。

提示：首先在一张有格的纸上画出方框，而且盒子的边与格线重合。把单位长度就取为格线间距。把原点取做想像中的包含所有盒子的方框左下角。

注意：很快你就能生成自己的有格子的页面，其中格线间距可以是任意希望的尺寸。



§6.4.3 直线

在 picture 环境中， LaTeX可以绘制任意长度的水平，竖直以及有限倾角的直线。这个图形元素的语法是：

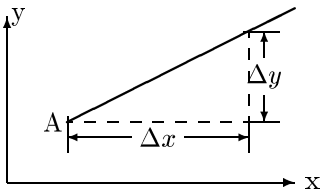
\line(Δx, Δy){ 长度 }

对于水平线和竖直线， 长度 定义了以单位长度为单位的线长。对于其它倾角的直线，它的意义就有点儿复杂，稍后加以解释。直线开始于由 \put 或 \multiput 命令中给出的安置坐标确定的点。




```
\thicklines
\put(0,0){\line(1,0){6}}
\put(0,0){\line(0,1){1}}
\put(6,0){\line(0,1){0.5}}
```

给制直线时的倾角是由斜率对 $(\Delta x, \Delta y)$ 给定的。斜率对 (1,0) 中 $\Delta x = 1$, $\Delta y = 0$ ，这样生成水平线，而 (0,1) 生成的则是竖直线。上面的例子说明了这一点。一般地 $(\Delta x, \Delta y)$ 具有下面的含义：



从直线上的 A 点开始，沿着 x 方向 (水平) 走 Δx 距离，那么 Δy 就是沿 y 方向 (竖直) 移动的距离，从而可以重回到直线上。

通过定义斜率对 (Δx, Δy) ，那么绘制出来的直线倾角应该满足上面的条件。

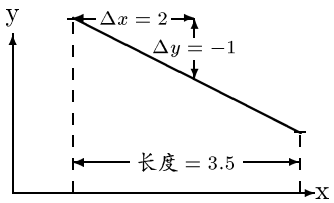
前面已经说过，只可以绘制有效倾角的直线。这是因为 Δx 和 Δy 的取值要遵从下面的规则：

1. 数值必须是整数 (负数或正数均可)。
2. 只可以取值 0,1, . . . , 6 。
3. 在数对中的两数不能有公因子。

因此类似 (3.5,1.2) (规则 1) 和 (7,0) (规则 2) 这样的数对是不允许出现的。同样 (2,2) 和 (3,6) 不符号规则 3 ，因为前一对中的两个数可以都被 2 整除，而后一对中的两数可以都被 3 整除。可以用 (1,1) 和 (1,2) 来得到同

样的倾角。因此这里一共有 25 种可接受的斜率对，其中 (1,0) 和 (0,1) 分别相应于水平线和竖直线。可以通过把所有的可能写出来以验证这一总数。

另外，在斜率对中的数值可以是负数，也可以是正数，例如 (0,-1) 和 (-2,-5) 也是允许的。在上面图示中，负的 Δx 意味着向左移动，而负的 Δy 意味着向下移动。因此 \put(2,3){\line(0,-1){2.5}} 的结果是一条开始于 (2,3) 的直线，其竖直向下伸展长达 2.5 单位。



对于有倾角的直线，参数值 *长度* 定义的是沿 x- 轴的投影长度。这一点可以借助于左边的图示更清楚地看出来。

```
\put(1.0,2.75){\line(2,-1){3.5}}
```

如果我们从两个端点竖直向下画虚线，那么在这两虚线之间的 x- 轴部分就是直线在 x- 轴上的投影。

倾斜直线的长度必须不能短于 10pt 或者 3.5mm ，否则不会生成任何结果。但是如果包含了 pict2e 软件包（ 6.5.6 节），就不会有这种限制。

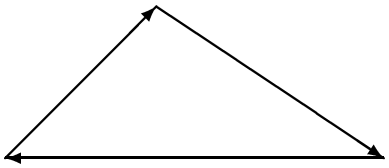
§6.4.4 箭头

箭头图形要素是用下面的命令生成的：

\vector($\Delta x, \Delta y$){ *长度* }

其作用方式同 \line 命令完全一样，而且参数值和其局限性也是相同的。这条命令从由 \put 或 \multiput 命令定义的位置开始画一条直线，然后在终点处画上箭头。

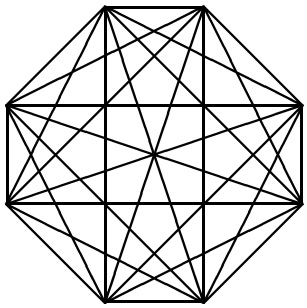
同直线一样，箭头的长度也不能少于 10pt 或 3.5mm 。规则 1–3 也同样适用于 Δx 和 Δy ，而且更进一步，要求可以取的值只能是 0,1,2,3,4 。这样当不考虑正负号时，只能画 13 种不同倾角的箭头。



```
\begin{picture}(5,2)\thicklines
  \put(5,0){\vector(-1,0){5}}
  \put(0,0){\vector(1,1){2}}
  \put(2,2){\vector(3,-2){3}}
\end{picture}
```

练习 6.4: 补上练习 6.3 中还没画的水平和竖直直线与箭头，完成那个演示图。

练习 6.5: 生成一张 6.5×9 英寸的方格纸，格线距离为 0.1in 。这只需要用两个 \multiput 命令就可以了。在这层网格上重叠同样全局尺寸，但是格线距离为 0.5in 的方格，而且线粗由 \thicklines 给出。



练习 6.6: 生成右边的插图。
顶点是 (0,5), (0,10), (5,15), (10, 15), (15,10), (15,5), (10,0) 和 (5,0) ，单位长度为 0.1in 。

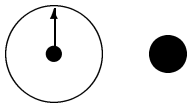
§6.4.5 圆

圆这种图形要素是用下面的命令得到的：

`\circle{` 直径 `}`

`\circle*{` 直径 `}`

利用 *- 形式的命令，可以画出内部被填充了的实心圆，而不是标准形式画出的那种轮廓线。只能画特定尺寸的圆，因此 LaTeX会选出与指定 直径 最接近的圆。（ 6.5.6 节介绍的 pict2e 软件包可以绘制任意尺寸的圆。）



```
\begin{picture}(3,1.6)
  \put(1,1){\circle*{0.2}}
  \put(1,1){\circle{1.2}}
  \put(1,1){\vector(0,1){0.6}}
  \put(2.5,1){\vector*{0.5}}
\end{picture}
```

在相应的 \put 命令中的安置位置对应于圆心。

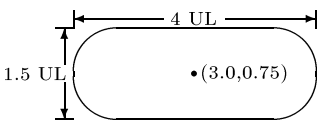
§6.4.6 卵形线与圆角

我们这里所说的卵形线指的是一种矩形，其顶角用四分之一圆角代替；这里对直径的选取是使所有边光滑拼接的最大值。生成卵形线的命令是

`\oval(x-` 尺寸， `y-` 尺寸 `)[` 部分 `]`

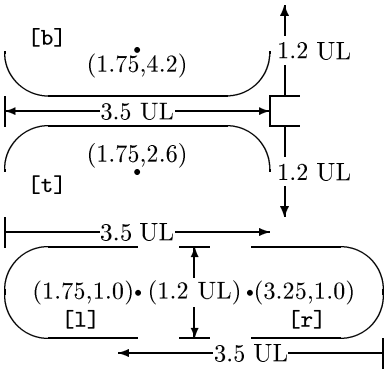
在相应 \put 命令中的安置坐标对应于卵形线的中心。

```
\put(3.0,0.75){\oval(4.0,1.5)}
```



这里我们取 x-尺寸=4.0 UL, y-尺寸=1.5 UL ，而单位长度 UL 已选择为 0.8cm 。卵形线的中心就是 \put 命令中的安置坐标 (3.0,0.75) 。

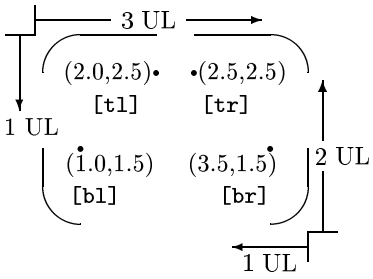
可省参数 部分 可以取值 `t` ， `b` ， `l` 或者 `r` ，以生成一半的卵形线。




```
\put(1.75,4.2){\oval(3.5,1.2)[b]}
\put(1.75,2.6){\oval(3.5,1.2)[t]}
\put(1.75,1.0){\oval(3.5,1.2)[l]}
\put(3.25,1.0){\oval(3.5,1.2)[r]}
```

半卵形线的宽度和高度与整个都要画出来时是一样的，即使这里只是画一半出来。类似地，在相应 \put 命令中的安置坐标仍然对应于完整卵形线的中心。（这里的单位长度为 UL=1 cm 。）

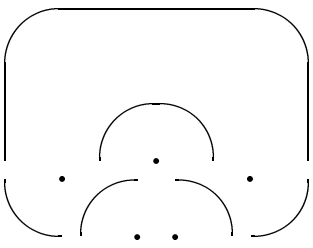
参数值 *部分* 也可以是四种组合 tl, tr, bl 或 br 中的一种，以生成四分之一的卵形线。这里两字母的顺序是无关紧要的，因此也可以用 lt, rt, lb 或 rb 。




```
\put(2.0,2.5){\oval(3.0,1.0)[tl]}
\put(2.5,2.5){\oval(3.0,1.0)[tr]}
\put(1.0,1.5){\oval(1.0,2.0)[bl]}
\put(3.5,1.5){\oval(1.0,2.0)[br]}
```

同样这里的尺寸定义仍旧参照完整卵形线而进行，即使画出来的只是一部分，在 \put 命令中的安置坐标也是指的整个卵形线的中心。

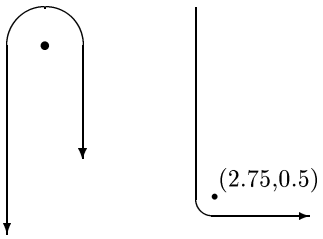
通过把卵形线中宽度与高度取成相等的值，可以得到四分之一或者半个圆周，但正如前面对圆的限制一样，也只能用特定的尺寸。下面的示例说明了部分圆周可以有小至 1.5cm 的尺寸。



```
\put(2.0,1.0){\oval(4.0,4.0)[t]}
\put(2.0,1.0){\oval(1.5,1.5)[t]}

\put(0.75,0.75){\oval(1.5,1.5)[bl]}
\put(1.75,0.0){\oval(1.5,1.5)[tl]}
\put(2.25,0.0){\oval(1.5,1.5)[tr]}
\put(3.25,0.75){\oval(1.5,1.5)[br]}
```

部分卵形线也可以与其它图形要素组合。当然这时可能需要对 \put 命令中的安置坐标进行相当仔细的考虑，以定位准确。



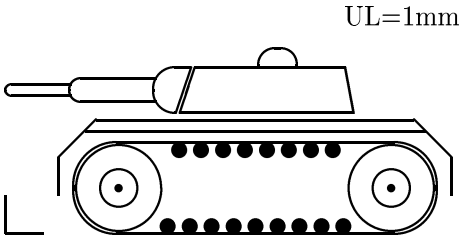

```
\put(0.5,2.5){\oval(1.0,1.0)[t]}
\put(0.0,2.5){\vector(0,-1){2.5}}
\put(1.0,2.5){\vector(0,-1){1.5}}
\put(0.5,2.5){\circle*{0.1}}

\put(2.5,0.5){\line(0,1){2.5}}
\put(2.75,0.5){\oval(0.5,0.5)[bl]}
\put(2.75,0.25){\vector(1,0){1.25}}
```

在上面所有例子中, 卵形线的中心都用一个点标记出来, 以说明其位置。通常它们并不是 \oval 图形要素的一部分。

练习 6.7: 虽然在冷战后年代, 画下面这个东西并不是很有意义, 但是它是一个相当棒的练习 picture 环境的机会。

提示: 用透明格纸罩在下面的图上, 可以很方便地进行定位和确定尺寸。



§6.4.7 竖直堆积文本

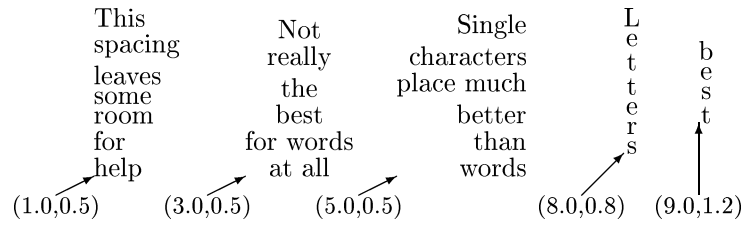
有时在演示图中需要沿竖直方向书写文本, 如本段右页那样所示。这可以用如下命令来做到

\shortstack[位置]{ 列 }

位置 参数值可以取 l, r 或 c 。标准值是 c , 表示居中。该命令类似于只有一列的 tabular 环境。 列 表示输入文本, 行与行之间用 \\ 分开。

y-axis

\shortstack 命令经常用来实现把单个字母上下安置, 即使是长文本也可以竖直堆积。行与行之间用尽可能小的间距分开。这就意味着没有向下或向上突出字母 (如 h 和 y) 的行离相邻行的距离要比有这样字母的行离相邻行近。



相应 \put 命令中的安置坐标对应于想像中的包含竖直堆积文本的盒子的左下角。上面例子中第一个的文本是左对齐的，第二个是居中，第三个是右对齐。最右面的两个是居中的。它们是用如下输入生成的：

```
\put(1.0,0.5){\shortstack[l]{This\\spacing\\leaves\\some\\ ...}}
\put(3.0,0.5){\shortstack{Not\\really\\the\\best\\ ...}}
\put(5.0,0.5){\shortstack[r]{Single\\characters\\ ...}}
\put(8.0,0.8){\shortstack{L\\e\\t\\t\\e\\r\\s}}
\put(9.0,1.2){\shortstack{b\\e\\s\\t}}
```

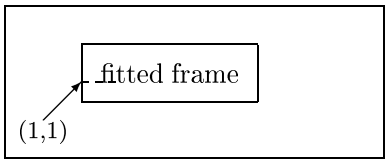
\shortstack 命令也可以用在 picture 环境外面的普通文本中。其中一个应用就是页边注，见 4.10.6 节。

§6.4.8 有框文本

\framebox 命令生成一个具有预先定义尺寸的有框盒子，其中的文本可以放在不同的地方（ 6.4.2 节）。在文本模式中，命令 \fbox 可以围绕文本画一个方框，而且其会与文本匹配得很好（ 4.7.1 节）。这条命令也可以用在 picture 环境中。

在盒子的框线与被包围文本之间的距离是由参数 \fboxsep 所确定的。利用 \put 命令来放置一个 \fbox ，其方式是出人意料的，请看下图：

```
\begin{picture}(5,2)
\setlength{\fboxsep}{0.25cm}
\put(0,0){\framebox(5,2){}}
\put(1,1){\fbox{fitted frame}}
\end{picture}
```



在演示图中通常不期望在框中出现多余的空白，特别当方框包围的是一幅图，而不是文本时更是如此。在这种情况中，可以使用命令

\frame{ 图形元素 }

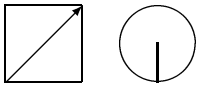
\put 命令中的安置坐标与通常一样，相应于左下角。

```
\put(0.0,0.5){\frame{TEXT}}
\put(1.5,0.0){\frame{\shortstack{W\\O\\R\\D}}}
```



\frame 命令中的内容并不一定只是文本，也可以是前面给出的图形元素。然而，很多情形中输出总有点儿不对头。

```
\put(0,0){\frame{\vector(1,1){1.0}}}
\put(2,0){\frame{\circle{1.0}}}
```



第一条命令得到正确的结果，而第二条命令却失败了。对此情况，我们可以尝试把图形对象放在适当尺寸 \makebox 中，然后把它作在 \frame 命令的参数值。然而，用 \framebox 命令代替 \frame{\makebox...} 会更合理。

§6.4.9 曲线

在 picture 环境中也可以用下面的命令画曲线：

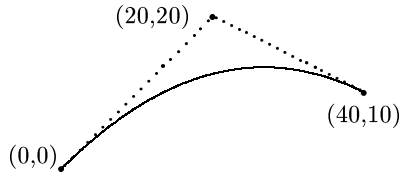
\bezier{ 数 }$(x_1,y_1)(x_2,y_2)(x_3,y_3)$

2ε \qbezier[数]$(x_1,y_1)(x_2,y_2)(x_3,y_3)$

它会画出一条从点 (x_1,y_1) 到 (x_3,y_3) 的二次 Bézier 曲线，而 (x_2,y_2) 是 Bézier 控制点。曲线实际上是用 数+1 个点画出来的。 \bezier 和 \qbezier 命令之间的唯一区别在于，对后者而言， 数 是一个可省参数值；如果省略了它，那么就会计算生成一条光滑曲线所需要的点数。之所以还保留 \bezier 命令，就是为了与老的 LaTeX2.09 文档兼容。

Bézier 控制点的意义可以从右面的例子看出来。输入文本为

```
\begin{picture}(40,20)
\qbezier(0,0)(20,20)(40,10)
\end{picture}
```



这条曲线是从 (0,0) 到 (40,10) , 那么在端点处的切线(虚线)就经过 Bézier 控制点 (20,20) 。另一种表述这一点的方法是：当要从第一个点移到第三个点时，我们首先面向第二个点前进，为了到达目的地，我们要背向第二个点前进。上面例子中的虚线并不是 \qbezier 函数绘制出来的，只是为了说明上述含义。

通过指定点 数 , 可以绘制出虚线, 要得到这种效果, 可能需要试验几次。

注意： \bezier 命令并不是积成在 LaTeX2.09 中的部分, 它是包含在一个叫 bezier.sty 的文件中，必须把 bezier 列在 \documentstyle 的选项中，这样才能读取它。在 LaTeX 2_ε 中为了保证兼容性，提供了同名的空文本。

§6.5 其它图形命令与示例

§6.5.1 直线粗细

对于图形元素 \circle 、 \oval 、 \vector 和斜线，存在两种可选择的线粗。即可以用

\thicklines 或 \thinlines

来选择粗线或细线。这两条命令的作用直到有相反命令为止，或者其所在环境结束。刚开始时 \thinlines 起作用。

水平或竖直直线的线粗可以用下面的声明来定义成任意期望的尺寸：

\linethickness{ 粗细 }

参数值 粗细 是正的长度定义。利用 \linethickness{1.5mm} 可以使得所有后面的水平或竖直线粗 1.5mm 。如果包含了 pict2e 软件包 (6.5.6 节), 这

个粗细定义也同样适用于斜线。

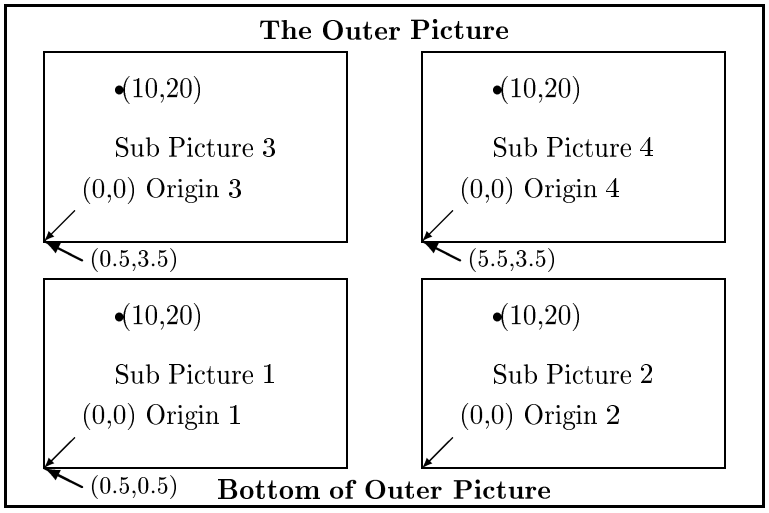
由于方框就是用竖直线和水平线构成的，因此命令 \linethickness 的定义对 \framebox 和 \dashbox 也同样有作用。

§6.5.2 嵌套图形

在 \put 或 \multiput 命令中的 *图形元素* 也可以是另一个 picture 环境。如此多重图形的语法是：

```
\put(x- 坐标， y- 坐标 ){\setlength{\unitlength}{ 单位长度 }
                        \begin{picture}{x- 尺寸， y- 尺寸 )
                        ... 子图形 ... \end{picture} }
```

在内部图形环境中的安置坐标是相对于它自己的原点的，即其左下角；而原点就对应于外面 picture 环境中 \put 命令的安置坐标。可以给内部 picture 环境选择不同的 \unitlength *单位长度* 值；然而，如果没有定义新的值，那么它就用与外面 picture 环境相同的 \unitlength 值。




```
\begin{picture}(10.0,6.6)
\thicklines\put(0,0){\framebox(10.0,6.6){}}
 \put(5.0,6.3){\makebox(0,0){\bfseries The Outer Picture}}
 \thinlines
 \put(.5,.5){\setlength{\unitlength}{1mm}
 \begin{picture}(50,25)
 \put(0,0){\framebox(40,25){Sub Picture 1}}
 \put(10,20){\circle*{1}} \put(10,20){\makebox(0,0)[l]{(10,20)}}
 \put(4,4){\vector(-1,-1){4}}
 \put(5,5){\makebox(0,0)[lb]{(0,0) Origin 1}} \end{picture}}
\put(5.5,0.5){ ... Sub Picture 2 ...  }
```

```
\put(0.5,3.5){ ... Sub Picture 3 ...  }
\put(5.5,3.5){ ... Sub Picture 4 ...  }
\put(5,0.1){\makebox(0,0)[b]
    {\bfseries Bottom of Outer Picture}}
\end{picture}
```

最外层图形的单位长度设为 UL=1 cm，该图形宽 10 cm，高 6.6 cm。它的图形要素中有一个粗线的方框，这个方框的尺寸与图形的尺寸一样，其它的元素有：两块文本 'The Outer Picture' 和 'The Bottom of the Outer Picture'，四幅小图形，每一幅的取 UL=1 mm 的单位长度和 40×25mm 的尺寸。在每个小图形中的对象相对于它们自己的原点定位。在每个小图形中 • 符号都具有相同的坐标 $(10, 20)$。

通过使用嵌套图形，可以简化逻辑上彼此相关的特定对象之间的相互定位，从而减少了定位时可能出现的错误。当调用 \put 命令时把 \unitlength 命令设成不同值，尤其如此。

§6.5.3 存贮部分图形

可以把图形中的一组图形元素用特定的名称保存成子图，然后就能随时调用整组命令，而不需一条一条地调用。

首先必须为每个子图起一个名称，所用命令为：

\newsavebox{\ 子图名称 }

这样就会创建一个名叫 \子图名称 的用于保存图形的盒子。然后，用下面的命令保存子图：

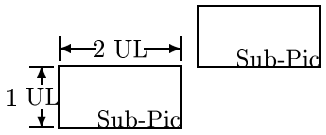
\savebox{\ 子图名称 }(x- 尺寸， y- 尺寸)[位置]{ 子图 }

这里的参数值 (x-尺寸,y-尺寸) 与 位置 同 6.4.2 节 \makebox 中的含义一样。

如果图形命令 \子图 只是一小块文本，那么这条命令就与 \makebox 命令几乎完全一样，除了文本不是显示出来，而是存贮到 \子图名称 中。子图可以用下面这条命令当做一个图形元素安置在主图内的任何地方。

\usebox{\子图名称}

```
\newsavebox{\sub}
\savebox{\sub}(2,1)[br]{\small Sub-Pic}
....
\put(0.7,0.0){\frame{\usebox{\sub}}}
\put(3.0,1.0){\frame{\usebox{\sub}}}
```



这个例子看不起并不怎样，因为这里的 \savebox 和 \usebox 命令如果用 \framebox 结合 \multiput 命令取代，简单地就可以得到同样的结果。然而，这两条命令的主要优势并不是安置多重文本，而是更复杂的 子图 组合。

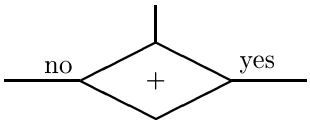
这里要指出，\savebox 命令可以在 picture 环境外面调用，甚至可以把它放在导言中。这样该子图就可以在整篇文档的所有 picture 环境中使用。

然而，如果 \savebox 是定义在一个环境中，那么其中的值当环境结束时也就没有了。

放在 \savebox 中的图形元素将会根据盒子被构造时候的单位长度确定尺寸。但不会受随后 \unitlength 改变的影响。

```
\newsavebox{\testcon}
\savebox{\testcon}(0,0){%
  \thicklines
  \put(0,0.5) {\line(-2,-1){1.0}}
  \put(0,0.5) {\line(2,-1) {1.0}}
  \put(0,-0.5){\line(-2,1) {1.0}}
  \put(0,-0.5){\line(2,1)  {1.0}}
  \put(0,0.5) {\makebox(0,0)[b]
     {\put(0,0){\line(0,1){0.5}}}}
  \put(1.0,0) {\line(1,0){1.0}}
  \put(-1.0,0){\line(-1,0){1.0}}
\put(-1.1,0.1){\makebox(0,0)[br]{no}}
\put(1.1,0.1){\makebox(0,0)[bl]{yes}}}
```

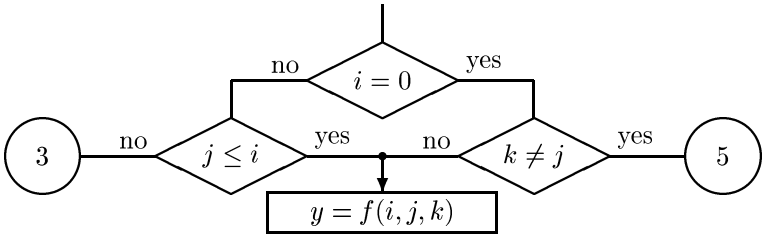
这定义了条件分枝符号，它经常用在计算机程序的流程图中。



+ 表示符号 \textcon 的参考点。

在上面的 \savebox 命令中，参数值 (0,0) 定义了一个零宽度和零高度的盒子，这样它的中心就放在 \put 命令中安置坐标定义的地方。因为我们想要菱形的中心就是这个参考点，因此菱形上面的竖直线必须没有高度，否则它的出现就会使得整幅图的中心偏离菱形的中心。因此竖线（ line(0,1){0.5} ）就放在一个零尺寸的 \makebox 盒子中。这样整幅图的参考点（如外面的 \put 命令中一致）就是 + 号所表示的地方。

现在 \testcon 命令可以很容易地与其它图形要素组合起来。




```
\begin{picture}(10,3)  \thicklines
\put(5,2){\usebox{\testcon}\makebox(0,0){$i=0$}}
\put(3,1){\usebox{\testcon}\makebox(0,0){$j\le i$}}
\put(7,1){\usebox{\testcon}\makebox(0,0){$k\neq j$}}
\put(0.5,1){\circle{1.0}\makebox(0,0){3}}
\put(9.5,1){\circle{1.0}\makebox(0,0){5}}
\put(5,1){\vector(0,-1){0.5}\circle*{0.1}}
```

```
\put(3.5,0){\framebox(3,0.5){$y = f(i,j,k)$}}
\end{picture}
```

这个例子说明，在一条 \put 命令中可以包含不只一条图形对象。这里把表示符号的 \usebox{\testcon} 命令同包含文本的 \makebox(0,0) 放在了一起。类似地， \circle{1.0} 命令分别同其中的居中数字 3 和 5 放在一条 \put 命令中。最后， \vector 和 \circle 命令也放在了一起。当多条命令放在同一 \put 命令中时，在图形元素之间必须不能有空格。

也可以用 \savebox 命令存贮一个完整的 picture 环境。这时参数值 (x-尺寸,y-尺寸) 和 位置 是被忽略的，因为在 picture 环境中就有这些尺寸定义。语法为

```
\savebox{\ 图形名称 }{\begin{picture}(x- 尺寸， y- 尺寸 )
                                    ... ... ...
                                    \end{picture} }
```

如果要把这样存贮的图形做为子图来使用， (x-尺寸,y-尺寸) 应预设为 (0,0) ，因为这样使得可以相对于子图定位。被保存的图形将会根据外面的 picture 环境进行放缩。前面的示例符号 \testcon 可以这样保存起来：

```
\savebox{\testcon}{\begin{picture}(0,0) ...
                                    \end{picture} }
```

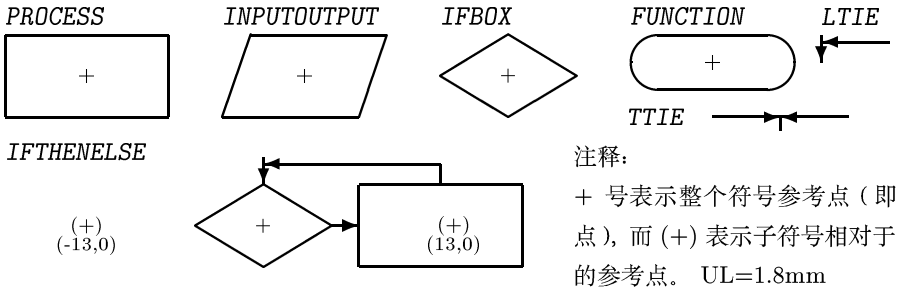
这里的 ... 代表前一页中定义 \testcon 时相同命令集合，从 \thicklines 开始，到最后一个 } 结束。

把子图作为 picture 环境保存起来的另一个好处是：即使点 (0,0) 并不是符号的真正中心，但也可以把它设成外部 \put 命令的参考位置点。这也就是说在 \testcon 的定义中，我们可以把

```
\put(0,0.5) {\makebox(0,0)[b]{\put(0,0){\line(0,1){0.5}}}}
```

改成 \put(0,0.5){\line(0,1){0.5}} 。也就是现在不需要 '隐藏' 会把符号中心从点 (0,0) 移开的竖直线。

练习 6.8: 下面这些符号经常出现在计算机程序的流程图中。生成给定的盒子，并把它保存为具有相应名称的符号。用这些较简单的符号组合成复杂的符号。注意连接时的相对坐标和参考点。

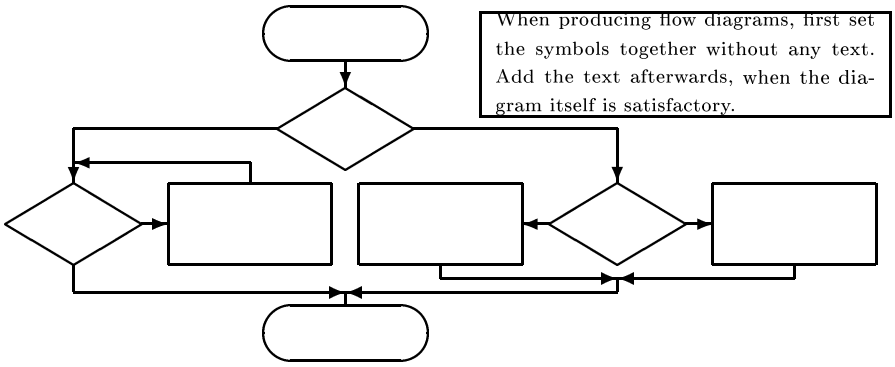


注释：
+ 号表示整个符号参考点 (即原点)，而 (+) 表示子符号相对于 + 的参考点。 UL=1.8mm

©TEXGuru, August 15, 1999



利用上面的符号生成下面这幅图.



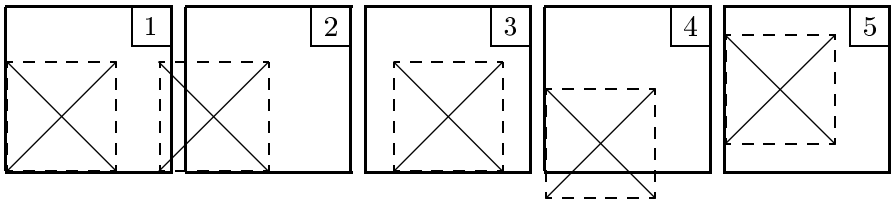
§6.5.4 图形环境的广义语法

在 picture 环境的广义语法中多了一对坐标，其为可省参数值:

\begin{picture}(x- 尺寸， y- 尺寸)(x- 偏移， y- 偏移)

图形命令 \end{picture}

在这种形式中，(x-偏移, y-偏移) 定义了左下角的坐标。这就是说对环境中所有 \put 命令，要从其定位坐标中减去 x-偏移 和 y-偏移，这样整幅图就向左平移了 x-偏移 单位，向下平移了 y-偏移 单位。



这里我们给出了五张图形，每个图形中有一个细线方框，其宽 3UL，高 3UL(UL=7.2mm)。在每个方框中, 用 \put 加入了一个子图, 其中有 2×2UL 的虚线框及两条对角线。每幅图除了偏移外都是一样的。

1. \put(0,0){\begin{picture}(2,2)(0,0) 子图 \end{picture}}
2. \put(0,0){\begin{picture}(2,2)(0.5,0) 子图 \end{picture}}
3. \put(0,0){\begin{picture}(2,2)(-0.5,0) 子图 \end{picture}}
4. \put(0,0){\begin{picture}(2,2)(0,0.5) 子图 \end{picture}}
5. \put(0,0){\begin{picture}(2,2)(0,-0.5) 子图 \end{picture}}

在情形 2 和 4 中，部分子图位于主图边界外面，当有偏移时必须考虑这一点。

§6.5.5 更多的例子

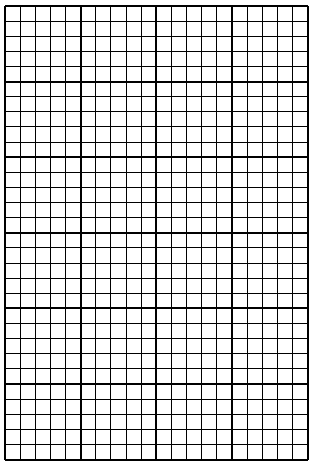
到现在为止，我们已经通过举例，相当详细地说明了各种图形元素。然而，我们有点儿冷落了 \multiput 命令，因此这里给出一些例子说明它的用法。（在 6.3 节中描述了 \multiput 命令。）

```
\multiput(0,0)(1,2){7}{\circle*{1}}
\multiput(8,0)(2,0){10}{\begin{picture}(0,0)
        \multiput(0,0)(1,2){7}{\circle*{1}}
                    \end{picture}}
```



第一条 \multiput 命令生成七个直径为 1mm 的点，它们从 (0,0) 开始，然后每次向右平移 1mm，同时向上平移 2mm。第二条 \multiput 命令生成 10 个子图，每个子图相对于前者向右平移 2mm。而子图本身就是前面第一条 \multiput 命令生成的七点。

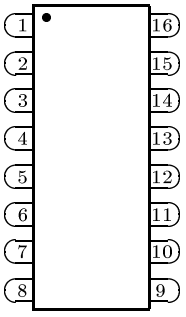
例：网格



```
\setlength{\unitlength}{0.1in}
\begin{picture}(20,30)
\linethickness{0.25mm}
  \multiput(0,0)(10,0){3}{\line(0,1){30}}
  \multiput(0,0)(0,10){4}{\line(1,0){20}}
\linethickness{0.15mm}
  \multiput(5,0)(10,0){2}{\line(0,1){30}}
  \multiput(0,5)(0,10){3}{\line(1,0){20}}
\linethickness{0.075mm}
  \multiput(1,0)(1,0){19}{\line(0,1){30}}
  \multiput(0,1)(0,1){29}{\line(1,0){20}}
\end{picture}
```

首先画出三条粗 0.25mm，长 30UL 的竖线，同样粗细的四条长 20UL 的横线，它们都是从 (0,0) 开始，彼此间距 10UL。而另两条竖线和三条横线粗 0.15mm，是从 (5,0) 和 (0,5) 开始画。最后我们画出了粗 0.075mm 的 19 条竖线和 29 条横线，它们彼此间隔 1UL。

IC 符号

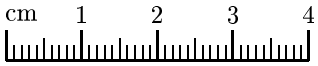



```
\newcounter{ic} \setlength{\unitlength}{1mm}
\begin{picture}(25,40)  \thicklines \scriptsize
\put(5,0){\framebox(15,40){}} \thinlines
\multiput(5,37.5)(0,-5){8}{\oval(8,3)[l]%
\stepcounter{ic}\makebox(0,0)[r]{\arabic{ic}\,\,}}
\put(20,2.5){\oval(8,3)[r]%
\stepcounter{ic}\makebox(0,0)[l]{\,\,\arabic{ic}}}
\multiput(20,7.5)(0,5){7}{\oval(8,3)[r]%
\stepcounter{ic}\makebox(0,0)[l]{\,\arabic{ic}}}
\put(6.5,38.5){\circle*{1}}
\end{picture}
```

©TEXGuru, August 15, 1999

这个例子中的两条 \multiput 命令都包含两个图形对象：半个卵形线和自动增加的文本。第一条 \multiput 命令从上到下生成左边的对象，而第二条是从下向上生成右边的对象。命令 \newcounter{ic} 建立一个叫 ic 的新记数器，然后用 \stepcounter{ic} 逐次增 1 ，并用 \arabic{ic} 来以阿拉伯数字显示出它的值。（在上面的示范源文本中，每条 \multiput 命令都占有两行；而事实上，这条命令的输入必须没有空格或回车以避免图形的水平移位！）

练习 6.9: 利用右面的厘米尺为模型，生成长 10cm 的横尺和竖尺。



练习 6.10: 改进练习 6.5 中的方格纸，方法是如上面例子那样把每个第 5 条和第 10 条直线变粗。并且沿外边界标出每个第 10 条直线。

§6.5.6 扩展软件包 2ε

在标准 LaTeX 2ε安装中，为了增强 picture 环境的功能，提供了两个软件包。

去掉某些限制

利用打印机驱动程序的一些功能，可以去掉对斜线长度和倾角的限制。软件包 pict2e 就是为了激活特定驱动程序的这种能力页设计。结果可能不会是真的与设备无关，但应该对于支持这种绘图操作的驱动程序都行得通。

利用这个软件包，圆也可以具有任意直径，而不只是局限于那些可用的特殊圆字体。

像通常那样，把这个软件包用 \usepackage{pict2e} 包含在导言中。

注意：在编写本书时 (August 15, 1999) 这个软件包还没有付诸于实用，因此这里无法给出其选项或驱动程序选择。

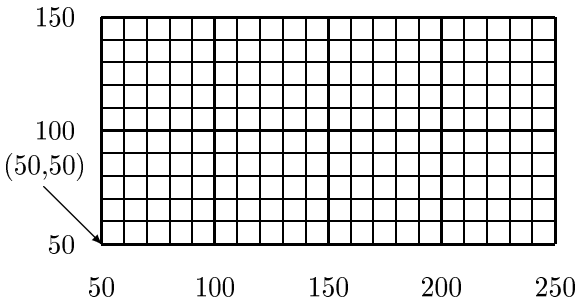
格纸

软件包 graphpap 增加了一条绘制格纸的命令

2ε \graphpaper[数](x,y)(lx,ly)

这条命令把方格的左下角放在 (x,y) 处，它有 lx 单位宽， ly 单位高。每隔给定的 数 单位就划一条格线，而每个第 5 条格纸要粗一些，并且加上标记。如果没有给出 数 ，那就假定它是 10 。所有参数值必须是整数，而不能是小数。

例如， \graphpaper(50,50)(200,100) 的结果为 (UL=0.3mm):



§6.5.7 一般性建议

在实际应用中，每个用户都会形成自己使用 picture 环境的风格。这里我们要讲述由 Leslie Lamport 给出的一些建议，我们对这些建议是非常赞成的，而且也增加了一些我们自己的看法。

1. 对于绘制和定位单个图形元素，各种网格尺寸的格纸是相当有用的。借助于这种网格，或者 \graphpaper 命令，用户可以生成任意尺寸的格纸，然后复制足够份，以备将来使用。
2. 网格的最小间距应就取为单位长度的大小。这样可以避免定位时出现小数。
3. 如果图形中没有斜线，那么图形元素的位置坐标可以直接从格纸上读出来。单个对象的参考点应是一个格点，并不是位于格点之间。
4. 对于斜线，只有有限的几个斜率（ 6.4.3 节）。当指定了一个可以接受的斜率后，那么直线的端点应该放在格点上。对箭头也要这样做（ 6.4.4 节）。（如果包含了 pict2e 软件包就不必这样做了。）
5. 图形应该利用子图构成，或者是子图的子图构成（ 6.5.2 节）。这样就可以通过相对于子图定位，从而简化图形绘制中的这种操作。
6. 经常用的子图应该保存在 \savebox 中，并收集到一个或多个独立的文件中。利用这种方法，经过几年的积累，我们可以建立自己的符号库。建立这些符号的文档时应该包含它们的名称、参考点和联系点，这样其它用户也可以用它了。

在 picture 环境一个很小的错误，也有可能导致毁灭性的后果。因此如果得到的结果完全出乎意料之外，也没有必要焦虑。例如，若一幅图形设计时是按照单位长度为 1cm 进行的，而没有设定 \unitlength ，这样 LaTeX就会取默认值 1pt ，从而使得图形几乎缩成一点，其中包含的文本不可想像地都显示在一点上。如果图形显示在主图区域外面，对于这种出乎意料的结果是由于不正确的定位造成的，可能是错误的符号或者漏掉了小数点。

有时候在水平定位中，尤其是几个图形用一条 \put 或 \multiput 命令安置时，会出现一些小错。这可能是在图形命令之间加了空格。这样的空格

也会被当做图形元素，从而把下一个符号从右进行了一点移位。因此在元素命令之间应该没有空格或回车（新行）。如果还是有错误，那就试着换一下安置命令中的元素顺序。如果这样做还行不通，那就只好每个元素用一条 \put 或 \multiput 命令了。

§6.6 浮动表格和插图

picture 环境生成的图形显示在输入文本的地方，前后都是其它文本。如果当前页上有足够的地方，这通常没有任何问题，而且也是我们所期望的。然而如果插图（或者表格）很高，在当前页上放不下来，那么就会结束当前页，把插图（或表格）放在下一页的顶部。这样当前页的格式就会很难看。

如果存在下面这种机制就相当好了：若当前页空间足够，就把插图或表格放在当前页上，否则就留到下一页上考虑，直到找到适当的地方放置。由于表格或插图通常带有标题或说明，这些东西也应随同移动。

在 4.8.4 节中已引进了相应于表格的这种机制。下面将给出浮动的完整描述。

§6.6.1 浮动安置

LaTeX确确实实地按照上面所要求的那样，做到了插图和表格连同标题和说明在一起的浮动。这种机制是用下面命令调用的：

```
\begin{figure}[位置]    插图   \end{figure}
\begin{figure*}[位置]   插图   \end{figure*}
\begin{table}[位置]     表格   \end{table}
\begin{table*}[位置]    表格   \end{table*}
```

*- 形式只适用于两列页面格式，它使得插图或表格占据两列，而不是正常情形的一列。当页面格式是单列时，它的作用同标准形式一样。

在上面的语法中，插图 和 表格 就是要浮动的内容，它是包含在 picture 或 tabular 环境中的，还有可能出现的 \caption 命令（我们稍后在 6.6.4 节中介绍这一命令）。

而参数值 位置 定义了插图或表格允许出现的地方。位置 由零到四个字母组成，因此取值有很多可能，字母的意义如下：

h here: 浮动对象可以位于环境输入时所处的地方；它不能用于 *- 形式；

t top: 浮动对象可以出现在当前页的顶部，条件是要有足够的空间以容纳它自己和其前面的文本；如果这行不通，就会把它加到下一页的顶部；页后续文本仍然显示在当前页上，直到正常地分页；（对于两列格式，上面叙述中的页换成列就可以了）；

b bottom: 浮动对象可以出现在当前页的底部; 后续文本继续到在当前页上有放下浮动对象的足够地方为止; 如果在当前页上已经没有足够地方, 浮动对象会被放到下一页的底部; 在 *- 形式中不能取这个值;

p page: 浮动对象可以放在一个特殊页 (或列) 上, 该页 (或列) 上只有插图和 / 或表格;

2ε ! 与其它字母的组合一起使用, 它会去掉在 6.6.3 节中描述的关于间距和数值限制。

参数值可以组合, 形成几种可能。如果没有给定任何值, LaTeX假定它是标准组合 tbp 。

安置参数值使得定位浮动对象有几种可能, 但是实际的插入点是符合下述规则的最早可能点:

- 在它定义的前面一页上再没有浮动对象;
- 插图和表格是按照在文本中定义的先后顺序输出的, 因此不会有浮动对象会在前面定义的同类型对象之前输出; 然而插图和表格输出顺序有可能混杂在一起; 在两列格式中的双列 *- 浮动对象也有可能不符合这一顺序;
- 浮动对象可会出现在安置参数值 *位置* 所允许的位置上; 若没有该参数值, 就会用标准组合 tbp ;
- 除非在 *位置* 中包含了 ! , 定位要遵从在 6.6.3 节中描述的样式参数的限制;
- 对于组合 ht , 优先考虑 h ; 即使在当前页顶部有足够的空间, 浮动对象也会被插入在定义点。

当使用了 \clearpage, \cleardoublepage 或者 \end{document} 命令时, 所有还没输出的浮动对象就会显示在单独一页或列上, 而不会考虑它的定位参数。

§6.6.2 延迟浮动

有的时候可能不希望浮动对象出现在某些页面上, 例如不要出现在标题页的顶部。(LaTeX会自动地校正这种情形。) 然而, 也有时候需要暂时抑制浮动。我们可能希望它位于页面顶部, 但是我们不希望它位于一节引用它的那节的前面。命令

2ε \suppressfloats[*位置*]

确保在当前页的 *位置* 定义的地方没有其它浮动对象。如果没有可省参数 *位置* , 就会抑制所有的浮动; 否则 *位置* 可以是 t 或 b , 但不会全有。

注意 \suppressfloats 并没有抑制当前页所有的浮动, 它只是抑制位于从调用该命令开始到该页结束之间的浮动。因此来自于前一节的浮动仍旧有可能出现在当前页上。

\suppressfloats 命令和 ! 位置参数都是在 LaTeX 2ε中才有的，它们的目的就在于尝试给作者更大的控制浮动安置中可能出现的反复无常行为。

§6.6.3 浮动中的样式参数

有很多影响浮动安置的样式参数，用户可以修改它们：

topnumber 可以位于一页顶部的最多浮动对象数。

bottomnumber 可以位于一页底部的最多浮动对象数。

totalnumber 不考虑位置，可以位于一页中最多的浮动对象数。

dbltopnumber 同 tatalnumber 一样，只是表限定的是双列格式中横跨两列的浮动对象数。

上述参数都是记数器，可以用命令 \setcounter{记数器}{数} 来给它设定新值，这里 记数器 指的是是记数器的名称，而 数 是将要设定的新值。

\topfraction 是一个小数，定义页面顶部多大部分可以放浮动对象。

\bottomfraction 是一个小数，定义页面底部多大部分可以放浮动对象。

\textfraction 该数规定了页面多大部分必须填充以文本。这表示一个最低限度，因此无论顶部，还是底部，总共放浮动对象的部分不能超过 1−\textfraction 。

\floatpagefraction 在开始新页之前浮动页中填充的浮动对象所占部分的最低限度。

\dbltopfraction 同 \topfraction 一样，只是它对应于两列页面格式中双列浮动对象。

\dblfloatpagefraction 同 \floatpagefraction 一样，只是它对应的是两列页面格式中双列浮动对象。

要用 \renewcommand{命令}{小数 } 来改变这些样式参数的值，这里 命令 表示参数名， 小数 是新数，每个都必须小于 1 。

\floatsep 出现在页面顶部或底部中的浮动对象之间的竖直距离。

\textfloatsep 页面顶部和底部中的浮动对象与文本之间的竖直距离。

\intextsep 当用 h 安置参数值时出现在一页中间的浮动对象与上下正文之间的竖直距离。

\dblfloatsep 与 \floatsep 一样，只是它对应于两列页面格式中双列浮动对象。

\dbltextfloatsep 同 \textfloatsep 一样，只是它对应于两列页面格式中的双列浮动对象。

这一组样式参数都是橡皮长度，可以用 \setlength 命令修改它们的值（2.4.2 节）。

2ε \topfigrule 在一页顶部浮动对象之后被执行的命令。可以用它加上标尺以把浮动对象与正文分开。但是这里加入的标尺必须是零高度。

2ε \botfigrule 类似于 \topfigrule，但它是在位于页面底部浮动对象之前被执行。

2ε \dblfigrule 类似于 \topfigrule，但它是相应于双列浮动对象。

这三条命令通常什么都不做，但必要时可以重定义它们。例如，可以用下面的输入在顶部浮动之后加上粗 0.4pt 的标尺：

```
\renewcommand{\topfigrule}{\vspace*{-3pt}
  \rule{\columnwidth}{0.4pt}\vspace{2.6pt} }
```

由于在 \vspace* 中是负的参数值，整个竖直距离还是零，这满足命令的要求。

所有单列样式参数在两列页面格式中也有作用，但它们只适用于填充一列的浮动对象。

§6.6.4 浮动对象中的说明

可以用下面的命令生成插图的说明或表格的标题：

\caption[短标题]{ 标题文本 }

要把这条命令放在 figure 或 table 环境中。 标题文本 就是与浮动对象显示在一起的文本，它可以相当长。 短标题 可以省略的，它是出现在插图或表格列表中的文本（ 3.4.4 节）。如果不给出这段文本，那它就等于 标题文本 。如果 标题文本 长度超过 300 个字符，或者比一行还长的话，那就应该给出 短标题 。

在 table 环境中，\caption 命令生成形如 'Table n: 标题文本' 的标题，而在 figure 环境中，则是形如 'Figure n: 标题文本' 的说明，这里的 n 是自动给出的顺序编号。在文档类 article 中，插图和表格的编号是从 1 开始，直到文档结束。而对于 report 和 book 类，每章单独编号，形式为 $c.n$ ，这里 c 表示当前章号， n 是顺序号，每章开始被重设为 1 。插图和表格的编号是相互独立的。

如果不想要编号的话，可以不用 \caption 命令，因此包含在浮动环境中的任意文本都会随着其它内容一起移动。而 \caption 比普通文本强的地方就在于自动编号，而且其会自动列在插图和表格列表中。然而，也可以按 3.4.4 节所讲的那样用下面的命令手工向列表中增加一些项：

\addcontentsline 和 \addtocontents

当 \caption 命令位于浮动环境中其它材料的前面，那就可以形成一个标题，即编号和文本显示在表格或插图的上方。如果该命令出现在所有其它浮动命令的后面，那么就会在对象的下方加上说明。也就是说， \caption 只是浮动中的一项，其中文本出现在顶部（ 标题 ）或底部（ 说明 ）与用户放的位置有关。

如果 标题文本 的长度不足一行，那它就会居中排列，否则它同正常段落

表 6.1: Computer Center Budget for 1995

Nr.	Item	51505	52201	53998	Total
1.1	Maintenance	130 000		15 000	145 000
1.2	Network costs	5 000		23 000	28 000
1.3	Repairs	25 000	6 000		31 000
1.4	Expendables		68 000		68 000
1.	Total	160 000	74 000	38 000	272 000

一样。可以通过把命令放在子段盒子或者小页中来相应于表格或插图调整总宽度。例如,

`\parbox{ 宽度 }{\caption{ 标题文本 }}`

下面的例子说明了如何把文本、表格和图形命令组合进浮动对象。

§6.6.5 浮动示例

前两个表格是用下述文本生成的:

```
\begin{table}   \caption{Computer Center Budget for 1995}
  \begin{tabular}{|l|l||r|r|r|r|} ... ... \end{tabular}
\end{table}
```

(这段文本是输入在上一节最后一段的前面。下面的第二个表格就输入在此处。)

```
\begin{table}
  \caption{\textbf{Estimates for 1996} {\em A continuation...}}
  \begin{tabular}{|l|l||r|r|r|r|} .. ... ... \end{tabular}
\end{table}
```

由于在 `table` 环境中没有给出安置参数值,因此这里用的是标准值 `tbp`。第一个表格被放在该页的顶部,因为它输入的很早(在上一节中),而且那里有足够的空间放下它。那么命令 `\suppressfloats`(6.6.2 节)被调用,以禁止在这一页上再出现其它的浮动对象。因此第二个表格就浮动到下一页的顶部。

窄的插图或表格可以并排放在一起,如下例所示(结果显示在下一页的底部)。

```
\begin{figure}[b]
\setlength{\unitlength}{1cm}
\begin{minipage}[t]{5.0cm}
\begin{picture}(5.0,2.5) ... ... \end{picture}\par
\caption{Left}
```

表 6.2: Estimates for 1996 ***A continuation of the previous budget is no longer practical since, with the installation of the new computing system in 1995, the operating conditions have been completely overhauled.***

Nr.	Item	51505	52201	53998	Total
1.1	Maintenance	240 000			240 000
1.2	Line costs	12 000	8 000	36 000	56 000
1.3	Training			50 000	50 000
1.4	Expansion	80 000	3 000		83 000
1.5	Expendables		42 000		42 000
1.	Total	332 000	53 000	86 000	471 000

```
\end{minipage}
\hfill
\begin{minipage}[t]{6.0cm}
\begin{picture}(6.0,3.0) ... ... \end{picture}\par
\caption{Right}
\end{minipage}
\end{figure}
```

这两幅图连同其说明分别放在宽 5cm 和 6cmminipage 环境中。两个小页之间用 `\hfill` 间距分开。定位参数值 `t` 使得小页环境的第一行是对齐的（ 4.7.3 节）。在 `figure` 环境中的所有结构当做一个整体进行浮动。

那么现在就有问题了，既然两幅插图的高度不等，这里要求它们的顶行对齐，那为什么是底边在同一水平线上呢？答案是 `picture` 环境建立起一个 LR 盒子（ 4.7.1 节），它包含所有的图形命令， LaTeX认为这些命令就是一行输入，基线就是图形的底边。在这里的两个小页环境中， `picture` 环境是其中的第一项，因此也就是文本的第一个逻辑行。从而它们的基线就用来进行小页的竖直对齐。

如果要把两个图形沿顶边对齐，那就需要在每个小页环境中 `picture` 环

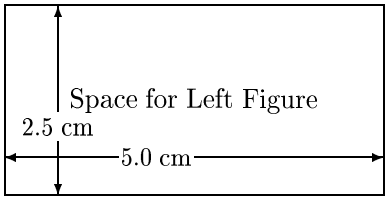


图 6.1: Left

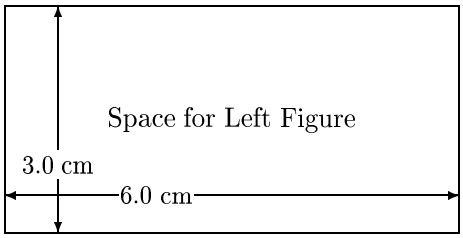


图 6.2: Right

境之前插入一个没有内容的第一行（ 4.7.4 节）。例如这可以是 \mbox{} 。

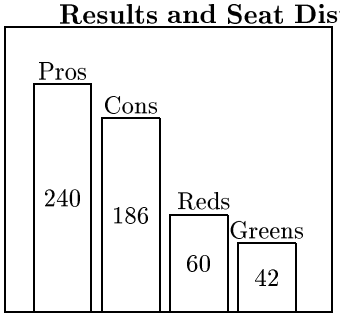
在浮动对象中应用盒子命令是可以完全自由定位的。如果说明文本位于表格或插图的一旁，而不是上面或下面，那么对象可以放在小页或子段盒子中，选择适当的对齐参数值。下面给出一个例子：

```
\begin{table}[b]
  \centerline{\bfseries Results and Seat Distribution of the ...}
  \mbox{\small
  \begin{minipage}[b]{7.7cm}
    \begin{minipage}[t]{4.4cm}
      \mbox{}\\  \unitlength0.75cm
      \begin{picture}(5.75,5.0) ... ... ... \end{picture}
    \end{minipage} \hfill
    \parbox[t]{3.2cm}{\makebox[0cm]{}\\{\bfseries Seat ...} ...}
    \\ \mbox{}
  \end{minipage}
  \hspace{-3cm}
  \begin{minipage}[b]{7cm}
    \parbox[b]{2.5cm}{{\bfseries Results:}\\ General Election...}
    \hfill
    \begin{tabular}[b]{|l||r|r|} ... ... \end{tabular}
  \end{minipage} }
\end{table}
```

在这里需要使用嵌套竖直盒子。最左边是由图形和右上角的标题组成，它位于一个宽 7.7cm 的小页中，其中的图形所在的另一个小页宽 4.4cm ，而文本又位于一个宽 3.2 cm 的子段盒子中。这两者是顶行对齐。为此需要在 picture 前面加上一个没有内容的行 \mbox{}（ 4.7.4 节），以使得顶行与子段盒子对齐。

右边是由一个宽 7cm 的小页组成，它向左平移了 3cm ，并与左面的小页

Results and Seat Distribution of the 1994 General Election



Seat Distribution in the Assembly of 1994–1998

Results: 1994 General Election

Party	Votes	%
Pros	15 031 287	37.7
Cons	12 637 418	31.7
Reds	6 499 210	16.3
Greens	4 486 514	11.3
Others	1 201 078	3.0

沿底边对齐。它由一个放文本宽 2.5cm 的子段盒子和一个表格组成，表格自动就是一个竖直盒子。这两者是底部对齐的。

§6.6.6 在正文中对插图和表格的引用

表格和插图的自动编号，就意味着在写作时作者并不知道它们的编号。而他 (或她) 又希望能够用这些对象的编号引用它们，如 'Figure 3' 或 'Table 5 illustrates'，因此需要找到一种引用的方法。而仅仅跟踪已调用的 `\caption` 中的编号又是不够的，因为文档可能不是按顺序写作的，修改时又可能插入或删掉插图或表格。

这个问题可以借助于 LaTeX的交叉索引系统来解决，在 8.3.1 节中将详细讲解。基本命令是

`\label{`*名称*`}` `\ref{`*名称*`}`

这里给正文中要用到的插图或表格编号赋予一个关键词 *名称*。关键词可以是字母、数字或符号的任意组合。赋值是用 `\label` 命令放在 `\caption` 命令的说明文本的任何地方就可以了；在正文中，命令 `\ref` 就会插入相应关键词所对应的编号。

下面用一个例子就可以很好地说明这一操作。在 159 页上的表格实际上写为

```
\caption{\label{budget95} Computer Center ... }
```

因此在正文中用 `Table \ref{budget95}` 可以生成 'Table 6.1'。

还有另一条引用命令 `\pageref`，它可以生成被引用对象所在那一页的页码。例如，刚刚几行前，就是用 '在 `\pageref{budget95}` 页上' 得到文本 '在 159 页上'。

第七章　用户定制 LaTeX

在 LaTeX中用户可以定义自己的命令和环境。然而这不可避免地要频繁用到 LaTeX的记数器和长度，因此我们首先详细讨论一下这些对象，并说明如何使用它们。

§7.1　记数器

§7.1.1　LaTeX记数器

LaTeX管理着大量的记数器，在启动时给出它们的初始值，通过调用特定命令可以改变它们的值。这些记数器的绝大多数都与可以改变它们的命令有相同的名称：

part	chapter	paragraph	figure	enumi
	section	subparagraph	table	enumii
	subsection	page	footnote	enumiii
	subsubsection	equation	mpfootnote	enumiv

从名称上就可以知道大部分记数器的意义，不需要再解释了。记数器 enumi ... enumiv 相应的是四个层次的 enumerate 环境（ 4.3.4 节和 4.3.5 节），而记数器 mpfootnote 控制 minipage 环境中的脚注编号（ 4.10.4 节）。

除这些记数器外，还可能存在用 \newtheorem 命令创建的记数器，它也具有与 *结构类型* 参数值相同的名称（ 4.5 节）。由于在 70 页上的 \newtheorem 命令例子，本书中也包含 theorem 和 axiom 两个记数器。

记数器的值必须是整数，通常也是非负的。一条命令可以同时输出几个值：当前的 \subsection 命令就输出 7.1.1，在这种情况中是调用了多个记数器。例如，\subsection 命令会给 subsection 记数器增 1，并显示 chapter, section 和 subsection 记数器的值，中间用句号分开。同时，命令还会把 subsubsection 记数器设为零，即使它没有出现。

§7.1.2　用户自定义的记数器

用户可以用下面的命令创建自己的记数器：

\newcounter{ *记数器名* }[*上级记数器*]

这里 *记数器名* 就是刚建立的记数器的名称。它可以是任一字母的组合，只要不与已存在的记数器名称相同就可以了。因此不能用列在上面的 LaTeX记数器或者前面已经定义的记数器名称作为 *记数器名* 。可省参数 *上级记数器* 是另一个已经存在的记数器（ LaTeX的或用户自定义的）的名称，其作用就在于只要 *上级记数器* 被 \stepcounter 或 \refstepcounter 命令（见下面）增 1，就把新建立的记数器重设为零。

当用 \newcounter 创建了一个新的记数器，它的初始值就是零。

\newcounter 记数器不能位于用 \include 命令（ 8.1.2 节）读进的文件中。因此最好把所有的 \newcounter 命令都放在导言中。

§7.1.3 改变记数器的值

无论是LaTeX记数器，还是用户自定义的记数器，都可以用下面的命令改变其值：

\setcounter{记数器}{ 数 }

这条命令的意义从字面上就可以知道了：指定的 记数器 被赋予给定的 数 值 (整数) 。

\addtocounter{记数器}{ 数 }

利用这条命令，指定 记数器 的值增加了给定 数 值， 数 可以是正值，也可以是负值。

\stepcounter{记数器}

指定 记数器 的值增 1 ，同时所有从属记数器（即所有把这个 记数器 作为自己 上级记数器 的记数器）的值被重设为零（见上）。

\refstepcounter{记数器}

这条命令的效果与 \stepcounter 相同，但它也同时把 counter 设为交叉索引命令 \label 中的当前记数器（见 8.3.1 节）。

例如，最后一条命令可以用在没有 \caption 命令 figure 或 table 环境中，这样也可以在正文中引用这些插图或表格的编号。那么放在浮动环境中的 \refstepcounter{figure} 或 \refstepcounter{table} 命令也可以使得相应的记数器变为正确的值，从而可以用 \label 命令给它赋予一个关键词（ 8.3.1 节）。

记数器的值可以用下面的命令当做一个数值处理：

\value{ 记数器 }

这条命令并不改变 记数器 的值。它通常与 \setcounter 或 \addtocounter 结合使用。例如，若用户已经创建了记数器 mypage ，那么就可以用命令

```
\setcounter{mypage}{\value{page}}
```

使它取与页码记数器 page 有相同的值。

通常 \protect 命令可以用来保护脆弱命令在传送过程被破坏，它同样也可以放在牢固命令前面，而不会有任何危害。然而 \value 是一个例外。从来不要在它前面加上 \protect 命令。

§7.1.4 显示记数器的值

在记数器中的值可以用下面的命令显示出来：

\arabic{counter}	以阿拉伯数字显示,
\Roman{counter}	以大写罗马数字显示,
\roman{counter}	以小写罗马数字显示,
\alph{counter}	以小写字母显示,
\Alph{counter}	以大写字母显示,
\fnsymbol{counter}	以脚注符号显示。

在命令 \alph 和 \Alph 中，数字 1...26 对应着字母 a...z 和 A...Z 。这就需要用户保证记数器的值位于这个范围里。对于 \fnsymbol ，数字 1...9 输出的符号分别是 * † ‡ § ¶ ‖ ** †† ‡‡ 。这里也同样要求用户保证记数器的值不要达到或超过 10 。

有一些记数器，还存在下面这样形式的命令：

\the 记数器

这里 \the 紧接着 记数器 的名称，如 \thepage 。这种命令通常与 \arabic{记数器} 是一样的，但也可以是几条记数器命令组成的。例如，在文档类 book 和 report 中，命令 \thesection 就是用章和节号组成的：

\arabic{chapter}.\arabic{section}

这里 \thesection 的结果就是 7.1 。

页码、公式或章节编号等等的自动显示，都是通过调用适当的 \the记数器 命令完成的。如果需要另一种不同格式的自动编号，比如说字母型的公式编号，相应 \the记数器 命令的定义就可以用 7.3 节中的方法进行修改。

练习 7.1: 使用标准练习文件 exercise.tex ，用 \arabic{counter} 在结尾出打印出 LaTeX记数器的值。利用 \setcounter 和 \addtocounter 命令改变一些记数器的值，然而再打印出结果。

§7.2 长度

我们在前面已经多次指出，类似于 \parskip 或 \textwidth 这样的长度参数的值可以用命令 \setlength 来设成新值。有些参数需要取可以伸展和收缩的橡皮长度。这些参数主要用来生成竖直或水平距离。在 2.4 节中已详细给出了长度（无论固定长度还是橡皮长度）单位的类型。这里不会再重复，在这一节中讨论其它的赋值和控制长度的命令。

给长度参数赋值的标准 LaTeX方法是用下面的命令

\setlength{\ 长度命令 }{ 长度指定 }

这里 长度指定 可以是指定长度（要有单位）或者另一个长度参数。在后一种情形中， \长度命令 就取这个参数的当前值。因此在一个 list 环境中用 \setlength{\rightmargin}{\leftmargin} 就可以使得右页边与左页边相同。

可以用下面的命令增加长度值：

\addtolength{\ 长度命令 }{ 长度指定 }

这条命令就把 长度指定 加到 \长度命令 参数上去。若 长度指定 为负值，就减去相应的量。同样，可以用另一个长度参数作为 长度指定，参数前面可以有负号，这样就可以加上或减去这个参数。在长度参数前面的数值会与参数中的值相乘： 0.5\textwidth 意味着文本列宽的一半，而 2\parskip 是段间距的两倍。

利用命令

\settowidth{\长度命令}{ 文本 }

可以使 \长度命令 参数的值等于处于 LR 模式（通常是从左到右）的一块 文本 的自然宽度。

类似地，命令

2ε \settoheight{\ 长度命令 }{ 文本 }

2ε \settodepth{\ 长度命令 }{ 文本 }

把 \长度命令 的值分别取为文本在基线上方或下方的高度与深度。

最后，命令

\stretch{ 小数 }

生成一个橡皮长度，其可展性是 \fill 的给定 小数 倍（ 2.4.2 节）。

用户要自己定义长度，可以用如下命令：

\newlength{\ 新长度命令 }

这样就可以建立起长度 \新长度命令，初始值为 0pt 。上面所讲的命令都可以用来处理它的值。

命令

\addvspace{长度指定}

会在其所处的地方插入给定 长度指定 的额外竖直距离。如果同时多次使用这条命令，那么实际被插入的间距是其中最大的那个，而不是所有间距的总和。这条命令只能用在两段之间。把它应用于用户自己定义的命令和环境中，可以使得生成的结构更像段落。

§7.3 用户定义命令

在 LaTeX中可以用下面的命令定义或重定义新的命令：

2ε \newcommand {\ 命令名称 }[参数个数][可省参数]{ 定义 }

2ε \renewcommand{\ 命令名称 }[参数个数][可省参数]{ 定义 }

或者

2.09 \newcommand {\ 命令名称 }[参数个数]{ 定义 }

2.09 \renewcommand{\ 命令名称 }[参数个数]{ 定义 }

这两组中的第一条命令是用来定义不存在的新命令。 命令名称 可以是字母的任意组合，只要不与别的命令重名即可。第二条命令是用来重定义一条已存在的命令。对这两种情形，如果调用了不正确的变量，都会给出一条错误信息。第一个可省参数 参数个数 是一个介于 1 到 9 之间的数，它规定了新定义的命令或者被改变了定义的命令中有多少个参数值。在 LaTeX 2ε中可以存在的第二个可省参数值 可省参数 给出了新命令可以为可省参数值取的默认值。命令的实际定义是包含在 定义 中。

§7.3.1 没有参数值的命令

我们首先演示没有可省参数值 参数个数 的 \newcommand 命令的用法。当一种固定的 LaTeX命令或用户命令组合被多次重复时，就可以用这种形式的命令给它赋一名称。例如，结构 $x_1,\ldots,x_n$ 称为 x- 向量，经常出现在数学公式中，它是用数学模式中的 x_1,\ldots,x_n 生成的。为此输入

```
\newcommand{\xvec}{x_1,\ldots,x_n}
```

就可以创建一个新的命令，名称为 \xvec 。此后就可以同其它命令一样调用这条新定义的命令。当调用它时，它就在自己所处的地方插入文本或命令序列（即这里的 x_1,\ldots,x_n ）。事实上，这里的过程是：当 \xvec 被调用时，它的定义就进入 LaTeX处理系统中。

由于新的 \xvec 命令定义中包含数学命令（下标命令 _ ），因此只能在数学模式中调用。从而在文本模式中需要用 $\xvec$ 来得到 $x_1,\ldots,x_n$ 。从这点来看，在定义中包含数学模式切换也不失为一个好主意：

```
\newcommand{\xvec}{$x_1,\ldots,x_n$}
```

这样 \xvec 就生成 $x_1,\ldots,x_n$ 。然而，这样一来，它就只能用在文本模式中，而不能再用在数学模式中了。下面是一种可以保证命令在两种模式中都可以使用的技巧：把命令定义为

2ε \newcommand{\xvec}{\ensuremath{x_1,\ldots,x_n}}

这样 \xvec 和 $\xvec$ 都是可以接受的，而且结果一样。

(在 LaTeX2.09 中，没有 \ensuremath 命令。此时在数学模式中我们可以用 \mbox{$...$} ，因为在文本模式中， \mbox 是被忽略的，但在数学模式中，它就可以暂时切换到文本模式中，而其中的 $ 符号又可以激活数学模式。这两种方法得到的结果并不一样，相比之下，在 LaTeX 2ε中的 \ensuremath 结果就要好得多。)

在文本中应用 \xvec 时应该写成 \xvec{} ，这是因为 TeX认为它是一个没有参数值的命令，当它遇到第一个非字母字符时就终止其名称。如果这里遇到的第一个非字母字符是空格，那它就结束命令名称，但并不插入单词间隔 (2.1 节) 。因此 \xvec and ... 的结果是 '$x_1,\ldots,x_n$and ...' ，其没有单词

间隔。只要在命令名称后面插入一个空格命令 \␣ 或者空结构 {} 就可以解决这个问题，因此这里可以用 \xvec\ 和 \xvec{}。

当然也可以在 \xvec 定义中就包含空格，即 {\ensuremath{x_1,\ldots,x_n} }。现在跟在命令名称后面的空格还是要被去掉的，但命令本身会插入一个空格。然而，在实际中我们不推荐这种做法，因为这个程序设计中的空格就总是存在的，即使接在命令后面的是标点符号或其它符号也不例外。

上面各个不同的例子中用的都是命令 \newcommand，但实际上这条命令只能用一次，以初始化用户定义的还不存在的命令。一旦 \xvec 已经被定义好，那么修正版本只能用 \renewcommand 命令来给出。实际上这里的第二个和第三个 \xvec 命令就是这样做的。

照这个例子的样子，用户可以通过 \newcommand(或者 \renewcommand) 命令来给命令和文本组合起一个新名称，然后在需要的时候调用它。利用这种方法，输入量会显著减少，而且也会减少出错的机会，特别是处理复杂数学结构的时候。

如果不确定给命令选择的名称是否已经存在，那么可以用

2ε \providecommand{\命令名称}[参数个数][可省参数]{定义}

这条命令的语法与 \newcommand 和 \renewcommand 命令完全一样。差别就在于如果命令已经存在，新定义就被忽略。想取得相反的效果 (在不用确知命令是否存在的条件下，覆盖命令的当前定义) 可以用如下方法得到：首先调用 \providecommand 确保命令存在，然后用 \renewcommand 命令给出真正的定义。然而，做这一操作时需要特别细心！

练习 7.2: 定义命令 \iint, \iiint 和 \idotsint，生成如右所示的显示公式中的多重积分符号，或者是下面这样的正文公式： $\iint$, $\iiint$, $\int\cdots\int$。

$$\iint \quad \iiint \quad \int\cdots\int$$

练习 7.3: 修改 \thechapter, \thesection 和 \thesubsection 命令的定义，使得在 book 和 report 文档类中章节号以大写字母显示，例如 B，而节号是用大写罗马数字接在章号后面，形式为 B-III，而小节编号用的是小写罗马数字，与前面之间用逗号分开： B-III,v。

提示：在 book 和 report 类中，这些命令的原始定义为

```
\newcommand{\thechapter}{\arabic{chapter}}
\newcommand{\thesection}{\thechapter.\arabic{section}}
\newcommand{\thesubsection}{\thesection.\arabic{subsection}}
```

现在用 \renewcommand 命令进行必要的修改。

§7.3.2 有参数值的命令

除了结构 $x_1,\ldots,x_n$ 外，在数学中还有 $y_1,\ldots,y_n$ 和 $z_1,\ldots,z_n$ 等等同样

的向量结构。因此我们可以按 \xvec 的样子定义命令 \yvec 和 \zvec 。然而，也可以用定义一个广义的向量命令，把变化部分作为参数值。对于当前这个例子，变化部分是字母 x, y 和 z 。只有一个变量的命令是用可省参数值 [1] 生成的。因此，

```
\newcommand{\avec}[1]{\ensuremath{#1_1,\ldots,#1_n}}
```

就定义了一般性的向量命令 \avec{参数值} 。这样当调用 \avec{x} 时结果就是 $x_1,\ldots,x_n$ ，而 \avec{y} 显示出 $y_1,\ldots,y_n$ 。在命令定义中的字符 #1 只是一个哑元，表示当命令被调用时，要用 参数值 的文本取代所有出现的 #1 。只要把定义中所出现的 #1 都想像成 x 或 y ，就可以相当容易地了解所定义的结构了。

在哑元 #1 中的数字 1 在这里好像没有什么用处。事实上，对于只有一个参数值的命令，它确实没有什么意义。然而，对于多参数值的命令，它的作用就很明显了。比如说，我们要定义一条命令，它既能生成结构 $x_1,\ldots,x_n$ ，也能生成 $v_1,\ldots,v_m$ 。这就需要有两个参数值，一个用来指定 u, v 等等，另一个用来确定最后的那一个下标 n, m ，等等。这样的命令是如下生成的：

```
\newcommand{\anvec}[2]{\ensuremath{#1_1,\ldots,#1_#2}}
```

这样 \anvec{u}{n} 就得到 $u_1,\ldots,u_n$ ，而 \anvec{v}{m} 得到 $v_1,\ldots,v_m$ 。\newcommand 命令中的可省参数值 [2] 说明要定义的命令中包含两个参数值；在定义部分， #1 表示第一个参数值， #2 表示第二个参数值。把 #1 想像成 u 或 v ，把 #2 想像成 n 或 m ，那么就可以很容易知道 \anvec{arg1}{arg2} 的操作方式了。

这种方式可以适用于有更多的参数值。做了如下定义后，

```
\newcommand{\subvec}[3]{\ensuremath{#1_#2,\ldots,#1_#3}}
```

命令 \subvec 就是有三个参数值的命令。从其定义易知 \subvec{a}{i}{j} 生成 $a_i,\ldots,a_j$ 。

只由单个字符构成的参数值不必放在大括号 { } 内，可以直接给出。如果它是第一个参数值，那必须像通常那样，用空格把它与命令名分开。因此 \subvec aik 的结果与 \subvec{a}{i}{k} 的一样，而 subvec x1n 生成与前面例子 \xvec 同样的结构 $x_1,\ldots,x_n$ 。

当参数值不只由一个字符组成时，它就必须放在大括号 { } 内，因为这里的大括号表示要把参数值内容当做一个整体处理。因此 \subvec{A}{ij}{lk} 的结果为 $A_ij,\ldots,A_lk$ 。这里的参数取代是 A 相应于 #1 ， ij 相应于 #2 ， lk 相应于 #3 。

可是为什么 \subvec{A}{ij}{lk} 的结果是 $A_ij,\ldots,A_lk$ ，而不是所期望的 $A_{ij},\ldots,A_{lk}$ 呢？这是因为虽然在大括号内的参数值被当做一个整体替换进定义文本中，但大括号自身并没有进去。这里替换后的命令文本实际上类似于 \ensuremath{A_ij,\ldots,A_lk} ，因此实际上只有接在下标符号 _ 后面

的第一个字母被降低。为了使两个字母都被降低，那在命令文本中它们就必须做为一个整体出现，也就是类似于 $A_{ij},\ldots,A_{lk}$ 。可以在 \subvec 命令的参数值中额外再加一对大括号就可以做到这一点，即 \subvec{A}{{ij}}{{lk}} 。而更好的解决方法是一开始就在定义中加上大括号：

```
\newcommand{\subvec}[3]{\ensuremath{#1_{#2},\ldots,#1_{#3}}}
```

这样即使每个参数值只有一层大括号，也能得到所期望的结果：

\subvec{A}{ij}{lk} 的结果为 $A_{ij},\ldots,A_{lk}$ 。

§7.3.3 有一个可省参数值的命令 2ε

我们已经知道有很多 LaTeX命令可以有可省参数值，这其中最典型的例子就是 \newcommand 命令自身。在 LaTeX 2ε中，同样也可以使用户定义的命令有一个可省参数值。这样做的好处就在于，虽然多提供了一个参数，但在绝大多数情况下它取得只是标准值，不需要显式改变其值。

例如，在上一节用户定义的命令 \subvec 中有三个参数值，分别相应于字母和第一个及最后一个下标。然而，通常字母就是 x ，因此把它做为一个可省参数不失为一个好主意，这样只有在字母不同时才需要指定。要做到这一点，利用如下命令：

```
\renewcommand{\subvec}[3][x]{\ensuremath{#1_{#2},\ldots,#1_{#3}}}
```

它与前面定义的区别就在于 [3] 参数值后面多了 [x] 。这就说明了三个参数值中的第一个是可省的，其标准值为 x 。现在 \subvec{i}{j} 的结果就是 $x_i,\ldots,x_j$ ，而 \subvec[a]{1}{n} 的结果为 $a_1,\ldots,a_j$ 。

在用户定义的命令中只能有一个可省参数，而且也必须是第一个，即定义中的 #1 。

§7.3.4 用户定义命令的其它样例

在上面用户定义命令的解释中，我们用向量结构这样很简单的情形做为示例。下面我们要演示的是更复杂的情况，其中要应用到记数器、长度，甚至一些特殊的 TeX命令。

例1: 在 5.4.6 节中，说明了 TeX命令 \atop 和 \choose 是相当有用的数学命令，即使在 LaTeX中也不例外。不幸的是这两条命令的语法同类似的 LaTeX命令 \frac 大相径庭。然而，

```
\newcommand{\latop}[2]{#1\atop\ #2}
\newcommand{\lchoose}[2]{#1\choose\ #2}
```

定义的两条命令 \latop 和 \lchoose 就生成同样的结果，而且语法也符合通常的 LaTeX结构化要求： \latop{上式}{下式} 。

例2: 在下面我们要定义两条命令， \defbox{取样文本} 会设置一个盒子，其宽度等于 取样文本 的长度。然后调用 \textbox{文本} 命令，就会在一个宽

度与 取样文本 相同的有框盒子中居中显示 文本 。

```
\newlength{\wdth}
\newcommand{\defbox}[1]{\settowidth{\wdth}{#1}}
\newcommand{\textbox}[1]{\framebox[\wdth]{#1}}
```

首先创建了一个新的长度参数 \wdth，然后定义 \defbox，使得 \wdth 就等于其参数值的长度（ 7.2 节），最后用 \textbox 生成一个相同宽度的有框盒子，其中文本居中。（不要用长度参数 \width，因为它已经存在，见 4.7.5 节。）

```
as wide as this text\\
\defbox{as wide as this text}\textbox{}\\
\textbox{text}\\
\textbox{longer text}
```

as wide as this text
text
longer text

例3: 我们要定义脚注命令 \myftnote，它与通常的 \footnote{文本} 命令一样把 文本 放到脚注处，但这里不是用数字做脚注标记，而是依次用符号 * ‡ ‡ § ¶ ‖ ** †† ‡‡, 在每一新页上都是从 * 开始。首先需要创建一个新的记数器，每当 page 记数器增 1 时，其都要被重置为零。做法为（见 7.1.2 节）：

```
\newcounter{myfn}[page]
```

这个用户自定义的记数器 myfn 每当 page 增 1 时都会自动重置为零。下面这条命令

```
\renewcommand{\thefootnote}{\fnsymbol{footnote}}
```

重定义脚注标记为记数器 footnote 所对应的上述符号（ 4.10.2 和 7.1.4 节）。现在可以如下构造新的脚注命令：

```
\newcommand{\myftnote}[1]{\setcounter{footnote}{\value{myfn}}%
  \footnote{#1}\stepcounter{myfn}}
```

其就会生成所期望的结果。 * 用户定义的命令 \myftnote 拥有一个参数值，当把 LaTeX记数器 footnote 的值与用户定义记数器 myfn 相同值时，把它传送给 LaTeX的 \footnote 命令。一旦执行了命令 \footnote，就用命令 \stepcounter{myfn} 给记数器 myfn 增 1 。但是一旦 page 记数器增 1，即生成一个新页，这个记数器就要被重置为零。

上面的脚注就是用命令 \myftnote 生成的。同样现在也是用的这条命令，† 再用一次。 ‡ 这演示了所定义符号的使用。

例4: 我们下面定义命令 \alpheqn，一旦它被调用，后续公式将具有相同的编号，只是后缀字母 a, b, . . . ，中间用连字符 '-' 分开。命令 \reseteqn 把

*放在第一行结尾处的 % 符号是为了去掉行尾隐藏的空格被解释成命令的一部分（ 4.11 节）。通常为了增加可读性，需要把命令定义分成几行。

†另一个脚注

‡还是一个脚注

编号框架重设为原来的样式。因此公式编号序列可能有如此形式： 4, 5, 6-a, 6-b, 7 。

```
\newcounter{saveeqn}%
\newcommand{\alpheqn}{\setcounter{saveeqn}{\value{equation}}%
\stepcounter{saveeqn}\setcounter{equation}{0}%
\renewcommand{\theequation}
      {\mbox{\arabic{saveeqn}-\alph{equation}}}}%
\newcommand{\reseteqn}{\setcounter{equation}{\value{saveeqn}}%
\renewcommand{\theequation}{\arabic{equation}}}}%
```

由于我们在 7.1 节中已讲解了上面所用的命令和记数器，因此这个例子是很好理解的。记数器 equation 的当前值被保存到记数器 saveeqn 中，然后增加 saveeqn , 而 equation 被重置为零。公式编号的形式 \theequation 利用这两个记数器进行了重定义。这样公式编号的方式就同原来一样，而 saveeqn 没有改变。重设命令 \reseteqn 就把 saveeqn 的值重新赋给 equation ，并恢复 \theequation 的定义。

这条命令只适合用在 article 文档类中。如果所用文档类是 book 或者 report ，那么 \theequation 的定义是

```
\arabic{chapter}.\arabic{equation|
```

这里需进行的必要修改，就留给读者做为练习。

在定义组合公式编号的第一个 \renewcommand{\theequation} 命令中的 \mbox 命令是必须的，因为结果要用数学模式显示，这样连字符 '-' 要被解释成二元运算符 (负号)，其前后与两个被操作数 \arabic{saveeqn} 和 \alph{equation} 之间就会有额外的间距。因此结果可能是 $6-a$ ，而不是 6-a 。 \mbox 命令就是为了暂时从数学模式切换进 LR 模式。

例5: 最后我们利用某些很基本 TeX命令，给出一个例子，这里无法详细解释这些命令。但是这里定义的命令对于那些输入文本中经常包含化学公式的情形是非常有用的。

在 5.4.9 节中指出在化学公式 $Fe_2^{2+}Cr_2O_4$ 中的下标不在同一水平线上，而且在文本模式中输入相对短的公式也很烦。下面这条命令就可以解决这些问题。

```
\newlength{\fntxvi} \newlength{\fntxvii}
\newcommand{\chemical}[1]
{{\fontencoding{OMS}\fontfamily{cmss}\selectfont
  \fntxvi\the\fontdimen16\font
  \fntxvii\the\fontdimen17\font
  \fontdimen16\font=3pt\fontdimen17\font=3pt
  $\mathrm{#1}$
```

```
\fontencoding{OMS}\fontfamily{cmys}\selectfont
\fontdimen16\font=\fntxvi \fontdimen17\font=\fntxvii}}
```

现在 `\chemical{Fe_2^{2+}Cr_2O_4}` 的结果就是正确的形式 $\mathrm{Fe}_2^{2+}\mathrm{Cr}_2\mathrm{O}_4$ 。

解释： TEX命令 `\fontdimen`n 描述了字符字体的特定性质，如 $n=16$ 和 $n=17$ 就确定指标（下标）的位置。在用 `\selectfont`（ 8.5.1 节讲到）结尾的那行上的命令使得当前数学符号字体连同其内部设计一起存贮在 TEX命令 `\font` 中。当前尺寸（这里是 10pt ）下数学公式中的所有指标都是一致降低 3.0pt 。关于其它尺寸，请见练习 7.7 的提示。由于对 `\fontdimen` 的改变总是全局性的（当从一个真正的或者隐式的环境中退出时，并不恢复以前的值），因此有必要首先保存当前值，然后再手工恢复。

练习 7.4: 同例 1 中定义 `\latop` 和 `\lchoose` 一样的方式，定义 LATEX命令 `\lbrack` 和 `\lbrace`，分别相应于 TEX命令 `\brack` 和 `\brace`。这两条 TEX命令的形式同 5.4.6 节的 `\choose` 一样，只是它们用中括号 `[\lbrack]` 或者大括号 `{\lbrace}` 把其中内容包围起来。

练习 7.5: 把例 4 中命令推广成 `\vareqn{`*数*`}{`*类型*`}`，使得后续公式的编号形式为 *数* 后接放在中括号内的活动编号，其中活动编号的显示由 *类型* 参数值确定。例如 `\vereqn{33}{\Alph}` 的结果可能为 33[A], 33[B] 等等。

练习 7.6: 推广练习 7.2 中的积分命令，引进一个参数值表示水平居中放在积分号下面的积分区域。因此 `\iint{(D)}`, `\iiint{V}` 和 `\idotsint{G}` 的结果应如右所示。

$$\iint_{(D)} \quad \iiint_{V} \quad \int\cdots\int_{G}$$

提示：第二条命令只要把下标放在中间那个积分号下面就可以了（见 5.2.5 节的 `\limits` ）。而对于其它两个，下标符号需要利用 `\hspcae{-..}` 进行负的水平位移。

练习 7.7: 在上面例 5 中化学公式下标下沉 3pt 是相应于 10pt 的字体尺寸的。对于 11pt 和 12pt，这个值是 3.3 和 3.6pt 就比较恰当。推广命令 `\chemical`，使得这个值成为一个可省参数值，其默认值为 3pt 。这就是说对于 12pt 的字体尺寸，调用方式应为 `\chemical[3.6pt]{...}` 。

§7.3.5 条件文本

有经验的 TEX用户对在 TEX和 LATEX中都可以用的条件文本是相当熟悉的。然而，这种用法并不是那么简单明了，需要对 TEX深层机制有相当的了解。 Leslie Lamport 提供了一个叫 `ifthen` 的软件包， David Carlisle 把它进行了推广，在 LATEX 2$_\varepsilon$中也可以应用，这个软件包不但简化了这种应用，而且符合 LATEX语法。

这个软件包可以像通常那样在导言中用下面命令调用：

2ε \usepackage{ifthen}

或者把它指定为 \documentstyle 的一个选项，即

2.09 \documentstyle[..,ifthen,..]{...}

这样就可以使用 \ifthenelse 和 \whiledo 这两条命令了，它们的语法是：

\ifthenelse{ 测试条件 }{then 文本 }{else 文本 }

\whiledo{ 测试条件 }{do 文本 }

在这两种语法中，测试条件 是一个逻辑声明(下面解释); 在第一条命令中, 如果这个声明等于 ⟨*true*⟩ , 那么就把 then文本 插入到正文中, 否则就把 else文本 插入到正文中。在第二条命令中，只要 测试条件 等于 ⟨*true*⟩ , do文本 被插入 (执行)。(do文本 必须改变对 测试条件 的输入，否则它永不会停止 !) 文本中也可以包含命令，甚至可以定义或重定义命令。

共有四种类型的逻辑声明，可以把它们组合起来，以形成复杂的声明。

测试数字

要比较两个数字，或者值为数字的命令，只要在它们中间放上关系运算符 <, = 或 > 就可以了，这三个符号分别代表小于，等于或大于。记数器中的值可以通过把它的名称作为 \value 命令的参数值来进行测试。例：

```
\newcommand{\three}{3}
\ifthenelse {\three = 3} {O.K.} {What?}
\ifthenelse {\value{page} < 100 }
  {Page xx} {Page xxx}
```

在第一种情形中是显示出 'O.K.' , 因为 \three 等于 3 ; 在第二种情形中, 如果当前页码小于 100 , 就会显示出 Page xx , 否则就显示出 Page xxx 。 (上面加的空格是为了明了，因为参数值中间的空格总是被忽略。)

要测试一个数是偶数，还是奇数，可以用 \isodd 2ε 命令：

```
\ifthenelse {\isodd{\value{page}}
   {odd} {even}
```

测试文本

若要测试两条命令得到的是否是相同一块文本，或者一条命令是否定义成特定的字符串，可以用

\equal{ 字符串一 }{ 字符串二 }

这里的 字符串一 和 字符串二 就是文本或者可以简化为文本的命令。例如, 给出如下定义后：

```
\ifthenelse {\equal{\name}{Fred}} {Fredrick} {??}
```

若 \name 已被定义成 Fred(用命令 \newcommand) , 那么就会插入 Fredrick , 否则就显示出两个问号。

©TEXGuru, August 15, 1999

测试长度

另一个逻辑声明用来比较两个长度。

2ε `\lengthtest{关系}`

这里的 *关系* 由中间用关系运算符 <, = 或 > 分开的两个长度或长度命令组成。例如,

```
\newlength{\horiz} \newlength{\vert}
\newlength{\min}
. . . . . .
\ifthenelse {\lengthtest{\horiz > \vert}}
  {\setlength{\min}{\vert}} {\setlength{\min}{\horiz}}
```

会把 \min 取成 \horiz 和 \vert 两者中较小的那个。

测试开关

所谓 Boolean 开关就是要么为 ⟨*true*⟩，要么为 ⟨*false*⟩ 的参数，也称为*标志*。存在处理标志的三条命令：

2ε	`\newboolean{字符串}`	创建一个新开关
2ε	`\setboolean{字符串}{数}`	赋值 `true` 或 `false`
2ε	`\boolean{字符串}`	测试其值

其中最后那个就可以用于 \ifthenelse 和 \whiledo 命令中的*测试条件*。

在 LaTeX中还有很多内部开关，可以测试它们的值 (但从不要重设它们的值 !)。其中最有用的就是 `@twoside` 和 `@twocolumn`，可以用来测试当前激活的是不是双面模式或者两列模式。由于其中包含字符 `@`，因此它只能用在类或宏包文件中 (附录 C)。

组合逻辑声明

可以把上面列出的逻辑声明利用下面的逻辑运算符，组合成更复杂的声明：

```
\and   \or   \not   \(   \)
```

只要熟悉 Boolean 逻辑，这些运算符的含义是非常明了的。例如，如果激活了两列模式或者 \paperwidth 大于 15cm 且同时页码记数器的值小于 100 时，就设置 \textwidth 等于 10cm，那么下面的定义：

```
\ifthenelse {\lengthtest{\textwidth > 10cm} \or
     \( \lengthtest{\paperwidth > 15cm} \and
        \vaule{page} < 100 \) }
     {\setlength{\textwidth}{10cm}} {}
```

就可以达到这个目的。

上面这个条件比较复杂，就非常适用于定义可以有不同行为的新命令，或者包含在宏包和类文件中（附录 C）。

下面给出一个相对简单的例子：假设作者不知道出版者用的到底是英国拼写，还是美国拼写。那么他可以在文档中把两种拼写都包含进去，在导言中加进一个开关，以得到最后的选择。

```
\newboolean{US}
\setboolean{US}{true} %For American spelling
%\setboolean{US}{false} %For British spelling
\newcommand{\spell}[2]{\ifthenelse{\boolean{US}}{#1}{#2}}
```

那么现在 \spell 命令就会显示出根据标志 US 的设置来确定显示第一个还是第二个参数值。因此在正文本中作者可以如下输入：

```
... the \spell{color}{colour} of music ...
```

这样如果指定的是 \setboolean{US}{true}，就会生成美国拼法，否则得到的就是英国拼法。事实上，不同工作部分之间可以用不同的拼法，只需简单地在适当地方改变开关的值就可以了。（在写书时这就是一种好方法，因为编辑在最终决定手稿之前是有可能改变拼法的。）

\whiledo 的一个例子：作者希望把某一文本重复写 n 遍，这里的文本和 n 是可变的，那么可以用如下方法：

```
\newcounter{mycount}
\newcommand{\replicate}[2]{\setcounter{mycount}{#1}
    \whiledo{\value{mycount}>0}{#2\addtocounter{mycount}{-1}}}
```

那么 \replicate{30}{?} 就会显示出 30 个问号。

§7.4 用户定义的环境

可以用下面的命令来创建或修改环境：

2ε \newenvironment {环境名}[参数个数][可省参数]
{开始的定义}{结束的定义}
2ε \renewenvironment{环境名}[参数个数][可省参数]
{开始的定义}{结束的定义}

或者

2.09 \newenvironment {环境名}[参数个数]
{开始的定义}{结束的定义}
2.09 2ε \renewenvironment{环境名}[参数个数]
{开始的定义}{结束的定义}

这里参数值意义如下：

环境名: 对于 \newenvironment，它不可以与任一已存在的 LaTeX或用户定义的环境或命令重名。另一方面，对 \renewenvironment，则就必须有同名的环境存在。要对 LaTeX环境进行任何修改，都要求用户对 LaTeX的内部工作机理有相当深入的了解。

参数个数: 介于 1 到 9 之间的数，说明环境有多少个参数值；如果忽略了这个可省参数值，环境就没有参数值。

2ε 可省参数: 如果第一个参数值（ #1 ）是可省的，这是它的默认值；它与 \(re)newcommand（ 166 页）中的同名参数值行为一样。

开始的定义: 当 \begin{环境名} 被调用时，被插入的初始化文本；如果这一文本中包含形如 #n， $n = 1, \ldots,$ 参数个数 项，那么环境就要用如下形式调用：

\begin{环境名}{参数 1}...{参数 n}...

在 开始的定义 中出现的每个 #n 都要用 参数n 取代。

结束的定义: 当调用了 \end{环境名} 时，要插入的结束文本；这里不要用哑参数值 #n，因为它们只允许出现在 开始的定义 文本中。

§7.4.1 没有参数的环境

同用户定义命令中的情形一样，这里首先讲的环境也是没有可省参数值 参数个数 的。用户若要定义一个自己的环境 sitquote，可以如下输入：

```
\newenvironment{sitquote}{\begin{quote}\small
    \itshape}{\end{quote}}
```

which sets the text appearing between \begin{sitquote} *text* \end{sitquote} *in the typeface* \small\itshape*, and indented on both sides from the main margins, as demonstrated here.*

在这里 开始的定义 由命令序列 \begin{quote}\small \itshape 组成，而 结束的定义 只有 \end{quote}。那么现在调用

\begin{sitquote} 文本 \end{sitquote} 就等价于

\begin{quote}\small\itshape 文本 \end{quote}

这样就得到了所期望的结果。

这个例子不是很好，因为只要在 quote 环境的开始加上 \small \itshape 就很容易得到同样的效果。然而，如果在文档中经常出现这样的结构，那么这个新环境对于简化输入和减少在输入 \small\itshape 时出错的机会方面就非常有效了。

我们现在把前面这个例子做如下的推广：

```
\newcounter{com}
\newenvironment{comment}
{\noindent\slshape Comment:\begin{quote}\small\itshape}
```

```
{\stepcounter{com}\hfill(\arabic{com})\end{quote}}
```

这里 开始的定义 就是由文本和命令组成：

```
\noindent\slshape Comment:\begin{quote}\small\itshape
```

而 结束的定义 组成为：

```
\stepcounter{com}\hfill}(\arabic{com})\end{quote}
```

其中 com 是一个由 \newcounter 创建的用户记数器。由于 \begin{comment} 命令要在环境开始处插入 开始的定义 文本， \end{comment} 在结束处插入 结束的定义 文本，因此很容易知道：

```
\begin{comment} This is a commen.
    Comments should ...
   ... in round parenthesis.
\end{comment}
```

的结果应是如下结构：

Comment:

> *This is a comment. Comments should be preceded by the word Comment: the text being in a small, italic typeface, indented on both sides from the main margins. Each comment receives a running comment number at the lower right in round parentheses.* *(1)*

读者应仔细对比一下环境定义中的命令序列与例子中文本位置的关系，这样可以准确地知道这个环境的效果。这里有两个非常明显的缺陷，那就是当调用 \begin{comment} 时其正好位于一行的中间，前面没有空行的情形，或者注释的最后一行太长，使得该行没有足够的空间把活动编号放在行尾时的情形。

下面这个修正版本就解决了这两个问题：

```
\renewenvironment{comment}
{\begin{sloppypar}\noindent\slshape Comment:
    \begin{quote}\small\itshape}
{\stepcounter{com}\hspace*{\fill}(\arabic{com})\end{quote}
    \end{sloppypar}}
```

这里利用 \begin{sloppypar} 命令使得环境总是开始一个新段，而且在断行是不会有过满行情形发生。如果注释的编号在最后一行放不下来，就会开始一个新行，编号是右对齐的，因为这里用了命令 \hspace*{\fill} 。这里同样希望读者仔细查看插入在 \begin{comment} 和 \end{comment} 之间的定义文本。

§7.4.2 有参数的环境

向环境中传递参数值，同命令中的情形是完全一样的。举例来说，我们对

上面的注释环境进行修正，使得注释人的姓名加在单词 Commnet: 的后面；当环境被激活时这个姓名就是其参数值。

```
\renewenvironment{comment}[1]
{\begin{sloppypar}\noindent\slshape Comment: #1
     \begin{quote}\small\itshape}
{\stepcounter{com}\hspace*{\fill}(\arabic{com})%
     \end{quote}\end{sloppypar}}
```

那么现在输入

```
\begin{comment}{Helmut Kopka} This is a modified ...
  ... environment argument \end{comment}
```

结果就是：

Comment: Helmut Kopka

> *This is a modified comment. Comments should be preceded by the word Comment:, followed by the name of the commenter, with the text of the comment being in a small, italic typeface, indented on both sides from the main margins. Each comment receives a running comment number at the lower right in round parentheses. The name of the commenter is transferred as an environment argument.* *(2)*

下面再对这个例子进行修正，交换注释编号与注释人姓名的位置。把活动编号放在单词 Comment: 后面是没有任何问题的，因为这只需要简单地把来自于 {结束的定义} 命令插入在上面 #1 哑元参数值的地方。然而，要把 #1 符号放在原来注释号所处的地方，在 LaTeX处理过程中就会出现错误信息，因为这与 \newenvironment 命令的语法相冲突：‘在 {结束的定义 } 中不能有哑元参数值’。如果哑元参数值位于 \begin{quote} 的后面，那么其位置就不是所期望的地方，而是处于注释文本的开始。

下面给出解决这个问题的技巧：

```
\newsavebox{\comname}
\renewenvironment{comment}[1]
{\begin{sloppypar}\noindent\stepcounter{com}\slshape
   Comment \arabic{com}\sbox{\comname}{#1}
   \begin{quote}\small\itshape}
{\hspace*{\fill}\usebox{\comname}\end{quote}\end{sloppypar}}
```

我们在 4.7.1 节中已讲述了 \newsavebox, \sbox 和 \usebox 命令。这里 \comname 就是用 \newsavebox 命令提前创建的盒子名称，其中放的是第一个参数值，即注释人的姓名。利用这个新的定义，注释的新样子如下：

Comment 3

> *In this form, every comment is assigned a sequential number after the word*

Comment. The comment text appears as before, while the name of the commenter is entered as the environment argument and is placed at the right of the last line. *Helmut Kopka*

要实现不只一个参数值的环境，步骤同这里是完全一样的，因此这里不再讲述。

§7.4.3 有一个可省参数值的环境 2ε

同命令一样，环境也可以有一个可省参数值。还是考虑上面那个例子，如果我们觉得绝大多数注释都是由 Helmut Kopka 给出的，这样我们就可以把定义的第一行改为

```
\renewenvironment{comment}[1][Helmut Kopka]{..}{..}
```

因此只有当注释人是其它人时才需要指定。

```
\begin{comment}[Patrick W. Daly] More than ...
 ... appropriate. \end{comment}
```

结果为

Comment 4

More than one person may want to make a comment, but perhaps one person makes more than others do. An optional argument for the name is then appropriate *Patrick W. Daly*

练习 7.8: 推广注释环境的定义，使得不能在标题 'Comment n' 与注释正文之间以及正文与注释人姓名之间有分页。

提示：解法应该利用 samepage 环境或者 \nopagebreak 命令。

练习 7.9: 利用 minipage 环境创建一个新的环境，名称为 varbox ，它具有一个参数值，该参数值是一个文本样本，小页环境的宽度由它确定。因此

```
\begin{varbox}{'As wide as this sample text'}
. . . . . .\end{varbox}
```

就会用一个宽度等于文本 'As wide as this sample text' 的自然宽度的小页环境包围其中文本。

提示：首先要建立一个用户定义的长度参数，名称可以是 \verwidth 。在 7.2 节中有关于文本宽度怎样赋给长度参数的讲解。

练习 7.10: 生成一个名称为 varlist 的环境，它有两个参数值，行为类似于 4.4.3 节的示例列表的推广。第一个参数值就是每次调用 \item 时显示出来的项词；第二个参数值项枚举的编号样式。例如，利用

```
\begin{varlist}{Sample}{\Alph}. . . . . \end{varlist}
```

在环境中依次调用 \item 时，生成序列 'Sample A', 'Sample B', . . . 。缩进应比项词的宽度大 1cm ，而项词本身是在标签盒子中左对齐。

提示：解法中还是要借助于用户定义长度，如 \itemwidth 。当项词的长度利用 \settowidth 保存起来后，缩进可以如下设置：

```
\setlength{\leftmargin}{\itemwidth}
```

这样缩进就等于项词长度，然后

```
\addtolength{\leftmargin}{1cm}
```

就可以使它增大 1cm 。对于 \labelwidth 和 \labelsep 的长度赋值也可以类似进行。关于其它的细节可以参考 4.4 节。

§7.5 对用户定义结构的解释

本节包含了用户定义 LaTeX结构的创建和使用方面的一些一般性注解。这没有严格的规则，只是阐述了我们根据据自己的经验而得到的一些想法。每个用户都会根据自己的经验，总结出自己的实用方法。

§7.5.1 保存用户定义的结构

用户创建自己定义的结构，可以简化大量的工作。经常使用的命令和 / 或文本应利用 \newsavebox, \newcommand 或 \newenvironment 定义单独写到一个文件中。这个文件可以用 \input 命令读入，从而在其它文档文件中也可以使用。

随着时间的推移，这样就会积累出大量的结构。若把它们全部放到一个文件中就会减慢处理过程，而且跟踪内容和命令名称也变得非常困难。因此我们推荐用户结构应根据应用领域保存在许多独立的文件中。

§7.5.2 缩写结构

用户定义命令的一个简单应用就是缩短 LaTeX结构的名称。例如，利用

```
\newcommand{\be}{\begin{enumerate}}
\newcommand{\ee}{\end{enumerate}}
```

只要输入 \be 就可以开始一个 LaTeX环境 enumerate ，而 \ee 用来结束这个环境。

这样的缩写可以显著地避免大量输入,但它并不适合于做为命令大量保存起来,因为容易忘记命令名称背后所包含的意义。LaTeX的作者 Leslie Lamport 已经精心选择了命令或环境的名称，以明白地表示它们的功能。这种明确完整的命名，要比不明了的任何缩写容易记住。但是可以在单个文档内容中使用某些缩写，只要不太多就可以。具体要视用户的输入能力来确定了。

§7.5.3 命令和记数器的名称相同

在前面的例子中，一些特定的命令或环境的应用中使用了许多记数器，例如 171 页上 \myfootnote 命令中的 myfn ， 177 页上 comment 环境中的

com 。在这些情形中，记数器与对应的命令或环境有不同的名称。但这并是一定要这样：记数器可以具有与命令或环境相同的名称。 LaTeX可以从上下文知道某一名称指的是记数器还是命令 / 环境。

在上面例子中之所以选择不同的名称，就是为了避免初学者混淆。事实上完全有理由把记数器的名称同其所对应的命令或环境的名称取成一样的，LaTeX本身就是经常这样做的（ 7.1.1 节）。在上面所讲到的例子中，记数器就可以命名为 myfootnote 和 comment ，以强调它与同名命令和环境之间的依赖关系。

§7.5.4 用户定义的作用范围

在导言中给出的用户结构对整篇文档都有效。在环境内给出的命令和环境定义有效性持续到包围环境的结束。甚至在定义所处的环境外面 LaTeX连它们的名称也不知道。因此如果在另一个环境又要进行同名的定义，应该用 \newcommand 或 \newenvironment ，而不是 \renew 形式的命令。

对于全局定义的命令和环境（即放在导言中），对它们进行新的定义需要用 \renew 形式的命令。然而这个定义只在其所处的环境内有效；在这个环境外面，原先的全局定义仍然有效。

这一规则同样适用于嵌套环境中的结构定义。在外部环境中定义在内部环境中有效，在深层必须有 \renewcommand 或 \renewenvironment 进行同名新的定义。当离开内层环境后，新定义不再有效，而重新恢复原先的含义。

警告：用 \newsavebox 或 \newcounter 创建的结构是全局定义的。如果在环境中给出，即使在环境外面它也是有效的。同样虽然 \savebox 并不是全局性的，但 \setcounter 却是全局性的。

§7.5.5 定义的顺序

用户定义的结构可以彼此嵌套。如果一个用户定义结构中包含另一个用户定义结构，通常内部的结构已经被定义了。然而，这并不是必需的。用户定义中可以包含后面定义的其它用户结构。重要的是当第一次调用命令之前其它结构已经有了定义。

例如，下面是一个正常的定义顺序：

```
\newcommand{\A}{ 定义 A}
\newcommand{\B}{ 定义 B}
\newcommand{\C}{\A \B}
```

这里 \C 是用 \A 和 \B 定义的。然而，也可以像下面这样输入：

```
\newcommand{\A \B}
没有调用 \C 的普通文本
\newcommand{\B}{ 定义 A}
```

```
\newcommand{\A}{ 定义 B}
```
其它文本，可以随意调用 \A, \B 和 \C.

§7.5.6 传递参数值

在命令和环境定义中的哑元参数值可以用作定义中其它命令的参数值。例如，如果命令 \A 和 \B 每个都有一个参数值，命令定义

```
\newcommand{\C}[3]{\A{#1}#2\B{#3}}
```

是可以接受的。这里 \C 的第一个和第三个参数值被传递给命令 \A 和 \B，只有第二个参数值直接进入定义。可以把直接参数值和传递参数值进行任意的组合。下面是一个比较具体的例子：

```
\newcommand{\sumvec}[4]{\anvec{#3}{#4} = #1_1+#2_1,\ldots,
        #1_#4+#2_#4}
```

这样调用 `$\sumvec xyzn$` 就得到 $z_1, \ldots, z_n = x_1+y_1, \ldots, x_n+y_n$, 这里 \anvec 就是在 7.3.2 节中定义的命令。

§7.5.7 嵌套定义

用户定义可以彼此嵌套。类似于下面这样的结构

```
\newcommand{\ 外部命令名 }{{\newcommand{\ 内部命令名 }...}}
```

是可以的。根据 182 页上关于定义作用范围的说明，用名称 {\内部命令名} 定义的命令只在命令 {\外部命令名} 内部才被知道和有效。虽然 TEX的宏利用了大量的嵌套命令定义，以限制临时性命令的寿命，但是我们并不推荐过分地使用 LATEX定义嵌套，因为这样太容易造成括号不匹配。忘记一个括号匹配，将会在第二次调用外部命令时给出错误信息，因为这时还残留第一次调用时一部分内部定义。但是我们还是给出下面这个例子：

```
\newcommand{\twentylove}
  {{\newcommand{\fivelove}
    {{{\newcommand{\onelove}
        {I love \LaTeX!}%
       \onelove\ \onelove\ \onelove\ \onelove\ \onelove}}}
    \fivelove\\ \fivelove\\ \fivelove\\ \fivelove\\ \fivelove}}
```

那么现在 `My opinion of \LaTeX:\\ \twentylove` 结果为：

My opinion of LATEX:

I love LATEX! I love LATEX! I love LATEX! I love LATEX! I love LATEX!
I love LATEX! I love LATEX! I love LATEX! I love LATEX! I love LATEX!
I love LATEX! I love LATEX! I love LATEX! I love LATEX! I love LATEX!
I love LATEX! I love LATEX! I love LATEX! I love LATEX! I love LATEX!

在定义中像上面那样把行缩进，可以帮助跟踪嵌套层次；同一层次中的每一行不考虑括号的前提下开始于同一列。

如果内部和外部定义中都有参数值，那么内外表示哑元参数值的符号必须不一样。内层定义的符号是 `##1 ... ##9`，而外层定义的符号 `#1 ... #9`。例如，

```
\newcommand{\thing}[1]{{\newcommand{\color}[2]{The ##1 is ##2.}
    \color{#1}{red} \color{#1}{green} \color{#1}{blue}}}
```

那么 `The colors of the objects are\\`

`\thing{dress}\\ \thing{book}\\ \thing{car}` 结果为

The colors of the objects are

The dress is red. The dress is green. The dress is blue.

The book is red. The book is green. The book is blue.

The car is red. The car is green. The car is blue.

像 7.5.5 节中那样分开定义和调用是比较容易接受的方法：

```
\newcommand{\thing[1]}{\color{#1}{red} \colr{#1}{green}
                       \color{#1}{blue}}
\newcommand{\color}[2]{The #1 is #2.}
```

§7.5.8 不期望的空格

有时候用户定义结构中会出现不期望的空格，或者出现比需要多的空格。这几乎总是由定义中空白或新行造成的，在定义中包含空格只是为了提高源文本可读性，当结构被调用时它们可能被解释成空白。

例如，如果在 171 页上的 `\myfootnote` 定义中去掉第一行的 `%` 字符，那么就会在此处加入一个分行符，从而转化成一个空白。这个空白要插入在接受脚注标记的前面单词与调用 `\footnote` 命令实际生成标记之间，结果就会使得标记同单词分开。

在此处我们想提醒读者许多 LaTeX命令是看不到的，即在调用它们的地方不会生成任何文本。如果在这样不可见命令与周围命令之间有空格，那么就可能出现两个空格。

```
For example \rule{0pt}{0pt} produces
```

结果为 'For example produces'，有一个两倍单词间距。没有参数值的不可见命令不会导致这种问题，因为跟在命令后面的空格被当做终止符，从而消失了。而且，下面的 LaTeX命令和环境总是去掉后续空格，即使它们有参数值：

```
\pagebreak     \linebreak     \label   \glossary    \vspace   figure
\nopagebreak   \nolinebreak   \index   \marginpar              table
```

§7.5.9 最后两个例子

通常的 `description` 环境（ 4.3.3 节）把标签设成黑体，所有接在第一行

后面的行都缩进一个固定长度。为了描述计算机代码中的命令名，可能更希望用打字机字体（ \texttt 命令或者 \ttfamily 声明）显示标签，后面的行缩进量等于最长标签的长度。为此我们现在定义一个新环境 ttscript，其使用方法为

```
\begin{ttscript}{取样文本}列表文本\end{ttscript}
```

参数值 取样文本 在使用 \texttt 命令时的宽度等于左边缩进量。 列表文本 由 \item[标签文本] 命令后接相应的解释性文本组成，其中每行相对于左边界缩进 取样文本 的宽度。 标签文本 在标签盒子中是用 \texttt 左对齐显示的。 ttscript 环境的定义为

```
\newenvironment{ttscript}[1]{%
  \begin{list}{}{%
  \settowidth{\labelwidth}{\texttt{#1}}
  \setlength{\leftmargin}{\labelwidth}
  \addtolength{\leftmargin}{\labelsep}
  \setlength{\parsep}{0.5ex plus0.2ex minus0.2ex}
  \setlength{\itemsep}{0.3ex}
  \renewcommand{\makelabel}[1]{\texttt{##1\hfill}}}}
{\end{list}}
```

这也很容易把标签的字体改成其它字体样式，重新给这个结构起个名字，把它保存起来以供一般性应用。

对上面定义中的前七行的理解应该没有任何问题。环境的定义包含一个参数值，它被如下传递：

```
\settowidth{\labelwidth}{\texttt{#1}}
```

这样就设置了标签的宽度。接着左边界的设置为：

```
\setlength{\leftmargin}{\labelwidth}
```

这样其值与标签 \labelwidth 是一样的，最后

```
\addtolength{\leftmargin}{\labelsep}
```

在左页边界中增加了标签分隔 \labelsep 的量。

第六行和第七行把 \parsep 和 \itemsep 设置成适当的值，用户可以按需要进行改变。如果必要的话，也可以包含其它的列表参数。

最后一行重定义了 \makelabel，这里需要加以解释。在 4.4.1 节中简要介绍了这一命令。它只在 list 环境中才有效，每当调用 \item 时就用它生成实际的标签。它通常的参数值就是 \item 命令的可省参数值。利用上面的重定义，现在的标签是包括字体命令 \texttt 后接 \hfill，这样在标签盒子中是左对齐的。由于重定义是嵌套在 ttscript 环境定义中，因此它的哑元参数值应是 ##1，而不是 #1，在上一节中有解释。

例如，给出如下文本：

```
\begin{ttscript}{description}
\item[list] environments are useful for . . .
\item[enumerate] environments number . . .
\item[itemize] environments mark the . . .
\item[description] environments label . . .
\end{ttscript}
```

那么我们可以得到如下关于LaTeX列表环境的描述：

`list` environments are useful for generating lists in which the various items are set off from one another for greater clarity;

`enumerate` environments number their items consecutively, starting at 1;

`itemize` environments mark the various items with a distinguishing symbol, often a bullet •;

`description` environments label the items with some text in bold face type, for greater stress.

练习 7.11: 把 `ttscript` 环境推广成更一般的 `varscript` 环境，使它具有两个参数值，第二个参数值确定标签的字体样式。

用户定义的命令和环境可以有多达 9 个参数值。一个结构拥有的参数值越多，其弹性就越大。另一方面，调用命令时就变得很复杂和冗长，因为参数的数目和顺序都必须严格与定义一致。

在 4.4.4 节中 68 页上的用户定义示例环境 `figlist` 中没有参数值。当调用它时，每条 `\item` 命令生成 **Figure 1**:, **Figure 2**: 等等标签。而且文本相对于包围文本左边界缩进 2.6cm，相对于右边界缩进 2cm。在列表声明中的类似于 `\labelsep`, `\parsep` 和 `\itemsep` 等其它列表参数取的是固定值。然而，进行了下述定义后，

```
\newcounter{itemnum}  \newlength{\addnum}
\newenvironment{genlist}[8]
  {\begin{list}{\textbf{#1 \arabic{itemnum}:}}
    {\usecounter{itemnum}
     \settowidth{\labelwidth}{\textbf{#1}}
     \settowidth{\addnum}{\textbf{\ \arabic{itemnum}: }}
     \addtolength{\labelwidth}{\addnum}
     \setlength{\labelsep}{#2}
     \setlength{\leftmargin}{\labelwidth}
     \addtolength{\leftmargin}{\labelsep}
     \setlength{\rightmargin}{#3}
     \setlength{\listparindent}{#4}
```

```
     \setlength{\parsep}{#5}
     \setlength{\itemsep}{#6}
     \setlength{\topsep}{#7}#8}}
{\end{list}}
```

就能生成一个广义列表,其中第一个参数确定每次调用 \item 命令时与顺序编号显示在一起的共同项名称。左边的缩进设成项名称宽度加上 \labelsep ,后者由第二个参数值给定。后面五个参数值确定各种列表参数值的长度。最后一个参数值为指定环境中文本的字体的声明。这个新环境的语法为:

```
\begin{genlist}{ 项名称 }{ 标签间距 }{右边距 }{ 列表段落缩进 }
  { 段落间距 }{ 项间距 }{ 顶部间距 }{ 字体样式 }
 \item 文本
  ... ...
 \item 文本
\end{genlist}
```

当如下调用

```
\begin{genlist}{Sample}{4mm}{1cm}{0pt}{1ex plus0.5ex}
     {0pt}{0pt}{\slshape}
  \item no value
  \item The enclosed coupon is worth a value of \$2.00 towards
        the purchase of your next order.
\end{genlist}
```

结果为:

Sample 1: *no value*

Sample 2: *The enclosed coupon is worth a value of $2.00 towards the purchase of your next order.*

在这个例子中,每个长度项都必须有单位。当然也可以在定义中直接包含单位,这样在调用时只需给出数值就可以了。同样竖直距离中的橡皮长度也可以用如下定义中的算法给出:

```
\setlength{\parsep}{#5ex plus0.3#5ex minus0.5#5ex}
```

这里要求第五个参数值为纯粹的数字。如果其值为 2 , \parsep 就会等于 2ex plus0.6ex minus1ex 。这里还要提出的是,一定在要简化项数与记住其含义的复杂之间取得折衷,特别是如果各种不同长度参数值具有不同的大小和算法的时候更要如此。

©TEXGuru, August 15, 1999

第八章　高级功能

本章描述 LaTeX用来调整所谓 '文档准备系统' 的那些功能，而前面几章主要着重于文本处理。修饰词 '高级' 可能会有误导作用，因为这里只是指这些功能对于高效地创建长而复杂的文录是相当重要的。这里的内容包括把一个文档分成几个文件，文档中不同部分的选择处理，章节、插图和公式的交叉引用，自动生成参考文献、索引和汇总，处理不同的字体集合，准备报告材料。

§8.1　处理文档的各个部分

我们已经多次指出，LaTeX文档是由导言和实际的文本两部分组成。对于较短的文档，如初学者通常处理的文档，可以用文本编辑器输入到单个文本中就可以了，而且可以在第一次显示之后再进行修改。当用户积累了足够的经验和信心后，LaTeX文档就可能在长度上急速膨胀，甚至有可能想生成长达 1000 页的整本书籍。

理论上这样长的文档是可以保存在一个文件中的，虽然这会使得整个操作变得非常笨拙。对于长文件，文本编辑器的功能大打折扣，而且 LaTeX处理起来也会相应地花费很多时间。比较好的主意是把工作分成几个文件，在处理时再由 LaTeX把它们合并起来。

§8.1.1　\input 命令

可以用下面这条命令把另一个文件的内容读进 LaTeX文档：

\input{ *文件名* }

这里另一个文件的名称是 *文件名*.tex 。只需要指定另一个文件的基本名，不需要扩展名 .tex 。在 LaTeX处理过程中，包含在这第二个文件中的文本将会被包含进来，放到第一个文件给出命令的地方。

\input 命令的结果与直接把文件 *文件名*.tex 中的内容输入到文档文件中该处是完全一样的。这条命令可以放在文档的任何地方，既可以放在导言中，也可以放在正文部分。

由于 \input 命令可以放在导言中，因此可以把整个导言本身放到单独一个文件中。这样 LaTeX实际处理的文件甚至可以只包含命令 \begin{document} ... \end{document} ，中间有很多 \input 命令。当有一系列的文档使用相同的导言时，把导言放在单独一个文件中就是非常合理的。这样即使修改了导言，也不必在所有文档中进行另外的修改。可以为各种不同类型的处理提供不同的导言文件，然后用 \input{*处理类型*} 选择。

一个用 \input 命令读入的文件中也可以包含 \input 命令。嵌套深度只受计算机能力的限制。

为了得到所有读入文件的清单，可以把命令

2ε \listfiles

放在导言中。当处理完毕，这个清单会同时出现在计算机显示器和抄本文件中。版本号和其它的上载信息也会显示出来。这提供了一种检查到底有哪些文件被输入的方法。详情请见 C.2.9 节。

练习 8.1: 把前面一致在用的练习文件 exercise.tex 中的导言放在单独一个文件 preamble.tex 中。把正文分成三个文件 exer1.tex, exer2.tex 和 exer3.tex 。那么为了保证 LaTeX处理整个练习文本，现在主文件中应该包含什么内容呢？

§8.1.2 \include 命令

把文档分成几个文件，对于输入和编辑来说是比较实用的，但是当通过 \input 命令把它们合并起来后，处理的仍是整个文档。这样即使在一个文件中进行了很小的改动，所有的文件都要被重新读入和处理。因此我们希望能够提供一种方法，可以只重新处理被修改的那个文件。

一个比较粗糙的方法是写一个临时性文件，只包含导言（无论如何，它都需要被读入）和一条 \input 命令读入那个特定的文件。这种方法的缺陷是所有页码、章节、插图和公式等等的自动编号都是从 1 开始的，因为所有来自于前面文件的信息都没有了。而且，来自于其它文件中的交叉引用也行不通了。

更好的方法是用 LaTeX命令

\include{ *文件名*}

这条命令只能用于文档的正文部分，另外命令

\includeonly{ *文件清单* }

只能位于导言中，*文件清单* 包含了所有要被读入的文件。文件名之间用逗号分开，后缀 .tex 不需要的。这两条命令要结合使用。

如果 *文件清单* 中包含 *文件名*，或者在导言中没有 \includeonly 命令，那么 \include{*文件名*} 等价于

\clearpage \input{ *文件名* } \clearpage

然而，如果 *文件名* 不包含在 *文件清单* 中，\include 等价于 \clearpage，其中内容并不被读入。

\include 命令要比 \input 命令少一些普遍性，因为它总是在新页上开始。因此文档应该在开始新页的地方分成文件，比如章与章之间。另一个局限性是 \include 命令不可以嵌套：它只能位于主处理文件中。然而，\input 命令可以出现在 \include 进来的文件中。

©TEXGuru, August 15, 1999

\include 命令的主要好处在于关于页面、章节和公式编号的附加信息由 \includeonly 命令提供的，因此选择处理时这些记数器的值也是正确的。来自于其它文件中交叉引用信息也是可用的，因此 \ref 和 \pageref 命令 (8.3.1 节) 生成正确的结果。所有这些值是由前面一次完整处理 (用 LaTeX编译文件) 确定下来的。

如果对被选择处理的文件进行了修改，导致页码的增加或减少，后面的文件也需要被重新处理以校正页码。如果增加或去掉了某些章节，或者公式、脚注和插图等等的编号发生了变化，也需要进行同样的操作。

例如，假设 文件三 在第 17 页上结束，但是经过选择处理后，它现在长度扩展到了第 22 页。但是后接的 文件四 还是从第 18 页开始，所有后面的其它文件也都具有自己的起始页码。如果 文件四 被选择处理，那么它将会得到正确的起始页码，即根据修改后的 文件三 保存的信息，确定现在是从第 23 页开始。其它依次类推。然而，如果在 文件三 后选择处理的是 文件六 ，那么它将得到来自于 文件五 的起始页码，而这个页码还没有修正，因此这之间会差 5 页。对其它结构、记数器也有同样的问题。只有当文件按正确的顺序进行了重新处理，才能保证它们取正确的值。

虽然有这些限制，对于长文档 \include 命令是相当有用的，它可以节省相当可观的用机时间。长文档在输入和编辑时通常要分很多步。\include 命令可以使得在很短时间内有选择地重处理改动之处，即使编号系统工作不正常也可以。随后进行一次完整的处理，即去掉导言中的 \includeonly 命令就可以做到这一点。

用 \include 读入的文件中不能包含任一 \newcounter 声明。这并不是一个过分的限制，因为通常它们就放在导言中。

本书的每一章就是用单独的文件输入的，它们名称分别是 chap1.tex, chap2.tex, ... 。处理文件本身就包含如下文本：

```
\documentclass{book}
  . . . . . .
\includeonly{...}
\begin{document}
\frontmatter \include{toc}
\mainmatter
\include{chap1} . . . \include{chap8} . . .
\backmatter \printindex
\end{document}
```

这里文件 toc.tex 就是由如下文本组成：

```
\setcounter{page}{7}
\tableofcontents \listoftables \listoffigures
```

通过在 \includeonly 命令中填加适当的项，就可以有选择地处理各章：例如，利用 \includeonly{toc,chap8} 就可以只处理目录表和第 8 章。

§8.1.3 终端输入和输出

有的时候希望在处理过程中 LaTeX能在计算机终端上显示出一条消息。这可以用如下命令来做到：

```
\typeout{信息}
```

这里的 信息 就代表要显示在计算机屏幕上的文本。当 LaTeX处理到这条命令时，就会显示出这段文本。同时，消息也写入到 .log 文件中 (8.9 节) 。

如果 信息 中包含用户定义的命令，那么它要被解释，并把翻译后的结果显示在屏幕上。对 LaTeX命令也要做同样的事情。如果命令 (无论是用户定义的，还是 LaTeX命令) 并不是可显示的，那么这会造成可怕的后果。要显示命令名，就在命令的前面加上 \protect 命令。

命令

```
\typein[\ 命令名 ]{ 信息 }
```

也会把 信息 显示在终端上，但是它会等待用户从键盘上输入一行文本，用回车结束。如果没有可省参数值 \命令名 ，那么这文本就直接插入在处理过程中。举个例子，这样我们就可以重复利用一封信的相同文本，而写上几个不同的地址，只要每次从键盘上输入不同信息就可以了。假设文本是如下组成的：

```
Dear \typein{Name:}\\ ...
```

那么就会在屏幕上显示：

```
Name:


\@typein=
```

此时，我们可以输入收信人的姓名。如果的接连几次处理中输入了 ‘George’, ‘Fred’ 和 ‘Mary’ ，那么结果就是同样的信件内容，只是称呼不一样，它们是 ‘Dear George’, ‘Dear Fred’ 和 ‘Dear Mary’ 。

如果 \typein 命令包含可省参数值 \命令名 ，那么就认为它等价于

```
\typeout{ 信息 }\newcommand{\ 命令名 }{输入的定义 }
```

这样就会交互式地把定义保存在名称为 \命令名 的命令中，可以在文档的其它部分同其它 LaTeX命令一样调用和执行它。

当对 LaTeX方法有了一定经验后，那么就会发现利用 \typein 命令进行交互处理是可行的。例如，如果导言中包含

```
\typein[\files]{Which files?}
\includeonly{\files}
```

那么就会在屏幕上显示如下信息：

```
Which files?


\files=
```

LaTeX等待用户输入一个或多个要处理的文件名(中间用逗号分开)。这就避免了每次必须用编辑器修改主处理文件。

本书就是用这种方法结合上一页的主处理文件生成的。

当一封礼仪信件要送给不同的收信人时，也可以采用类似的过程。我们可以交互式地输入收信人的姓名、地址，甚至包括称呼。整封信由 LaTeX用这种方法处理，用户从键盘上输入各项信息。

警告： `\typein` 命令不能用作其它 LaTeX命令的参数值！然而它可以出现在类似于 `minipage` 这样的环境中。

练习 8.2: 修改练习 8.1 中的主文件，使得文件 `exer1.tex`, `exer2.tex` 和 `exer3.tex` 可以用 `\include` 命令读入。而且你可以交互式地确定要处理的文件。

练习 8.3: 生成下面这种结构：

Certificate

Olympic Spring Games

Waterville 1992

Finger Wrestling

Gold	A. T. Glitter	AUR	7999.9	Points
Sliver	S. Lining	ARG	7777.7	Points
Bronze	H. D. Tarnish	CUP	7250.0	Points

使得在屏幕上依次显示下列要求：

```
信息        命令 =      输入
Sport:      \@typein=   Finger Wrestling
Unit:       \unit    =  Points
Gold:       \@typein=   A. T. Glitter
Country:    \@typein=   AUR
Value:      \@typein=   7999.9
Silver:     \@typein=   S. Lining
.  .  .     . . .    =  .  .  .
```

这样可以交互式生成所需的条目。第三列中包含生成上面输出所需的输入。用不同的条目重复上面的程序。尽你能力想像输入。

§8.2 在 LaTeX中包含 TeX命令

TeX应用的快速扩展主要归功于 LaTeX的实用性，因此有些用户可能根本不知道 TeX 。他们认为 LaTeX就是一个单独可执行的程序，是与 TeX并存

的。实际上， LaTeX只是用户与 TeX处理程序之间的一个界面，它显著简化了 TeX操作。它把输入的逻辑结构翻译成起作用的 TeX命令，在内部通过调用 TeX来处理它们。

因此可以在 LaTeX内部包含纯粹的 TeX命令。这理所当然地适合于所有 TeX原语命令。除了大约 300 条 TeX原语命令外，还有另外 600 个左右的定义在 Plain TeX格式中的宏。运行 LaTeX和 Plain TeX之间的正式差别就在于装载的格式文件是 `latex.fmt`（或者 LaTeX2.09 中的 `lplain.fmt` ），而不是原来的 TeX格式文件 `plain.fmt` 。然而，由于 LaTeX格式包含了 Plain TeX中绝大多数宏定义，注意并不是全部，绝大多数 TeX宏也可以在 LaTeX内部调用。

那些不能用在 LaTeX中的 TeX宏列在 **??** 节。那些就是唯一在 LaTeX中不能用或者功能有些不同的 TeX命令。

尽管可以在 LaTeX文档中使用 (Plain) TeX命令，我们还是推荐只要有可能，就尽量用高级命令。这是对可移植性和稳定性的最好保证。然而，由于有很多精细的功能（或技巧）只能在 TeX的层次上得到，因此禁止这种应用又是相当不明智的。关于编码指南方面的详细讨论请见 C.1.4 节，在 C.2.12 节中有一些有用的 TeX命令。

§8.3 正文内的引用

在长的文本中我们经常需要引用在其它地方进行了描述的章、节、表格和插图，或者页。同时也需要生成一个索引记录，它是对整篇文档中特定关键词的引用。在电子文本处理以前的时代中，如此的交叉引用和索引意味着要耗费作者或秘书大量的工作。现在计算机可以承担这一负担的绝大部分。

在以前岁月里，引用前面文本部分的页码虽然是费事的，但总是可行的。但是为了说明将要讲述的内容，引用还没有写出来的部分就只能局限于章节编号了，因为页码还不知道，或者留下空白以备将来填上页码。

编写一本书通常是一个渐进的按步就班的工作。手稿可能根本就不是按顺序写的，而且当初稿完成后，由于基于作者新的考虑或者来自于评论者的有见地的建议，有可能对它进行很大的修改。修订、删除或插入某些节，甚至某些章都是完全有可能的，更不用说有可能交换文本某些部分的顺序了。

LaTeX把所有这些由于大改动造成的问题都变成了历史。无论作者进行了怎样的改动，交叉引用和索引记录所需要的信息都被保存起来，以供在正文中任何地方使用。

§8.3.1 交叉引用

前面有多处地方已经提到过，命令

```
\label{记号}
```

用来给它所位于的文本中该点设置一个标记,以便在其它地方引用这一文本。区别标记的符号是 *记号* 文本，它可以是字母、数字和字符 (除了那些表示单个字符命令的特殊符号： \ # $ % & ~ ^ _ { }) 的任意组合。

\label 命令被调用时所处的页码可以用下面命令来显示：

\pageref{ *记号* }

其可以位于文档的任何地方。

如果 \label 命令在章节命令后面给出，或者位于 equation, eqnarray 或 enumerate 环境中，或者是在 figure 或 table 环境中的 \caption 的参数值中，命令

\ref{ *记号* }

就会用正确的格式显示出 *记号* 被定义处的章节、公式、插图、表格或枚举的编号。对于 enumerate，显示出来的编号是 \label 所处的 \item 命令生成的编号。对于由 \newtheorem 命令创建的定理类型结构，如果 \label 命令出现在定理命令的文本中，那么也可以引用它。例如，在 70 页上 Bolzano-Weierstrass 定理的内容中用 \label 放上 *记号*bo-wei：

```
\begin{theorem}[Bolzano-Weierstrass]
                    \label{bo-wei}...\end{theorem}
```

因此输入文本

```
Theorem~\ref{bo-wei} on page~\pageref{bo-wei}
```

将得到输出 'Theorem 1 on page 70'。类似地，输入文本

```
for Table~\ref{budget95} on page~\pageref{budget95},
see also Section~\ref{sec:figref}
```

结果为 'for Table 6.1 on page 159, see also Section 6.6.6'，因为那一节包含记号 \label{sec:figref}。

标记命令 \label 可以位于章节命令的参数值内，也可以位于该节正文中的其它地方。由相应的 \ref 命令显示的编号是 \label 出现的最内层那节的编号。为了避免可能出现的混淆，我们建议把 \label 命令就放在被引用章节命令后面。

如果安装了 LaTeX，那么处理文件 lablst.tex— 就可以得到一张清单，其中包括所有的 *记号*， *记号* 转换后的结果，以及页码。

了解交叉引用信息的管理方式是相当有用的。实际上 \label 命令就是把 *记号* 文本连同相应记数器的值和当前页码写到一个辅助文件中，这个文件的基本名与正在处理的文档文件相同，后缀为 .aux。 \ref 和 \pageref 命令就是从这个 .aux 文件中得到相关信息的，这条命令是由 \begin{document} 命令开始读入的。在创建目录表时出现的情形，这里也会发生：在第一次运行过程中， .aux 文件并不存在，因此不会输出任何交叉引用信息；而只是收集信息，并在第一次运行结束时写入一个新的 .aux 文件中。如果辅助文件已

发生了改变，需要再运行一次，在本次处理结束时会给出一条警告信息。

§8.3.2 参考文献的引用

在 4.3.6 节已讲述了参考文献的创建及对它的引用，并在 323 页上给出了演示，这里重复讲述，只是为了给出完整的对交叉引用的讨论。参考文献是如下生成的：

```
\begin{thebibliography}{ 标签样本 }
  \bibitem[ 标签一 ]{ 关键词一 } 文本条目一
  \bibitem[ 标签二 ]{ 关键词二 } 文本条目二
  . . . . . .
\end{bibliography}
```

在 4.3.6 节解释了参数值的意义，这里只解释一下引用关键字，不再重复其它各项。标记 *关键词* 标演了 `\label` 命令中 *记号* 同样的角色。而且它可以是字母、数字和字符的任意组合，只是不能用逗号。在正文中对参考文献的引用是用如下命令给出的：

```
\cite[ 附加信息 ]{ 关键词 }
```

它把相应 `\bibitem` 命令的标签放在中括号内。输入：

```
For additional information about \LaTeX\ and \TeX\
see~\cite{lamport} and \cite{knuth,knuth:a}.
```

那么结合在 4.3.6 节的参考文献样例，就会得到如下输出：

For additional information about LaTeX and TeX see [1] and [6,6a] 。

引用标记 `lamport`, `knuth` 和 `knuth:a` 被转化成相应的参考文献中的引用标签。

如果在 `\cite` 命令中包含了可省参数值 *附加信息* ，那么这块文本就会加在标签的后面，但仍然是在中括号内。

```
The creation of a bibliographic database is described in
\cite[Apendix B]{lamport}, while the program \BibTeX\
itself is explained in \cite[pages 74,75]{lamport}.
```

The creation of a bibliographic database is described in [1, Apendix B], while the program BibTeX itself is explained in [1, pages 74,75].

在 `thebibliography` 环境中作者可以利用 `\bibitem` 命令逐项建立参考文献。另外还有一个单独的程序 BibTeX，它通过从一个或多个参考文献数据库中搜索出现在 `\cite` 命令中的 *关键词* 标记自动生成参考文献。这里的关键词必须与数据库中的一致。

在正文中没有引用的项也可以包含在参考文献中。这是利用下面的命令：

`\nocite{` *关键词* i, *关键词* j, ...`}`

把它放在正文任何地方都可以，这里 关键词i, 关键词j, ... 就是这些附加项的 关键词 。参考文献自身现在是用命令

\bibliography{ 数据库 a, 数据库 b, ...}

生成的，这里 数据库a, 数据库b, ... 就是包含要被搜索的参考文献数据库文件的基本名。这些文件的扩展名为 .bib 。在 323 页上有一个关于如何使用BibTeX的例子。

在附录 B 中讲述了如何建立参考文献数据库，如何使 BibTeX程序正常运行。概略地讲，第一次在LaTeX上运行之前，必须用文档基本名单独执行一次 BibTeX程序。然后，LaTeX必须至少执行两遍，第一次建立参考文献，建立关键词与标签之间的对应关系，第二次在 \cite 命令的关键词地方，插入标签。如果加进了新的引用或者去掉了原来的引用关键词，必须再执行一次BibTeX 。

§8.3.3 索引记录

LaTeX并不会像处理目录表那样自动生成一个索引记录，但它可以帮助作者建立关键词和页码的索引。

索引记录是用下面的环境生成的：

\begin{theindex} 索引条目 \end{theindex}

这个环境切换到两列页面格式，它具有一个活动标题 *INDEX* 。在索引记录第一页上有一个标题 **Index** ，它的尺寸在 book 和 report 文档类同章标题的一样，在 article 中同节标题的一样。（更精确地说，实际显示出来的单词是包含在命令 \indexname 中，可以重定义它，以适合其它语种。）每一项都是用如下命令

\item \subitem \subsubitem 和 \indexspace

后接关键词和相应的页码组成的。例如，

commands, 18
    as environments, 42
    arguments, 19, 101
        multiple, 103, 104
        replacement symbol, 20
    used as arguments for sectioning commands, 41,42
displayed text, 21-32

```
\item commands, 18
\subitem as environments, 42
\subitem arguments, 19, 101
\subsubitem multiple, 103, 104
\subsubitem replacement symbol, 20
\subitem used as arguments for
   sectioning commands, 41, 42
\indexspace
\item displayed text, 21--32
```

如果文本项太长，在一行中放不下，就会断开，接在下一行上，并且向内缩进，深度比其它行都大，如上面例子中的 'used as argument for sectioning commands, 41, 42' 。命令 \indexspace 在索引记录中插入一个空白行。

theindex 环境只是建立起索引记录的一个适当格式。其中的每一项，包括页码，都必须手工输入。然而，在确定页码方面LaTeX可以提供一些帮助。

可以在正文中任何地方调用下面这条命令

\index{ 索引条目 }

这里 索引条目 可以是任何文本，但它应是后面索引中的一项。它可以是字母、数字和符号的任意组合，甚至可以包含命令字符和空格。这也就是说命令也可以包含在 索引条目 中，如 \index{\section}, \index{\[} 或 \index{% }。即使从不能用作参数值的命令 \verb 也可以包含进来。然而，如果 索引条目 包含命令，那么它就不能再做为其它命令的参数值。另一个限制就是如果在 索引条目 中有一个左大括号，那就必须同时给出右大括号。因此不能用 \index{\{}，但可以用 \index{\{\}}。

除非导言中包含下面这条命令，否则所有 \index 命令都会被 LaTeX忽略：

\makeindex

这一命令激活所有的 \index 命令，并打开一个文件，其基本名与文档文件相同，后缀 .idx 。现在 \index 命令就会向这个文件中写入 索引条目 和当前页码，格式为：

\indexentry{ 索引条目 }{ 页码 }

对于最简单的情形，可以输出 .idx 文件，从而得到一张索引项与对应页码的清单。利用这张清单，作者就可以用前面给出的那种方式生成 theindex 环境中的各项。这应该是文档编写已到了最后阶段时的工作，因为否则页码的改变会导致可怕的更正工作量。

在 LaTeX的安装文件中有一个叫 idx.tex 的文件，利用它可以提高 .idx 文件的可读性。当处理 idx.tex 文件（即调用 latex idx ）时，用户会被提示输入要列出来的 .idx 文件名：

```
*******************************
* Enter idx file's first Name. *
*******************************

\filename=
```

当从键盘上输入了 .idx 文件的基本名后，就会输出两列页面格式，其由出现在 Page n 形式页标题下面的 索引条目 文本组成。从这个有格式的列表中再得不到其它的信息，但它要比直接利用输出的 .idx 文件容易得多。

即使当导言中没有 \makeindex 命令，从而使得 \index 命令无效的时候，在编写文档之初就应包含这些文本，这就不失为一个好主意。当文档终稿确定后，再包含 \makeindex 命令。最后就可以利用来自于 .idx 或 idx.dvi 文件的各项来组织 theindex 环境。这仍旧是一件烦人的工作，需要相当大的耐心和集中的精力，因为 \item, \subitem 和 \subsubitem 项必须按字母顺序， .idx 只是按在文档中的顺序给出了所需信息。

然而，还存在着更好的方法的！有一个可用的程序，它从 .idx 文件生成 theindex 环境，就如同 BibTeX程序生成参考文献一样。在下一节讲述这一

工具。

在LaTeX安装文件中也包含一个软件包 showidx，它会把 \index 命令中的索引项显示为边注，在它所处的那页边界从页顶开始。当浏览文档的初始版本，以看看所有的索引项是否位于正确的页面或者是否生成了新增项，这一方法是相当有用的。这个软件包是用 \usepackage 命令调入的，或者把它放在 \documentstyle 的选项列表中。

如果用了 showidx，那么在导言中用 \marginparwidth 声明(4.10.7 节) 增加宽度以适应边注宽度不失为一个好主意。不幸的是，现在书籍格式并不适合于进行这种演示。

§8.3.4 汇总

所谓汇总，就是一类特殊的索引，它是术语和短语按字母顺序排列，连同其解释一起组成的。为了帮助建立一个汇总，LaTeX提供了命令

\makeglossary　　放在导言中，

\glossary{汇总条目}　　放在正文部分

这两条命令的作用方式同组织索引记录是一样的。当在导言中使用了命令 \makeglossary 时，就会把各项写到一个后缀为 .glo 的文件中。文件中来自于每条 \glossary 命令的条目格式为

\glossaryentry{ 汇总条目 }{ 页码 }

在 .glo 文件中的信息可以用来建立汇总。然而，并不存在着汇总的与 theindex 环境等价的结构，因此可以用 description 环境 (4.3.3 节) 或者特殊的 list 环境 (4.4 节)。

§8.4 MakeIndex— 关键词处理器

如果可以使用 MakeIndex 程序，那么利用 theindex 环境生成一个索引记录的这种烦人苦差事就会免掉。这个程序是 Pehong Chen 在 Leslie Lamport 的帮助下做出的。我们这里只是给出其用法的简略描述。在随程序软件所带的文档中有详细的讲解。

程序 MakeIndex 处理 .idx 文件，得到输出文件，其基本名与文档文件的相同，但是后缀 .ind，这个文件由完整的 theindex 环境组成。程序的调用方法为：

makeindex 基本名 .idx 或者简单地 makeindex 基本名

接着当给出 \printindex 命令时，LaTeX就经过处理，在该命令处给出索引，这条命令连同 \see 命令一起，都是在软件包 makeidx.sty 中定义的。因此要用这种方法得到索引记录，需要用 \usepackage 命令装载 makeidx 软件包。

MakeIndex 软件包希望 \index 项是下列三种形式之一：

```
\index{主条目}
\index{主条目!子条目}
\index{主条目!子条目!子子条目}
```

每个 *主条目*, *子条目* 或者 *子子条目* 可以是除 `!`, `@` 和 `|` 字符外的任意组合。在这里惊叹号是各个条目之间的分隔符。如果 `\index` 命令中只有一个主条目，那么它就会成为 `\item` 命令中的文本。主条目将以字母顺序排列。

如果 `\index` 命令由主条目和子条目组成，那么子条目的文本给赋给相应主条目下面的 `\subitem` 命令。 `\subitem` 的文本也要以字母顺序排列。同样地，子子条目的文本将会跟在子条目文本下面的 `\subsubitem` 命令后，并以字母顺序排列。

主条目和子条目也可以包含特殊字符，甚至可以是在排序过程会被忽略的 LaTeX命令。其标志是条目的形式为 *排序列条目*`@`*显示条目* ，这里 *排序条目* 是用来进行字母排序的，而 *显示条目* 是实际要输出的文本。例如，本书的索引就是用 MakeIndex 生成的，所有的命令名都做为源文本出现，在排序时忽略前面的 `\` 字符。这些条目就是用类似于 `\index{put@\verb=\put=}` 这样的文本得到的。

条目也可以用字符序列 `|(` 或 `|)` 结尾，以标志页码范围的开始与结束。例如，如果

```
在第 136 页上有 \index{picture!commands|(}
在第 146 页上有 \index{picture!commands|)}
```

结果就是在主条目 'picture' 下面的子条目 'commands' 后面显示页码为 136–146 。

可以不在条目后面显示页码，代之以对另一条目的引用。例如，利用

```
\index{space|see{blank}}
```

就会在索引中得到 'space, *see* blank' 。（更准确地说，是存贮在 `\seename` 命令中的文本显示在 *see* 的地方；这样可以改变它以适应其它语言。）

对于 MakeIndex 程序， `!`, `@` 和 `|` 这三个字符因此具有特殊的作用。为了按原样在文本在显示这三个字符，不让它们发挥本来的作用，需要在其前面加上引号符号 `"` 。例如， `"!` 表示的就是一个惊叹号，而不是项分隔符。

因此引号符号就成为第四个特殊符号，必须用输入 `""` 得到本来的样子。然而，在 MakeIndex 语法中有一条特殊的规则，那就是在引号前面加上反斜杠，就会认为它是命令的一部分，因此在条目中就可以用 `\"` 来得到德语重音（如 `\index{Knappen, J\"org}` ）。这条特殊的规则有时会导致另外的问题：在本书中索引条目 `\!` 就必须输入为 `"\"!` 。

可以把页码指定为不同的字体。例如，本书的索引记录中，黑体的页码用来表示命令第一次被解释或定义的地方。这是用如下形式的项得到的：

```
在第 9 页上有 \index{blank}
```

在第 17 页上有 `\index{blank|bb}`

第二种情形中把该项的页码作为命令 `\bb` 的参数值。在 `theindex` 环境中的对应行变为

```
\item blank, 11, \bb{20}
```

这一方法只有当已经在导言中进行了 `\newcommand{\bb}[1]{\textbf{#1}}` 时才可行。注意：在 `\index` 项中的竖线并不是排版失误，只是在这种情况下用来代替 LaTeX命令符号反斜杠。

也可以定义其它的页码样式，如

```
\newcommand{\ii}[1]{\textit{#1}}
\newcommand{\zz}[1]{\textsl{#1}}
```

分别表示斜体和 slanted 字体。在 `\index` 命令中就是用 `|ii` 或 `|zz` 结尾该项以进行相应的定义。

在 MakeIndex 程序中的字母排序通常是根据标准的 ASCII 码，首先是符号，然后是数字，最后是字母，而且大写字母在小写字母前面。空格是包含在符号中。有许多选项可以改变这些规则。当调用程序时如何给出选项与计算机类型有关，这里我们假设在选项字母前面加连字符，如

```
makeindex -g -l 基本名
```

最重要的选项有：

`-l` 字母顺序：排序时忽略空格；

`-c` 压缩空格：同通常的 LaTeX一样，忽略多于一个的空格或者前导空格。

`-g` 德语顺序：根据德语顺序，符号在字母前面，小写字母在大写字母前面，然后是数字；而 `"a`, `"o`, `"u` 和 `"s`（在德语版的 LaTeX中表示 ä, ö, ü 和 ß ）就当做它们是 `ae`, `oe`, `ue` 和 `ss` 处理，这与标准的德语中的用法一致。

`-s` 样式定义：允许给出索引格式文件的名称，以包含进来重定义的 MakeIndex 功能。

`-s` 选项读入一个索引样式文件，这个文件由定义 MakeIndex 程序输入和输出的命令组成。例如，可以用另外的字符来替换特殊字符 `!`, `@`, `|` 和 `"` ，这样新的字符具有原来的作用，而原来的字符成为纯粹的文本。

样式定义文件由一串关键词 – 属性对组成。属性可以是在单引号内的一个字符组成（例如 `'z'` ），也可以是在双引号的字符串组成（如 `"a string"` ）。最重要的关键词，连同其默认定义为：

`quote`	`'"'`	定义引号符号；
`level`	`'!'`	定义项分隔符号；
`actual`	`'@'`	定义词汇切换符号；
`encap`	`'|'`	定义页面格式中的虚命令符号。

有相当多的关键词用来定义复杂输出结构。这些关键词在随 MakeIndex 程序软件包一起的文档中有讲解。

文件 makeindex.tex 包含了 Leslie Lamport 给出的一个简略手册（其中没有提到样式定义）。

§8.5 新字体选择框架 (NFSS)[2ε]

当刚发明 TEX和 LATEX的时候，其可用的字体在数量上是很有限的。正是由于这个原因，在 LATEX2.09 中所用定义字体的系统弹性很差，因为我们需要改变它们的必要性实在不是很明显。高级字体命令（如 \large 和 \bf ）同最终选定的外部字体名称之间的关联是严格地固定在文件 lfonts.tex 中，而这个文件就组合在 lplain 格式中。

到了今天，可用的字体很多，其中有些字体可以与标准 CM 字体一起共用，而其它一些则需要取代某些 CM 字体。例如， E.7 节中的 Cyrillic 字体须平行于 Latin 字体加进来，但是要想在标准 LATEX尺寸命令下自动对它进行操作，则是相当复杂的过程。（我们已经知道这是可以做到的！）同样，安装 PostScript 字体激活了对错综复杂的界面宏的调用，这些宏无疑是来自于 PostScript 驱动软件包，但是它包涵了一批 LATEX软件开发人员的大量心血。

在 LATEX2.09 中另一个问题是字体样式和尺寸命令的行为 (4.1.5 节和 4.1.2 节) 。字体声明 \rm, \bf, \sc, \sl, \it, \sf 和 \tt 中的每一个都激活与当前选定的尺寸有关的特定的字体。这些声明中每一个都是排它的。因此 \bf\it 的组合实际上就与 \it 相同。在这个框架中是无法选择黑体斜体字体的。而且，从 \tiny 到 \Huge 的尺寸声明都是自动切换到 \rm ，因此 \bf\large 并不生成所期望的黑体大号字体。最后，这里使用的是声明，而不是有一个参数值的命令，这与 LATEX的基本哲学相矛盾。也就是说，为了强调单个词，输入 \emph{single} 就要比 {\em single\/} 更合理些。（有经验的 LATEX用户可能会否认这一点，但只是因为他们已完全习惯于原来的那一套。）

在 1989 年， Frank Mittelbach 和 Rainer Schöpf 为 LATEX提出了一个新字体选择框架 (NFSS) ，并在 1990 年给出了一个初步的测试软件包。在 1993 年年中，第二个版本 (NFSS2) 问世，这一版本做了相当大的改进。在 1994 年 4 月正式发行的 LATEX 2ε中， NFSS已经在新标准中占稳了脚跟。新字体声明和命令在 4.1.3 节和 4.1.4 中已做了讲述。这里我们更仔细地讲解这个系统，给出一些低级字体选择命令。

§8.5.1 在 NFSS中的字体属性

按照 NFSS框架，每个字符集可以按照如下五种属性分类：编码、族、序列、形状和尺寸，可以用下面的命令来选择这些属性：

\fontencoding{ 编码 } \fontfamily{ 族 } \fontseries{ 权与宽度 }

\fontshape{形状} 和 \fontsize{尺寸}{基线间距}

编码属性是在 NFSS第二版本中新出现的。它定义字体中字符的布局。在表 8.1 中给出了它的可能取值。我们一般不大可能需要在一篇文档中改变编码，当然激活 Cyrillic 字体除外。这个特征就是为了允许程序开发者可以在系统中安装新的字体。

表 8.1: NFSS编码 框架

编码	描述	字体样例	页码
OT1	来自于 Knuth 的原始文本字体	cmr10	368
OT2	Washington 大学的 Cyrillic 字体	wncyr10	373
T1	Cork(DC) 字体	dcr10	—
OML	TEX数学字母字体	cmmi10	371
OMS	TEX数学符号字体	cmsy10	371
OMX	TEX数学扩展字体	cmex10	372
U	未知编码	—	—

在 \fontfamily 命令中的 族 参数值表示字体的一组基本属性，或来源。对于计算机现代字体，列在 E.3.1 节中所有 serif 字体都属于 cmr 族。而族 cmss 包含 E.3.2 节中的所有 sans serif 字体，族 cmtt 包含 E.3.3 节中的所有打字机字体。在 E.4 节的一些特殊装饰性字体是它们所在族的唯一成员。在表 8.3 中根据族和其它属性列出了所有的 CM 字体。

注意：字体属性 '族' 与原来 TEX中同名概念之间没有任何关系。一个TEX的族可以由在数学公式中做为普通文本、第一层和第二层下标所用的三种不同尺寸的字体组成。

在 \fontseries 中的参数值 权与宽度 表示字符的权（ = 粗细程度）和宽度。它们是由表 8.2 中所给的 1 到 4 个字母来定义。

\fontseries{权与宽度} 的参数值是由对应于权的字母后接对应与宽度的字母组成。因此 ebsc 表示权为较黑，宽度为较松，而 bx 意味着权为黑，宽度为松。同任何非正常权或宽度组合时，字母 m 可以忽略；如果两者都是正常，那么只要给出 m 就可以了。

在 \fontshape 中，参数值 形状 是 n, it, sl 或 sc 字母组合中的一种，分别表示正常（直立）、斜体、 slanted 或小体大写字母。

\fontsize 属性命令有两个参数值，第一个参数值 尺寸 表示字体以点为单位的大小（不显式地给出 pt 单位），而第二个参数值 基线间距 是从一个基线到下一个基线的竖直距离。第二个参数值成为 \baselineskip 的新值（ 3.2.3 节）。例如， \fontsize{12}{15} 选择 12pt 的字体尺寸，行间距为 15pt 。（第二个参数值可以给出单位，例如 15pt ，但如果没有给出单位，就

表 8.2: NFSS序列 属性

权类		宽度类		
超轻	ul	超紧	50%	uc
较轻	el	较紧	62.5%	ec
轻	l	紧	75%	c
半轻	sl	半紧	87.5%	sc
中间（正常）	m	中间	100%	m
半黑	sb	半松	112.5%	sx
黑	b	松	125%	x
较黑	eb	较松	150%	ex
超黑	ub	超松	200%	ux

认为是 pt 。）

一旦五个属性都已设置好，就可以用 \selectfont 命令选择字体。这里的新特征是各种属性是彼此独立的。改变其中一个，并不会改变另一个。例如选择：

```
\fontfamily{cmr} \fontseries{bx} \fontshape{n} \fontsize{12}[15]
```

已经生成了一种直立、黑体、松的罗马字体，尺寸为 12pt，行间距 15pt，那么当后来用 \fontfamily{cmss} 选择一种 sans serif 字体，那么属性中的权和宽度 bx，形状 n，尺寸 12（ 15pt ）在进行下一 \selectfont 调用时继续有效。

等价地，可以利用下面这条命令来定义除尺寸外的所有属性，并同时马上激活字体；

\usefont{ 代码 }{ 族 }{ 序列 }{ 形状 }

下面这个表格 (表 8.3, 由 F. Mittelbach 和 R. Schöpf 提供) 列出了根据 \fontfamily, \fontseries 和 \fontshape 属性对计算机现代字符集的分类。有相当多的属性组合并不对应任一 CM 字体，这看起来好像是 NFSS系统的一种缺陷，但我们要知道，这一设计是为将来考虑的。它也可以应用于正变得越来越普及的 PostScript 字体，以开发其完整用途。

形式上是可以任意设置属性的组合的；然而，可能不存在一种字体对应于所有选择的属性。如果出现这种情况，那么当调用 \selectfont 时，LaTeX会给出一条警告信息，告诉你它用什么字体取代所需字体。在 \fontsize 命令中的字体 尺寸 属性通常可以取 5, 6, 7, 8, 9, 10, 10.95, 12, 14.4, 17.28, 20.74，但也可以加上其它值。第二个参数值，即 基线间距，可以取任何值，因为它并不是字体固有的性质。

利用 \begin{document} 命令，LaTeX给五种属性设置当前特定默认值。

表 8.3: 计算机现代字体的属性

序列	形状	外部字体名称示例
计算机现代罗马字体 —(\fontfamily{cmr})		
m	n, it, sl, sc, u	cmr10, cmti10, cmsl10, cmcsc10, cmu10
bx	n, it, sl	cmbx10, cmbxti10, cmbxsl10
b	n	cmb10
计算机现代 Sans Serif 字体 —(\fontfamily{cmss})		
m	n, sl	cmss10, cmssi10
bx	n	cmssbx10
sbc	n	cmssdc10
计算机现代打字机字体 —(\fontfamily{cmtt})		
m	n, it, sl, sc	cmtt10, cmitt10, cmsltt10, cmtcsc10

这通常就是标准编码 OT1，族 cmr，中间序列 m，正常形状 n 和选择的基本尺寸。用户可以在导言中改变这些值，或者用特殊选项把它们设置成不同的值，例如当已经选定了一种 PostScript 字体的时候。

§8.5.2 简化的字体选择

属性命令 \fontencoding, \fontfamily, \fontseries, \fontshape 和 \fontsize，连同命令 \selectfont，都是新字体选择框架中的基本工具。用户并不需要直接使用这些命令，而可以利用列在 4.1.2 节和 4.1.3 节的高级声明。事实上，类似于 \itshape 这样的字体声明就是定义为

\fontshape{it}\selectfont 。

用来选择字体尺寸的高级命令有：

\tiny	(5pt)	\normalsize	(10pt)	\LARGE	(17.28pt)
\scriptsize	(7pt)	\large	(12pt)	\huge	(20.74pt)
\footnotesize	(8pt)	\Large	(14.4pt)	\Huge	(24.88pt)
\small	(9pt)				

当在 \documentclass 命令中选择 10pt（默认值）做为基本尺寸选项时，上面命令相应的尺寸就是列在括号内的数值；当选择了 11pt 或 12pt 时，这些尺寸就会相应的放大。

族声明及对应的标准族属性值为

\rmfamily (cmr) \sffamily (cmss) \ttfamily (cmtt)

这分别相应于计算机现代族中的罗马、 Sans Serif 和打字机字体（ E.2 节）。

序列声明及其初始值为：

```
\mdseries (m)     \bfseries (bx)
```

这就是说在标准中只提供了中间和黑松。

最后，形状声明和相应属性值为：

```
\upshape  (n)     \itshape  (it)
\slshape  (sl)    \scshape  (sc)
```

这可以选择直立、 slanted 、斜体和小体大写字母。

注意对编码并没有高级声明。这是因为通常并不需要在文档中改变编码。当要用 Cyrillic 字体（编码 OT2 ）时就是一个例外，这时可以进行如下定义：

```
\newcommand{\cyr}{\fontencoding{OT2}\selectfont}
\newcommand{\lat}{\fontencoding{OT1}\selectfont}
```

这样就可以方便地来回切换了。

利用 \normalfont 命令，可以随时把族、形状和序列属性值重设为标准值，这也激活了当前尺寸的那种字体。

对于上面每种字体属性声明，也存在着一个相应的字体命令 (4.1.4 节)，用以设置其参数值的字体。因此 \textit{text} 就与 {\itshape text} 差不多一样，唯一的差别就在于命令中自动包含倾斜校正。这些命令的完全清单如下：

```
族：     \textrm  \textsf      \texttt
序列：   \textmd  \textbf
形状：   \textup  \textit      \textsl  \textsc
其它：   \emph    \textnormal
```

在 4.1.1 节中描述了 \emph 命令；\textnormal 把其参数值设为 \normalfont.

§8.5.3 默认属性值

在前面一节中的字体属性声明并没有显式地设置它们的值，而是用某种默认的命令来进行。因此 \itshape 的真正定义是：

```
\fontshape{\itdefault}\selectfont
```

可用的默认命令有：

```
族：     \rmdefault  \sfdefault  \ttdefault
序列：   \mddefault  \bfdefault
形状：   \updefault  \itdefault  \sldefault  \scdefault
```

当调用了 \normalfont 命令时，需要定义标准属性值。它们是由如下四个默认值组成的：

```
\encodingdefault   \familydefault   \seriesdefault|
\shapedefault
```

所有这一切都使得我们觉得好像高级命令与特定字体之间的联系是非常复杂的。实际上，它确实提供了足够的弹性和模块化结构。作者只需要知道这三个族、两个序列和四种形状是可以用的，而不用关心它们实际是什么。程序设计者用低级命令定义它们的默认值。

在 212 页上有一个例子，通过重定义三个族的默认值，来说明了如何利用 PostScript 字体取代所有四种标准字体。这样重定义比改变包括 \normalfont 在内的字体声明要简单得多。这些定义实际上要远比这里提到的复杂，然而默认命令确实就如这里所指出的那样简单。

§8.5.4 定义字体命令

有很多命令可以用来定义新的字体声明和命令。这些命令主要是为 LaTeX 宏包开发者提供的，但也可以用在普通文档中。

2ε \DeclareFixedFont{\命令}{编码}{族}{序列}{形状}{尺寸}

把 \命令 定义为一个选择具有指定属性字体的声明。所有属性都是严格固定的。这与 \newfont 基本等价，除了这里的字体是由属性而不是由名称确定的。

2ε \DeclareTextFontCommand{\命令}{字体指定}

定义 \命令 为一个字体命令，它按照 字体指定 设置其参数值。在内部就是用这条命令来定义所有类似于 \textbf 的命令，而定义 \textbf 时 字体指定 为 \bfseries 。

2ε \DeclareOldFontCommand{\命令}{文本指定}{数学指定}

定义 \命令 定义了按照 LaTeX2.09 方式可以用在数学模式中的字体声明。注意它是一个声明，不是一条命令。这对于定义与原来版本兼容命令是相当有用的，但应尽量避免使用。例如， \it 的定义为

```
\DeclareOldFontCommand{\it}{\normalfont\itshape}{\mathit}
```

§8.5.5 数学字母表

已被激活的用于文本处理的字体对数学模式中的字符及其字体并没有作用，因为此时用的是特殊的数学符号字体。如果想使一个公式显示为黑体，那么就必须用 \boldmath 命令 (5.4.9 节)，该命令的作用持续到调用相反命令 \unboldmath 为止。这些声明都必须在数学模式外面给出。

在 NFSS下也可以通过同样方式应用这些声明。然而，内部的数学字体选择命令为

\mathversion{变体名称}

这里的参数值 变体名称 通常就取 normal 和 bold 。\boldmath 和 \unboldmath 声明就是用这一命令定义的。现在有人已计划提供一些特殊的宏包文件，从而可以使用其它的数学符号集合。

另一方面，下列数学公式中的字体命令也可以在数学模式内部调用，把字母设置成特定的字体（ 5.4.2 节 ）：

```
\mathrm \mathcal \mathnormal \mathbf \mathsf \mathit \mathtt
```

这些都是对参数值有作用的命令，而不是声明，这一点与 LaTeX2.09 中的不同。

新数学字体字母表也可以由用户定义。例如，为了定义 slanted 数学字体 `\mathsl` ，可以用

```
\DeclareMathAlphabet{\mathsl}{OT1}{cmr}{m}{sl}
```

这就意味着新数学字体命令 `\mathsl` 选择族为 `cmr` ，权为 `m` 和形状为 `sl` 的字体，在通常字体定义中，这就是适当尺寸的 `cmsl` 字体。然而，在所有数学变体下，都可以选择这种字体，尽管当 `\mathversion{bold}` 有效时，选择的是黑体字体时它可能更恰当。为此可以在 `\mathsl` 定义中加入该选项：

```
\SetMathAlphabet{\mathsl}{bold}{OT1}{cmr}{bx}{sl}
```

这样就定义 `\mathsl` 为期望的只适用于 `bold` 变体，权为 `bx` 的字体。对于普通的字体定义，这就是当前尺寸的 `cmbxsl` 。

可以用下面的命令创建新的数学变体：

`\DeclareMathVersion{` *变体名称* `}`

属于它的字体是由对每个数学字母表或符号字体调用 `\Set...` 命令确定的。

§8.5.6 数学符号字体

数学符号的定义方式必须与文本字符完全不同：它们拥有一个命令名 (如 `\alpha`) ，可能来自于不同的字体，根据不同的类型有不同的行为，可以显示为不同的尺寸。在 LaTeX2.09 中，符号名称固定为计算机现代数学字体，但 NFSS为其它 (或者代替) 符号字体提供了一个相当大的弹性。

用下面这条命令声明一个符号字体名称：

2ε `\DeclareSymbolFont{` *符号字体名称* `}{` *编码* `}{` *族* `}{` *序列* `}{` *形状* `}`

这样就把 *符号字体名称* 与给定的属性集联系起来。这一名称不是一条命令，而是一个内部用来定义符号的标识。所选定的字体对所有变体都有效，除非在其它变体中指定了同名的不同字体。

2ε `\SetSymbolFont{` *符号字体名称* `}{` *变体* `}{` *编码* `}{` *族* `}`
`{` *序列* `}{` *形状* `}`

可以用来重定义一种变体下的 *符号字体名称* 。

标准 LaTeX安装中有如下声明

```
\DeclareSymbolFont{operators}{OT1}{cmr}{m}{n}
\DeclareSymbolFont{letters}{OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}{OMS}{cmsy}{m}{n}
\DelcareSymbolFont{largesymbols}{OMX}{cmex}{m}{n}
```

这一串声明是深建在LaTeX内部的，这也是它们之所以重要的原因。

一旦已定义了符号字体名称，可以用它来构造数学字母表和各种不同类型的符号。

2ε \DeclareSymbolFontAlphabet{\数学字母表}{数学字体名称}

把 \数学字母表 定义成基于内部名称 数学字体名称 的数学字母表。如果相应于这个数学字母表的字体合适属性已经存在，那么这条命令就要优于命令 \DeclareMathAlphabet 。

定义符号的主要命令是

2ε \DeclareMathSymbol{\符号}{类型}{符号字体名称}{位置}

这使得 \符号 显示在字体 符号字体名称 中处于给定 位置 的符号。这里 位置 是一个数，可以用十进制（如 10 ），八进制（如 '12 ）或十六进制（如 "0A ）表示。 类型 定义符号的功能，它可以取如下一个值：

\mathord	普通符号
\mathop	大运算符，如 $\sum$
\mathbin	二元运算符，如 $\times$
\mathrel	关系运算符，如 $\geq$
\mathopen	左括号，如 {
\mathclose	右括号，如 }
\mathpunct	标点符号
\mathalpha	字母表字符

数学字母表命令只作用在类型为 \mathalpha 的符号上；对于其它类型，在给定的数学变体里，在所有数学字母表中都生成同样的符号。

上面的数学字体声明中并没有指定尺寸。这是因为通常有四种尺寸可以使用，具体与数学样式有关，可看 5.5.2 节的解释。然而，这些尺寸必须在某个地方进行指定。这是用如下命令完成的：

2ε \DeclareMathSizes{正文}{数学文本}{上下标}{双重上下标}

这里的四个参数值都是数值，给出尺寸的点数。当正常文字体的尺寸是 正文pt 时， \textstyle 就会是 数学文本 尺寸， \scriptstyle 是 上下标 尺寸，而 \scriptscriptstyle 是 双重上下标 尺寸。例如，

```
\DeclareMathSizes{10}{10}{7}{5}
```

所有的 \Declare... 和 \Set... 命令都只能在导言中调用。

§8.5.7 处理属性值

在程序设计中，有时要利用属性的当前值，而并不知道它们的值是多少。这些值实际上保存在如下内部命令中的：

\f@encoding	\f@shape	\tf@size
\f@family	\f@size	\sf@size

```
\f@series     \f@baselineskip   \ssf@size
```

永远不能直接改变这些命令的值。然而，可以测试它们，以确定它们是否具有某个值。由于其名称中都包含字符 @，因此它们只能用在类或宏包文件中，而不能直接用在主文档文件中（附录 C）。

§8.5.8 定义 NFSS中的字体

在 NFSS中，如果需要在文档中指定字体，那么就先给出所需的属性，然后调用 \selectfont。这样确定的字体属性集合是如何与附录 E 中所讲的特定外部字体名称联系起来的呢？这是通过字体定义命令做到的，它通常保存在扩展名为 .def 和 .fd 的文件中。

首先利用声明

2ε \DeclareFontEncoding{ 编码 }{ 正文集合 }{ 数学集合 }

建立一个叫 编码 的新编码属性；无论何时选择这种编码中的一个字体，就会执行 正文集合，以重定义重音命令或其它与编码有关的事情；同样地，对于这种编码中的每个数学字母表都会执行调用 数学集合。也可以如下定义默认的 正文集合 和 数学集合：

2ε \DeclareFontEncodingDefaults{ 正文集合 }{ 数学集合 }

可以用这条命令声明一般的文本和数学模式设置，而更具体的，在后面执行的设置是位于 \DeclareFontEncoding 中的。

如果对应于指定的属性，不存在任何字体，

2ε \DeclareFontSubstitution{ 编码 }{ 族 }{ 序列 }{ 形状 }

声明出要被取代的属性值；取代是按照从 形状, 序列, 然后 族 的顺序进行的；编码从不会被取代。如果这样还找不到对应字体，那么

2ε \DeclareErrorFont{ 编码 }{ 族 }{ 序列 }{ 形状 }{ 尺寸 }

就确定出最终迫不得已时使用的字体。

指定编码框架中的一个新族是用如下命令建立的：

2ε \DelcareFontFamily{ 编码 }{ 族 }{ 选项 }

这里的 option 是一组命令，每当选择该族和编码中的一种字体时就会被执行。

把字体属性与外部字体名称关联的主要字体定义声明是：

2ε \DeclareFontShape{code}{family}{series}{shape}
{font_def}{option}

这里的 选项 是另外的命令，每当选择其中一种字体时就会执行它。

定体指定 包含一列尺寸 / 字体联系，每一个联系都是由一个尺寸对、一个函数、一个可省参数和一个字体参数值组成。例如，

```
\DeclareFontShape{OT1}{cmr}{m}{n}
   {  <5> <6> <7> <8> <9> <10> <12> gen * cmr
```

```
<10.95> cmr10
<14.4>  cmr12
<17.28> <20.74> <24.88> cmr17}{}
```

就说明 cmr 族的中间序列、正常形状成员用外部字体 cmr5 ... cmr12 表示 5–12pt 的尺寸，用 cmr10 放大到 10.95 以接近 11pt 的尺寸，等等。如果指定的尺寸不存在，就会用特定限度内最接近的尺寸。

尺寸部分由一个或多个放在尖括号内的表示尺寸 (pt) 的数字组成。括号内也可以包含范围，如 <-10> 就表示所有小于 10pt 的尺寸，而 <10-14> 表示从 10pt 到小于 14pt 的尺寸，而 <24-> 表示 24pt 或更高。可能的函数有：

(empty) (上面例子中的第二、三、四行) 上载指名的字体，并放缩到要求的点数尺寸；如果在字体名称前面有位于中括号内的可省参数值，那它就是一个额外的放缩因子：

<11> [.95] cmr10 就会上载 cmr10 并放缩到 11pt 的 95% ；

gen * (上面例子中的第一行) 把点数尺寸加到字体参数值后，生成字体名称；

<12> gen * cmr 上载 cmr12 ；

sub * 用不同的字体代替，这种字体的属性在形式为 *族/序列/形状* 的字体参数值中给出；

<-> sub * cmtt/m/n 当没有字体具有要求的属性时，最好用这种字体；会在监视器和抄本文件中给出一条信息；

subf * 类似于空函数，但是如果上载了一种显式取代字体时会给出一条警告信息；

fixed * 以正常尺寸上载指定字体，忽略尺寸部分；如果给出了可省参数值，那这个参数值就是字体要放缩到的点数尺寸，如

<10> fixed * [11] cmr12 当要求的是 10pt 时，就在 11pt 尺寸下上载 cmr12 。

上面所有函数都可以前缀一个 s(表示无声) ，以禁止显示在屏幕上的信息。因此 sub * 的无声形式为 ssub * ，无声的空函数是 s * 。

现在给出另一个例子，考虑黑斜打字机字体的定义，因为在计算机现代字体集中没有这种字体：

```
\DeclareFontShape{OT1}{cmtt}{bx}{it}{
  <-> ssub * cmtt/m/it }{}
```

这会把所有尺寸 (<->) 的字体 (无声地) 用中间斜体打字机属性取代。这就是那些由 \DeclareFontShape 命令确定的相关属性的字体。

字体定义命令也可以在宏包文件中调用，甚至在文档中也可以使用。然而，通常的过程是把先把每条 \DeclareFontEncoding 命令存贮在一个名为 *编码*enc.def (例如 OT1enc.def 对应于 OT1 编码) 的文件中，然后把命令 \DeclareFontFamily 和 \DeclareFontShape 放在一个文件中，其名称由编

码加上族标识，后缀 .fd 组成。例如，编码 OT1 和族 cmr 的形状定义可以在 OT1cmr.fd 文件中找到。当选择的编码和族组合并没有定义，LaTeX就会尝试找到相应的 .fd 文件做为输入。因此并不需要显式地输入这个文件，因为需要时可以自动调入。

然而，事先声明编码是重要的。如果在当前格式中并不知道编码，就必须调用 \DeclareFontEncoding，方法是或者显式地调用，或者上载 编码enc.def 文件。例如，做到这一点的方法就是调用已存在的宏包 fontenc：

```
\usepackage[OT2,T1]{fontenc}
```

这里所期望的编码列在中括号做为选项，这组选项就是当前的编码。

普通用户从来不必担心这些问题。然而，LaTeX程序开发者将会发现这些事情是相当容易的。例如，在NFSS中安装 PostScript 字体是相当平凡的。一些常用的 PostScript 字体已经在它们自己的 .fd 文件做为单独的族定义好了；例如 OT1ptm.fd 就把 PostScript 的新罗马字体与一个名为 ptm 的族关联起来。激活这些字体的宏包是在名为 times.sty 的文件中，其中主要包含如下几行：

```
\renewcommand{\rmdefault}{ptm}
\renewcommand{\sfdefault}{phv}
\renewcommand{\ttdefault}{pcr}
```

这就使得新罗马 ptm 成为默认罗马字体族，用 \rmfamily 命令调用，而 Helvetica phv 成为默认的 sans serif 字体族（用 \sffamily 调用），Courier pcr 为默认的打字机字体族（用 \ttfamily 激活）。

另外两个关于NFSS使用的例子是华盛顿大学的 Cyrillic 字体 (E.7 节) 和 Cork 编码中的扩展 DC 字体。这两个字体都可以在文档中只要选择另一个编码 OT2(对应于 Cyrillic)，T1(对应于 DC 字体) 就可以激活，(这些编码必须首先进行声明，例如利用上面说明的 fontenc 宏包。)

§8.5.9 编码命令

在LaTeX中，必须用命令来得到特殊的字符和重音，例如 \O 显示出 Scandinavian 字母 Ø。这个字符在字体表格中的位置有编码有关（在 OT1 中是第 31 个字符，而在 T1 中是第 216 个字符），因此当编码改变了时，需要重定义所有这样的符号命令。这可以借助于某种编码命令来完成，这些命令通常与 \DeclareFontEncoding 命令一起位于 编码enc.def 文件中。

为了定义一个在不同编码下功能不同的命令，可以用：

2ε `\ProvideTextCommand{\命令}{编码}[参数个数]`
`[可省参数]{定义}`

2ε `\DeclareTextCommand{\命令}{编码}[参数个数]`
`[可省参数]{定义}`

其行为同 \providecommand 和 \newcommand 命令一样，只是只有当 编码 被激活时 \命令 具有这里给定的 定义 。因此对每种编码 \命令 就具有不同的定义。

2ε \DeclareTextSymbol{\ 命令 }{ 编码 }{位置 }

定义 \命令 为当 编码 被激活时字体中给定 位置 的字符。

2ε \DeclareTextAccent{\ 命令 }{编码 }{ 位置 }

定义 \命令 为一个重音命令，当 编码 被激活时使用位于给定 位置 处的字符作为重音符号。

2ε \DeclareTextComposite{\ 命令 }{ 编码 }{ 字母 }{ 位置 }

2ε \DeclareTextCompositeCommand{\ 命令 }{ 编码 }{ 字母 }{ 定义 }

定义命令 \ 命令及后接的单个 字母 的行为是显示在字体给定 位置 处的字符或者执行 定义 。在 T1 编码中这些声明是相当有用的，这时许多重音字母就是单独一个符号。因此 \‘{e} 在字母 e 上放尖重音（字符 19 ），而在 OT1 中它显示的是位置 233 处的字符。这个行为是如下得到的：

```
\DeclareTextAccent{\'}{OT1}{19}
\DeclareTextComposite{\'}{T1}{e}{233}
```

相应于给定编码的这条命令必须已经有定义，方法是用 \DeclareTextAccent 或者 \DeclareTextCommand ；对后一种情形，必须把它定义成具有单个参数值的命令。

上面所有定义命令都生成相应于指定编码中的新命令。如果这里定义的命令在其它编码中调用，会给出错误信息。可以为未指定的编码给出默认定义：

2ε \DeclareTextCommandDefault{\ 命令 }[参数个数][可省参数]{ 定义 }

2ε \ProvideTextCommandDefault{\ 命令 }[参数个数][可省参数]{ 定义 }

2ε \DeclareTextAccentDefault{\ 命令 }{ 编码 }

2ε \DeclareTextSymbolDefault{\ 命令 }{ 编码 }

这里前两条命令创建适用于所有未指定编码的默认定义，而后两条命令声明把哪种编码取为默认值。

注意： \DeclareTextCompositeCommand, \Provide... 和默认命令是于 1994 年 12 月 1 日加到 LaTeX 2ε中的。因此任何使用这些命令的文件应包含 \NeedsTeXFormat{LaTeX2e}[1994/12/01] 。

§8.6 输入代码 2ε

在不同的计算机系统上有可能直接向输入文件中输入一些 2.5.5–2.5.7 节的特殊符号，而且在用文本编辑器准备 LaTeX文件时这些符号会显示在屏幕上。有些 TeX实现版本确实可以把这些特殊符号转化成对应的命令。不幸的是这些特殊字符并没有标准代码，因此这样的文件是与系统有关的。例如，

特殊字母 Æ是 ISO-Latin1 代码方案的第 198 位字符，而在 IBM PC 扩展系统中却是第 146 位。

`inputenc` 软件包可以解决这个与系统有关的问题。其调用方式为

`\usepackage[` *编码* `]{inputenc}`

这样就依照代码方案 *编码* 来定义第 128–255 位输入字符的代码。*编码* 可能取的值有 `ascii`(没有扩展字符), `latin1`, `latin2` (ISO-Latin1 和 2 代码), `cp437`, `cp850`(IBM 代码页中第 437 页和 850 页) 以及 `applemac`(Apple Macintosh 代码) 。

在文件 *编码*`.def` 中用如下命令定义了扩展字符：

`\DeclareInputText{` *位置* `}{` *文本* `}` 或者

`\DeclareInputMath{` *位置* `}{` *数学* `}`

这样就把 *文本* 或 *数学* 的代码指定给给定 *位置* 的字符。例如， `latin1.def` 中包含

```
\DeclareInputText{198}{\AE}
```

这样就把输入的第 198 位字符翻译成 LaTeX命令 `\AE` 。

这也就是说为了得到 the rôle of the æther ， 可以直接输入 `the rôle of the æther` (不必用 `the r\^{o}le of the {ae}ther`) , 这样即使同样的输入，在不同的计算机上可能结果完全不一样。

§8.7 特殊符号的替代 2ε

浏览一下 CM (第 368 页) 和 DC 的字体布局方案，那你就会发现有很多符号，是不能直接用通常的命令输入的。例如， ¡ 和 ¿ 认为是输入文件 `!‘` 和 `?‘` 的连写，其它特殊符号类似。有一些符号对应的命令只能用在数学模式中。

利用连写生成符号的一个问题是有些特殊字体可能并不支持，即使在相应的位置上有这个符号。

任何符号都可以通过 `\symbol` 命令直接使用，这条命令要求布局位置做为参数值，但实际生成的字符与字体编码有关。一些 `\text...` 命令可以用来直接显示所有这些符号，而与当前编码无关。

命令	符号	替代
	连写	
\textemdash	—	---
\textendash	–	--
\textexclamdown	¡	!‘
\textquestiondown	¿	?‘
\textquotedblleft	“	‘‘
\textquotedblright	”	’’
\textquoteleft	‘	‘
\textquoteright	’	’
	数学符号	
\textbullet	•	$\bullet$
\textperiodcentered	·	$\cdot$
	杂类	
\textvisiblespace	␣	\verb*+ +

另外还有：

\textcompwordmark 表示分开连写的不可见字符

\textcircled{字符} 用圆把字符包起来，如 ⓧ 。

^{字符} 生成当前文本 (不是数学) 字体中的上标。

§8.8 其它标准文件

基本的 LaTeX格式包含了需要生成通常文档所需的绝大多数功能。然而还存在许多其它功能，在某些应用中是不可缺少的，可以用 \usepackage 命令 (3.1.2 节) 把它们包含进来。各个方面的用户编写个上百个这样的宏包，可能通过计算机网格得到这些宏包 (D.5 节)；还有一些宏包是由 LaTeX3 项目组的成员开发的，与标准安装版本一起发行；最后要指出，有些宏包就是标准安装版本的一部分。

§8.8.1 特殊文档

在标准安装版本中有许多特殊的‘文档’，这些文件的后缀为 .tex 。它们有：

small.tex[2.09] LaTeX2.09 中一个简短的示例文档；

sample.tex[2.09] LaTeX2.09 中一个较长的示例文档；

small2e.tex[2ε] LaTeX 2ε中一个简短的示例文档；

sample2e.tex[2ε] LaTeX 2ε中一个较长的示例文档；

lablst.tex (8.3.1 节) 显示出所有的交叉引用表的翻译及对应页码；它会交

互式地询问要列出的文档名称；

idx.tex (8.3.3 节) 显示出文档中所有的 \index 项，以两列方式同时显示出页码；它也要交互式地询问要列出的文档名称；

testpage.tex 测试文本在页面上的位置；这也是一个打印机驱动器的检测，以确保页边界正确，放缩显示恰当；在 LaTeX2.09 中，这个文档只适用于美国信纸，而在 LaTeX 2ε中，它会交互式地询问所希望的纸张尺寸；

nfssfont.tex [2ε] 这个文档显示字体表格，示例文本以及其它各种检测；它会交互式询问字体的名称； \help 命令会显示出所有可用的命令；

docstrip.tex [2ε] (D.3.1 节) 如其说这是一个文档，不如说是一个程序；它是从文档源文件中解开得到操作文件的基本工具；它不但去掉注释，而且会根据不同的选项加上替代编码。

§8.8.2 其它类 [2ε]

除了我们已经讨论的标准类 (book, report, article, letter) 外，在标准安装版本还有如下类：

proc 生成会议学报的照像制版拷贝；它是 article 类的两列模式的变体；它包含额外的一条命令 \copyrightspace，用以在第一列的底部留下显示版权信息的空间；(在 LaTeX2.09 中，这是 article 样式的一个选项，在 LaTeX 2ε中也可以这样用，但这只是为了兼容的需要)；

slides (8.10 节) 相应于 LaTeX2.09 中的 SliTeX，用来生成演讲材料；

ltxdoc (D.3.2 节) 生成类和宏包文件的文档。

§8.8.3 标准宏包

那些成为 LaTeX标准安装版本一部分的宏包，要考虑得相当广泛，这样我们就可以期望总是可以用它。而其它的宏包，即使是 LaTeX3 项目组的，也都只是供选择应用的。

有些标准宏包我们在其它地方已做了描述。标准宏包有：

alltt 给出了 alltt 环境，它非常类似于 verbatim 环境，只是 \, { 和 } 的行为与通常的一样；即在文本中的命令要被执行，而不是照字样显示；

bezier [2.09] (6.4.9 节) 提供了在 picture 环境中画二次曲线的能力；这个功能现在内植于 LaTeX 2ε中，只是为了兼容原来的文档，而提供了一个空文件；

doc [2ε] (D.3.2 节) 提供了建立 LaTeX宏包文档的特殊功能；

exscale [2ε] 允许以不同尺寸显示的数学符号也根据在 \documentclass 命令中的字体尺寸选项进行放缩；通常这些符号的尺寸是固定的，与文本的基本尺寸无关；

fontenc [2ε] (8.5.8 节，第 212 页) 通过上载 .def 文件来声明字体编码；编码

的名称列在选项中；

graphpap[2ε] (6.5.6 节) 给出了命令 \graphpaper，以生成 picture 环境中的坐标网格；

ifthen (7.3.5 节) 允许文本或命令的定义与各种条件的状态有关；我们可以测试 Boolean 值开关的状态，记数器或长度的值，或者某一命令的定义文本；

inputenc[2ε] (8.6 节) 允许在制作 LaTeX文档时特殊字符的输入采用的是与系统无关的编码方案；

latexsym[2ε] 用 lasy 字体 (E.5.1 节) 上载 11 个特殊的 LaTeX符号，这些符号现在并不是内植于 LaTeX 2ε中；在兼容模式中这个文件是自动读入的；

makeidx (8.4 节) 给出了当用 MakeIndex 程序生成关键词索引时可能用到的命令；

newlfont[2ε] 重定义了两字母声明，使得它们的行为与第一版的 NFSS中的一样；这些字体声明只改变一个属性；即 \bf 类似于 \bfseries，其余类似；

oldlfont[2ε] 定义两字母 (如 \bf) 声明的行为同在 LaTeX2.09 中文本和数学模式中的一样；这些声明现在并不内植于 LaTeX 2ε中，但所有的标准类中都有其它义；保留这个宏包只是处于兼容性的考虑；

pict2e[2ε] (6.5.6 节) 通过去掉直线长度、粗细、斜角以及圆的半径等方面的限制，来增强某些 picture 环境命令的功能；它利用了与驱动器有关的功能；

showidx (8.3.3 节) 使得 \index 项显示为边注；

shortvrb [2ε] 提供了 \MakeShortVerb 和 \DeleteShortVerb，这两条命令用来标记 (和删掉) 某一特定字符的功能与 \verb 功能，用以显示源文本。这个字符前缀反斜杠做为命令的参数。因此如果进行了如下调用：

```
\MakeShortVerb{\|}
```

那么 |\cmd{}| 等价于 \verb|\cmd{}|，结果都是 \cmd{}；

syntonly[2ε] 提供了命令 \syntaxonly，在导言中调用它，不把结果输入到 .dvi 文件中，但会给出错误和警告信息；这样会比正常的情形快四倍；当并不急于得到输出时，这种方法可以用来检验处理过程；

Tienc[2ε] 把 DC 字体编码设为标准；

tracefnt[2ε] 是一个检验 NFSS字体的对话式工具；在 \usepackage 命令中可以用下面这些选项控制其行为：

errorshow 不在屏幕上显示所有的警告和提示信息，但会送到抄本文件中；在屏幕上只显示错误信息；

warningshow 在屏幕上显示警告和错误信息；这种形式与完全没有用这个宏包很相像；

`infoshow` (默认选项) 在显示器上显示字体选择信息，通常这些信息只是写到抄本文件中；

`debugshow` 在每次改变字体，都会向抄本文件中写入相当多的信息；这样会得到一个很大的抄本文件；

`pausing` 把所有的警告信息都转化为错误信息，这样会使得处理暂时中止，等待用户的反馈；

`loading` 显示对外部字体的上载。

§8.8.4 附属软件

在可以找到 LaTeX的网络服务器 (D.5 节) 上有很多平行的目录，其中包含相当多的独立于标准安装版本之外的附属软件。这些软件都属于半正式状态。稍后在附录 D 中会描述其中一些软件对标准 LaTeX的功能扩展。

amslatex 是一组由美国数学学会提供的用于高级数学排版的宏包。

babel 提供了 LaTeX中的多语言支持，在 D.1.3 中对它进行了描述。

graphics 使得可以利用一些宏包，来进行一些图形处理：插入、旋转和处理外部图形和图示，以及彩色。

mfnfss 提供在 NFSS中安装其它 METAFONT(位图) 字体的宏包 (见下面的例子) 。

psnfss 提供在 NFSS中安装特定 PostScript 类型 1 字体的软件包，见 D.2.1 中的说明。

tool 是一组由 LaTeX3 项目组成员编写的实用宏包，在 D.3 节中有描述。

上面宏包中大多数都有完整的文档，在 D.3.2 中描述了如何利用文档化源代码的技术。把源文件在 LaTeX中进行处理，以得到手册和 / 或代码的详细解释，而运行安装文件可以从源文件中得到各种文件 (`.sty`, `.fd` 等等) 。

§8.8.5 捐献的宏包

并不是只有 LaTeX3 项目组的成员在开发 LaTeX类和宏包文件。在网络服务器 (D.5 节) 上有成千上万个已公开的个人宏包。其中大多数很有用的宏包在 The LaTeXCompanion (Goossens 等著， 1994 年) 一书中有介绍。至于时间方面，这些宏包大多数是为 LaTeX2.09 编写，但在 LaTeX 2_ε中功能也相当不错。当然我们希望作者们能很快把它们更新到新的版本。

可以编写自己的类和宏包的能力 (在附录 C 中详细介绍) 是 LaTeX的巨大能量之一。与其期望初始发明者把文本处理中每件事都做得很好，不如实际用户针立自己专业需要，寻找方法，并使得其它用户也可以应用自己的这一方法。

©TEXGuru, August 15, 1999

§8.9 各种 LaTeX文件

在 LaTeX的处理过程中要用到各种各样的文件：有些是从中读取信息，而有些是新创建的，存贮供下一次运行使用的信息。这些文件的名称都是由两部分组成：

基本名. 扩展名

对于每个 LaTeX文档，都有一个主文件，当调用 LaTeX程序时只需用它的基本名。其扩展名通常为 .tex，而其它文件可以用 \input 或 \include 命令来读入。(如果它们有另外的扩展名，那就必须显式地给出了。)

在 LaTeX运行期间创建的文件通常都具有与主文件相同的基本名，而只是扩展名不同，具体与文件的功能有关。如果主文件中包含 \include 命令，那么就会创建另外一个同被包含的 .tex 文件基本名一样的文件，扩展名为 .aux 。

其中有些文件每次运行 LaTeX时都会被创建，而其它的文件只有当调用了特定 LaTeX命令时才会出现，如 \tableofcontents 或 \makeindex 命令。对于 .aux 文件以及那些通过特殊 LaTeX命令生成的文件，都可以通过在导言中包含下面这条命令来抑制其生成：

\nofiles

如果文档频繁被订正和重处理，那么这条命令就相当有用，因为这时特殊文件中的信息还没有定案，因此不必管它。在 TeX层次上创建的文件总是会生成的。

其它的扩展名描述的文件用来包含格式信息、其它指令（编码）或数据库。这些文件名称中的基本名并不与任何文档有关。这种类型的一个例子是 article.cls 文件定义了 article 类。

本节余下部分是一个列表，包含 LaTeX文本扩展名和其在 LaTeX处理过程中作用的一个简短说明：

.aux 这是由 LaTeX生成的辅助文件，包含交叉引用信息以及其它生成目录表和其它列表所需的命令。除了每次用 \include 命令读入一个文件时会为每个文件创建一个 .aux 文件，也会为主文件创建（重建）一个 .aux 文件。

当用 LaTeX第一次处理一个文档时，并没有 .aux 文件，因此交叉引用命令并没有作用。这时会向屏幕上显示如下信息：

No file 主文件名.aux

.aux 文件是由 \begin{document} 命令读入的。当其被读入时，它们的名称会以下面的形式显示在终端屏幕上：

(主文件名.aux)(从属文件名 1.aux) ...

每次运行 LaTeX进行处理时都会生成新的 .aux 文件。相应于主文件的新

.aux 文件是从 \begin{document} 命令开始的，而那些相应于 \include 命令的文件是由相应的 \include 命令初始化的，当读完文件时被关闭。主 .aux 文件是由 \end{document} 命令关闭的。

可以在导言中给出命令 \nofiles 来不生成辅助性 .aux 文件。

.bbl 这个文件不是由 LaTeX创建的，而是由程序 BibTeX生成的。它与主文件有相同的基本名。实际上 BibTeX只是从 .aux 文件中读取信息。 .bbl 文件是由 \bibliography 命令读入到第二次 LaTeX处理过程中的，从而生成参考文献列表。

.bib 参考文献数据库的扩展名为 .bib 。 BibTeX读取这些文件以提取生成 .bbl 文件所需的信息。其基本名描述了数据库信息，而不必与任何文件相关。数据库的名称是在文本中用 \bibliography 命令包含进来的。

.blg 这是来自于 BibTeX运行时的抄本文件。它与输入文件有相同的基本名。

.bst 这是参考文献样式文件，它是 BibTeX的输入，用以确定参考文献的格式。 .bst 文件的名称在正文中是用 \bibliographystyle 命令包含进来的。

.cfg [2ε]有些类允许局部把纸张设置成其它尺寸，方法是把这些定义放到一个与类有相同基本名的 .cfg 文件中。 ltxdoc 类就是这种类型的类 (D.3.2 节) 。

.clo [2ε]这是 LaTeX 2ε中的类选项文件，由可能不只适用于一个类的特定选项的编码组成。并没有特殊命令来输入它。

.cls [2ε]这是 LaTeX 2ε类文件，定义文档的全局格式。它是由 \documentclass 命令读入的。 .cls 文件也可以用 \LoadClass 命令来输入。

.def [2ε]各种类型的额外定义文件都具有 .def 的扩展名。例如 T1enc.def (定义 T1 编码) 和 latex209.def (保持 LaTeX 2ε和 LaTeX2.09 中的兼容性) 。

.drv [2ε]不用直接处理 .dtx 文件以得到档案，而是处理 .drv 驱动器文件。档案通常是用 DocStrip 从 .dtx 文件中提取的。而现在这种方法的好处在于，当为了满足不同的页面尺寸，或者其它要求时，用户可以按手册的指导或者利用代码的解释来编辑这些驱动程序。 (在早先的版本中，这时得到档案文件的唯一方法。)

.dtx [2ε]在 LaTeX安装中有档案化的源文件。如果直接处理 .dtx 文件，就会显示出档案文件。这个文件是一个解释性的手册和 / 或对代码的详细解释。通过利用 DocStrip 处理 .dtx 文件可以得到类、宏包、 .ltx 、 .fd 或其它文件 (D.3.1 节) 。

.dvi 这是 TeX的输出文件，其由有格式的与输出打印机无关的被处理后的文本组成，因此称之为 '与设备无关' 文件。每次运行 TeX处理文件时都会生成这个文件，不能用 \nofiles 禁止生成它。当这个文件被关闭时，会在屏幕上显示如下信息：

Output written on 主文件名.dvi(n pages, $m$bytes).

当有错误出现时，没有向 .dvi 文件中写入任何信息，这时会显示如下信息：

No pages of output

必须用打印机驱动程序进一步处理 .dvi 文件，以把它转化为所用输出设置 (打印机) 上的特殊命令。

.fd [2ε] 这个文件包含 NFSS命令，这些命令把外部字体名称与字体属性关联起来 (8.5.8 节)，这个文件被称为字体定义文件。其基本名由编码和族标识组成，如 OT1cmr.fd 。当第一次使用给定族的一个字体时，就会自动地读入相应 .fd 文件 (如果存在的话) 。

.fdd [2ε] .fd 文件的档案化文件具有扩展名 .fdd ，它只是一种特殊类型的 .dtx 文件。为了得到档案文件，可以处理它，或者用 DocStrip 提取 .fd 文件。

.fmt 这个扩展名属于一种 TeX格式，它是用特殊的程序 initex(1.2.1 节) 创建的。为了快速上载在格式文件中以紧致编码存贮基本指令。LaTeX也就是TeX运行时使用格式 latex.fmt [2ε] 或者 lplain.fmt [2.09] 。(Plain TeX利用的格式文件是 plain.fmt 。)

.glo 只有当导言中有指令 \makeglossary 时，才会生成这个文件。它具有与主文件相同的基本名，整个文件是由源文本中的 \glossary 命令生成的 \glossaryentry 命令组成。如果在导言中有 \nofiles 命令，那么即使用了 \makeglossary ，也不会生成这个文件。

.gls 这个文件就是汇总中的 .ind 文件。它也是由 MakeIndex 程序生成的，以 .glo 文件做为输入。这里的输入和输入文件扩展名都必须显式给出。不存在标准的 LaTeX命令用来读取 .gls 文件。在 doc 宏包中用它来记录改变历史。

.idx 只有当导言中有指令 \makeindex 时才会生成这个文件。它与主文件有相同的基本名，只是由源文本中的 \index 命令生成的 \indexentry 命令组成。如果在导言有 \nofiles 命令，那么即使用了 \makeindex 命令，也不会生成这个文件。

.ilg 这是运行 MakeIndex 时的抄本文件。与输入文件有相同的基本名。

.ind 这个文件由 MakeIndex 程序生成，以 .idx 文件为输入。它是由 makeidx 软件包中定义的 \printindex 命令读入的。其由一个 theindex 环境组成，这是利用输入文件得到每个项的。

.ins [2ε] 为了便于在 .dtx 文件上运行 DocStrip ，就在安装文件中放一些必须的命令 (和选项) 。只要简单地 (利用 TeX或 LaTeX) 处理 .ins 文件，就可以从中提取出所需指令。

.ist 这是索引样式文件，由为 MakeIndex 提供的样式设置组成。基本名反映

出样式的性质，与任一文本文件没有关系。对于 doc 宏包，为指定的索引格式提供了 gind.ist 和 gglo.ist 文件。

.lis 在有些系统上这就是 .log 文件的替身。

.lof 这个文件由插图列表的信息组成。其行为与 .toc 文件完全一样，只是它是由 \listoffigures（不是 \tableofcontents ）命令打开的。

.log 这个文件由 TEX处理过程中的协议组成，也就是说显示在终端上的信息以及当出错时，只有对 TEX专家才有用的其它信息，都会写到这个文件中。即使用了 \nofiles，也会生成这个文件。当该文件被关闭时，会在终端上显示出如下信息：

```
Transcript written on 主文件名.log
```

.lot 这个文件由表格列表信息组成。它同 .toc 文件的行式完全一样，只是它是由 \listoftables 命令打开的，不是用的 \tableofcontents 命令。

.ltx [2ε] 在 LATEX 2ε中，某些输入给 initex 程序的文件具有扩展名 .ltx 。这些文件建立起生成 latex.fmt 格式的基础。这个文件的主文件名称为 latex.ltx 。

.sty 这是 LATEX 2ε中的宏包文件，用 \usepackage 调入。宏包是由定义新功能或修改已存在功能的其它命令组成。其中不能包含任何可显示的文本。也可以用命令 \RequirePackage 使它成为另一个宏包的输入。

在 LATEX2.09 中， .sty 文件既是样式选项文件 (宏包)，也是主样式文件 (与类等价)。这是两种截然不同的性质，但是它们却具有相同的扩展名。它们是由 \documentstyle 命令读入的。

.tex 包含用户源文本的文件应该具有扩展名 .tex 。每个 LATEX文档都至少要有一个 .tex 文件。如果只有一个 .tex 文件，那么它就也是决定其它所有文件基本名的主文件。如果在源文本中有 \input 或 \include 命令，那么在文档中就会有其它基本名不同，但后缀仍是 .tex 的文件。主文本就是调用 LATEX程序时用到的名称。其中有最高层次的 \input 命令，而且 \include 命令也只能出现在主文件中。

.toc 该文件由目录表信息组成。如果 .toc 文件存在，那么 \tableofcontents 命令就会从其中读入信息，然后输出目录表。同时，打开一个新的 .toc 文件，写入当前运行过程所有新的目录信息。该文件由 \end{document} 命令关闭，因此如果在 LATEX运行完毕之前通过其它方法终止的话，那么就会失去 .toc 文件。

只有文档中包含 \tableofcontents 命令，而且没有调用 \nofiles 命令时，才会生成 .toc 文件。它与主文件具有相同的基本名。

§8.10 报告材料的准备 (SliTeX)

LaTeX2.09 提供了与自己平行的称为 SliTeX的版本, 用来生成报告材料, 如彩色的幻灯片或视图。这个特殊的 LaTeX变体利用了不同的字体集, 这比字母通常的书籍样式更适合于进行投影。然而, 这需要另一种 TeX格式 (1.2.1 节) , 因此原来版本的 LaTeX把某个字体集显式地程序设计在格式文件中。因此用 SliTeX以示与通常的 LaTeX的区别。运行的是 SliTeX , 而不是 LaTeX , 实际上差别就在于前者用的是 splain 格式, 而后者用的是 lplain 格式来调用 TeX程序。

在 LaTeX 2ε中, 利用新字体选择方案, 在一个 LaTeX格式中替换所有的字体再不会有任何问题。因此 SliTeX再不只是与 LaTeX相似, 但独立之外的事物, 而且也不需要再给另外一个名称了。因此我们用 slides 类生成报告材料。在 LaTeX 2ε中该类同其它类的选择方式完全一样, 而在 LaTeX2.09 中, slides 样式只能在 SliTeX中选用, 而且事实上它也只接受这一种样式。

从此当我们用 SliTeX这一名称时, 它只是相对于 LaTeX2.09 而言的, 而它与 LaTeX 2ε中的 slides 类有相当大的差别。

§8.10.1 slides 类

我们所谓的报告材料, 就是那些要投影给观众看的幻灯片或者视图。我们当然可以利用通常的 LaTeX命令编写这些材料, 但是如果想得到适当的字体尺寸, 如何安排覆盖, 确保文本不会出现不期望的分页, 书籍样式的字体是否适合于投影方面存在一些问题。 LaTeX的类 slides 就是尝试解决这些问题的。

在这个类中用的是一组不同的字体集, 其中的小写字母比通常的字体要大一些, 而且基本尺寸更大。这当然会限制了一页 / 片上的文本量, 但是这对做报告材料就非常好。这里的文本应局限于关键词和简略的句子。满满的一页普通文本即使投影在屏幕上, 也不会有观众去看。

投影片也应该利用彩色。从这一点上来看, SliTeX就更应该更新了。当在八十年代中期人们创造出第一个 LaTeX时, 还没有理想价格的彩色打印机可以使用。一种变通的方法是利用彩色层, 也就是每种颜色的文本 (用黑白色) 单独打印出来, 然后把它复印到透明片上, 接着叠加起来就可以了。到了今天, 有更好的可以直接生成彩色投影片的方法。虽然对彩色的支持并不是 LaTeX 2ε核心的一部分, 但是宏包 color 却可以针对某些驱动器提供彩色命令 (D.3.3 节)。在 LaTeX 2ε中任何地方都可以用这个宏包, 并不是只能用在 slides 类中的。

即使没有彩色的功能, slides 在利用特殊字体和其它优势, 生成黑白视图方面也是非常有用的。通常 LaTeX的格式命令大多都可以使用, 只有分页

和章节命令例外。在下面一节中描述只属于 slides 的特殊功能。

在 LaTeX 2ε中并没有提供彩色层命令，除非你用的是兼容模式才可以用。

§8.10.2 制作幻灯片的环境

生成幻灯片的源文件在结构上同通常文档类似，只是这里用的是 slides 类：

```
2ε \documentclass{slides}
导言文本
\begin{document}
幻灯片文本
\end{document}
```

如果希望得到真正的彩色（不是黑白加彩色膜），那么可以如下在导言中加入 color 宏包：

```
2ε \usepackage[dvips]{color}
```

因为这个软件包并不是专为 slides 类定义的，因此我们在 D.3.3 节对它进行讲述。

在导言中可以包含某些全局性规定，如像通常那样改变纸张尺寸或选择纸张样式。在这里不能包含任何可显示的材料。

任何出现在 \begin{document} 命令之后，但在下面将要描述的特殊幻灯片环境之前的文件，都输入到一个没有编号的首页上，它位于所有幻灯片的前面。可以用它做为幻灯片的封面。

在组织幻灯片不同部分时，有三个环境可以使用：主幻灯片自身，可能用的覆盖，和对幻灯片的注解。

幻灯片

幻灯片或视图是用如下环境生成的：

```
2ε \begin{slide} 文本和命令 \end{slide}
```

slide 环境中的内容可以是任何所期望的文本。注意 slides 类使用的是自己字符字体集。正常尺寸中的标准字体大致相当于 LaTeX的 \LARGE 尺寸的 sans serif 字体 \sffamily 。可以使用 4.1 节中的通常字体命令和声明，以及第 4 章中的其它显示和列表环境。然而，在环境中不能出现分页，因为整个文本希望在一页 (幻灯片或视图) 上结束。后面的 slide 环境会顺序编号的。如果一页出现了溢出，给显示出警告信息。

可以使用来自于 color 环境的彩色命令，前提条件是这个软件包已经被调入。这样就会直接输出彩色，会不必需要以往 SliTeX中的彩色膜。

覆盖

所谓 *覆盖* 就是幻灯片的补充，它可以放在幻灯片上面，以弥补某些不

足。基本想法就是在报告时通过稍后才填上某些关键词，或者能够替换某些文本，来创造出某种悬念。

slides 类是利用 overlay 环境生成覆盖的，其作用方式同 slide 环境完全一样，只是编号是前面幻灯片的子编号。因此跟在第 6 号幻灯片后面的覆盖编号为 6-a, 6-b 等等。

2ε \begin{overlay} 文本和命令 \end{overlay}

生成视图的 slide 环境若要有覆盖，那它应在相应的 overlay 环境之前。幻灯片和覆盖环境中应包含相同的文本，只是某些部分用下面的声明以不可见的墨水显示：

\invisible 和 \visible

这两条命令的作用方式与字体声明一样。可以把它们放在大括号 {} 内以限制其作用范围，也可以用于整个环境。通常在 slide 环境中只会使某几个单词不可见，而在相应的 overlay 环境中，在开头处放上 \invisible 声明，而只有幻灯片中看不见的部分才是可见的。具体见 227 页的样例。

覆盖的一种应用就是提供替换文本。例如我们可以显示一张成本估计的表格，然后把图示放在覆盖上。通过交换这些覆盖，那么只要进行某些程序修改，所得到的新数可能适合于同样的表格。

注解

在报告的过程中，经常需要在进行实际的投影幻灯片时引用一些关键词或其它注解。slides 类提供了 note 环境，可以生成这样的给报告人的提示。

\begin{note} 文本 \end{note}

同 overlay 一样，注解也是相对于前面的幻灯片进行子编号的，只是这个用的是数字，不是字母。例如，接在第 4 号幻灯片后面，注解的编号为 4-1, 4-2, 等等。

§8.10.3 其它功能

页面样式

LaTeX命令 \pagestyle{样式} 也可以用在 slides 类中。可以使用下面这些 样式：

plain 所有的幻灯片、覆盖和注解的编号都是显示在右下角。

headings
同 plain 一样，只是如果选择了 clock 选项 (见下)，就会在注解的左下角显示一个时间标记。这也是默认的页面样式。

(headings 和 plain 样式在 SLiTeX中有很大的差别，前者这时还会绘制对齐标志。)

empty 打印出来的页面上没有页码 (或者对齐标志)。

在通常的 LaTeX中， \pagestyle 命令是在导言中调用的，从而对整个文档都有效。单个页面可以利用 \thispagestyle 命令给出不同的样式，这条命令只对当前一页有效。在上面环境内部不能用这条命令，而 \pagestyle 命令却可以用在 slide, overlay 和 note 环境外面。

时间注解

在 \documentclass 的选项中可以选择 clock ；它激活如下两条命令：

2ε \settime{ 秒数 }

2ε \addtime{ 秒数 }

这会以分钟为单位在注解的底部显示出时间。通过这种方法，可以提醒报告人在该处可以做怎样程度的讲述。时间标志的值是用上面那两条命令进行设置和增加的，这里的参数值以 秒 为单位。记时器的初始值为 0 。

有选择地处理幻灯片

如果幻灯片文件要处理很多的 slide 环境，而且有可能用户只想改变其中很少一部分幻灯片，这样就没有必要再全部重新输出一遍。这可以用下面的命令做到这一点：

\onlyslides{ 幻灯片清单 }

把这条命令放在导言中。这里的 幻灯片清单 表示升序的一组幻灯片编号，例如， 2，5，9-12，15 ，用来指定要处理的幻灯片或幻灯片范围。

不存在的幻灯片编号也可以出现在 幻灯片清单 中。比如说在幻灯片文件中只有 20 张幻灯片，那么 \onlyslides{1,18-999} 命令将会使得只有第 1 号和 18–20 号幻灯片被处理。属于这些幻灯片的覆盖也同时被输出。

最后，在导言中的命令

\onlynotes{ 注解清单 }

使得只有列在 注解清单 中的注解才会被输出。假设第 5 号幻灯片有三个与之相关联的 note 环境，那么 \onlynotes{5} 将会使得只有页码为 5-1, 5-2 和 5-3 的注解被输出。

\onlyslides 和 \onlynotes 命令同在 8.1.2 节中描述的 \includeonly 命令在功能上很相像。也可以同 8.1.3 节中描述的通过利用 \typein 命令一样，实现幻灯片和注解的交互式选择：

```
\typein[\slides]{Which slides to do?}
\onlyslides{\slides}
```

在处理过程中会在屏幕上显示出 'Which slides to do?' 的信息，这时用户就可以输入所期望处理的幻灯片。对 \onlynotes 也可以类似操作。

幻灯片文件样例

下面给出一个使用 slides 类的输入文件例子，其中包含了首页、覆盖和

注解，也使用了 clock 选项。同时在图 8.1 中显示了四页的输出结果。

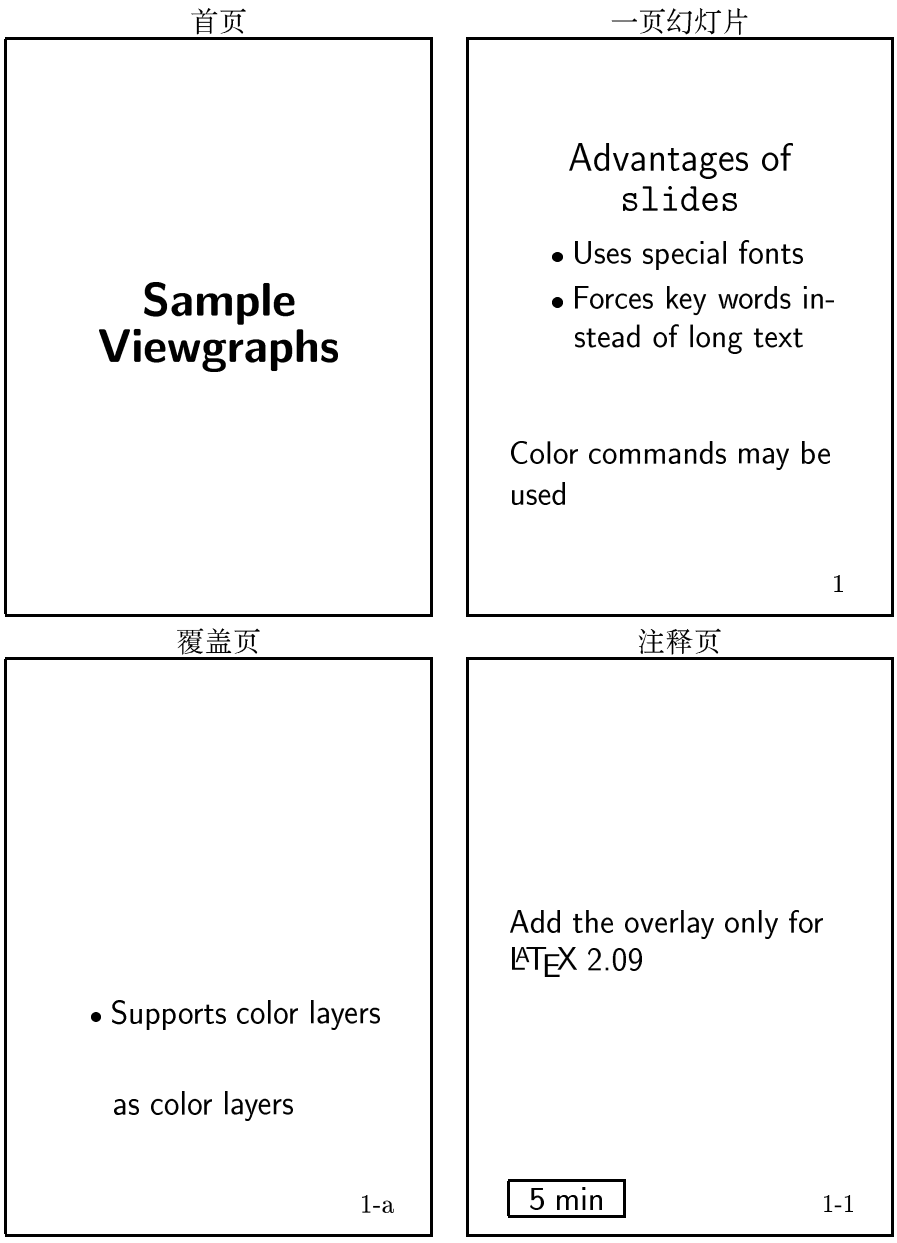


图 8.1: 幻灯片文件样例

```
\documentclass[a4paper, clock]{slides}
\begin{document}
\begin{center}\Large\bfseries
Sample Viewgraphs
```

```
\end{center}

\begin{slide}
\begin{center}\large Advantages of \texttt{slides}\end{center}

\begin{itemize}
\item Uses special fonts
\item Forces key words instead of long text
\invisible
\item Supports color layers
\visible
\end{itemize}

Color commands may be used {\invisible as color layers}

\end{slide}

\begin{overlay}
\invisible
\begin{center}\large Advantages of \texttt{slides}\end{center}

\begin{itemize}
\item Uses special fonts
\item Forces key words instead of long text
\visible
\item Supports color layers
\invisible
\end{itemize}

Color commands may be used {\visible as color layers}
\end{overlay}

\addtime{300}
\begin{note}
Add the overlay only for \LaTeX~2.09
\end{note}
```

©TEXGuru, August 15, 1999

```
\end{document}
```

©TEXGuru, August 15, 1999

第九章　错误消息

在制作长的 LaTeX文档时很容易出现错误。这里的错误可以是各种类型，从最简单的命令名称输入错误到忘记了某些命令必须配对，或者忘记了复杂命令的语法等等种类的错误都有可能发生。

在 LaTeX处理过程中的错误会在屏幕上显示出一长串消息，这些信息对初学者而言，是完全不可理解的。即便是高级用户在确定出错的具体地方时也会有一些困难。然而，这其中包含的关于基本结构信息，可以帮助有经验的 TeX使用者深入了解问题本质。

另外，错误消息中也包含对初学者而言相当有用的信息。本章的目的就是解释一下这些对非程序设计者有帮助的消息。

§9.1　错误消息的基本结构

错误消息有两个来源：一些来自于 LaTeX，另一些来自于基本的 TeX程序。由于 LaTeX是在更高层次上进行操作，因此 LaTeX消息之后通常接着 TeX消息。

§9.1.1　TeX错误消息

我们从一个简单的错误例子开始。

```
\documentclass{article}
\begin{document}
The last words appear in \txetbf{bold face}.
\end{document}
```

其中 `\textbf` 命令被误输入为 `\txetbf`。在处理过程中，LaTeX认为作者想调用的是 TeX命令 `\txetbf`。由于 LaTeX不认识 TeX命令，它只是把这条命令送给 TeX处理，在这里可以确定其指令系统中并没有这条命令。那么就会在屏幕上显示出下面的错误消息：

```
! Undefined control sequence.
l.3 The last words appear in \txetbf
                                    {bold face}.
?
```

程序运行到此处就停下来，等待用户给出反馈。这条消息即使是初学者也能理解。其中包含由惊叹号 `!` 开始的错误标识。这里的标识为 `! Undefined control sequence`，指的是未知的命令名称（控制序列）导致出错。接下来是两行文本，第一行前缀 `l.3`，指的是错误出现在输入文本的第 3 行。而错误本身就是在这一行最后那个符号中遇到的。而下面一行显示的是当遇到这

个错误时正在处理的输入行的其它部分，这里是单词 {bold face}。在继续向下处理之后，消息中最后一行的问号标志表示 TEX需要用户的反应。

再输入一个 ?，并回车，那么就会显示出如下信息：

```
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore next 1 to 9 tokens of input,
H for help, X to quit
?
```

下面就是用户可能给出的反应清单：

1. ⟨回车⟩：简单地输入回车键，让 TEX在经过一番按照预先设计好的规则，处理这个错误后，继续向下处理。在这种出现未知命令名称的情形中，处理错误的方式就是这样来忽略它。
2. S 卷动模式： TEX继续向下处理，当再次遇到错误时，还会在屏幕上显示出消息，但并不等用户做出反应。这就好像在所有后续错误中按的都是⟨回车⟩键。
3. R 运行模式： TEX如同 S 一样继续向下处理，但是即使遇到类似于在 \input 或 \include 命令没有文件名这样的错误也不停下来。
4. Q 安静模式：同 R 一样，只是不会再向屏幕上显示错误消息。然而这些消息要写到 .log 文件中。
5. I 插入：通过插入正确的文本来校正错误。TEX把从键盘上输入的文本行取代出错的地方，然后继续处理下去。对于这样的校正，实际上 .tex 文件中的原始文本并没有改变，还是需要用编辑器进行修改。输入 I\stop 会使得在 .dvi 文件的当前页上终止程序。
6. 1 ...：输入一个小于 100 的数，从而在后续文本中删掉许多字符和命令。程序还是会停下来等用户的反应。
7. H 帮助：在屏幕上显示出对问题的进一步考虑，它由相对于简短错误标志要多很多的信息组成，从而可能包含改正错误的有用提示。
8. X 退出：在该点停止执行 TEX处理过程。当前页并不出现在 .dvi 文件中。
9. E 编辑：同 X 一样，处理过程中止，并显示信息，说哪个文件的哪一行出现了错误。在有些实现版本中，可能自动调用编辑器，而且直接跳到出错行。

上面的反应字母既可以是大写的，也可以是小写的。只有按了⟨回车⟩键，反应才有作用。

在前面那个示例错误消息中，若按 H 或 h 以寻求帮助，会得到如下文本：

```
The control sequence at the end of the top line
```

```
of your error message was never \def'ed. If you have
misspelled it (e.g., `\hobx'), type `I' and the correct
spelling (e.g., `I\hbox'). Otherwise just continue
and I'll forget about whatever was undefined.

?
```

这里更详细地描述了错误情形：在上面一行尾部的命令名称是未知的；如果这只是一个输入错误，那么就用 `I` 输入正确的文本，这里是 `I\textbf` 。否则，按〈回车〉键，有错的命令被忽略。对这行的处理就如同是 `The last word appears in bold face` 。当然，这里并不会真的得到黑体。

TEX的错误消息结构总结如下：

每条错误消息都以错误指示开始，其第一行的开头有 `!` 号。所谓指示文本就是对问题的简要说明。接下来是一行或多行的输入文本。其中第一行的最后那个符号使得 TEX停下来，并显示错误消息。最后一行由接下来要处理的文本或命令组成。 TEX要等待用户的反应。如果这里的反应是寻求帮助的 `H` ，那么就是在屏幕上显示出更详细的说明，这可能会给出一些提示，而且TEX还会等待进一步的反应。

§9.1.2 LATEX的错误消息

在 LATEX2.09 与 LATEX 2_ε之间的一个主要差别就是它们的错误消息样子不同。在原来的版本中会显示出许多行难以理解的文本，以揭露出导致出错的深层内部代码，而新的版本只是显示出错误大概发生于其中的几行文本。在下面一节我们会讲到来自于 LATEX2.09 的错误消息例子，除此之后所有其它的例子都是针对于 LATEX 2_ε的，发行期为 `<1994/06/01>` 。

为了给出一个有 LATEX错误的文本例子，我们如下输入：

```
\documentclass{article}
\begin{document}
\begin{qoute}\slshape
  Text indented at both ends
\end{quote}
\end{document}
```

这里 `\begin{quote}` 调用中错误输入为 `qoute` 。当 LATEX处理这段文本时会显示如下错误消息：

```
! LaTeX Error: Environment qoute undefined

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
```

```
 ...

l.3 \begin{qoute}
                \slshape
?
```

这段消息的第一行就用一个简短的错误指示，说明这是 LaTeX自己发现的一个错误，这里是 `Environment qoute undefined` 。所有的 LaTeX错误都是以类似这样的一行开始的，接着告诉若想得到详细解释，请看 LaTeX手册（在本书的 9.3 节也可以看到这一介绍）。文本的第三行提示利用反应 H⟨回车⟩ 可以得到其它的说明。

有三个点 `...` 的那一行表示这里没有写出来的有关内部代码的许多行。在 LaTeX2.09 中会显示出这些内部代码行，即使在 LaTeX 2_ε中，也可能有选择地显示出一些。请见 9.1.3 节。

接下来的两行显示的是当发现错误时处理过程所停住的地方。同 TeX消息一样，文本的当前输入行也是断在出错的地方，前面部分放在第一行上，这里的断点为 `\begin{qoute}`，其它部分放在下面的行中。行指示符 `l.3`（小写字母 L，不是数字 1）表示是在输入文件的第 3 行上遇到错误的。

LaTeX现在等用户给出一个反应。输入 H⟨回车⟩ 会得到额外的信息：

```
Your command was ignored.
Type I <command> <return> to replace it with another command,
or <return> to continue without it.
?
```

这就是说以 `l.3` 开始的两行中上面这行的最后那条命令还没有进入处理过程中。因此可以用 只适用于当前处理过程 的 I\begin{quote}⟨回车⟩ 方法来校正。但是这里文件中的拼写错误仍然没改正过来，需要稍后再运行编辑器把它改过来。

然而，如果只是简单地输入 ⟨回车⟩ 键，处理过程就会继续下去，这样出错的 `\begin{qoute}` 就会被忽略，好像它在文件中不存在一样。这样当遇到 `\end{quote}` 命令时就会直接导致另一个错误，因此这里没有配对的 `\begin{quote}` 命令。这第二条错误消息的内容为：

```
! LaTeX Error: \begin{document} ended by \end{quote}

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.5 \end{quote}
```

```
?
```

这条消息中照样还是包含标准的 LaTeX声明，即有错误指示的第一行，接下来提醒参考手册，以及输入 H 可以得到帮助。在这种情形中的错误指示为

```
\begin{document} ended by \end{quote}
```

这是因为当 LaTeX遇到 \end{quote} 时，它会检查当前环境的名称。由于没有 \begin{quote}，因此匹配的 \begin 命令就是 \begin{document} 声明，从而得到一个匹配错误的 \begin...\end 。

最后两行显示的仍然是当发现错误时，处理已进行到了哪里。这里整个当前行都已被考虑进来了，因此另一行是空的。

在这时的反应 H⟨回车⟩ 会得到与第一个错误相同的消息。这里利用 I 进行改正并不会取得好效果，因为这里再不可能把不存在的 \begin{quote} 插入到环境文本的前面。输入 I\begin{quote} 只是用它代替 \end{quote}，这并没有解决实际问题。现在最好的反应就是输入 ⟨回车⟩，这样 \end{quote} 命令也被忽略，处理过程继续下去。

这样到现在为止，有错的 \begin{qoute} 和正确的 \end{quote} 命令都被去掉了，处理过程就如同在源文件中没有用 quote 环境一样。

如果是在 \end 处犯同样的拼写错误，而在 \begin 处则拼写正确，那么会得到如下错误消息：

```
! LaTeX Error: \begin{quote} on input line 3 ended by \end{qoute}

See the LaTeX manual or LaTeX companion for explanation.
Type  H <return>  for immediate hlp.
 ...

l.5 \end{qoute}

?
```

前面的解释对读者理解这条消息的内容应该是足够了。这里的错误指示为

```
\begin{quote} on input line 3 ended by \end{qoute}
```

最后两行文件表明问题出现在第 5 行，导致麻烦的命令是 \end{qoute}。现在明显可以采取的方法就是用 I\end{quote} 进行改正， H 消息也支持采取这一操作。然而，现在会在屏幕上显示出新的错误消息：

```
! Extra \endgroup
<recently read. \endgroup
l.5 \end{qoute}
```

```
?
```

这里除了以 `1.5` 开始的最后两行外，其它的都一点儿道理也没有。错误指示 `! Extra \endgroup` 好像没有任何意义。这是一个 TEX错误，而不是 LATEX错语，它对我们没有任何帮助。

此时不应责备受了挫折的用户。这里的反应是相当合理的，虽然它还是错的。只有在具备了丰富的经验后，我们才会知道此时最好的方法就是按 ⟨回车⟩ 。这样就会关掉 `quote` 环境，当然与 `\end{quote}` 相联系的任何特殊操作也随之去掉了。

此时的帮助消息是：

```
Things are pretty mixed up, but I think the worst is over.
```

这一消息至少是一个鼓励，读者不要灰心。最好的方法就是继续按 ⟨回车⟩ ，以结束这次运行。

这里我们给出如何选择反应的两条建议，一条非常具体，另一条很一般。具体的这条是：

> 如果出错的环境名位于 `\begin` 命令中，正确的改正错误的方法是
> `I\begin{正确的名称}`
> 如果拼写错误出现在 `\end` 命令中，那么最好的处理方法就是按 ⟨回车⟩ 。这样就会关闭这个环境，任何局部声明或定义也会终止作用。然而，如果 `\end` 命令执行了某命令，或显示出一些文本，那么这些结果也同时消失了。

一般性建议是：

> 如果借助于错误消息，用户知道了如何改正错误，那么可以用
> `I 修正的文本`
> 来进行。否则，用户可以按 ⟨回车⟩ ，等等看会出现什么结果。即使出现更古怪的 (TEX) 错误，用户也可以持续按 ⟨回车⟩ 键，直到处理过程被终结束。那么接下来的输出结果会指出错误原因。

当然也可以不按 ⟨回车⟩ 键，而是先输入 `S`, `R` 或 `Q` ，再按 ⟨回车⟩ ，以加速对错误的处理 (9.1.1 节) 。在这里，如果只是按 ⟨回车⟩ ，错误的命令并没有被忽略。 TEX在尝试猜测用户此处想进行的操作，从而进行一些改正。只有这一点行不通时， TEX才会完全忽略这条命令。例如，如果错误指示为

```
\begin{ 环境 } ended by \end{ 环境 }
```

这里至少有一条 `\begin` 命令中有一个非法的环境名称。这样就可以假设在 `\end` 命令中的环境名也是错误的。 LATEX就会尝试利用当前环境的名称来执行这条命令。

©TEXGuru, August 15, 1999

§9.1.3 来自于 LaTeX2.09 中的错误消息

为了进行对比，我们先把 233 页上输入例子中的 \documentclass 换为 \documentstyle，然后我们看看来自于 LaTeX2.09 中同样错误的消息结构。第一行消息显示为：

```
! LaTeX Error.   See LaTeX manual for explanation.
                 Type  H <return>  for immediate help.
! Environment qoute undefined.
\@latexerr ...diate help.}\errmessage {#1}

l.3 \begin{qoute}

?
```

这里主要的差别就是错误指示现在位于第三行上，而不是做为第一行上 LaTeX错误声明的一部分。另外一个差别就是第四行上的古怪文本，开头为 \@latexerr ... 。这实际上是生成这个消息的内部代码，只要简单地忽略它就可以了。

当没有做任何修改，只是按 ⟨回车⟩ 键，那么会得到第二条错误消息，它与 LaTeX 2_ε中的样子极端不一样：

```
LaTeX error.   See LaTeX manual for explanation.
               Type  H <return>  for immediate help.
! \begin{document} ended by \end{quote}.
\@latexerr ...diate help.}\errmessage {#1}

\@checkend ...urrenvir \else \@badend {#1}
                                       \fi
\end ...me end#1\endcsname \@checkend {#1}
                                       \expandafter ...
l.5 \end{quote}

?
```

在 \@latexerr... 和 l.5... 之间多出来的那些行对显示的是 \end 命令的内部代码，这些内容对通常 LaTeX用户来说，没有任何兴趣。它们只会令人感到混乱。如果你仍然用的是 LaTeX2.09，而且遇到了这样的错误消息，那么你就应该把这些行都忽略过去。除此之外，错误指示、用 l.n 开始的输入行、帮助消息和需要的反应都与 LaTeX 2_ε中完全一样。

有时候需要深入地观察错误消息，这一点对那些知道这些消息意味着什么的有经验 TeX专家特别有用。在这种情况中，可以利用下面的指令来在

LaTeX 2ε中补上去掉的代码行：

```
\setcounter{errorcontextlines}{ 数 }
```

这里的 数 就是在错误之处宏被解码的深度层数。在默认情形中，LaTeX 2ε为 数= −1，LaTeX2.09 为 数= 5。有经验的用户可以把它设为 5 或其它的值以获得额外的信息。

§9.1.4 来自于 TeX宏的错误消息

绝大多数的 TeX命令和实际用的 LaTeX命令都可以称为 TeX宏。它们就是原语命令的组合，从而具有一个新的命令名称，这样就可以整体调用它们。TeX宏在结构上类似于 LaTeX中用 \newcommand 命令定义的对象。可以向它传递多达 9 个的参数值，这也与 LaTeX命令一样。然而，生成宏的相应 TeX命令要比 \newcommand 更具一般性。

事实上，大约有 900 条 Plain TeX命令可以使用，其中只有 300 条是 原语，或者称为基本命令。其余 600 条都是宏。如果在宏中出现了错误，那么它里面的其它命令也可能受牵连。

为了明白起见，这里给出一个例子。命令 \centerline 就是如下定义的宏：

```
\def\centerline#1{\@@line{\hss#1\hss}}
```

这里 \@@line 本身还是一个宏，而 \hss 为一个 TeX原语，它是一个橡皮长度，可以无限伸展或收缩。为了避免误导读者进入太深层的 TeX命令，这里我们只是指出上面定义的宏就基本上等价于 LaTeX命令序列：

```
\newcommand{\centerline}[1]{\makebox[\textwidth][c]{#1}}
```

下面给出示例文本：

```
\documentclass{article}
\begin{document}
\centerline{This is an \invalid command}
\end{document}
```

其中在单词 invalid 前面加了一个 \，从而生成一个错误命令 \invalid。在 LaTeX处理它时，会给出如下 TeX错误消息：

```
! Undefined control sequence.
<argument> This is an \invalid
                                command
l.3 ...erline{This is an \invalid command}

?
```

这条消息现在应该很容易理解的。错误指示同 9.1.1 节例子中的一样：

```
! Undefined control sequence.
```

接下来两行文本说明是在处理完 \invalid‘命令’时发现错误的，接下来要读入的文本是单词 command 。同时，在上面那行开头部分的 <argument> 说明这部分文本是某个命令的参数值。

再下面两行文本类似的：错误出现在输入文本的第 3 行，而且整行文本（包括命令和参数值）都已经被读入并进行了处理。

§9.2 一些错误样例

§9.2.1 错误的传播

有不正确 \begin{qoute} 环境的例子表明如果只是给出一个简单地反应 ⟨回车⟩，尽管 \end{quote} 命令是正确的，也会导致第二个错误消息。这种错误地纠正导致进一步错误的现象要比想像中多很多。

现在看一些源文本：

```
\documentclass{article}
\begin{document}
\begin{itemie}
  \item This is the first point in the list
  \item And here comes the second
\end{itemize}
\end{document}
```

这部分文本中的唯一错误就是把环境名 itemize 错写为 itemie 。在 LaTeX处理时，首先生成与前面不正确的 quote 环境中一样的错误消息：

```
! LaTeX Error: Environment itemie undefined

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.3 \begin{itemie}
                \slshape
?
```

这时候，如果用户反应为 I \begin{itemize}，那么就会纠正这个错误，从而处理过程很顺利地结束。然而，如果输入的是 ⟨回车⟩，那么就会得到一条新的错误消息：

```
! LaTeX Error: Lonely \item--perhaps a missing list environment.

See the LaTeX manual or LaTeX Companion for explanation.
```

```
Type  H <return>  for immediate help.
 ...

l.4  \item T
           his is the first point in the list
?
```

之所以会出现这样的结果，是因为没有 \begin 命令，那么这就是在列表环境外面用了 \item，这样它没有任何意义。（实际上，它在这里是有意义的：它显示出了一条错误消息！）现在要插入不存在的 itemize 环境开始部分就太晚了。输入 H⟨回车⟩ 可以得到如下帮助：

```
Try typing  <return>  to proceed.
If that doesn't work, type  X <return>  to quit.
?
```

遵照这条建议，按了 ⟨回车⟩ 键，那么就会又得到同样的错误消息，但这次是出现在第 5 行上，相应于第二条 \item 命令。继续按 ⟨回车⟩，就会得到：

```
! LaTeX Error: \begin{document} ended by \end{itemize}

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.6 \end{itemize}

?
```

现在 itemize 环境总算结束了，但是因于其开头不正确，LaTeX抱怨遇到了不匹配的 \begin 和 \end 命令。在这里最后一次按 ⟨回车⟩，就会使处理继续进行下去。当然列表环境中的内容是不会有正确的格式的，但是其它部分的文档不会受到影响。

在这个例子中，源文本的一个错误生成了三个其它错误消息。这并不是不常见的。有些 LaTeX错误可以导致上百条后续错误。甚至有可能错误链永不会中止，处理过程不再向前进展。在这种情况中，没有别的办法了，只能终止程序。这可以在错误消息后面输入反应 I\stop 来做到。有可能需要给出几次这种反应，其才会发生作用。如果这还行不通，也就是说每次还是出现同样的错误消息，那么用反应 X⟨回车⟩ 就可以马上结束程序的运行。

用 I\stop 要比用 X 结束程序好，因为这样在输出中会包含最后一页的结果。这对于要推断错误来源时是非常有用的。

本节最后要告诉你的一条经验就是：即使遇到成群的错误，也不要惊慌！

坚持按 ⟨回车⟩ 键，继续处理下去。

如果按的是 S⟨回车⟩ 会在屏幕上得到同样的错误消息，但是其间不会再停下来等待用户的反应（ 9.1.1 节 ）。

§9.2.2 典型的严重错误

有的时候，用户可能忘记了 \documentclass 或 \begin{document} 命令，或者忘记了整个导言。当一个 LaTeX文件本来是要用 \input 或 \include 读入的，但错误地直接用 LaTeX程序调用它，后者这种错误更容易发生。如果一个文件中的文本为

```
This file has no preamble.
```

它被直接用 LaTeX进行处理，那么就会在屏幕上显示如下错误消息：

```
! LaTeX Error: Missing \begin{document}.

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.1 T
     his file have no preamble.
?
```

在输入 H⟨回车⟩ 后得到的相应帮助消息为：

```
You're in trouble here.  Try typing  <return>  to proceed.
If that doesn't work, type  X <return>  to quit.
?
```

从这条错误消息我们知道 LaTeX在读入第一个字母的时候就发现了一个错误。这里的帮助消息也不怎么样。在这里持续按 ⟨回车⟩ 是没有什么用的。反而我们应该用 X 或 E 马上停下来，因为这里得不到任何有用的东西了。

即使上面的例子文件中包含如下环境：

```
\begin{document}
   This file has no preamble.
\end{document}
```

也不可能得到一个正确的处理。现在的 TeX错误消息为：

```
! LaTeX Error: The font size command \normalsize is not defined:
there is probably something wrong with the class file.

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
```

```
 ...

l.1 \begin{document}

?
```

这里的错误指示相当古怪，因为在输入中从没有用 `\normalsize` 。只有消息中的最后两行还好理解。它们说明是在读入第 1 行的 `\begin{document}` 命令后发现错误的。实际上从文本的第一行就可以知道这个处理不会得到任何结果，因此在它前面并不存在不可缺少的 `\documentclass` 命令。

这时候，没有别的选择，就只能用 `X` 或 `E` 终止程序，因此继续处理下去也没有任何意义。

(这里之所以出现如此奇怪的错误指示，是因为有许多格式化参数需要用 `\begin{document}` 声明进行初始化，其中就包含标准字体。因此这里是内部调用了 `\normalfont` ，这条命令必须在类文件中进行定义，在这里并不存在类文件。而其它的初始化命令是在 LaTeX格式中定义的，因此它们不会导致错误。)

如果文档类的名称输入错误，例如：

```
\documentclass{fred}
```

那么就会得到如下错误消息：

```
! LaTeX Error: File ‘fred.cls’ not found.

Type X to quit or <RETURN> to proceed,
or enter new name. (Default extension: cls)

Enter file name:
```

这条消息应该是相当明白的：程序正在寻找一个叫 `fred.cls` 文件，但没有找到，因此它希望用户输入另外一个名称。如果新的文件具有所要求的扩展名 (`.cls`) ，那么就不需要显式给出。在现在情形中，任何标准 LaTeX类的名称，如 `article`, `report`, `book` 或 `letter` ，都可以做为输入，当然也可以输入其它可用类的名称。当然，文档应该是相应于该类而编写的。

只要系统找不到要读入的文件，就会显示同样的错误消息。会导致这个问题的命令有 `\input`, `\include` 或 `\usepackage` 。如果这个文件确实不存在，那么就可能是因为它并不位于 TeX寻找文件的地方。通常在安装时要建立一个系统参数，告诉 TeX在哪儿找文件。如果你的文件在其它地方，那就应该输入完整的文件名，即包含目录或路径标识符。

如果 LaTeX要求你输入的文件并不存在，或者是你根本就不知道的文件，那你就应该马上终止处理或者告诉 LaTeX忽略这个烦人的文件。这时，你还有

一个问题。LaTeX坚持要求得到一个合法的文本名。输入 X⟨回车⟩ 只是还会得到同样的消息，这时它会说它找不到文件 x.ext 。有些安装版本中提供了一个无内容的文本，叫 null.tex ，而且还可以用一个扩展软件包（ D.3.4 节），它提供了一组文件，依据标准反应字母而命名为 x.tex, e.tex, r.tex, h.tex, s.tex ，这样来模拟处理普通错误消息时的反应。

紧急停止：有时候我们会遇到一种情形，即使用 I\stop 或 X 也不能在出现错误后令程序终止。这时，就必须借助于操作系统的程序中断了。细节请看计算机中心或 PC 机的手册。

§9.2.3 数学错误

对数学公式命令的不正确应用，会导致多得令人吃惊的错误，这一点对没有经验的用户更是如此。更经常见到的在数学公式中的错误是无意识的错误，如忘记了一个右大括号 } ，或者没有及时切换回文本模式。另外一种常见的错误是在文本模式中用了只能出现在数学模式中的符号。下面我们给出几个典型的例子。

如果我们想得到：‘The price is $3.50 and the order number is type_sample’，那么可能会输入如下源文本：

```
The price is $3.50 and the order number is type_sample.
```

这块文本中包含两处错误，而且第一个错误取消了第二个错误： $ 符号是在文本中生成数学公式的数学模式切换开关（ 5.1 节）。在文本中生成真正的美元符号是输入 \$ 。然而，在这段示例文本中，单独的 $ 是切换进入数学模式，把其后的内容做为公式处理。然而，这里没有关闭的切换开关 $ ，当这段结束时 TeX才会注意到这一点。

```
! Missing $ inderted.
<inserted text>
                $
l.5

?
```

如果只是用⟨回车⟩做为反应，TeX会在该点插入一个 $ 。对于我们的示例文本，这就是在空行前面的文本结尾时插入的。也就是说从第一个 $ 到这段结束之间的文本都被当做正文公式处理了：

The price is 3.50$andtheordernumberistype_sample$.

从这个结果马上就可以看出用户犯的错误了。在靠近行的尾部，有一个字母 s 显示为下标了。这就是该例中的第二个错误。下划线字符 _ 只能用在数学模式中，这里应该输入为 _ 。然而，由于第一个 $ 已（不正确地）切换进入数学模式， _ 符号成为合法的了，这样其后字母 s 就成为下标了。

如果现在在文本中用 \$ 代替 $，那么 _ 符号就不是处于数学模式中，会生成如下错误消息：

```
! Missing $ inserted.
<inserted text>
                $
l.5 ...$3.50 and the order number is type_
                                          sample.
?
```

在这里按 ⟨回车⟩，告诉 TEX在数学命令 _ 前面插入 $，从而改正这个错误。这样处理过程就会继续下去，当到了当前段结束前的某处，TEX就会注意到并没有闭 $ 符号。这样就会显示与前面那种情形相同的错误消息。再按一次 ⟨回车⟩ 键，处理就会继续，得到如下结果：

The price is $3.50 and the order number is type$_s ample$.

在所有这三种情形中，都可以用 H 寻求更多的帮助，内容为

```
I've inserted a begin-math/end-math symbol since I think
you left one out. Proceed, with fingers crossed.
```

上面最后一句话就是我们所推荐的处理数学错误的方法：持续按 ⟨回车⟩ 或者 S，以使处理完成，这时再看打印的结果，以找到错误。

§9.2.4 来自于多文件的错误

如果文档被分成许多文件，每个用 \input 或者 \include 命令读入，那么错误消息中的行号相应于正在读的文件。利用反应 E ⟨回车⟩，将会调用编辑器程序，并打开文件，跳到发现错误的指定行。利用其它反应，那就必须单独使用编辑器，而且显式地给定出错文件名称。

通过查看处理消息或者 .log 文件，可以知道出错时正在处理的是哪个文件。当文本被打开供读取时，TEX会在屏幕或 .log 文件中写一个左小括号 (以及文件的名称。当文件被关闭时，会写右小括号) 。输出页码还是像通常那样显示在中括号内，这些内容都是同时显示在屏幕和写进 .log 文件中的。例如，如果屏幕上有如下处理消息：

```
..(sumfile.tex [1] [2] [3] (part1.tex [4] [5]) (part2.tex [6] [7]
! Undefined control sequence
l.999 \finish
?
```

那么可以有如下解释：一个叫 sumfile.tex 的文件正在被读取，而且在输出完第 1,2,3 页后，文本 part1.tex 被读取 (在文件 sumfile 中用 \input 和 \include 命令) ，这时输出第 4,5 页，然后关闭 part1.tex ，接着打开 part2.tex 文件做输入。在这个文件的第 999 行上发现一个错误。如果从键

盘上改正了这个错误，或者如果处理过程还是继续下去了，屏幕上会有如下消息：

```
[8] [9]) [10]
! Too many }'s
l.217 \em sample}
```

在第 9 页后面的右小括号表示 part2.tex 文件已被关闭。接下来在主文件 sumfile.tex 的第 10 页上又遇到一个错误，因为这里并没有) 表示该文件已被关闭。错误是在这个文件的第 217 行上遇到。

§9.3 LaTeX错误消息清单

下面列出 LaTeX错误消息，分成一般、宏包和字体错误消息三组。在每组中，按错误指示的字母顺序排序。然后给出可能导致该错误的描述和解决方法。

有许多消息是新出现在 LaTeX 2$_\varepsilon$中的，也有一些相比于 LaTeX2.09 版本稍有些改变。在 LaTeX 2$_\varepsilon$中的所有消息都有一个前缀词 `! LaTeX Error:` ，而在 2.09 版本中错误指示只是前缀惊叹号，并且位于第三行上。如果这是两者的唯一差别，那么我们就不显示 2.09 版本的内容。

当不同版本有其它不同时，这里还要用通常的标志来指明。

§9.3.1 一般 LaTeX错误消息

下述错误消息来源于那些没有调用字体选择或者定义，或者不涉及 LaTeX 程序类和宏包文件（附录 C ）的特殊功能时所出现的问题。

`! LaTeX Error: ... undefined.`

\renewcommand 或者 \renewenvironment 的参数值前面还没有定义。应在前面插入相应的 \new... 命令。

`! LaTeX Error: \< in mid line.`

tabbing 环境中的 \< 命令却出现在一行的中间。这条命令只能位于一行的开头（ 4.6.3 节 ）。

`! LaTeX Error: Bad \line or \vector argument.`

\line 或 \vector 命令的第一个参数值指定直线和箭头的倾角。这条消息说明所选择的角度非法。见 6.4.3 和 6.4.4 节。

`! Bad use of \\.` （ LaTeX2.09 ）

2.09 (这条消息在 LaTeX 2$_\varepsilon$中已经被 `There's no line here to end` 取代。)

`! LaTeX Error: Bad math environment delimiter.`

LaTeX在不正确的模式中遇到了数学切换命令：即数学模式中遇到了 \[或 \(，或者文本模式中遇到了 \] 或 \)。这是因为数学模式切换开关的配对不正确，或者某些大括号对 {...} 不匹配。

! LaTeX Error: \begin{...} on input line ... ended by \end{...}.
! \begin{...} ended by \end{...}. （ LaTeX2.09 ）

LaTeX遇到了一条 \end 命令，其没有与之对应的同名 \begin 命令。这可能是由于把环境名输入错了，或者前面漏掉了一个 \end 命令。避免这种错误的好方法就是一旦开始 \begin 命令，就马上输入 \end 命令，然后再在这两者之间插入实际的环境文本。对于长的嵌套环境，这一方法相当有用。这样也减少了在 \end 命令中输错环境名的机会。

! LaTeX Error: Can be used only in preamble.

有许多命令只能用在导言中。它们是： \documentclass, \usepackage, \nofiles, \includeonly, \makeindex, \makeglossary 以及其它一些。有些命令只有在类或宏包文件中才有意义，如 \ProvidesClass 和 \ProvidesPackage （ C.2.1 节），还有许多 \Declare.. 和 \Set.. 命令也只能用在导言中。如果在 \begin{document} 之后调用了这些命令，就会显示上面的消息。

! LaTeX Error: Command ... invalid in math mode.

2ε 在数学模式中调用了一条命令，而这条命令只有在文本模式中才有意义，如 \item 或者 \circle 。 LaTeX 2ε的字体命令 \itshape, \bfseries 等等在数学模式中也会导致这种错误，因为这时应该用的是数学字母表命令 \mathit, \mathbf 等。

! LaTeX Error: Command ... already defined.
! Command name ... already used. （ LaTeX2.09 ）

用户利用 \newenvironment, \newcommand, \newtheorem, \newsavebox, \newfont, \newlength, \newcounter 或 \DeclareMathAlphabet 等命令重定义已存在的结构。这时可以另选择一个不同的名称，或者当处理的是命令或环境时，可以用 \renew... 形式的命令。（注意当定义了一个名为 sample 的环境时，也同时创建了 \sample 和 \endsample 命令。）

! LaTeX Error: Command ... undefined in encoding ....

2ε 指定的命令只是已经相应于某一种 NFSS编码 (比如说是 OT1) 利用命令 \DeclareTextCommand 进行了定义，但是却在另一种编码（比如说是 T1 ）中使用它，因为在后一种编码中并没有相应的定义。

! LaTeX Error: Counter too large.

要以字母形式显示其值的记数器中的值大于 26 。

! LaTeX Error: Environment ... undefined.

LaTeX遇到了一条 \begin 命令，它不知道该环境名称。这可能是由于输入错误造成的。只要在处理时用反应 I 跟上正确的名称就可以校正这个错误。(这并没有改变源文件，错误仍然还在那儿。)

```
! LaTeX Error: File `...' not found
Type X to quit or <RETURN> to proceed,
or enter new name.  (Default extension: ...)
Enter file name:.
```

2ε 用输入命令上载一个文件，但是却没有找到这个文件。这里可以输入一个新的文件名，或者退出，或者不管它继续处理下去。如果给出了正确的文件名，而这个文件不存在，那么它可能是由于它不处在LaTeX寻找文件的地方。确保它在其中的一个正确目录中。此时用户有机会从键盘输入一个替代文件，或者改正输入错误。输入文件名，加上可省的扩展名，然后按 ⟨回车⟩。LaTeX给出一个与要求相同的可省扩展名。文件名的基本部分并没有默认值。

```
! LaTeX Error: Float(s) lost.
```

figure 或 table 环境或者 \marginpar 命令在竖直盒子 (\parbox 命令或者 minipage 环境) 中给出，或者这些结构出现在LaTeX命令中，而这条命令内部被竖直盒子调用，例如脚注。LaTeX是在输出一页后才会发现这个错误，因此实际的来源可能在指定有错文本前面好几行。这种错误的结果是有许多表格、插图或者边注丢失，但并不是它们导致这个错误。

```
! LaTeX Error: Illegal character in array arg.
```

tabular 或 array 环境中包含一个未知的列格式字符 (见 4.8.1 节)，或者在 \multicolumn 命令中做为第二个参数值的格式项有错。

```
! LaTeX Error: \include cannot be nested.
```

2ε 在一个已经要用 \include 命令读入的文件中又用了命令 \include。所有的 \include 命令都必须是在主文件中调用 (8.1.2 节)。

```
! LaTeX Error: LaTeX2e command ... in LaTeX 2.09 document.
```

2ε 如果正处在兼容模式 (即文档中用的是 \documentstyle 命令，而不是 \documentclass 命令)，却使用了 LaTeX 2ε 命令，就会出现这种错误。这些命令有 \LaTeXe, \usepackage, \ensuremath, lrbox 环境，还有相应于 \newcommand 和 \newenvironment 命令增强的一些新语法。出错的原因是在兼容模式中作者可以确保文档适用于 LaTeX2.09，这样可以把它送给只有原来版本的用户手中。

```
! LaTeX Error: Lonely \item--perhaps a missing list environment.
```

在列表环境 (4.3 节) 外面用了 \item 命令。这要么是因为把 \begin 中的环境名拼写错了，并且没有从键盘上改正它，要么是忘记了 \begin 命令。

`! LaTeX Error: Missing @-exp in array arg.`

`tabular` 或 `array` 环境的列格式化参数值包含了 `@` 符号, 但后面没有必须的放在大括号 { } 内的文本 (关于 `@-` 表达式见 4.8.1 节) , 或者在 `\multicolumn` 命令中的第二个参数里发生同样的错误。

`! LaTeX Error: Missing begin{document}.`

这要么是忘记输入 `\begin{document}` 命令，要么是在文档的导言中有可打印的文本。对于后者，这有可能是由于一个不合语法的声明造成的，例如命令的参数值没有放在大括号 { } 内，或者命令名前面没有 \ 。

`! LaTeX Error: Missing p-arg in array arg.`

`tabular` 或 `array` 环境中的列格式化参数值包含 `p` 符号，但没有必须与其一起出现的宽度定义 (4.8.1 节)，或者在 `\multicolumn` 命令的第二个参数值中发生同样的错误。

`! LaTeX Error: No counter '...' defined.`

`! No such counter. ( LaTeX2.09 )`

调用 `\setcounter` 或者 `\addtocounter` 命令，但所引用的记数器并不存在。这很可能是名称被输错了。如果发生错误时正在读 `.aux` 文件，而且记数器的名称肯定正确，那么 `\newcounter` 命令就可能是在导言外面。因此我们强烈建议总是把 `\newcounter` 命令放在导言中。(如果其它的 LaTeX记数器命令用的是一个没有定义的记数器名称，那么就会显示出一长串有趣的 TeX错误消息。)

`! No theorem environment '...' defined. ( LaTeX2.09 )`

2.09 (在 LaTeX 2_ε中，这条消息被 `No counter defined.` 代替)

在 `\newtheorem` 命令中给出了可省参数值，它指出另一个定理类型的声明, 以使两者共享同一个记数器, 但是这另一个定理结构并不存在(4.5 节)。这要么是写错了定理的名称，要么就是前面还没有定义。

`! LaTeX Error: No \title given.`

2ε 在给出 `\title` 声明之前就用了 `\maketitle` 命令。

`! LaTeX Error: Not in outer par mode.`

把 `figure` 或 `table` 环境或者 `\marginpar` 命令放在数学模式或者竖直盒子 (`\parbox` 命令或者 `minipage` 环境) 里面，会导致这种错误。对于第一种情形，通常是由于忘记了数学切换开关命令。

`! LaTeX Error: Page height already too large.`

2ε 用 `\enlargethispage` 命令来扩大页面的竖直尺寸，但是 LaTeX 认为这个尺寸太大了。

`! LaTeX Error: \pushtabs and \poptabs don't match.`

在 tabbing 环境中 \poptabs 命令与前面已经出现的 \pushtabs 命令在数目上不一样（ 4.6.4 节）。

`! LaTeX Error: Something's wrong--perhaps a missing \item.`

导致这个问题的原因很可能是由于列表环境（ list, enumerate 或者 description) 中的文本不是由 \item 命令开始的。如果在 thebibliography 环境中没有参数值 {样本标签}（ 4.3.6 节）时也会出现这种错误。

`! LaTeX Error: Suggested extra height (...) dangerously large.`

2ε 当用 \enlargethispage 命令扩大页面的竖直尺寸时，超出了 LaTeX认为合理的范围。

`! LaTeX Error: Tab overflow.`

在 tabbing 环境中最后一个 \= 命令超过了在 LaTeX可允许的制表位数目。

`! LaTeX Error: There's no line here to end.`

`! Bad use of \\.` （ LaTeX2.09 ）

在 \par 或者空行后面调用命令 \newline 或者 \\ ，因为在这里它们没有任何意义。如果要在这里插入额外竖直间距，应该用 \vspace 命令。

`! LaTeX Error: This may be a LaTeX bug.`

这条消息说明 LaTeX已完全糊涂了。这可能是由于前面出现错误时，用户的反应是按了 ⟨回车⟩。对于这种情形，应该用 I\stop, X 或者 E 把处理过程终止下来，纠正前面的错误。虽然有可能（但可能性不大）这确实是 LaTeX本身的一个漏洞。如果这是处理过程中的第一条错误，而文本看起来却令人满意，那么这个文件应该保存起来，交给计算中心，以供进一步研究。

`! LaTeX Error: Too deeply nested.`

把列表环境（ description, itemize, enumerate 或者 list ）彼此嵌套的层次太多。可以这样嵌套的最多层数与安装版本有关，但应至少为四层。

`! LaTeX Error: Too many columns in eqnarray environment.`

2ε 在 eqnarray 环境中每行只能有三列。这可能是你忘记用 \\ 开始新行了，或者在一行里多放了 & 。

`! LaTeX Error: Too many unprocessed floats.`

如果在一页上有太多的 \marginpar 命令时会出现这个错误。然而，更可能是 LaTeX维持了超过其能力的太多的 figure 和 table 浮动对象。这种情况可能发生在浮动对象被输出之前，提交了太多对象（ 6.6 节）。对于这种情况，就应该在文本中稍后加进最后这个对象。另外一种可能的原因是插图或者

表格在正常的页面中放不下来，而它适合于放在文本结尾的特殊浮动页或者 \clearpage 或 \cleardoublepage 命令后面。由于插图和表格的输出顺序与输入顺序是相同的，因此这样的一个对象就会阻塞整个队列。而 \clearpage 或 \cleardoublepage 命令就可以化解这种阻塞。

```
! LaTeX Error: Undefined tab position.
```

在 tabbing 环境中用 \>, \+, \- 或者 \< 命令把制表位移动到了一个不存在的地方（ 4.6 节）。

```
! LaTeX Error: \verb ended by end of line.
```

2ε 行内原文照排命令 \verb+...+ 中的文本超过一行文本。在 LaTeX2.09 中这是可以的，但在 LaTeX 2ε 中，这是禁止的，因为这样很容易捕捉一个常见的错误：少掉了终束字符。为了避免这种错误，确保在初始字符和结束字符之间的文本都在一个输入行上。

```
! LaTeX Error: \verb illegal in command argument.
```

2ε \verb 命令不能用在除 \index 和 \glossary 之外所有命令的参数值中。例如它就不能用在章节标题或者脚注中。

§9.3.2 LaTeX宏包错误

那些用来控制类和宏包文本的特殊程序设计命令（附录 C ）有它们自己的错误消息集。如果这些功能出现了个严重错误，那么很少有用户能解决这个问题，只有把它告诉文件的作者了。另一方面，有些错误是源于对所提供的文件和选项不正确使用造成的。

通常类和宏包遇到其感到奇怪的文本和内容时，会有其自己的错误或警告消息。在结出这些消息时，会有类或宏包的名称。例如，宏包 mypack 可能显示出如下错误：

```
Package mypack Error: cannot mix options ‘good’
(mypack)              and ‘bad’.
```

当按了 H 时，应该会给出帮助消息。显然我们不可能在这里解释这些错误 (警告) 消息，因为它们完全与所考虑的宏包有关。

```
! LaTeX Error: \LoadClass in package file.
```

2ε 在宏包文件中调用了 \LoadClass 命令，这是不允许的。只能从一个类文件调用另一个类文件。

```
! LaTeX Error: Option clash for package ....
```

2ε 指定的宏包被重复调用，而且两次所用的选项不同。宏包文件只能被调用一次，第二次调用被忽略。因此，如果 \usepackage 或 \RequirePackage 命令调用的是同一个宏包，但是选项不同，这就会导致冲突。此时需要尽力安排一致的选项。按 H 可以得到帮助，它会显示出这两组冲突选项。

`! LaTeX Error: \RequirePackage or \LoadClass in Options Section.`

2ε 这两条命令不能位于由 `\DeclareOption` 命令生成选项的类或宏包定义中。选项应该设置一些标志或者其它指示，以供在调用这里所考虑的命令之前进行测试。

`! LaTeX Error: This file needs format '...' but this is '...'`

2ε `\NeedsTeXFormat` 命令指定了一个不同于正在用的格式。所谓格式，就是预先保存起来的指令集，用来确定 TeX运行的类型。对于 LaTeX 2ε，指定的格式名称必须是 `LaTeX2e` 。错误所指的意思是用当前的格式根本不能处理这个文件。

`! LaTeX Error: Two \documentclass or \documentstyle commands`

2ε 一个文档中只能有一条 `\documentclass` 或者 `\documentstyle` 命令。如果在主文档中确实只看到一条这样的命令,那么检查一下其它上载的文件,确保其中没有与之冲突的第二条命令。

`! LaTeX Error: Two \LoadClass commands.`

2ε 在类文件中包含多于一条的 `\LoadClass` 命令，而这是不允许的。所用的类文件一定在编写上有问题。

`! LaTeX Error: Unknown option '...' for package '...'.`

2ε 在 `\usepackage` 命令中指定了一个选项，但是相应的宏包中并没有定义这个选项。查看一下宏包的指令，以及有没有输入错误。

`! LaTeX Error: \usepackage before \documentclass.`

2ε `\usepackage` 命令不能出现在 `\documentclass` 的前面。必须在所有宏件包之前上载类文件。

§9.3.3 LaTeX字体错误

下面的错误可能在用新字体选择框架(8.5 节)定义或选择字体时出现。这些消息中有些会说字体还没有正确地建立，那么这可能是因为字体描述文件已崩溃或者有错。对于这两种情形，系统管理员必须找一组正确的文件,进行重新安装。

`! LaTeX Error: ... allowed only in math mode.`

2ε 类似于 `\mathbf` 这样的数学字母表命令用在了文本模式中。这可能是忘记了一个 `$` 。

`! LaTeX Error: Command '...' not defined as a math alphabet.`

2ε `\Set..` 或 `\Declare..` 命令以数学字母表名称做为参数，但是当用这两个命令时，指定了一个不存在的字母表的名称。数学字母表名称在使用之前，必须先用 `\DeclareMathVersion` 命令进行声明。

```
! LaTeX Error: Command ... not provided in base LaTeX2e.
```

2ε 有一些符号是基本 LaTeX2.09 的一部分，但 TeX中并不存在，那么它们不再自动包含在 LaTeX 2ε中。这时需要利用软件包 latexsym 。

```
! LaTeX Error: \DeclareTextComposite used on inappropriate
                command ....
```

2ε \DeclareTextComposite（ 8.5.9 节）用来重定义特定编码中一个已存在的重音命令，以显示单个字符。如果这个重音命令并不存在，那就会给出这条错误。

```
! LaTeX Error: Encoding scheme `...' unknown.
```

2ε 进行了一个使用不存在的编码方案的字体声明或者选择命令。这很可能是由于输入错误导致的。

```
! LaTeX Error: Font family `...+...' unknown.
```

2ε 对字体编码和族组合调用了 \DeclareFontShape 命令，但是这种组合之前并没有用 \DeclareFontFamily 命令进行声明（ 8.5.8 节）。

```
! LaTeX Error: Font ... not found.
```

2ε 找不到具有指定属性的字体，也无法进行适当的替换。这时会用由 \DeclareErrorFont 定义的字体。

```
! LaTeX Error: Math alphabet identifier ... is undefined
               in math version `...'.
```

2ε 调用了一种对当前数学变体还没有定义的数学字母表 (8.5.5 节) 。这就是说已经用 \DeclareMathAlphabet 创建了字母表，其形状属性为空，但没有用 \SetMathAlphabet 声明把它定义成可以在选定数学变体中可用。

```
! LaTeX Error: Math version `...' is not defined.
```

2ε \Set.. 或 \Declare.. 命令以数学变体名称做为参数，但是当用这两个命令时，指定的一个不存在的字母变体名称。数学变体名称在使用之前，必须先用 \DeclareMathVersion 命令进行声明。

```
! LaTeX Error: *** NFSS release 1 command ... found
               *** Recovery not possible. Use ....
```

2ε 用了第一版 NFSS中的一条命令，而这条命令现在不再可用了。用建议的替换命令。

```
! LaTeX Error: Not a command name: `...'.
```

2ε \DeclareMathAccent 命令的第一个参数值必须是一个命令名称，如 \acute 。这时很可能是忘记输入反斜杠 \ 了。

```
! LaTeX Error: Symbol font `...' not defined.
```

2ε在 \Set.. 或 \Declare.. 命令中指定了一个不存在的符号字体名称，这两条命令需要符号字体名称做为参数值。字体名称在做为符号字体使用之前，必须用 \DeclareSymbolFont 进行声明。

```
! LaTeX Error: The font size command \normalsize is not defined:
               there is probably something wrong with class file.
```

2ε在类文件中必须定义 \normalsize 命令，它表示文档中文本的标准尺寸。这是根据 LATEX2.09 中的经验进行的一个修改。如果类文件中没有进行这个定义，而只是定义了 \@normalsize ，那就必须进行修正。如果没有 \documentclass 命令，也会出现这条消息，因为空的类文件当然是错误的。

```
! LaTeX Error: This NFSS system isn't set up properly.
```

2ε在 .fd 文件中的字体描述有错误，或者没有由 \DeclareErrorFont 声明的合法字体。这是一个严重的错误，应向系统管理员报告这一信息。

```
! LaTeX Error: Too many math alphabets used in version ....
```

2ε只可能有多达 16 个的数学字母表，这是由 TEX本身设定的限度。其它数学字母表定义都会被忽略。

```
! LaTeX Error: Unknown symbol font `...'.
```

2ε用一个不存在的符号字体名称做为了 \DeclareSymbolFontAlphabet 命令的参数值。这里的字体名称必须在使用前用 \DeclareSymbolFont 进行声明。

§9.4 TEX错误消息

本节列出一些比较常见的 TEX错误消息，按错误指示进行字母排序，而且同时给出一个导致该错误的简短描述。这里每一个前面只是以惊叹号开始。由于这些消息是由 TEX生成的，由此在 LATEX2.09 和 LATEX 2ε之间没有任何差别。

```
! Counter too large.
```

做为 TEX错误，这个消息是说当脚注标记用的是字母或符号时，记数器的值超过了 26 或 9 。如果在标题上有很多 \thanks 命令时也可能出现这个错误。

```
! Double subscript.
```

在数学公式中一个变量后面有两个下标，例如， x_2_3 或 x_{2}_{3} 。而要得到 x_{2_3} 的正确方法是 x_{2_3} 或者 x_{2_{3}} (5.2.2 节)。

```
! Double supscript.
```

在数学公式中一个变量后面有两个上标，例如， `x^2^3` 或 `x^{2}^{3}` 。而要得到 x^{2^3} 的正确方法是 `x^{2^3}` 或者 `x^{2^{3}}` （ 5.2.2 节 ）。

`! Extra alignment tab has been changed to \cr.`

在 `tabular` 或 `array` 环境中的一行里包含比列定义时还要多的 `&` 命令。这个错误可能是由于忘记在前一行尾部输入 `\\` 造成的。

`! Extra }, or forgotten $.`

在数学模式中，要么忘记输入左大括号 `}` ，要么就是多输入了右大括号 `}` 。另外一种可能就是忘记输入数学模式切换命令，如 `$`, `\[` ，或者 `\(` 。

`! Font ... not loaded: Not enough room left.`

文本处理中要求上载的字符字体超过了由于内存限制 TEX所能控制的数量。如果文档某一部分需要不同的字体，那你可以把它分成几部分，分别进行处理。

`! I can't find file '...'.`

这条消息实际上已经被 LATEX 2ε的等价消息 `File ... not found` 取代，而且作用也非常相似。要读入指定的文件，但是在搜索目录中却找不到这个文件。这时用户有机会输入另一个替换文件名称（或者修改输入错误）。在显示出错误指示后， TEX会问另一个要读入的文件名称：

`Please type another input file name:`

这就是等你再输入一个新名称，并按〈回车〉键。这与 LATEX 2ε消息有一点儿不同，默认的文件扩展名总是 `.tex` ，并不要求输入扩展名。

`! Illegal parameter number in definition of ....`

这个错误很可能是由于 `\newcommand`, `\newenvironment`, `\renewcommand`, `\renewenvironment` 命令造成的，即其中的替换字符 `#` 用法不对。这个字符在定义文件中只能以 `#`n 形式出现，这里的 n 就是介于 1 到在命令中指定的参数值个数之间的值。而要在文本中用到 `#` 字符，就必须用 `\#` 。如果在后一个参数值 {end_def} 中用到了替换字符（ 7.4 节），也有可能导致这个错误。

`! Illegal unit of measure (pt inserted).`

如果这条消息恰好是出现在

`! Missing number, treated as zero.`

错误消息的后面，那么问题就是由于这前一个错误造成的（见下面）。否则，错误就在于 TEX此时要求一个长度定义，但是只给出一个数，而没有长度单位。这通常是由于要把长度设为零，只给出了 `0` ，而不是 `0pt` 或 `0mm` 。如果是这种情形，那么用反应〈回车〉，就会得到正确的结果，因此对于零值，任何长度都得到同样的长度定义。如果完全忘记输入长度单位时，也可能出现这种错误。

©TEXGuru, August 15, 1999

```
! Misplaced alignment tab character &.
```

不能把单个字符命令 & 放在 tabular 或 array 环境外面的普通文本中。可能的动机是本来想显示 &，那么就应该输入为 \&。在响应错误消息时，可以用 I\& 来订正这一点。

```
! Missing control sequence inserted.
```

这个错误很可能是由于在 \newcommand, \renewcommand, \length，或 \newsavebox 命令的第一个参数值前面漏掉了反斜杠 \。只要用 ⟨回车⟩ 做为反应，就会正常地结束处理过程，因为 TEX假定你是少掉了反斜杠，从而替你补上了。

```
! Missing number, treated as zero.
```

这个错误很可能来源于需要一个数值或长度做为参数值的命令，而却少掉了这个参数值。或者也有可能是一条命令有可省参数值，而其后所接的文本开头就是 [，这也可能造成这种错误。最后一种可能是在长度或者 \value 命令前面有 \protect 命令。

```
! Missing { inserted.
! Missing } inserted.
```

当显示这两条消息之一的时候，就表明 TEX已完全糊涂了。这时显示出来的行号往往并不是出现错误的地方，这是由于少掉个左大括号或右大括号造成的。如果错误不是很明显，那就按 ⟨回车⟩ 让处理继续下去，从打印出来的结果中推断错误的地方。

```
! Missing $ inserted.
```

很可能是在普通文本中用了一个只能出现在数学模式中的符号或命令而造成这种错误。查看一下在第 5 章中列出的那些除非有特别说明，否则只能用在数学模式中的命令。如果在数学公式中插入了一个 \mbox 命令，那么其参数值就暂时从数学模式切换到文本模式。如果在数学公式中出现一个表示新段的空行，或者在公式的尾部没有必要的 $ 符号也会出现这种错误。

```
! Not a letter.
```

\hyphenation 命令的单词列表中包含一个不被认为是字母的字符，例如 \'{e} 这样的重音命令。这样的单词可以通过显式插入可选连字符来给出建议断点（ 3.6.1 和 3.6.2 节）。

```
! Paragraph ended before ... was complete.
```

命令的参数值中包含空行或者 \par 命令，而这是不允许的。很可能是漏掉了右大括号 }。

```
! \scriptfont ... is undefined (character ...).
! \scriptscriptfont ... is undefined (character ...).
```

`! \textfont ... is undefined (character ...).`

当在 LaTeX2.09 中数学公式使用了一个符号，其字体并不是针对于数学模式而设计的，如 \sf(sans serif)，就会显示这种消息。这样的字符字体必须用 \load{尺寸}{样式} 命令才可以使用，如 \load{\normalsize}{\sf} 。如果这样的公式出现在脚注中，就必须事先给出 \load{\footnotesize}{\sf} 命令。在 LaTeX 2_ε中，只要用的是 \mathsf 这样的数学字母表命令，就不会再出现这种问题。

`! TeX capacity exceeded, sorry [...].`

TeX在进行文本处理时，会在计算机内存中建立起各种存贮缓冲区。当这些缓冲区有的已满了，从而不能再用时，就会显示这条消息。缓冲区的名称及其最大尺寸显示在错误指示的中括号内。显示过这条消息， TeX的处理被终止。这个问题的来源几乎从不会是内存不足，无论处理的文本有多复杂，而多是文本自身的错误。在 9.6 节描述的方法可以用来探察真正的错误原因。

下面描述一下各种缓冲区，来帮助你确定是否真的是因为 TeX分配缓冲区能力太小，以及应该怎样校正这个错误。

`buffer size` 这个问题可能是由于章节命令、\caption 、\addcontentsline 或 \addtocontents 命令中的参数值太长造成的。这条消息通常只有当 \end{document} 命令被处理时才会出现，但是在遇到 \tableofcontents, \listoffigures 或 \listoftables 时也有可能出现。避免这种错误的方法就是对太长的标题文本，利用表示短标题的可省参数值 (3.3.3 和 6.6.4 节) 。因为这样的太长的项出现在目录表也是很烦人的，总是要把它缩短。一旦在源文本中进行了这种修正，必须删除已有的 LaTeX .aux 文件。

在 PC 机上如果用的是字处理程序，而不是文本编辑器来生成源文本，有可能出现这个问题。因为有的这样程序把整个段落放在单独一行上，即使显示在屏幕上是分成几行的。

`exception dictionary` 用 \hyphenation 命令输入的连字符例外清单太大了。那些使用频率很小的单词应该去掉，而用 - 进行显式地指定单词断点。

`hash size` 在源文件中包含了太多的命令定义或者交叉引用标志。这并不是说源文件需要所有这些命令，而可能是由于用户开发出一个很大的自己专用的命令集，并把它存贮在一个文件中，读入到每个文档中，而不管用不用到这些命令。

`input stack size` 这个缓冲区的溢出可能是命令定义中出现了错误。例如，如果用下面这样的命令定义：

```
\newcommand{\com}{One more \com}
```

就会得到 One more {One more {...One more \com}...}}，永远这样持续下去，因为它自己调用了自己。实际上它并不会永远持续下去，直至

缓冲区变满。

`main memory size` 这个缓冲区包含当前正在处理页面中的文本。如果调用了一个递归定义的命令，它也会溢出。然而，更常见的原因是： (1) 在一页上定义了很多相当复杂的命令； (2) 在一页上有太多的 `\index` 或 `\glossary` 命令； (3) 页面自身太复杂，以致于充满分配的缓冲区空间。

对于前两种情形的解决方法是非常简单的：减少该页的命令定义和 / 或 `\index` 与 `\glossary` 命令。对于第三种情形，原因可能是由于有太长的 `tabbing`, texttttabular, `array` 或 `picture` 环境，或者等待输出的浮动对象 (插图或表格) 发生了堵塞。

为了验证内存溢出是否确实是由于复杂的页面造成的，那么在出现溢出的地方前面加上一条 `\clearpage` 命令。如果在接下来的处理运行中不再出现这种错误，那么就可以确定这真的是由于该页面对 TEX而言确实太复杂了。然而，如果仍然有溢出，那么就可能是由于源文本中的错误造成的。必要的话，可以用 9.6 节给出的方法确定错误的位置。

如果确实是由于对 TEX处理而言，页面太复杂了，那就必须想法进行简化。然而要注意到在输出页面之前要把最近的那整个段落处理完，即使分页点可能出现在这个段落的开始部分。利用 `\newpage` 命令，可能会解决这个问题，因此应该在进行烦人的文本重组织之前，尝试一下这条命令。如果错误是由于浮动对象阻塞造成的，那么把浮动对象移到后面文本中，或者改变一下定位参数值 (6.6.1 节)，可能会解决这个问题。如果整个文本还没有结束，那么现在可以尝试用 `\clearpage`，以清除被阻塞的浮动对象，然后在完成文本后给出浮动对象的一个更合理的顺序。

`pool size` 很可能是由于有太多的命令定义和 / 或标签, 或者是其名称太长。缩短一下名称长度看看。如果在诸如 `\setcounter`、`\newenvironment` 或 `\newtheorem` 等命令中的记数器命令参数值漏掉了右大括号 } 时也会出现这种错误。

`save size` 当命令、环境和声明的范围嵌套层数太多时, 这个缓冲区会溢出。例如，`\multiput` 命令的参数值中包含 `picture` 环境，而这个环境中又在另一个 `\multiput` 命令中用了 `\footnotesize` 声明，以此类推。这样的嵌套必须简化，除非实际导致问题的原因是因为忘记了右大括号 }，而使得结构变得相当复杂。

`! Text line contains an invalid character.`

在源文本中包含一个特殊符号，TEX无法识别。这可能是编辑器本身的问题，是它插入了这个多余的字符。如果检查源文本时无法看到这个字符，那只有寻求计算中心的帮助了。

`! Undefined control sequence.`

每个 TEX用户都会在经常遇到这条错误消息。它通常是由于不正确输入命令名称造成的。可以在处理过程中用 `I` 及正确的命令名称，再加上 ⟨回车⟩ 进行更正。如果正确地输入了一个 LATEX命令，那就可能是由于在不正确的环境中调用造成的，因为在这个环境中这条命令是不允许使用的（即还没有定义）。

`! Use of ... doesn't match its definition.`

如果 '...' 是一个 LATEX命令的名称，那它就很可能是 6.3 和 6.4 节中的图形环境中的命令调用时参数值语法不对。如果名称是 `\@array`，那么在 `tabular` 或 `array` 环境中有一个有错的 `@-` 表达式（ 4.8.1 节）。这可能是由于在 `@-` 表达式中有一个脆弱命令，但没有前缀 `\protect` 命令。

`! You can't use 'macro parameter #' in ... mode.`

在普通文本中用了特殊符号 `#`。这可以是需要用 `\#` 以生成 # 自身。可以在处理过程用 `I\#` 加 ⟨回车⟩ 纠正这个错误。

§9.5 警告

TEX和 LATEX错误都会使得处理过程暂时停下来，并等待用户的反应，或者程序完全中止。而另一方面，警告只是提示用户输出结果中可能有些不对头，需要进一步修正。警告消息是连同其所在的页码一起显示在屏幕上，但不是会使处理过程停下来。当然这些消息也会同时写到 `.log` 文件中，这样处理结束后可以查看该文件，甚至打印出来。 TEX或者 LATEX都有可能导致警告。

§9.5.1 普通 LATEX警告

LATEX的警告是以 'LaTeX Warning:' 为标志的，它位于消息的开头，后接警告消息。

```
LaTeX Warning: Citation '...' on page ... undefined on
               input line ....
```

在 `\cite` 命令中的关键词并没有对应的 `\bibitem` 命令进行定义（ 4.3.6 和 8.3.2 节）。

```
LaTeX Warning: Citation '...' undefined on input line ....
```

2ε 在 `\nocite` 命令中的关键词还没有定义。

```
LaTeX Warning: Command ... has changed.
               Check if current package is valid.
```

[2ε]用 \CheckCommand 语句来测试一个命令是否已经具有特定的定义。如果这个测试失败，就会给出这条警告消息。这可以用来确认给定宏包的设计者到底想做什么。

```
LaTeX Warning: Float too large for page by ..pt on input line ...
```

[2ε]浮动对象（ figure 或 table 环境）太大，在一页上放不下来。当然浮动对象总是会被显示出来的，但是会超出正常页面的边界。

```
LaTeX Warning: ‘h’ float specifier changed to ’ht’.
```

[2ε]浮动对象的定位指定 ‘h’ 表示 ‘这里’（ 6.6.1 节），并不一定能精确地放在浮动对象所处的位置上。只有在当前页面有足够的空间时，才会把它放在 ‘这里’，否则它会出现在下一可用页面的顶部。这个消息就是对这一事实的提醒。

```
LaTeX Warning: inputting ‘...’ instead of obsolete ‘...’.
```

[2ε]有些专门为 LaTeX2.09 编写的宏包显示要求输入特定的文件，例如 article.sty 已经有一个新的等价文件, 但名称不同 (此时为 article.cls)。那么原文件名称就对应一个空文件，这里给出这个消息，要求输入正确的文件。

```
LaTeX Warning: Label ‘...’ multiply defined.
```

两条 \label 或 \bibitem 命令定义的标记有相同的名称 (8.3.1 和 8.3.2 节)。即使进行了改正，这条消息还会再显示一次，因为这次能从上次运行生成的 .aux 文件中找到重名标记。当第三次运行时，它应该没有了。

```
LaTeX Warning: Label(s) may have changed.
               Return to get cross-references right.
```

从 \ref, \pageref 和 \cite 命令中得到的输出可能不正确，因为在处理中它们的值已发生了变化。这就必须再运行一次 LaTeX，以使用正确的值。

```
LaTeX Warning: Marginpar on page ... moved.
```

在指定页上的边注已经被向下移动了，以防边注之间离得太近。这就是说这个边注并不是从 \marginpar 命令实际所位于的那行开始显示。

```
LaTeX Warning: No \author given.
```

[2ε]调用了 \maketitle 命令，但是前面却没有 \author 命令。与没有 \title 命令不同的是，这并不是一个错误，只是它觉得有点儿奇怪。只所以显示这个消息，就是以防你忘记输入作者信息。

```
LaTeX Warning: Optional argument of \twocolumn too tall on page..
```

[2ε]\twocolumn 命令 (3.2.5 节) 开始新页，并切换进入两列格式。在可省参数值中的文本以单列方式显示在两列文本上面的。如果这部分文本太长，在一页中放不下来，就会显示出这条警告。

```
LaTeX Warning: Oval too small on input line ....
```

在 \oval 命令中指定的尺寸太小，LaTeX无法找到一个相应的四分之一圆周。

```
LaTeX Warning: Reference `...' on page ... undefined on
               input line ....
```

在 \ref 或 \pageref 命令中的标志名在前一次运行时还没有用 \label 命令进行定义 (8.3.1 节)。如果在下一次运行中还会出现这条消息，那就是少掉了相应的 \label 命令。

```
LaTeX Warning: Text page ... contains only floats.
```

2ε 这条消息指出在指定页上，浮动对象样式参数已经把所有其它正常文本排除在外了。这并不是不好，但通常你需要自己看看这一页面样子。

```
LaTeX Warning: There were multiply-defined labels.
```

2ε 如果有 \label 或 \bibitem 命令多次使用同名的标志，就会在LaTeX运行结束时显示这条消息。在运行到每个重复标志附近时，也会显示一条警告消息。

```
LaTeX Warning: There were undefined references.
```

2ε 如果在前一次运行中，有的 \pageref 命令中的标志还没有定义，就会在LaTeX运行结束时显示这条消息。在每次用到一个还没有定义的标志时也会显示一条警告消息。

§9.5.2 LaTeX宏包警告

类和宏包警告主要就是当要求上载的文件版本和名称与实际不符，或者选项使用不当，或者不知道 filecontents 环境已经向一个文件中写入了某些文本时显示的消息。这里所提及的命令在 C.2.9 节有介绍。

类和宏包也可以有自己独特的警告消息。这里不可能列出这些消息，因为它们是与类和宏包密切相关的。

```
LaTeX Warning: File `...' already exists on the system.
               Not generating it from this source.
```

2ε 由于 filecontents 环境发现在系统中已存在一个同名的文件，因此它并没有从主文件中提取文本。

```
LaTeX Warning: Unused global option(s):....
```

2ε 在 \documentclass 声明中指定的选项是全局性的，即它要应用于所有的类和后面调用的宏包。然而，如果对某一个选项，所有的类或宏包都不能识别，就会显示出这条警告消息，然后列出没有用的选项清单。

```
LaTeX Warning: Writing file `...'
```

[2ε]filecontents 环境 (C.2.9 节) 正从主文档中提取信息, 并写到一个给定名称的文件中。

```
LaTeX Warning: You have requested class/package `...',
               but the class/package provides `...'.
```

[2ε]由内部识别命令 \ProvidesClass 或 \ProvidesPackage 给出的类或宏包名称并不与 \usepackage 或 \RequirePackage 命令中的名称一致.

```
LaTeX Warning: You have requested, on input line ..., version
               `...' of class/package ...,
       but only version `...' is available.
```

[2ε]由内部识别命令 \ProvidesClass 或 \ProvidesPackage 给定的类或宏包日期, 早于 \usepackage 或 \RequirePackage 命令要求的日期。此时类或宏包可能不具备输入文件所要求的全部功能。

```
LaTeX Warning: You have requested releases `...' of LaTeX,
               but only release `...' is available.
```

[2ε]你拥有的 LaTeX版本比某些输入文件中 \NeedsTeXFormat 命令要求的早 (C.2.1 节), 因此它可能不会提供该文件要求的所有功能。

§9.5.3 LaTeX字体警告

字体警告是在调用 NFSS命令 (8.5 节) 时出现的。其标志为 LaTeX Font Warning: 后接警告文本。

```
LaTeX Font Warning: Command ... invalid in math mode.
```

[2ε]只能用在文本模式中的命令, 被用在数学模式中。这条命令就会被忽略。 \boldmath, \unboldmath 和 \em 都会导致这条消息。也有一些命令会产生同样内容的错误消息。

```
LaTeX Font Warning: Command \tracingfonts not provided.
(Font)              Use the `tracefnt' package.
(Font)              Command found: on input line ....
```

[2ε]字体跟踪诊断工具 \tracingfonts 只有当上载了 tracefnt 宏包 (217 页) 时才可以使用。否则就会忽略该命令。

```
LaTeX Font Warning: Encoding `...' has changed to `...' for
(Font)              symbol font `...' in the math version `...'.
```

[2ε]要想在给定的数学变体中应用某一特定的符号字体, 需要暂时修改字体编码。

```
LaTeX Font Warning: Font shape `...' in size <...> not available
(Font)              size <...> substituted.
```

2ε 没有为要求的尺寸和形状定义字体，因此用了一个替换尺寸。

```
LaTeX Font Warning: Font shape `...' undefined
(Font)              using `...' instead.
```

2ε 要求的形状属性是未知的，或者还没有定义，因此使用一个替换形状。

```
LaTeX Font Warning: *** NFSS release 1 command ... found
(Font)              *** Update by using release 2 command ....
(Font)              *** Recovery is probably possible.
```

2ε 使用了一条来自于 NFSS第一版本的命令，而这条命令现在是不可用的。但是 LaTeX会尝试插入新的等价命令，并继续处理下去。

§9.5.4 TeX警告消息

TeX警告消息的标志为它并不是一个错误消息 (没有前缀 !)，处理过程也不会停下来。最常见的 TeX警告消息是：

```
Overfull \hbox ....
```

TeX无法以一种合理的方式断开该行，从而该行有部分内容伸展进右边界。消息中的其它内容可以给你一些帮助。

例如，下面是一条完整的警告消息：

```
Overfull \hbox (17.2122pt too wide) in paragraph at lines 4--6
[]\OT1/cmr/m/n/10 If T[]X can-not find an ap-pro-pri-ate
 spot to di-vide a word at the end of the line, as right
 here aaaaaaaaaaaaaaaaaaaaaaaaa
```

从这条消息我们可以知道当前行长度约超过了 17.2 pt(6 mm)，这也就是进入右页边界的长度。这一行位于从 4–6 行所在的段落中。所用字体的属性标志为 `\OT1/cmr/m/n/10`。有问题行的文本是 `If ... ... right here aaaaaaaaaaaaaaaaaaaaaaaaa`。供选择的断词点用连字符表示，如 `di-vide`。最后那个单词无法断开，也就是它导致当前的问题。

解决这个问题的一种方法是利用建议断词点，即在单词内部插入 `\-`，例如 `aaaaaaaaa\-aaaaaaaaaaaaaaaaa`。同样，在有问题单词前面插入 `\linebreak` 命令，或者把整个段落放在 `sloppypar` 环境中，也会去掉这条警告消息。

如果一个断词不是很好的行只稍稍长一点儿，比如说是 1 pt 或更少，在绝大多数情形中很少会注意到它，从而我们就不必在意这一点。

```
Overfull \vbox ....
```

很少会出现这条消息。它表示 TeX不能很好的分页，文本伸展到了页面底部。TeX通常宁可在一页上少放些内容，也不会放太多的东西。因此只有当一页上有一个太大 (超过 `\textheight` 的当前值) 的竖直盒子，例如一个长表格时，才会显示出这条警告消息。

`Underfull \hbox ....`

这是与 `Overfull \hbox` 警告相对的一条消息。当 TEX对一行进行了填充，以使得左右对齐，但是这时的单词间距太大，它认为结果不太令人满意时就会显示这条消息。这通常是 `sloppypar` 环境，`\sloppy` 声明或者 `\linebreak` 命令的后果。也有可能是对 `\\` 或 `\newline` 的不恰当应用造成的，例如连续给出了两条 `\\` 命令。警告中的其它消息由格式化结果不好的文本以及对单词间距糟糕程度的估计组成的。

如果对于上面的 `Overfull \hbox` 例子，在 `...` 插入命令 `\linebreak`，即 `as right\linebreak here`, 那么警告消息为

```
Underfull \hbox (badness 5504) in paragraph at lines 4--6
[]\OT1/cmr/m/n/10 If T[]X can-not find an ap-pro-pri-ate
 spot to di-vide a word at the end of the line, as right
```

这条消息说明在第 4 行到第 6 行上的段落包含了一个输出行，其单词间距太宽，不能令人满意。这行文本的内容为 `If ...  ..., as right`，当前字体为 `\OT1/cmr/m/n/10`。估计值 `badness 5504` 是对间距糟糕程度的度量，这个值越小，结果就越好。

为了对这个 badness 对一个大致的印象，那就必须知道它到底是如何计算出来的。所有的单词间距都有一个基本尺寸，以及它可以伸展或收缩的最佳限度。对于每个输出行，就会求出达到理想伸展和收缩后的实际单词间距。badness 的值为

badness $= 100 \times$ (实际的收缩量/最佳的收缩量)^3 或者

badness $= 100 \times$ (实际的伸展量/最佳的伸展量)^3

这里 '实际的收缩量' 和 '实际的伸展量' 就是在每行两端加上或减去的距离。

通常 TEX容忍把单词间距伸展到 badness=200，即真正的伸展是最佳伸展的 1.26 倍。在 `sloppypar` 环境或者 `\sloppy` 声明后，文本行是可以无限伸展的。当 badness 超过 1000 时就会显示出 `Underfull \hbox` 警告消息，这意味着伸展值为最佳值的 2.15 倍。如果上述公式结果超过 10000，就简单地取 badness 为 10000。

在实际应用过程中，可以容忍 badness< 2000，即使这时不理想的单词间距是很容易注意到的。应该打印出结果看看到底怎样。

`Underfull \vbox ....`

已进行了分页，并对顶部与底部进行了调整，但 TEX认为段落间距可能不太令人满意。这里的 `badness` 数相应于 `Underfull \hbox` 警告消息中同名数量。

§9.6 搜索顽固错误

有的时候，你会遇到一个错误，怎么也找不到出错的地方。对于这种可恶的错误，我们推荐如下的搜索策略：

1. 把正在处理的文档复制两份，一份做为旧版本，一份做为工作版本 (原来的版本仍保留住，在下面的搜索过程中不要改动它) 。
2. 在工作版本文档中，找到出错地方的最外层环境，去掉一个或多个内层环境。如果其没有内层环境，就缩短余下的文本。利用 LaTeX再次处理这个文件。
3. 如果仍然有那个错语，就把缩短后的工作版本复制到旧版本中，重复第 2 步。如果第 2 步中外层环境是 `\begin{document} ... \end{document}`，那么就可以通过在某些点简单地插入 `\end{document}` 来缩短文本。
4. 如果在缩短后的工作版本中不再有那个错误，那就把旧版本内容复制到工作版本中，这样错误仍然在这个版本中。这次比上次少去掉一些文本，重复 2 到 4 步。
5. 如果按这种方法，发现错误是在下一个最外层环境中，那就重复从 2 到 4 的同样操作。

利用这种方法，就可以把错误局限在一条命令或者只剩下很少一部分结构的环境中。如果尽管错误已被准确定位，当仍然不知道为什么错了，那就需要得到有经验的团体或计算中心的帮助了。然而，通常只要一找到出错的地方，就很容易知道为什么错了。

有时也确实会即使已改正了错误，在下次运行 LaTeX时还会出现这条错误消息。这主要是因为 LaTeX通过辅助文件进行信息转递，而这些文件的更新是在当前处理过程结束。例如，若某一个章节命令中有错误，那么即使已修正了这个错误，在 `.toc` 文件中仍旧是那个有错误的条目。如果文档中包含 `\tableofcontents` 命令，那么 LaTeX就会在下次运行时读入这个 `.toc` 文件，从而仍然给出那条错语消息。只有当成功处理完一次文件后才会得到新的 `.toc` 文件。

对于这种情况，应当编辑 `.toc` 文件，去掉那个错误。如果没法这样做，那就删掉这个 `.toc` 文件，并用 LaTeX编译修正后的文档 *两遍* 。如果错误是在 `\caption`, `\addcontentsline` 或者 `\addtocontents` 命令中，也要对 `.lof` 或 `.lot` 文件做同样的处理。

有时我们也必须删掉 `.aux` 文件，以杜绝虽然已改正了 `.tex` 文件中的错误，但仍然显示同样的错误。这时如果导言中有命令 `\nofiles` 时用户必须更加小心，因为此时再运行一次 LaTeX也不会生成正确的 `.aux` 文件。

附录A　书信的编辑

除了前面讲到的三种标准 LaTeX文档类外，还有另外一种文档类，名叫 letter.cls，专门用来编辑书信。但是 letter.cls 只限于写书没有任何装饰（例如信笺头或者公司名称）的私人书信。显然那些经验丰富的 LaTeX专家可以很容易地改变这一局限。

我们首先讲解标准的 LaTeX letter 类，然后演示在我们研究所里广泛使用的学术信件样式的编写方法。正如我们一贯倡导的那样，我们建议哪怕做了很小的一点儿改变，也应该把类存贮到另一个文件中，让 letter.cls 仍然表示原来的标准 LaTeX文件档类。

§A.1　LaTeX 的 letter 类

letter 文档类就是用来写信的。单个输入文件中可以包含不只一封信和地址的文本，这些信都是来自于同一个寄信人。如果愿意的话，可以同时自动打印出地址标签。绝大多数普通的 LaTeX命令在 letter 类中的功能仍像通常一样。然而一个例外就是章节命令，在这里它会导致错误消息：！ Undefined control sqeuence。实际上在书信中也没多大道理要划分章节。另一方面，也有许多只适用于这一样式的特殊命令。

一封信的源文件开头与所有 LaTeX文档类似，即

```
\documentclass[选项]{letter}
```

这里的 选项 可以取列在 3.1 节中的所有种类，只是 twocolumn 和 titlepage 例外，因为在书信中没有必要用到它们。在 LaTeX 2_ε中也可以用 twoside 选项，但在 LaTeX2.09 中不能用这一选项。

每封信都必须有寄信人的姓名和地址，这两类信息是通过如下放在导言中的命令来给出，从而适用于同文件中的所有信件：

```
\address{寄信人地址}
\signature{寄信人姓名}  或者  \name{寄信人姓名}
```

寄信人地址 通常由几行组成，行与行之间用 \\ 分开，例如：

```
\address{Max-Planck-Institut f\"ur Aeronomie\\
         Postfach 20\\
  D--37189 Katlenburg--Lindau\\Germany
```

如果给出了 \name 命令中的条目，那它将被用做信头的回信地址。而放在 \signature 命令中的条目将显示在书信结尾处给作者签名用空白的下面。如果没有给出 \signature，就会在这里插入 \name 中的内容。这样就可以得到一种非常正式的回信地址以及另一种不同形式的（可能是多行的）签名：

```
\name{Prof.\ M.\ Ostmann}
```

```
\signature{Martin Ostmann\\Project Leader}
```

当在导言中调用了上述命令，那它们就会对文档中的所有信件都适用，除非某些信件中以新内容调用了这些命令。因此有的信件可以拥有与其它信件不同的签名。这些条目的适用范围只延续到调用它环境的结束（见 7.5.4 节）。

在标准 LaTeX的 letter 类中还可以包含另外两个寄信人条目。可以利用它们进行适用于局部范围的修改。基本想法就是如果没有调用 \address，那就生成已设计好的公司信笺头以及寄信人的房间号和 / 或电话号码。因此提供了命令

```
\location{房间号} 和 \telephone{电话号码}
```

当使用的是 letter.cls，而且没有调用 \address 时就会在每页的底部显示出 房间号 和 电话号码。

导言中也可以包含 \pagestyle 命令，选项与通常的一样，即 plain, empty 或 headings。第一个也是缺省值，在第一页后面所有页的底部居中加上页码。 headings 页面样式会在第一页后面所有页的顶部在一行中加上收信人姓名，日期以及页码。

在导言命令后面，就是同所有 LaTeX文件一样，用 \begin{document} 命令开始实际的正文。正文由一封或多封信件组成，每封信用如下语法包围在 letter 环境中：

```
\begin{letter}{ 收信人信息 } 信件内容 \end{letter}
```

其中 收信人信息 由收信人的姓名和地址组成，可以用 \\ 命令生成几行。

```
\begin{letter}{Mr. Donald J. Burns\\
               Ontario Institute of Physics\\
 41 Adelaide St.\\
 London, Ontario\\Canada N4R 3X5}
\end{letter}
```

信件内容 通常以 \opening 命令开头，以 \closing 命令结尾，这两条命令之间就是信的主体，可以包含所有要用的 LaTeX命令。这两条命令的语法是

```
\opening{ 问候 }
\closing{ 祝好 }
```

其中 问候 是信的开头称呼，例如 Dear Mr. Tibs，而 祝好 表示正文的结束，例如 Yours sincerely,。 \opening 命令中也可以包含其它的文本，例如，可以用它生成一个主题行，而真正的问候语是放在接下来的正文中。

LaTeX把寄信人的姓名和地址放在第一页的右上角，其下面是右对齐的当前日期。然后把收信人的姓名和地址靠左边摆放，后接问候语与信件正文。整个信件以结束语来完成，然后是寄信人的姓名或签名，它们相对于中心线左对齐，并与正文间留下足够的竖直间距，以供手写签名。

在 \closing 命令后，还有一些命令可以使用，它们也是信件的一部分。

其中一条命令是 \cc，它生成该信件的分发名单：

\cc{ 姓名一 \\ 姓名二 \\ ... }

文本 'cc:'（或者更准确地说，是在 \ccname 中定义的文本）显示在左页边，然后向里缩进一点，并显示信件分发名单。

另一条命令是 \encl，它给出随信附件清单：

\encl{ 附件一 \\ 附件二 \\ ... }

单词 'encl:'（在 \enclname 中的文本）显示在左页边，随后接上附件的清单。

最后，命令 \ps 可以在签名后加上一条附言。这条命令自身并不生成任何文本，也没有任何参数。附言文本可以是介于 \ps 和 \end{letter} 命令之间的任何内容。

通常信件是自动标上当前日期的。然而，如果希望信件的日期向后推迟一下，或者把日期固定下来，那么可以用命令

\date{日期文本}

其中 日期文本 就放在日期应该位于的地方。

一个书信文件中可以包含任意数目的 letter 环境，每个环境对应于一封信。正如前面已讲到的，当在导言中调用了 \address, \name 和 \signature，那么它们就会对整个文件中所有信件有作用。当然也可以在一个 letter 环境内部 \opening 命令前面重新调用上述命令，以改变寄信人的信息，这些修改就对这封信有效。如果已经声明了 \name 和 \signature，那么后者就会显示在签字空白的下面。

信件的首页总是没有页码的，后面的几页才有一个在底部的居中页码（缺省值）或者在顶部有一个包含收信人，日期，以及页码的页眉（页面样式 headings）。

我们在图 A.1 中给出一封示例信件，它是用如下输入生成的：

```
\documentclass[a4paper,11pt]{letter}
\name{Dr P. W. Daly}
\address{Max-Planck-Institut f\"ur Aeronomie\\
         Postfach 20\\
  D--37189 Katlenburg--Lindau\\Germany}
\signature{Patrick W. Daly}
\begin{document}
\begin{letter}{\TeX proof Ltd\\P.\,O. Box 123\\
               9876 Wordtown\\Textland}
\opening{Dear Sir;}
We are most pleased to be able to answer your request for
information about the use of \LaTeX{} for general text processing
at a scientific institute.
```

图 A.1: 利用标准 `letter` 类生成的信件示例

Max-Planck-Institut für Aeronomie
Postfach 20
D-37189 Katlenburg–Lindau
Germany

August 15, 1999

TEXproof Ltd
P. O. Box 123
9876 Wordtown
Textland

Dear Sir;
We are most pleased to be able to answer your request for information about the use of LaTeX for general text processing at a scientific institute.

1. After some initial trepidation on the part of the secretarial staff, which was mainly due to the first experience with a computer in any form, the system is now fully accepted and appreciated.
2. Much to the surprise of many secretaries, they find that they are able to set the most complicated mathematical formulas in a reasonably short time without difficulties. The same applies to the production of detailed tables.
3. Creating cross-references and index registers no longer causes horror, even when the boss is well known for demanding constant changes.
4. Finally, the high quality appearance of the output has assisted in winning acceptance for LaTeX in our house.

An additional positive note is the ability to write business letters readily, making use of the `letter` class provided with LaTeX. In our institute, we have designed a special version to print our own letterhead, saving the need to have special letter paper printed.

Yours truly,

Patrick W. Daly

encl: Listing of our `mpletter.cls`
Sample output
cc: H. Kopka
B. Wand

©TEXGuru, August 15, 1999

```
\begin{enumerate}
  \item
  After soem initial trepidation on the part of the secretarial
  . . . . . .
  in winning acceptance for \LaTeX{} in our house.
\end{enumerate}
An additional positive note is the ability to write business
  . . . . . .
special paper printed.
\closing{Yours truly,}
\encl{Listing of our \texttt{mpletter.cls}\\Sample output}
\cc{H. Kopka\\B. Wand}
\end{letter}
\end{document}
```

而在导言中利用命令 \makelabels 用户可以打印出地址标签。其上的地址信息由 letter 环境中的收信人姓名和地址参数值组成。标准的 letter 类设计的标签尺寸为 $4\frac{1}{2} \times 2$ 英寸，并以两列形式排列。在其它格式中可以改变这一点。要想打印出一张没有对应信件的标签，可以采用如下形式的空白 letter 环境：

```
\begin{letter}{收信人}\end{letter}
```

§A.2 局部范围的信件样式

上面的示例信件演示了标准 letter 类的功能。可以在导言中利用适当的命令或声明来修改文本的高度和宽度。其中的英文单词和日期样式的显示转换到其它语言也没有什么问题，因为这些信息都是包含在特殊的命令中，可以很容易进行重定义。（这实际上只适用于 1991 年 12 月 1 日以后的 LaTeX版本，在此之前都是直接应用英文单词。）

在 letter 类中如果忽略了 \address 命令，那就会显示出公司的信头。当然这要求在进行安装时已经预先进行了相应的设置。每位职员都可以应用这种局部范围里的书信样式，同样给出必要的私人信息，例如房间号和 / 或电话号码。在标准的 letter 类中已提供了这些命令。

在我们研究所就有这样一个局部范围使用的样式，我们下面来演示一下。因为我们发现在新样式中必须能提供更多的个人信息，如 'Our Ref.', 'Your Ref.'，以及电子邮件地址。而且，这里增加了一条可以显示出 'Subject:' 的命令。

为了把我们的局部范围使用的样式与 LaTeX标准区分开，我们把它命名

为 mpletter 。它拥有比 letter 更多的功能，因为它实际上是先读入那个标准的类文件，然后进行了一些修改和增加。现在并不需要 \address 命令，因为所有的信件都显示出研究所的信头，以及地址。收信人的姓名和地址来自于 letter 环境的参数值，并竖直居中地放在信头空白处。

作者的姓名和电话号码是用如下命令输入的：

\name{作者} 和 \telphone{通讯号码}

如果它们在导言中被调用，就适用于文件中所有的信件。如果不同的信件有不同的作者，那就必须在不同的 letter 环境中 \opening 命令前面调用它们。mpletter 所拥有的新条目命令为

\yref{对方号码}

\ymail{对方日期}

\myref{我的号码}

\subject{主题文本}

它们会生成单词：

Your Ref.:, *Your letter of*:, *Our Ref.*:, *Subject*:.

并与相应的文本参数值一起放在信头的下面。如果没有给出某一命令，在信件中也就不会出现相应的单词。

由于我们是一个德国研究所[1]，那么我们要有一个 german 选项，它把上述所有单词翻译成相应的德语单词。而条目命令有相同的名称。

在标准的 letter 类中，当前日期是自动显示出来的。如果想把日期向后推迟一下，或者在信件内部固定下来日期，也可以用下面的命令把它改成任何希望的时间：

\date{日期文本}

如果保留了信件的电子版本，这就非常方便。即使信件已过了几个月，也可以用新的日期打印出来。

以 \name 命令参数值形式输入的 作者 条目以寄信人身份显示在信件上方。同时也会显示在签字空间下面的地方，除非这一信息被命令 \signature 覆盖，因为调用后者可以给出另外不同形式的签字名称。

这里给出的示例信件，就是用我们的样式生成的，源文本如下：

```
\documentclass[12pt]{mpletter}
\name{Dr P. W. Daly} \signature{Patrick W. Daly}
\myref{PWD/sib}
\subject{\LaTeX{} information}
\telephone{279} \internet{daly}
\ymail{May 28, 1999} \yref{GFM/sdf}
\begin{document}
```

[1]这里我们是直接翻译，以示例如何设计一个适用于局部范围信件样式的方法

MAX-PLANCK-INSTITUT FÜR AERONOMIE
Postfach 20
Katlenburg–Lindau
GERMANY

MPI für Aeronomie, Postfach 20, D-37189 Katlenburg–Lindau

Dr P. W. Daly
Tel.:05556-401-279
INTERNET:
daly@linmpi.gwdg.de

Mr George Murphy
35 Waterville Rd.
Centertown, Middlesex
United Kingdom

August 15, 1999

Your Ref.: GFM/sdf *Your letter from*: May 28, 1999 *Our Ref.*: PWD/sib
Subject: LaTeX information

Dear George,

Thank you for your inquiry about the latest version of the LaTeX installation and additional package. As you may know, there was a major update last year with the release of LaTeX 2ε on June 1, 1994, which became the official standard at the time. It is intended that updates be issued every six months, in June and December, with absolutely urgent fixes coming out as patches when necessary.

I am sending you a copy of the current version of the entire LaTeX package on an MS-DOS floppy disk, as you requested. In a separate directory named `bibtex` you will find the special bibliography formatting style files mentioned in 'A Guide to LaTeX'. I hope you will find these of use.

Do not hesidate to get in touch with me again if you have any further questions about the installation or running of the package.

Regards,

Patrick W. Daly

encl: 1 diskette with LaTeX package
cc: H. Kopka

Telephone	05556-401-1	Bank	Train Station
Telefax	05556-401-240	Kreis–Sparkasse Northeim	Northeim
Telex	9 65 527 aerli	41 104 449(BLZ 262 500 01)	(Han.)

http://202.38.68.78/texguru Email: texguru@263.net

```
\begin{letter}{%
Mr George Murphy\\35 Waterville Rd.\\
Centertown, Middlesex\\United Kingdom}
\opening{Dear George,}
Thank you for your inquiry about the latest version of the
\LaTeX{} installation and additional package. As you may know,
. . . . . .
Do not hesitate to get in touch with me again if you have any
further questions about the installation or running of the
package.

\closing{Regards,}
\encl{1 diskette with \LaTeX{} package}
\cc{H. Kopka}
\end{letter}
\end{document}
```

正如所示例的那样，在这里第一页的信头上并没有页码。如果信件内容超过了一页，那么后面几页的信头如下：

MAX-PLANCK-INSTITUT FÜR AERONOMIE

To Mr George Murphy *August 15, 1999* *Page 2*

显示在这里的的收信人地址是来自于 `\begin{letter}` 中 `收信人` 参数值的第一行。当 LaTeX处理信件时，它会把这个参数值分成两部分，其中第一行就包含在 `\toname` 命令中，其余几行包含在 `\toaddress` 命令中。而单词 'To' 和 'Page' 是在标准命令 `\headtoname` 和 `\pagename` 中，可以按照 C.3.3 节所讲的那样改成适合于任何语言的单词。

§A.3 定制信件样式

对一个有经验的 LaTeX程序设计者而言，修改 `letter.cls` 类文件，以满足某种特定的需要，应该是不会有多大困难的。即使普通用户也可以在本节例子的帮助下对类文件进行必要的修改。

我们这里给出用来生成 271 页上那封信的样式文件 `mpletter.cls` 。这要大量利用在 C 介绍的 LaTeX程序设计方法。为了理解这里所提到的内容，读者应该对 C.2 节的内容相当熟悉。不必对 `letter.cls` 文件自身做什么修改，因为这里所有的修改都是放在单独一个文件中，而这个文件要读入原来的类文件。

新的类文件命名为 `mpletter.cls` 。开头即指出它所需要的 TeX格式，并

标明了自己的身份。

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mpletter}
```

由于这里需要执行条件判断，因此我们要上载 7.3.5 节描述的 ifthen 宏包。我们也需要一个标志来确定是否信件用的是德语，这由选项来确定。下面就是创建标志，并定义选项 german 来设置标志的方法：

```
\RequirePackage{ifthen}
\newboolean{@german}
\setboolean{@german}{false}
\DeclareOption{german}{\setboolean{@german}{true}}
```

所有在标准 letter 类中可以用的选项这里也都可以使用，因此就只需简单地用缺省选项把它们传递给 letter 类。然后在用 a4paper 选项上载 letter 之前处理所有的选项。在我们研究所只有 A4 纸。

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{letter}}
\ProcessOptions
\LoadClass[a4paper]{letter}
```

这就完成了准备工作。到现在为止，我们已经读入了标准 letter 类，以及 ifthen 宏包，并定义了一个在原来类中没有提供的新选项。否则到现在为止任何选项和功能都没有改变。

现在我们定义新的 'name' 命令，使它的内容与语言有关，例如在基本类中并没有提供的 '*Subject*' 。实际的定义是用 \englishnames 和 \germannames 这两条命令来执行的。

```
\newcommand{\englishnames}{%
 \newcommand{\yrefname}{\textsl{Your Ref.}}
 \newcommand{\ymailname}{\textsl{Your letter from}}
 \newcommand{\myrefname}{\textsl{Our Ref.}}
 \newcommand{\subjectname}{\textsl{Subject}}
 \newcommand{\telephonename}{Telephone}
 \newcommand{\stationname}{Train Station}
 \newcommand{\germanname}{GERMANY}
 \newcommand{\telcode}{[49]-5556-401}
 \newcommand{\postcode}{D--37189}
}

\newcommand{\germannames}{%
. . . . . .
```

```
 \newcommand{\telcode}{(05556) 401}
 \newcommand{\postcode}{37189}
}

\ifthenelse{\boolean{@german}}
  {\RequirePackage{german}\germannames}{\englishnames}
```

最后一行就是用来测试是否已（用 german 选项）设置了 @german 标志，如果这样做了，那就上载 german 宏包，并定义德语名称，否则就用英文名称。宏包 german 已经把标准的名称命令 \toname, \headtoname 和 \pagename 翻译成了德语，因此在 \germannames 中没有重定义它们。

解决好了语言问题，我们下面对付在信头上输入信息的命令。这其中每条命令都在一条内部命令中存贮了其文本参数值，以供后面使用。首先必须创建内部的存贮命令。

```
\newcommand{\@yref}{}      \newcommand{\@ymail}{}
\newcommand{\@myref}{}     \newcommand{\@subject}{}
\newcommand{\@internet}{}
\newcommand{\yref}[1]{\renewcommand{\@yref}{\yrefname: #1}}
\newcommand{\ymail}[1]{\renewcommand{\@ymail}{\ymailname: #1}}
\newcommand{\myref}[1]{\renewcommand{\@myref}{\myrefname: #1}}
\newcommand{\subject}[1]{\renewcommand{\@subject}
                          {\subjectname: #1}}
\newcommand{\internet}[1]{\renewcommand{\@internet}{#1}}
\newcommand{\INTERNET}{@linmpi.gwgd.de}
```

我们下面设置每页上正文与页边的尺寸。这些数值都是相应于 A4 纸的（这实际上也就使得 a4paper 选项成为多余的）。

```
\setlength{\textheight}{215mm}  \setlength{\textwidth}{160mm}
\setlength{\oddsidemargin}{0pt} \setlength{\evensidemargin}{0pt}
\setlength{\topmargin}{-20pt}   \setlength{\headheight}{12pt}
\setlength{\headsep}{35pt}
```

接下来的一步就是定义信头中需要的一些固定字体。我们是通过字体的NFSS属性直接引用它们；此时我们用的都是不同尺寸的计算机现代 sans serif 字体。这样即使正文用了不同族的字体，这些字体也不会改变。

```
\DeclareFixedFont{\xviisf}{OT1}{cmss}{m}{n}{17}
\DeclareFixedFont{\xsf}{OT1}{cmss}{m}{n}{10}
\DeclareFixedFont{\viiisf}{OT1}{cmss}{m}{n}{8}
```

信头是分为两部分的，左边那部分包含研究所的名称，以大号字母显示，

右边那部分以小号字体显示出地址。在第一条水平线下面，左边部分列出收信人的姓名和地址，而且定位恰好在信封的小窗口里，右边部分则是写信人的信息，包含姓名，电话，计算机地址等等。这些部分的宽度是如下设定的。

```
\newlength{\leftfield}    \setlength{\leftfield}{117mm}
\newlength{\rightfield}   \setlength{\rightfield}{43mm}
```

这两部分的总宽度等于 160mm ，这也就是 \textwidth 的值。

然后，我们把研究所的名称和地址放到几个存贮盒子中。

```
\newsavebox{\FIRM}        \newsavebox{\firmaddress}
\newsavebox{\firm}        \newsavebox{\firmreturn}

\sbox{\FIRM}
  {\parbox[t]{\leftfield}
       {\xviisf MAX--PLANCK--INSTITUT F\"UR AERONOMIE}}

\sbox{\firm}
  {\xsf MAX--PLANCK--INSTITUT F\"UR AERONOMIE}

\sbox{\firmreturn}
  {\viiisf\underline{MPI f\"ur Aeronomie, Postfach 20,
                     \postcode{} Katlenburg--Lindau}}

\sbox{\firmaddress}
  {\parbox[t]{\rightfield}{\viiisf\baselineskip10pt
    POSTFACH 20\\Max--Planck--Stra{\ss}e 2\\
    \postcode{} Katlenburg--Lindau\\ \germanname}}
```

把这些盒子做为构造模块，我们可以在另外两个存贮盒子中搭建起实际的信头和信脚。

```
\newsavebox{\firmhead}   \newsavebox{\firmfoot}

\sbox{\firmfoot}
  {\parbox{\textwidth}{\usebox{\FIRM}\raisebox{6pt}
     {\usebox{\firmaddress}}\\[3pt] \rule{\textwidth}{1pt}}}

\sbox{\firmfoot}
  {\parbox{\textwidth}{\rule{\textwidth}{0.6pt}\\[5pt]
     \viiisf\setlength{\baselineskip}{12pt}%
    \begin{tabular}[t]{@{}ll}
```

```
      \underline{\telephonename}    & \telecode-1 \\
      \underline{Telefax}           & \telcode-240\\
      \underline{Telex}             & 9\,65\,527 aerli
   \end{tabular}\hfill
   \begin{tabular}[t]{l}
      \underline{Bank}\\
      Kreis--Sparkasse Northeim\\
      41\,104\,449 (BLZ 262\,500\,01)
   \end{tabular}\hfill
   \begin{tabular}[t]{l@{}}
      \underline{\stationame}\\
      Northeim\\
      (Han.)
   \end{tabular} }}
```

这里 \firmhead 的意义相当明了：它是一个宽为 \textwidth 的子段盒子，内容由在一条横线下面的 \FIRM 和 \firmaddress 盒子并列组成。 \firmfoot 也是一个同样宽度的 \parbox ，但内容是由三列包含在 tabular 环境中的研究所信息组成。

现在余下的事情是要把信头和信脚盒子只放在第一页上。在 letter 类中，有一个名称为 firstpage 的特殊页面样式，信件的第一页总是要调用该样式。因此我们要重定义它。

```
\renewcommand{\ps@firstpage}
  {\setlength{\headheight}{41pt}\setlength{\headsep}{25pt}%
   \renewcommand{\@oddhead}{\usebox{\firmhead}}%
   \renewcommand{\@oddfoot}{\raisebox{-20pt}[0pt]
                            {\usebox{\firmfoot}}}%
   \renewcommand{\@evenhead}{}\renewcommand{\@evenfoot}{}}
```

这个页面样式必须（重）定义 \@oddhead 和 \@oddfoot 命令，这两条命令就插入在奇数页的顶部和底部，从而放置我们的特殊 \firmhead 和 \firmfoot 。偶数页并不重要，因为第一页肯定是奇数页。这里需要加大 \headheight 和 \headsep 以放得下来大盒子。

接下来的页面就设置成 headings 或者 plain 样式。我们想修改前者，使其还要包含公司的地址。

```
\renewcommand{\ps@headings}
   {\setlength{\headheight}{41pt}%
    \renewcommand{\@oddhead}
```

```
    {\parbox{\textwidth}{\usebox{\firm}\\[5pt]
       \slshape \headtoname{} \toname\hfill\@date\hfill
                          \pagename{} \thepage\\
       \rule[3pt]{\textwidth}{1pt}}}
  \renewcommand{\@oddfoot}{}
  \renewcommand{\@evenhead}{\@oddhead}
  \renewcommand{\@evenfoot}{\@oddfoot}
```

现在还有一个小问题：当第一次调用其中某一个页面格式命令时，信头和信脚命令可能还没有定义，那么 \renewcommand 命令就会发出抱怨。为此我们在重定义的前面加上 \providecommand(7.3.1 节) 。

```
\providecommand{\@evenhead}{}\providecommand{\@oddhead}{}
\providecommand{\@evenfoot}{}\providecommand{\@oddhead}{}
```

那么现在把 headings 设成起作用的样面样式.

```
\pagestyle{headings}
```

现在还有一件事要做，那就是重定义显示收信人地址和称呼的 \opening 命令。我们要给它加一点儿功能，要求其中还要包含寄信人的信息，以及其它信息。地址放在左边部分，寄信人信息放在右边部分。其它信息放在标尺下面一行，后接主题行。要首先对这些条目进行测试，如果非空才显示出来。这儿的几个存贮命令也是标准 letter 类的一部分，例如 \toname 和 \toaddress 。而 \@date 则就是 \today 或者存贮在 \date 中的文本。

```
\renewcommand{\opening}[1]{\thispagestyle{firstpage}%
  \parbox[t]{\leftfield}
     {\usebox{\firmreturn}\\
      \parbox[b][3.5cm][c]{\leftfield}{\toname\\ \toaddress}}%
  \parbox[t]{\rightfield}
     {\fromname
      \ifthenelse{\equal{\telephonenum}{}}
         {}{\\ Tel.: \telcode-\telephonenum}
      \ifthenelse{\equal{\@internet}{}}
         {}{\\{\viiisf E-mail: \@internet\INTERNET}}
      \\[5mm] \@date}
  \par
  \rule{\textwidth}{0.6pt}
  \makebox[\leftfield][l]
     {\ifthenelse{\equal{\@yref}{}}
        {\@ymail}{\@yref\hfill\@ymail\hfill}}
```

```
\@myref\par
\ifthenelse{\equal{\@subject}{}}
   {}{\@subject\par}
\vspace{2\parskip} #1 \par\nobreak}
```

在 271 页上可以看到格式化的结果。显然若想把代码改成适合于其它组织，应不会有多大困难。

收信人的姓名和地址已精心进行了定位，这样对于正确的折叠方式，它们就会恰好位于信封的小窗口里。从这个窗口也可以看到较小的回信地址，因此也就没有必要再打印额外的地址标签了。

练习 1.1做一个练习，用户试设计文档类 `pletter`，以书写私人信件。利用前面格式公司信件的启发，加下如下形式的信头：

Sheila Joan McDonald

Tel.: 234-9871
31 Maple Drive
Willowtown

提示：这里的姓名字体是利用 `\newfont` 命令上载的 `cmdunh10 scaled \mapstep4`。这样的 `pletter` 类必须保存在用户个人的目录中。因为所有人都用同样的信头那也太令人尴尬了。

©TEXGuru, August 15, 1999

附录B　参数文献数据库的处理

在科技出版物的创作中，参考文献的组织是一个不可缺少的标准过程。我们已经在 4.3.6 节和 8.3.2 节中讲解了如何利用 thebibliography 环境来排版参考文献，以及如何在正文中引用其中的条目。有时候作者会发现在绝大多数文章中他经常引用的是同样作品。与此类似，在同一个领域工作的研究人员参考的论文也大致相同。这也就是说在不同文档的 thebibliography 环境中经常有大量重复的条目，在一个研究所中的同事与同事之间处理的文档里也有类似的情形。

如果我们能把参考文献条目全部存贮在一个数据库文件里，使得所有的文档都可以引用其中的一组文献，这就会节省大量的时间。借助于 LaTeX宏包中的 BibTeX程序我们就可以建立起这样的一个数据库。把各种出版物的信息存贮在一个或多个后缀为 .bib 的文件中。每篇文献有一个 关键词 来与其它文献区分开，在正文中可以通过它来引用该文献。这样的文件就称为 参考文献数据库 。

§B.1　BibTeX程序

BibTeX是 LaTeX的一个辅助程序，它通过搜索一个或多个数据库，自动为 LaTeX文档构造参考文献。要做到这一点，LaTeX文件中必须在参考文献所位于的地方调用命令

```
\bibliography{数据库一,数据库二,...}
```

其中的参数值 数据库一, 数据库二, ... 就是要搜索的参考文献数据库的基本名，中间用逗号分开。并不需要显式地给出 .bib 后缀。

在 LaTeX正文任何地方都可以通过下面命令引用数据库中的一篇文献：

```
\cite{关键词}
```

这与 8.3.2 节中的说明一样。其中 关键词 就是文献的标志，这当然要求用户必须预先知道这一信息。在第一次运行 LaTeX后，就必须执行 BibTeX程序。至于如何调用这个程序，那就要看所处的计算机操作系统了，但通常的方法就是操作命令 bibtex 后接 LaTeX文件的基本名。假设这个基本名是 comets，那么

```
bibtex comets
```

就会生成一个名为 comets.bbl 的新文件，其中就包含从数据库中提取的相应于 \cite 引用的参考文献信息，并包装到一个 thebibliography 环境中，下次运行 LaTeX时就会把它输入到文档中。

有时候需要包含进一个在正文中没有引用的文献。这可以用如下命令来加入该文献：

```
\nocite{关键词}
```

可以把这条命令放在主文档内的任意地方。它并不生成任何文本，只是告诉BibTeX也要把这一项放到参考文献中。而命令 `\nocite{*}` 会把数据库中所有项都包含到文档中，这对于要生成所有条目及其关键词是非常有用的。(** 对 BibTeX0.98 版及以前版本这条命令是没有用的. **)

在执行 BibTeX后，就会生成 `.bbl` 文件，这就需要至少执行两次 LaTeX以建立起参考文献和正文中正确的引用记号。参考文献就显示在 `\bibliography` 命令被调用的地方；这条命令实际上是把 `.bbl` 文件包含到正文中。

可以用下面的声明来选择参考文献的样式：

```
\bibliographystyle{样式}
```

可以在导言的任何地方调用这条命令。 *样式* 参数可以取如下的几个值：

`plain` 参考文献中的条目按字母顺序排列；每项条目都有一个活动编号，并放在方括号内，这就是在正文中引用该文献（即 `\cite` 命令调用的地方）时的标记。

`unsrt` 条目按第一次出现在 `\cite` 和 `\nocite` 命令中的顺序排列。在第一个 `\cite` 中的条目编号为 1，下一个具有不同关键词的 `\cite` 条目编号为 2，依此类推。其它标记和列表方式与 `plain` 样式相同。

`alpha` 参考文献的顺序与 `plain` 样式相同，但是记号用的是作者姓名的缩写加上出版年代。对 Smith(1987) 的引用将会是 [Smi87] 。

`abbrev` 顺序和标记与 `plain` 相同，但是参考文献中要对作者的名字，月份，以及杂志名称进行缩写，以缩短长度。

在用户使用的计算机上可能还有其它可用的样式。特别地，正如 B.3.1 节所描述的 *作者–年代* 索引，我们可以从标准 LaTeX宏包中衍生出大量的不同样式。参考文献样式是包含在后缀为 `.bst` 的文件中。

下一节就讲解供 BibTeX 程序 0.99 版本使用的参考文献数据库。在这个版本中有很多 0.98 版本所没有的功能，我们用双星号 ** 来标明这些新功能。那些在旧版本中可以使用的数据库文件 (`.bib` 文件) 在 0.99 版本中的功能仍然一样。然而，适用于一个版本的 `.bst` 参考文献样式文件在另一个版本中可能就行不通了。要想知道你所用的到底是哪个版本，只要看一下 BibTeX运行时屏幕输出或草案文件 `.blg` 的第一行。

§B.2 创建参考文献数据库

创建参考文献数据库看起来好象要比向 `thebibliography` 环境中输入一串引用文献的工作量大很多。但是这样做的最大优势就是一旦把某条目放到参考文献数据库中，也就一劳永逸了，以后所有的文章都可以引用它。即使以后需要另一种不同的参考文献样式，所有已在数据库中的条目也是可以用

的，这就比用另外的格式重写 thebibliography 环境要方便得多。实际上，据我们的经验，即使正在处理的只有一篇文章，向数据库中加进一项条目，也要比向文献列表中加一项简单，因为后者需要准确而灵巧的定位，而且要考虑到标点符号和作者名字的位置。如果有了在 B.2.6 节中演示的一般性模板，数据条目的处理就会变得非常简单，而且速度很快。

在参考文献中的条目具有如下的形式

```
@BOOK{knuth:86a,
  AUTHOR = "Donald E. Knuth",
  TITLE = {The \TeX{}book},
  EDITION = "third",
  PUBLISHER = "Addison--Wesley",
  ADDRESS = {Reading, Massachusetts},
  YEAR = 1986  }
```

第一个单词，其前缀 @，确定了 *条目类型*，我们将在下节详细解释。*条目类型* 后接放在大括号 { } 中的条目信息。其中第一项就是 *关键词*，这也就是在 \cite 命令中引用的名称。在上面的例子中，其为 knuth:86a。*关键词* 可以是字母、数字以及符号（逗号除外）的任意组合。然后就是真正的索引信息，它们分放在各个 *域* 中，相互之间用逗号分开。每个 *域* 由 *域名*，一个等号 =（其两边可以有空格）以及 *域文本* 组成。上面给出的 *域名* 有 AUTHOR, TITLE, PUBLISHER, ADDRESS 和 YEAR。*域文本* 必须放在大括号或者双引号内。然而，如果这部分文本只是由数字组成，比如上面的 YEAR 后的内容，那么可以不要括号或引号。

在输入条目时，有些域是不可少的，还有一些域则是可以省略的，而其它的则要被 BibTeX忽略的。在下面与条目类型一起列出了各种域。如果缺少了某个不可少域，BibTeX运行时就会给出一条错误消息。如果使用了可省域，那么它们在参考文献中就会包含所提供的信息，但是并不必要一定给出这些域的内容。而要被忽略的域可以在数据库中包含不输出的额外信息（例如注释或者论文摘要），这也是非常有用的。要被忽略的域也可以提供给其它的数据库程序使用。

在参考文献数据库中条目的一般语法为

```
@条目类型{关键词,
  域名  = {域文本}
  ....
  域名  = {域文本} }
```

在 *条目类型* 以及 *域名* 的名称中既可以用大写字母，也可以用小写字母，甚至是两者的混合。因此 @BOOK, @book, @bOOk 都表示同样的类型。

在整个条目最外面的括号对既可以是如上所示的大括号 { }，也可以是

小括号 ( )。对于后者，一般的语法为

@条目类型(关键词, ...)

域文本 却只能放在大括号 {...} 或者双引号 "..." 内，如上例所示。

§B.2.1 条目类型

下面按字母顺序列出了标准的条目类型，并给出类型功能的简短描述，以及其可包含的不可少域和可省略域名称。我们将在下一节解释域的意义。

@article 条目为期刊或杂志上的一篇文章。

不可少域 author, title, journal, year.

可省略域 volume, number, pages, month, note.

@book 条目为有确定出版社的书籍。

不可少域 author 或 editor, title, publisher, year.

可省略域 volume 或 number, series, address, edition, month, note.

@booklet 条目为印制的有封皮的作品，但没有出版社或赞助机构的名称。

不可少域 title.

可省略域 author, howpublished, address, month, year, note.

@conference 与下面的 @inproceedings 相同。

@inbook 条目为一本书的一部分 (章，节或某些页) 。

不可少域 author 或 editor, title, chapter 和 / 或 pages, publisher, year.

可省略域 volume 或 number, series, type, address, edition, month, note.

@incollection 条目为一本书中有自己题目的一部分。

不可少域 author, title, booktitle, publisher, year.

可省略域 editor, volume 或 number, series, type, chapter, pages, address, edition, month, note.

@inproceedings 条目为会议论文集中的一篇文章。

不可少域 author, title, booktitle, year.

可省略域 editor, volume 或 number, series, pages, address, month, organization, publisher, note.

@manual 条目为科技文档。

不可少域 title.

可省略域 author, organization, address, edition, month, year, note.

@mastersthesis 条目为硕士论文。

不可少域 author, title, school, year.

可省略域 type, address, month, note.

@misc 条目为不属于其它任何类型的作品。

不可少域 没有。

可省略域 author, title, howpublished, month, year, note.

@phdthesis 条目为博士论文。

不可少域 author, title, school, year.

可省略域 type, address, month, note.

@proceedings 条目为会议论文集。

不可少域 title, year.

可省略域 editor, volume 或 number, series, address, month, organization, publisher, note.

@techreport 条目为学校或其它研究机构印制的报告。

不可少域 author, title, institution, year.

可省略域 type, number, address, month, note.

@unpublished 条目为有作者和标题的还未出版的作品。

不可少域 author, title, note.

可省略域 month, year.

在每项条目中还可以有可省略域 key 和 crossref 。前者提供当没有作者信息时字母排序用的附加信息。作者信息通常就放在 author 域中，但也有可能放在 editor 域或者 organization 域中。这个 key 域与 \cite 命令中所用的识别条目的 关键词 没有任何关系。而 crossref 域给出另一个条目的 关键词 ，使得两者共享某些域的信息，见 B.2.3 节。

§B.2.2 域

在一项参考文献条目中可能使用的域及其含义见下面的列表。而域的形式总是为

域名 ={ 域文本 } 或者

域名 =" 域文本 "

address
: 出版社或者研究所的地址。对于大的出版社，只要给出所在的城市就可以了。而对于小的出版社，则建议给出详细的地址。

annote
: 在非标准参考文献样式中可以使用评注，这样可以得到有注释的参考文献。在标准 BibTeX样式中这个域是被忽略的。

author
: 给出作者的姓名，见 B.2.4 节。

booktitle
: 当只引用了一本书的某一部分时，应给出书的题目。参考 B.2.4 一节中有关大写方面的特殊考虑。

chapter
: 章节的编号。

`crossref`
在数据库中另一项的关键词, 这样两者可以共享一些相同的域文本。见 B.2.3 节。

`edition`
书籍的版本, 通常采用完整的首字母大写的单词表示, 如 'Second' 。必要时标准样式可以把它改成小写。

`editor` 在 B.2.4 节描述的形式中的编辑姓名。如果还有 `author` 域, 那么用这个域给出引用书籍或作品集的编辑。

`howpublished`
说明出版方法的非同寻常之处。应该首字母大写, 如: ' Privately published' 。

`institution`
科技报告的举办机构。

`journal`
期刊或杂志的名称。对于那些相当常见的刊物名称, 有相应的缩写 (见 B.2.5 节)。

`key` 当没有给出作者信息时, 用此信息进行字母排序。这与 `\cite` 命令中区分条目的 *关键词* 没有任何关联。

`month` 作品出版时的月份, 或者指还未出版的作品写完时的月份。这里可以用月份 (英文) 名称的前三个字母进行缩写。

`note` 应加上的其它信息。首字母应该大写。

`number` 期刊, 杂志, 技术报告, 或者一系列作品中的一部的编号。期刊通常用卷和期来标识; 技术报告则由研究机构赋予一个编号; 系列书籍中的某部有时也有一个编号。

`organization`
会议或报告的主办机构。

`pages` 页码或者起始页码, 形式为 `32,41,58` , `87--101` 或者 `68+` 。最后那一种形式表示 68 页及以后所有页。单个连字符表示页码范围, 要被转化为标准样式中的双连字符形式, 并生成一个小破折号, 如 '87–101' 。

`publisher`
出版社的名称。

`school` 写作论文时所在的学术机构的名称。

`series` 一系列书或一套书的名称。当从这个系列中引用一本书时, `title` 域给出书的题目, 而可省的 `series` 给出整系列书的标题。

`title` 作品的题目, 要遵从 B.2.4 节给出的大写规则。

`type` 技术报告的类型, 例如 'Research Note' 。

`volume` 期刊或者多卷书籍的卷号。

year 作品印刷的年代，或者未出版作品的完成年代。年代通常要求有四位数，例如 1999 。

另外还有一些域名，BibTeX通常就把它忽略掉。例如，要想向数据库中加进文章的摘要，可以用

```
abstract ={摘要文本}
```

这对于我们的应用不只是管理这个数据库时，就非常有用了。

§B.2.3 域的交叉引用

** 本节内容只适用于 BibTeX 0.99 版本及以后版本。 **

如果参考文献数据库中有许多条目有相同的域信息集合（例如出现在同一会议论文集中的几篇作品），那么就可以用 crossref 域来引用另一个条目中的与自己相同的域信息。例如，

```
@INPROCEEDINGS{xyz-1,
  crossref = {xyz-proceedings},
  author = {J. S. Jones},
  title = {The First Results from the {Appleville Experiment}}
  pages = {34--38} }
. . . . . .
@PROCEEDINGS{xyz-proceedings,
  editor = {C. H. Kelvin},
  title = {Proceedings of the First Conference on the
             {Appleville Experiment}},
  booktitle = {Proceedings of the First Conference on the
               {Appleville Experiment}}
  year = 1991   }
```

第一项关键词是 xyz-1 ，利用 crossref 域从关键词为 xyz-proceedings 的第二个条目得到自己所缺少的域信息。这里缺少的域有 editor, booktitle 和 year ，这也是对会议论文集中所有论文都一样的域。注意在 @PROCEEDINGS 中 booktitle 域是被忽略的，而这儿必须包含它，因为 @INPROCEEDINGS 需要这一个域的信息。

如果一个条目被其它两个或更多条目引用了，即使没有 \cite 或 \nocite 命令用其关键词做参数值，这项也会包含在参考文献中。

为了使整个系统工作正常，在数据库中那些被引用的条目必须放在引用它们的条目后面。因此我们建议把所有这些只被其它条目引用的条目放在数据库尾部。交叉索引不能嵌套。

§B.2.4 特殊的域格式

对输入到 author, editor, title 和 booktitle 中的域文本必须遵从如下几条规则。当 BibTeX处理姓名时，它会根据样式文件中的指令，首先给出作者的姓，然后是名的首字母缩写。因此如何告诉系统哪是名，哪是姓就非常重要了。对标题也有类似的问题，要根据样式和 / 或条目类型进行首字母大写，因此 BibTeX必须知道哪些单词要被大写。

姓名

在 author 和 editor 域中的姓名既可以输入为 {*名姓*} 的格式，也可以是 {*姓*，*名*} 格式。也就是说，如果没有逗号，BibTeX就认为第一个首字母大写的单词是姓；否则就把逗号前面的文本当做姓。因此姓名文本 "John George Harrison" 和 "Harrison, John George" 都指的是 J. G. Harrison 先生。然而，如果一个人有复姓，中间又没有连字符隔开，那就必须采用第二种方式，或者复姓放在大括号内，如

"San Martino, Maria" 或 "Maria {San Martino}"

都表示 M. San Martino 夫人.

对于姓中首字母没有大写的辅词，例如 *von* 或 de，则可以采用任一种方式输入：

"Richard von Mannheim" 或 "von Mannheim, Richard"

"Walter de la Maire" 或 "de la Maire, Walter"

放在大括号内的任意文本都当做一项处理，这可以避免有时候会出现的歧义问题，例如姓名中包含逗号或单词 and 。

"{Harvey and Sons, Ltd}"

如果在姓名前要加上 Junior，那会使事情变得更复杂。如果在 Jr 和名字之间有逗号，则就应该写成如下形式：

"Ford, Jr, Henry"

然而如果是逗号，那就必须把 Jr 看成是复姓的一部分：

"{Filmore Jr}, Charles" 或 "Charles {Filmore Jr}"

**BibTeX 0.99 版本或以后版本 ** 姓名中的用反斜杠命令构成的重音应该放在大括号内，并且反斜杠应该是紧接左大括号的第一个字符。通过这种方法，字母排序和在 alpha 参考文献样式中对标签的格式化都会工作正常。例如，

```
author = "Kurt G{\"o}del",
year = 1931
```

就会生成所希望的标签 [Göd31] 。重音命令被大括号包围的深度不能超过这里所示例的。（在所有版本的 BibTeX中，都可以把重音包含在姓名文本中，

其在参考文献中的显示是相当正确的。而这里新增加的功能就是对标签和字母排序的处理。)

**BibTeX 0.99 版本及以后 ** 名字中间有连字符，也能正确地缩写。因此 "Jean-Paul Sartre" 成为 'J.-P. Sartre'。(在以前的版本中，其结果为 'J. Sartre')。

如果 author 或者 editor 域中包含不只一个姓名，那么两个姓名之间要用 and 分开。例如，

author = "Helmut Kopka and Daly, Patrick William" 或者

AUTHOR = {Peter C. Barnes and Tolman, Paul and Mary Smith}

如果 and 真的是姓名的一部分，那么就必须把整个姓名放在大括号内，见前面的示例。如果作者清单太长，无法列出所有的姓名，那么可以用 and others 表示结束。根据样式文件的规定，这将会被转化为 *et al.* 。

题目

是否对题目首字母进行大写，与所用的参考文献样式有关：通常书籍的题目首字母要大写，而文章的题目则不这样做。在 title 和 booktitle 域中的文本应写成大写的形式，这样必要时 BibTeX可以把它转化为小写形式。

在使用英语的国家，大写题目的一般规则是：题目的第一个单词，冒号后的第一个单词，以及其它单词中除冠词，没有重音的介词及连词外都要首字母大写。例如，

title="The Right Way to Learn: A Short-Cut to a Successful Life"

那些总是首字母要大写的单词(例如专用名词)应该放在大括号内。实际上只要把总是要大写的字母放在大括号内就可以了。下面两个例子是等价的：

title = "The {Giotto} Mission to Comet {Halley}" 或

TITLE = {The {G}iotto Mission to Comet {H}alley}

§B.2.5 缩写

在输入域文本时通常可以使用缩写表示实际的文本。有些缩写，例如月份名称和某些标准期刊名称就可以使用缩写，当然用户也可以定义自己使用的缩写。

名称的缩写可以是字母、数字或符号的任意组合，当然下面的符号除外：

" # % ' ( ) , = { }

缩写是用下面的命令定义的：

@string{ 缩写名称 = { 文本 }} 或

@string(缩写名称 = { 文本 })

其中 *缩写名称* 就是缩写后的形式，而 *文本* 是替换文本。例如，如果定义了下面这样的缩写：

```
@string{JGR = {Journal of Geophysical Research}}
```

那么下面这两条声明是一样的：

```
journal = JGR
journal = {Journal of Geophysical Research}
```

缩写名称不要放在大括号或双引号内部，否则就会把它按字面解释成域文本。在 `@string` 命令和缩写名称中，都不区分大小写。因此上面的缩写也可以如下定义：

`@STRING{jgr = {Journal of Geophysical Research}}` 或者

`@StrinG{jGr = {Journal of Geophysical Research}}`

在域中就可以用 JGR, JGr, JgR, Jgr, jGR, jGr, jgR 或者 jgr 来引用它。

**BibTeX 0.99 版本及以后 ** 可以在缩写之间加上 # 来把它们结合在一起。因此如果定义了

```
@string{yrbk = {Institute Yearbook}}
```

那么这个缩写就可以同其它缩写或文本结合在一起，例如

```
title = "Max-Planck~" # yrbk # 1993
```

的结果为 'Max-Planck Institute Yearbook 1993' 。

月份名称是有标准缩写的，其由原来名称的前三个字母组成，例如 jan, feb 等等。对于某些标准期刊，存在着一些预先定义的名称缩写。要想知道到底有哪些缩写，那就需要查询机房使用手册了，因为这与安装版本有关。

`@string` 命令可以放在数据库中任两项条目之间，但是必须是定义在使用之前。因此把所有的缩写定义放在数据库开头就是相当合理的。

§B.2.6 使用模板

向参考文献条目中输入文本似乎是一件相当复杂的工作，因为要记住那么多的事情，比如哪些域是不可少的，哪些域是可省的，它们的格式怎样，不一而足。简化这件工作的一种方法就是为经常使用的条目创建 *模板* 。所谓模板，也就是一个基本完整的条目，只是所有的域都是空的。把模板存到单独一个文件中，只要想生成一项新条目，就把它插入到文件中。然后再输入域文本。

相应于 `@article` 条目类型的一个比较适当的模板可以是

```
@ARTICLE{<key>,
   AUTHOR  = {},
   TITLE   = {},
   YEAR    = {},
   JOURNAL = {},
```

```
        VOLUME  = {},
        NUMBER  = {},
        MONTH   = {},
        PAGES   = {}
    }
```

其中 <key> 是提醒用户这里必须要输入一个关键词，然后首先是不可少的域，并向里缩进，再后接可省域，并进一步缩进。这里没有包含域 note，因为虽然所有条目类型中都可以包含它，但很少会用到。

最后提一下，BibTEX是由 Stanford 大学的 Oren Patashnik 在 Leslie Lamport 的密切协助下开发的。安装了 BibTEX后应该有这两个文件 btxdoc.tex 和 btxhak.tex，其中就是一些关于 BibTEX的信息。特别地，btxhak.tex 中有关于编写参考文献样式文件的指令。用 LaTEX处理这两个文件，就可以看到其中的内容了。

§B.3 扩展 BibTEX

§B.3.1 作者–年代 参考文献样式

本书中所有的参考文献样式同绝大多数的自然科学期刊一样，与 LaTEX所提供的标准样式不同。在这种样式中，对其它出版物的索引是通过引用作者姓名和出版年代进行的。而且，引用既可以是放在方括号内的，如 [*Jones et al.*, 1990]，也可以是平铺直述的，如 *Jones et al.* [1990]。这种样式也可以有许多变体：例如，本书中是用圆括号取代了方括号，而且名称用的是直立字体。

在标准的 LaTEX中，引用是放在方括号内的一个活动编号，如 [10]，也允许逗号后面加上页码，如 [10, page 188]，但不能用文本引用。这里的编号也就是标签，并放到参考文献中，后接索引的文献信息。而 作者–年代 参考文献中没有标签，从而直接利用 作者 和 年代 可以容易知道引用与索引之间的关系。

实际上有许多非标准的作者 – 年代参考文献样式可以使用，所有这些样式都要利用一些宏包文件来重新设计 thebibliography 环境从而能解释修正后的 \bibitem 命令。

这其中最简单的一种样式就是由 BibTEX开发者 Oren Patashnik 所设计，称为 apalike.bst 和 apalike.sty。在这种样式中，\bibitem 有一个包含作者和年代的可省参数值，如

```
\bibitem[Jones et al., 1990]{jone90}
```

\cite 命令的使用与通常的一样，也就是只可以使用插入式的引用。至少有 8 个 .bst 文件遵从这种框架。

还有另外一组 LaTeX和 BibTeX样式，称为 newapa.sty 和 newapa.bst，由 Stephen N. Spence 开发出来， Young U. Ryu 对其进行了修改，最后一次更新是在 1991 年 6 月进行的。这是一个具有相当弹性的系统，引用既可以是插入式的，也可以是平铺直述的，而且作者的名单可以用全名，也可以缩写。这时 \bibitem 命令的形式为

```
\bibitem[\protect\citeauthoryear{Jones, Smith, and Harris}
                {Jones et al.}{1990}]{jone90}
```

\bibitem 命令的可省参数中的特殊命令 \citeauthoryear 是用来以三个独立项的形式向 LaTeX文档传递作者全名清单，缩写清单，年代。这条命令是在相应的 .sty 宏包文件中定义的。它同时提供了大量的 \cite 命令变体，用来显示名单与年代的组合，这既可以是插入式的，也可以是平铺直述的。有许多其它的参考文献样式，其中有引人著目的 chicago 族，也利用了这条命令。

而对于 newapa，也可以很容易地改变引用中的标点符号，比如说把 (Jones & Smith: 1990, pages 60–65) 改成 [Jones and Smith, 1990: pages 60–65]。这是利用 \citepunct 命令来做到的，利用它可以得到任何所希望的标点符号用法。

在参考文献样式中还有一个叫 astronomy 的家族，其中至少有 7 位成员。它们都应用了一条特殊命令 \astroncite，它实际上是与前面提到的命令 \citeauthoryear 一致，但它只有两个参数值，分别相应于简短作者名单和年代。要用它们需要宏包 astronomy.sty。

另外一组属于 作者–年代 参考文献样式的应当是 harvard 族了。它们利用 \harvarditem 命令取代了 \bibitem，但功能与 \citeauthoryear 一样。应用它们时，需要由 Peter Williams 和 Thorsten Schnier 所写的宏包 harvard.sty。这个宏包已经进行了更新，从而适用于 LaTeX 2_ε，现在它包含了许多新功能来定制引用。其已成为了一个功能强大的宏包，选用时，相比上面所提到的其它宏包，应当优先考虑这一个。

另外，也有一些样式是在 \bibitem 的可省参数中应用命令 \citename。然而，其语法与其它的不同，而且据我们的看法，其也没有涵盖所有可能的引用形式。

Patrick W. Daly 则完成了一个更一般的宏包，它接受上面所有 \bibitem 中 作者–年代 格式，也接受数字引用框架。这个宏包名称为 natbib.sty，它实际上是把年代在圆括号内的可省参数取作 作者–年代 的基本形式，

```
\bibitem[Jones et~al.(1990)]{jone90}
```

并用这种格式定义了所有其它 作者–年代 命令。而且也可以把所有括号和标点记号的类型，以及其它所需要的特殊功能，与 .bst 文件的名称联系起来，这样当调用 \bibliography 命令时就会调用它。因此所有的使用同样参考文

献样式的文档都会拥有正确的引用标点符号，甚至不用对作者部分做任何其它修改。而且对数字和 作者–年代 样式的选择是完全自动的。这个宏包已经相应于 LaTeX 2ε进行了更新，拥有与 harvard.sty 同样的功能，因此可推荐为所有参考文献样式的普适性宏包。

在 C.3.4 节给出了 natbib.sty 的一个简化版本，并同时给出一个基本的 作者–年代 参考文献样式。

所有在这里提到的样式，包括 natbib.sty 在内，都可以从 CTAN 文件服务器上的 BibTeX和 LaTeX相应目录中取到，我们在 D.5 列出了这些服务器，

§B.3.2 定制参考文献样式

好像每份期刊和出版社都有自己的参考文献格式规则。这其中的差异实际上是微乎其微的，比如说此处应该用逗号还是冒号，或者编号字体为黑体还是斜体。但是，虽然差异很小，每个出版社对格式的要求却是非常严格的。

标准的 LaTeX只提供了四种参考文献样式，它们的差别也就只限于排序和标签等琐碎的细节。我们可以通过修改已有的样式文件，来设计自己的 .bst 文件；然而，这需要对独特的 BibTeX程序设计语法有相当的了解，因为这种语法甚至会使有经验的 LaTeX用户也感到迷糊。在 TeX文件服务器上的 BibTeX目录中大约有 50 个 .bst 文件，而在 LaTeX相关的目录中可能有更多的此类文件。其中每个文件都是专对于某一期刊而设计的，因此无法保证能被另外的期刊所接受。如果出版社要求的格式，在那些地方却找不到，该怎么办呢？而且又应怎样在这些格式中搜索自己想要的格式呢？

从 custom-bib 宏包中可以得到一些帮助，这个宏包是由 Patrick W. Daly 写成，可以从文件服务器上的 LaTeX相关目录中找到。这个宏包的核心就是一个通用的（或者说主要的）参考文献样式 .mbs，其中包含了相当多的针对于不同参考文献功能或选项的替换代码。要用 DocStrip 程序处理它，这个程序能根据给定的选项来包含进来替换代码（见 D.3.1 节）。

由于提供了大量的选项（约有 100 个），因此设计了一个菜单驱动的界面，称为 makebst.tex。当用 TeX或 LaTeX处理这个“程序”时，它就会首先问你要读入哪个 .mbs 文件，接着就会交互构造出一个 DocStrip 批处理作业，以根据包含在 .mbs 文件中的菜单信息生成所选择的参考文献特征。这也就是说可以有很多 .mbs 文件供选择，每一个都通过 makebst 向用户解释自己的选项集合。

事实上，merlin.mbs 是第三代公开发表的参考文献样式，它取代了原来的 genbst.mbs 以及多语言部分 babel.mbs。对其它语言的支持现在是通过另外的 .mbs 文件提供的，每个相应于一种语言，其中包含对类似于 *volume*, *editor*, *edition* 等等单词的翻译。

由通用参考文献样式文件所提供的一些功能如下：

- 数字或 作者–年代 引用；对后者，用户可以选择所使用的 \bibitem 样式；
- 索引的顺序：排序可以按引用顺序，也可以按作者姓名的字母顺序，或者根据作者 – 年代标签字母顺序；
- 作者姓名的格式：名加姓，名首字母加姓，姓加名首字母，只把第一作者的首字母颠倒过来，等等。
- 在 et al. 前面可以给出的作者姓名个数；
- 排版作者姓名的字样；
- 日期的位置，是否把年份放在圆括号或方括号内；
- 期刊的卷，期以及页码的格式；
- 以句子或者标题样式对文章标题进行大写；
- 在单词间用 *and* 还是用 *&*；
- 用逗号取代单词 *and* ；
- 是否缩写 *editor*, *volume*, *chapter* 等；
- 给出起始，结束页，还是只给出开始页；
- 常见期刊名称的简写形式；
- ……

©TEXGuru, August 15, 1999

附录C　LaTeX程序设计

LaTeX 2ε的一个主要改进之处就是为类和宏包作者(简言之,即LaTeX程序开发者)提供大力支持。绝大多数用户并没有体味到新版本相比于LaTeX2.09的改进之处，只是知道多了一些新的字体命令，文档开头之处的声明换成了 `\documentclass`，用 `\usepackage` 命令上载原来的 *选项文件*。虽然在其它地方也有可能发现多了点新东西，但大致的感觉就是新版本并没有带来多少新事物。然而，对LaTeX程序开发者而言，尤其是对曾经编写过选项文件，或者安装过新字体框架的人员来说，他们就会非常欣赏NFSS以及新的类与宏包控制功能。随着时间的推移，基本LaTeX安装所提供的新扩展功能逐渐变得与LaTeX 2ε密不可分，从而想使用新功能的用户就不可避免地把自己的系统更新到新版本上。

在8.5节中描述了新字体选择框架。本附录讲解设计类与宏包文件的特殊命令，并给出了几个实用宏包的设计示例。

§C.1　类与宏包文件

§C.1.1　LaTeX2.09 中的样式文件

类与宏包文件是新出现在LaTeX 2ε中的概念，它取代了LaTeX2.09中的 *主样式* 和 *样式选项* 文件。在原来的系统中，要想选择主样式，就要利用声明

2.09 `\documentstyle[`*选项清单*`]{`*样式*`}`

这样就上载了一个名为 *样式*`.sty` 的文件，其定义了文档所需要的全局格式。几个主要的样式为 `article`, `report` 和 `book`，这也是LaTeX 2ε中最重要的几个类。在样式文件的某个地方，要用命令 `\@options` 如下处理 *选项清单* 中的选项：

1. 如果存在命令 `\ds@`*选项*，那就执行这条命令；否则，就把该选项放到第二个清单中；
2. 当遍历完 *选项清单*,就考虑第二个清单，对每一项，把文件 *选项*`.sty` 输入进文档。

这一过程使得选项可以定义在样式文件内部，也可以存贮在与选项同名的单独样式选项文件(扩展名为 `.sty`)中。这样做的想法就是有些选项的代码与主样式文件无关，从而可以存贮在外部的样式文件中。

这就可能是采取选项文件概念的初始动机；其直接结果就是有大批的爱好者开发了数目可观的添加‘选项’，并把它们存贮在一个文件中，读入到其它的文档中。通过网络服务器，添加的选项就传播开来，得到广泛的应用。由于其中有些只是拙劣地修补了LaTeX的内部命令，不能只是简单地用 `\input`

命令把它包含进来，而要把文件名后缀指定为 .sty，因此可以按照上面的第 2 点以准选项角色包含进来。但它们实际上并不是真正的选项，只是新增加的代码或功能。

在编写主文件样式时还会出现一个新的难题。如果需要适合于某期刊的文章样式，或者某出版社的书籍样式，那么可以利用已有的 article.sty 或 book.sty 样式为基础，进行必要的改动，得到新的主样式，但是不能保证在对原主样式做更新后，改动仍然有效。但是有时候一些重大的更新就与对LaTeX进行的改动是一致的。我们可以只是写一个 “选项” 来包含所做的改动，也可以写一个新的主样式，输入原来的样式。然而，它并不是相应于其选项清单的真正主样式。

§C.1.2 LaTeX 2ε的新概念

当 Leslie Lamport 发布 LaTeX时，他无论如何也不会预见到随后出现了那么多的 LaTeX程序。这一切现在已成为不争的事实，而且这也是该系统的魅力所在。LaTeX 2ε则不但包容这些 “外来” 成员，而且实际上它进一步支持和鼓励这种趋势，其中一个明证就是在 *The LaTeX Companion* (Goosens et al., 1994) 一书所介绍的大量宏包。

这就是事物发展的趋势。扩展的功能由那些需要该功能的人设计的，因为他们意识到 LaTeX缺少了某些对他们而言是很重要的功能。另一方面，要是把所有这些扩展的功能都加入到基本的 LaTeX安装中，那就会使得 90% 以上的用户虽然上载了它们，但从来不会用它们。现在解决这个问题的方法就是 LaTeX提供了一个基本的核心 (或称内核)，再首先用标准的类文件扩展其功能，然后利用那千变万化的宏包和类增加功能。

而 LaTeX Team 的任务就是建立程序设计的方针，从而确保宏包不会与内核或者其它宏包发生不必要的冲突，而且提供一种基本的稳定性，使得那些实用的宏包将来在更新后的内核和标准类下也工作正常。在 LaTeX2.09 中就缺少了这种安全机制，在那个版本中，程序设计者们被迫自己寻找门路，这实际上并不是真的程序开发。 LaTeX 2ε在类和宏包控制方面的新功能，随同一组程序开发工具，应该在宏包内部相互作用以及对内核更新的适应等方面达到比原来更强的可靠性和持久性。

§C.1.3 命令的层次

命令有许多层次，它们的安全程度也相应不同。

用户命令 (最高级命令) 在本书及其它手册中进行了描述，其名称由小写字母组成，例如 \texttt，它是永久被支持的 LaTeX外部定义；

类与宏包命令 其名称要稍长一些，而且大小写混杂 (如 \NeedsTeXFormat)，主要是为程序设计人员提供的，而且也是有保障的；绝大多数是只能用

在导言中的命令，但在类和宏包文件中并没有这个使用限制；

LaTeX内部命令 名称中包含 @ 字符，只能用在类与宏包文件中；虽然其中有些命令对得到特殊效果是密不可分的，但也无法保证永远可用；开发人员要使用该命令，那就有可能将来某一天自己设计的宏包变得不再能用了；

TeX低级命令 名称也是由小写字母组成，而且没有 @；即使 LaTeX继续演化，其功能也应该是稳定的，但这也不是绝对的；只要有可能，就尽量避免使用它们，见下面的解释；

内部专用命令 是用在其它人员开发的类与宏包文件内部的命令；建议所有命令都前缀大写字母 (以表示宏包的名称) 后接 @，这样可以避免与其它宏包发生冲突；例如， showkeys 宏包中有一条命令为 SK@cite 。

一个令 LaTeX开发人员感到迷惑的问题是 LaTeX内部命令在类和宏包文件中的应用范围到底有多大。总是存在着可能，将在某一天在新版本不存在这些新命令了，因为在正式的说明书中从没有给出其说明。而诸如下节讨论的 TeX命令，我们不应杜绝使用它们，但我们必须明白，用它们也同时伴随着一定程度的风险。

LaTeX Team 建议的方针就是只要有可能，就尽量使用 LaTeX高级命令。

- 要用 \newcommand 和 \renewcommand，而不用 \def；如果要使用某一个 TeX的定义命令 (因为调用某模板，或者因为必须用 \gdef 或 \xdef)，那么先调用一个空的 \newcommand 命令，以检测名称是否有冲突。如果无法确定命令名称是否存在，而且该命令不是很重要的，那么就调用一条空的 \providecommand，然后再调用 \renewcommand 。现在高级命令中可以定义有一个缺省值的命令，这使得原来经常要用低级命令的理由中缺少了重要的一条。
- 利用 \newenvironment 和 \renewenvironment 命令，而不用 \自己的环境 和 \end自己的环境 命令对。
- 要用 \setlength 命令给长度和橡皮长度赋值，而不要用直接等号方式。
- 避免使用 TeX盒子命令 \setbox, \hbox 以及 \vbox；而要用诸如 \sbox, \mbox, \parbox 一类的命令。利用 LaTeX 2_ε提供的可省参考值，原来对等价 TeX命令的需求现在大大降低，而且 LaTeX版本相比起来要透明得多。另外，当用了 color 宏包时， LaTeX的盒子仍然工作正常，而其它的命令结果就无法预料了。
- 如果想给出错误和警告消息，就用 \PackageError 和 \PackageWarning, 不要用 \@latexerr 或 \@warning；前面两条命令也同时告诉用户消息的来源，来不是只把它们标为 LaTeX消息。
- 我们不会建议你只使用 ifthen 宏包 (7.3.5 节) 中的 \ifthenelse 命令，以代替 TeX的条件命令。但是似乎用这个宏包可以简化对条件的应用，

而且符合 LaTeX的语法。本书所有的例子都用的是这个宏包，这就不需要再解释相应的 TeX命令。

遵从这些以及与之类似的规则，有益于在将来即使 LaTeX内核进行了更新，宏包也能继续保持有效的功能。

§C.1.4 TeX命令

为什么要避免用那些基本的 TeX命令呢？要想定义一条新命令，如果用 \def 就至少不会比用 \newcommand 差，而且有时候还必须用前者。那么在将来出现的 LaTeX3 中这条命令有可能被去掉吗？基本命令 (原语) 是所有 LaTeX风味得以建立的构造模块，因此它们一定会维持不变的。

而这并不是这样做的关键所在。基本 TeX命令构成了所有格式的基石，从而所有用它们直接定义的命令，其功能就永远与开发人员所期望的一样。然而，等价的 LaTeX工具所能做的事情却会随着时间的发展而增多。例如，\newcommand 命令可以检测新定义的命令是否与已有命令发生冲突。而且，以后完全有可能会加进一种调试机制，它可以跟踪所有的重定义；而用 \def 定义的命令则是排它的。而且现在的 LaTeX 2_ε中也有一种机制，它可以跟踪所有中级和高级命令的文件输入。

另外一个低层次程序开发人员容易误入歧途的例子的就是 LaTeX 2_ε中牢固命令处理的情形。有很多命令本质上是脆弱的，也就是说当把它们用做其它命令的参数值时，会过早被解释，可如果给它们前缀 \protect，一般可以使其变得牢固。在 LaTeX2.09 中，有几条脆弱命令在定义时采用的是一种牢固方式，即定义中包含了 \protect，例如 LaTeX的标志命令：

```
\def\LaTeX{\protect\p@latex}
\def\p@latex{...}
```

真正的定义是在内部的 \p@latex 中，并不是在外部的 \LaTeX 中。由于如此的标志定义中有很多缺陷，因此有几个宏包引入了一个改进的版本。它们只是重定义 \LaTeX，从而使得这条命令变成脆弱的了；因此聪明的方法是重定义 \p@latex，这就利用了隐藏在 LaTeX 2_ε后台的结果，但是命令却以一种完全不同 (而且更好) 的方式变得牢固了。(顺便说一下， LaTeX标志的内部定义已有了很大的改进。)

虽然我们希望只使用正式发布的 LaTeX命令，但也有很多情形，我们必须用 LaTeX内部命令或者 TeX基本命令。在目前阶段，为了得到一个工作稳定的宏包，我们就必须考虑在将来可能会不兼容的风险。而且如果有等价的高级命令可用，我们就不应冒这个风险。

©TeXGuru, August 16, 1999

§C.2 LaTeX 2ε程序设计语言

本节描述的所有命令都是在 LaTeX 2ε中新增加的。虽然它们对类和宏包文件而言，并不是很本质的，但它们确实扩展了类与宏包的用途，并保证使用时的正确性。

§C.2.1 文件识别

有三条命令用来测试类或宏包插入时所处的外部环境是否正确。其中第一条为

2ε `\NeedsTeXFormat{`格式`}[`版本`]`

在类或宏包中的第一条语句就应该是所需要的 TeX格式声明。虽然已有很多其它名称的格式，但只有名为 LaTeX2e 的格式才认识这条声明。而所有其它格式都会给出错误消息：

```
! Undefined control sequence.
1.1 \NeedsTeXFormat
                      {LaTeX2e}
```

此时，这条消息实际上就给出了提示信息。

在 LaTeX 2ε中这条命令可能更有用的地方是它的可省参数值 版本，该参数的形式必须为表示发行日期的 yyyy/mm/dd。如果一个宏包利用的功能是在某一个版本中才引进的，那就必须给出这个日期，因此如果用的是 LaTeX 2ε一个更早的版本，就会显示出一条警告。例如，在 LaTeX 2ε起初的测试版本中并没有提供命令 `\DeclareRobustCommand`，只是在 1994 年 6 月 1 日正式发行时才有了这条命令。因此使用了这条命令的宏包就应该以下面这条语句开头：

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
```

这里日期的形式是很重要的，必须有零和斜杠。

这条声明并不是只限于用在类和宏包文件中，也可以在文档开头调用它，以保证用正确的 LaTeX处理该文档。然而调用的位置就必须是在导言中。

下面两条命令用来标明类或宏包文件自身：

2ε `\ProvidesClass{`类`}[`版本`]`

2ε `\ProvidesPackage{`宏包`}[`版本`]` 在这两条命令中，版本 都是由三部分组成的：日期，版本号以及附加信息。日期与上面的格式相同，而版本号可以是任何没有空格的标志，附加信息可以是有或没有空格的文本。例如，

```
\ProvidesPackage{shortpag}[1995/03/24 v1.4 (F. Barnes)]
```

LaTeX只会检查其中的日期部分，也就是说如果使用了 `\usepackage` 命令并给出了日期，那么 LaTeX就会对两处的日期进行比较。如果调用了 `\listfiles`，

那么会显示出来版本号和附加信息部分。然而，对于 doc 宏包 (D.3.2 节) 中的 \GetFileInfo 而言，上述格式就是必须的了。

\documentclass 和 \usepackage(以及 \LoadClass 和 \RequirePackage) 命令都可以包含一个可省参数，以指定类 / 宏包可接受的最早发行日期。例如，当声明为

```
\documentclass[12pt]{article}[1995/01/01]
```

这时如果使用的 article 类文件中包含

```
\ProvidesClass{article}[1994/07/13 v1.2u
   Standard LaTeX document class]
```

那么就会显示出一条警告消息。对于 \usepackage 和 \ProvidesPackage 命令，也是同样的机理。

版本检测机制使得文档可以索取处理自己的合适版本的类和宏包文件。当然这里要假设所有以后版本都与以前的版本完全兼容。

对于那些用 \input 命令上载的普通文件，还有一条识别命令：

2ε `\ProvidesFile{`*文件名*`}[`*版本*`]`

此时不会检测名称或版本，但是利用 \listfiles 可以列出这两部分信息。

§C.2.2 上载其它类和宏包

在主文档文件中，类的读入是利用初始化 \documentclass 命令来实现的，而宏包用的则是 \usepackage 命令。在类和宏包文件内部，就必须使用下述命令：

2ε `\LoadClass[`*选项*`]{`*类*`}[`*版本*`]`

2ε `\RequirePackage[`*选项*`]{`*宏包*`}[`*版本*`]`

其中第一条命令可使得一个类文件上载另一个类文件，并且需要的话，可以给出选项；而第二条命令使得类和宏包文件上载其它的宏包。在任何类文件中只能有一条 \LoadClass 命令；不能在宏包文件中使用。这两条命令都可以用在文档文件中。其中的 *宏包* 参数值可以是几个宏包名称组成的清单，中间用逗号分开。

可省 *版本* 参数与相应的 \Provides.. 命令之间的关系在前一节中做了介绍；而我们下面将介绍 *选项* 参数的处理方式。

§C.2.3 选项的处理

在类和宏包中都可以有选项，其定义方式为

2ε `\DeclareOption{`*选项*`}{`*代码*`}`

其中 *选项* 就是选项的名称，而 *代码* 就是选项要执行的指令集。在 LaTeX内部，实际上创建了一条叫 \ds@*选项* 的命令。通常这些代码并不做任何事，只

是设置一些标志，或输入一个选项文件。(\RequirePackage 不可以用在选项代码中！) 在 article.cls 文件中的两个示例为

```
\DeclareOption{fleqn}{\input{fleqn.clo}}
\DeclareOption{openbib}{\setboolean{\@openbib}{true}}
```

可以用 \DeclareOption* 定义一个缺省选项，这条命令并不需要选项名称，只是指定适用于所有被调用的未定义选项的执行代码。

有两条特殊命令，可能用在缺省选项定义的 *代码* 中：

\CurrentOption 由正在被处理的选项名称组成；

\OptionNotUsed 把 \CurrentOption 声明为未处理的。

例如，若想有一个类文件，模拟 LaTeX2.09 在所有未定义选项上载同名的 .sty 文件时的行为，可以如下定义：

```
\DeclareOption*{\InputIfFileExists{\CurrentOption.sty}%
                {}{\OptionNotUsed}}
```

这样就会首先检测是否存在指定名称的 .sty 文件，如果不存在，就把选项声明为没有使用的。要求的选项没有使用（处理）的话，就会列在一条警告消息中。

接下来就用下面的命令处理选项：

2ε \ExecuteOptions{ *选项清单* }

2ε \ProcessOptions

2ε \ProcessOptions*

其中 \ExecuteOptions 会为 *选项清单* 中的每个选项调用 \ds@*选项* 命令。在默认方式下通常就是建立起特定的选项配置。 \ProcessOptions 按照所有选项定义的顺序执行调用的选项，然后删除它们。这也就是说这条命令只能执行一次。有星号的命令功能类似，只是它是按调用的顺序执行。为了与 LaTeX2.09 样式兼容，现在仍然保留了命令 \@options ，它只是 \ProcessOptions* 命令的另一个名称而已。

也可以用下面的命令为类或宏包定义选项：

2ε \PassOptionsToClass{ *选项* }{ *类名* }

2ε \PassOptionsToPackage{ *选项* }{ *宏包名称* }

其中 *选项* 是一串指定类或宏包文件可以识别的合法选项。这两条命令可以用在其它选项的定义中。最常见的用法就是把缺省选项传递给另一个类，如下例所示：

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
```

这里所定名的类或宏包必须稍后用 \LoadClass 或 \RequirePackage 命令进行上载。

如果类和宏包文件的缺省选项并没有用 \DeclareOption* 进行改变，那么处理未定义的被调用选项的标准程序如下：

- 所有在 \documentclass 语句中的选项标记为 *全局的*; 认为其要应用于后面所有宏包，但用 \LoadClass 上载的类除外；如果在主类中没有定义该选项，不会给出错误或警告消息；
- 所有其它语句（包括 \LoadClass 和 \PassOptionsTo..）给出的选项都是 *局部的*；如果在相应的类或宏包中没有定义该选项，会给出一条错误消息；
- 当所有宏包都读进来后，如果某一个全局选项还从来没有用过（从没有被定义），就会给出一条警告消息；
- 无论是全局选项，还是局部选项，都按照定义的顺序执行（除非调用了 \ProcessOptions* ）。

§C.2.4 延期处理

有时候，为了得到某种特殊效果，或者避免与其它宏包发生冲突，希望有些命令是在类或宏包结束处执行，或者在文档的开头或结尾处执行。这可以用下面的命令实现这种效果：

2ε \AtEndOfClass{ 命令集 }

2ε \AtEndOfPackage{ 命令集 }

2ε \AtBeginDocument{ 命令集 }

2ε \AtEndDocument{ 命令集 }

前两条命令把 *命令集* 保留到类或宏包结尾时才执行。配置文件利用它们在开始部分读入，但它由在结尾处所应进行的修改组成，这样在结束时就不会由于缺省方式而被重写。后两条声明把 *命令集* 分别保留到 \begin{document} 和 \end{document} 中执行。上述所有命令都可以不只调用一次，这样 *命令集* 执行时就会按照调用的顺序进行。

保存在 \AtBeginDocument 中的 *命令集* 是会准确地插入在导言的处理流中，但它是在 \begin{document} 命令已几乎做完自己该做的事情之后。因此可以认为 *命令集* 是正文的一部分，但是那些只能用在导言中的命令也可以用在其中。

§C.2.5 牢固的命令

在 7.3 节中我们详细讲述了如何使用 \newcommand 定义新的命令，使用 \renewcommand 重定义命令，以及使用 \providecommand 临时创建命令。实际上还有另外两条语法相同的定义命令的语句：

2ε \DeclareRobustCommand{ *命令名* }[*参数个数*][*可省参数*]{ *定义* }

用来定义或重定义一个名为 *命令名* 的命令，而且它是 *牢固的*，也就是说它可以用做其它命令的参数值，前面不需前缀 \protect 。如果命令已经存在了，就会向抄本文件中写入一条消息，并覆盖原来的定义。

另外一条命令可以用来检测 \命令名 的当前定义。

2ε \CheckCommand{ 命令名 }[参数个数][可省参数]{ 定义 }

如果命令的定义与 定义 不同，或者参数个数不同，等等，就会给出一条警告消息。这可以用来确认系统的状态是我们所希望的样子，并不存在已上载的宏包修改了某些重要的定义。

\DeclareRobustCommand 和 \CheckCommand 命令都可以用在文档的任何地方。

§C.2.6 有短参数值的命令

通常在用户定义命令的参数值中可以包含用 \par 或空行表示的新段落。按 TeX的行话来说，这些命令都是 '长' 的。这也不是用 \def 定义命令的标准行为，因为用它定义的命令必须是短的，这样可以检测是否遗漏了右大括号。

到了 1994 年 6 月 1 日，新发行的 LaTeX 2ε提供了所有定义命令的星号形式：

```
\newcommand*              \renewcommand*
\newenvironment*          \renewenvironment*
\providecommand*
\DeclareRobustCommand*    \CheckCommand*
```

上述命令创建的命令都具有 '短' 参数，从而行为与 \def 一样。

我们建议总是用星号形式命令来定义新的命令，除非有足够的理由取某些参数为 '长' 的，即参数中包含段落。长参数应当是例外，而不是规则。

§C.2.7 给出错误和警告消息

在设计类和宏包时，也应当使它们具有自己的错误和警告消息。这对帮助我们辨别到底哪个文件发出这条消息是非常有用的。

错误消息是用下列命令生成的：

2ε \ClassError{ 类名 }{ 错误消息文本 }{ 帮助 }

2ε \PackageError{ 宏包名 }{ 错误消息文本 }{ 帮助 }

其中 错误消息文本 就是显示在监视器或抄本文件中的消息，而 帮助 就是当用户反映为 H 时显示的进一步文本。如果文本中包含命令名，而且要按原样显示，那就必须前缀 \protect；空格是用 \space 生成的，新行用 \MessageBreak 开始。例如，

```
\PackageError{ghost}{%
  The \protect\textwidth\space is too large\MessageBreak
  for the paper you have selected}
  {Use a smaller width.}
```

就会生成如下错误消息：

```
! Package ghost Error: The \textwidth is too large
(ghost)                for the paper you have selected.

See the ghost package documentation for explanation.
Type H <return> for immediate help.
```

当LaTeX停下来等用户给出一个反应时，若按9.1节中描述那样输入H⟨回车⟩，就会得到

```
Use a smaller width.
```

在类和宏包中也可以按类似的方法给出警告消息。差别就在于后者没有*帮助*文本，而且处理过程也不会停下来等待反应。可以包含警告消息出现时在输入文件中所处的行号。

2ε \ClassWarning{*类名*}{*警告消息文本*}

2ε \ClassWarningNoLine{*类名*}{*警告消息文本*}

2ε \PackageWarning{*宏包名*}{*警告消息文本*}

2ε \PackageWarningNoLine{*宏包名*}{*警告消息文本*}

例如，当定义了

```
\PackageWarning{ghost}
   {This text is haunted}
```

就会得到了消息

```
Package ghost Warning: This text is haunted on input line 20.
```

而且处理不会停下来。警告消息可以用 \MessageBreak 分成几行，这一点与错误消息中类似。

此类型的最后两条命令是

2ε \ClassInfo{*类名*}{*信息文本*}

2ε \PackageInfo{*宏包名*}{*信息文本*}

它只把文本写到抄本文件中，不会显示在监视器上。从其它角度来看，就如同没有 NoLine 的警告消息。

§C.2.8 输入文件

不是类和宏包的文件也可以输入到文档中，当这样做的时候，通常必须事先保证文件是存在的。或者，根据文件是否存在来决定要采取的进一步行动。这一目标是用如下命令实现的。

2ε \IfFileExists{*文件名*}{*真*}{*假*}

2ε \InputIfFileExists{*文件名*}{*真*}{*假*}

这两条命令都会在LaTeX文件搜索路径中看看有没有指定的*文件名*，如果找到了文件，就执行*真*，否则就执行*假*。而且，在 \InputIfFileExists 命令

中，执行了 真 后还会读入该文件。

这些命令并不限于只用在导言中，也不限于只用在类或宏包文件中。实际上，通常的 \input 命令就是用它们定义的。

许多特殊的类利用这些命令读入局部的配置文件。例如，在类 ltxdoc 中包含语句

```
\InputIfFileExists{ltxdoc.cfg}
    {\typeout{Local config file ltxdoc.cfg used}}
    {}
```

它就放在 \ProcessOptions 前面。这样可以有一个局部配置文件，其相应于欧洲安装版本，指定

```
\PassOptionsToClass{a4paper}{article}
```

而不用修改用 ltxdoc 类处理的文件。

§C.2.9 检测文件

我们在此描述两条跟踪使用文件的命令，它们并不是程序设计的一部分。其中第一条命令就是我们在 8.1.1 节已提到的命令

2ε \listfiles

这条命令可以放在导言中，甚至 \documentclass 命令的前面。在处理过程结束后，它会生成并显示出所有输入文件的清单，同时包括文件的版本和发行数据。用这种方法，我们就可以得到所有被包含进来的文件记录，当要把一个文件送到另外的地方，用不同的安装版本进行处理时，这一记录信息就可能非常有用。由于非标准文件也有可能被包含进来，那么从上面的清单中可以很容易识别出来。

例如，输入下面这个简单的文档文件：

```
\documentclass{article}
\usepackage{ifthen}
\listfiles
\begin{document}
  \input{mymacros}
   This is \te.
\end{document}
```

就会得到如下的清单

```
*File List*
article.cls 1994/07/13 v1.2u Standard LaTeX document class
 size10.clo 1994/07/13 v1.2u Standard LaTeX file (size option)
 ifthen.sty 1994/05/27 v1.0i Standard LaTeX ifthen package (DPC)
```

```
mymacros.tex
 **********
```

在这种情形里，局部文件 mymacros.tex 中不包含版本信息，因为其中没有用 \ProvidesFile 命令。

如果要把上面这个文档文件关到其它地方进行处理，那么该如何处理像上面 mymacros.tex 那样的局部文件呢？当然可以与主文件一起寄送该局部文件，那么这就需要告诉收件人更多指令，以确定该如何行事。另外一种方法就是为了寄送文件，把局部文件内容直接包含在主文件中。对于宏包文件，这样做可就不是那么容易了，因为内部命令中有 @ 符号，会造成一些麻烦，从而不能正确处理一些选项。现在 LaTeX 2ε提供了如下环境：

2ε `\begin{filecontents}{`文件名`}`

文件内容

`\end{filecontents}`

这个环境可以用在文档开头，即 \documentclass 命令的前面。这个环境首先会检测系统中是否存在一个文件，名为 文件名，如果不存在，它就会把 文件内容 照原样写到那个名称相应的文件中。该文件可以是一个宏包，随后要用 \usepackage 上载它。利用这种方法，少掉的非标准文件就可以与主文档文件一起寄送了。

我们推广上面那个简单例子，在开头部分输入

```
\begin{filecontents}{mymacros}
\newcommand{\te}{the end}
\end{filecontents}
```

那么新生成的 mymacros.tex 内容为

```
%% LaTeX2e file `mymacros'
%% generated by the `filecontents' environment
%% from source `mydoc' on 1994/09/27.
%%
\newcommand{\te}{the end}
```

注意 filecontents 环境加进了一些注释行，说明新文件来自于何处。如果不希望加进这些注释行，那可以用 filecontents* 环境。

§C.2.10 兼容模式

为了使得专门为 LaTeX2.09 编写的老文档也可以用 LaTeX 2ε进行处理，现在存在着一个 兼容 模式，它用 \documentstyle 取代 \documentclass 。这样整篇文档就可以只遵从原来的标准，但不能使用 LaTeX 2ε的所有新功能。

然而，兼容模式仍旧上载新的类文件，而不是原来的样式文件，因为这

些样式文件将来可能不会存在的。该模式首先查找后缀为 .cls 的文件，只有当该文件不存在时，它才会上载 .sty 文件。这就是为了迁就原来的非标准样式，使其功能就像类一样。

类似于 article 这样后缀为 .cls 的标准类文件，甚至用 \documentstyle 也可以上载。但是这时其功能必须与将要废除的 .sty 文件功能一样。实际上它们是有很多差别的，例如如何设置页边和文本尺寸。不但如此，而且我们要求兼容模式的输出必须与用 LaTeX2.09 以及 article.sty 处理时完全一样。为了做到这一点，在兼容模式中把 boolean 开关 (7.3.5 节)@compatibility 设置成 ⟨真⟩，这样任一类或宏包都可以检测该模式。检测的形式为

\ifthenelse{\boolean{@compatibility}}{ 真 }{ 假 }

其中 真 表示只适用于兼容模式的命令，而 假 表示那些可以用在真正 LaTeX 2ε 模式中的普通命令。

对于所有的标准类文件，仍然存在着 .sty 文件，例如 article.sty，只不过内容是空的，它只是给出一条警告消息，并上载类文件。这是为了兼容那些旧宏包，它们有可能显式上载这些文件。

§C.2.11 有用的内部命令

尽管在 C.1.3 节中的方针建议最好避免用 LaTeX内部命令，但是其中有些命令是非常基本的，它们组成了 LaTeX内核和许多标准宏包的构造模块。由于它们终究还是内部命令，因此无法保证在以后的更新中它们还是存在的。然而，如果真的没有了它们，那么由 LaTeX Team 所提供的大量的有趣扩展功能宏包就要进行全面的大检修。我们这里只是为那些勇敢的用户们简单地介绍一下这些命令。在 LaTeX 2ε和 2.09 中都存在这些命令：

\@namedef{ 命令 }{ 定义 }

\@nameuse{ 命令 }

就会定义并执行名为 \命令 的新命令，其中命令名称中并不包含反斜杠。这个名称中可以包含任意字符，即使通常在命令名称中禁止使用的字符也可以。

\@ifundefined{ 命令 }{ 真 }{ 假 }

如果 \命令 不存在，就执行 真 ，否则执行 假 。同样这里 命令 中也不包含反斜杠，而且任何字符都可以出现在命令名称中。这个检测语句通常用来有条件地定义命令，其功能现在已经被 \providecommand 代劳。也可以用它确定主类是否是 article： \@ifundefined{chapter}{..}{..} 可以用来检测 \chapter 命令是否存在。

\@ifnextchar 字符 { 真 }{ 假 }

用来检测下一个字符是否是给定的 字符 ，如果是的话，执行 真 ，否则执行 假 。这条命令通常用来定义有可省参数的命令，这时 字符 就是 [。新扩展的 \newcommand 命令用高级方法得到了这一效果。

`\@ifstar{ 真 }{ 假 }`

用来检测下一个字符是否是星号 `*`，如果是的话，就执行 *真*，否则执行 *假*。可以用它来定义带星号的命令和环境，这是高级命令做不到的。

`\@for \ 对象 := \ 列表 \do { 命令 }`

其中 `\列表` 就是一条命令，被定义成一串用逗号分开的元素，而 `\对象` 就相继取值等于每个元素，并对每个元素执行一次 *命令* 代码。例如，

```
\newcommand{\set}{start,middle,end}
\@for \xx := \set \do {This is the \xx. }
```

就会显示出 'This is the start. This is the middle. This is the end.'

§C.2.12 有用的 TeX命令

LaTeX中许多复杂的功能以及宏包都只能用 Plain TeX命令开发出来。这些命令的描述不但可以在 Knuth 的 *The TeXbook*(1984 年) 一书中找到，而且还有一本相当好的参考书，那就是 Eijkhout 的 *TeX by Topic, a TeXnician's Reference* (1992 年)。

我们不打算要把本书写成 TeX手册；不但如此，而且常用的出现在许多宏包以及后面示例中的 TeX命令数目也很少。只要给出一个简短的描述，就对理解这些命令的作用非常有帮助。真正的 TeX专家和 TeX技术人员可以略过这一节。

`\def\ 命令 #1#2..{ 定义 }`

是 TeX中标准的定义命令。它等价于 `\newcommand`，只是它不会检查是否有名称冲突，而且参数的指定也不同。例如，显示科学记数法的命令 `\Exp` 可以如下定义：

```
\def\Exp#1#2{\ensuremath{#1\times10^{#2}}}
```

或者

```
\newcommand{\Exp}[2]{\ensuremath{#1\times10^{#2}}}
```

对于这两种定义方式，`\Exp{1.1}{4}` 的结果都是 1.1×10^4。然而，`\def` 还可以做得更多。它可以把参数放在一个模板中，例如

```
\def\Exp#1(#2){\ensuremath{#1\times10^{#2}}}
```

这样可以得到更方便的记号 `\Exp1.1(4)`，而这是 `\newcommand` 命令所做不到的。当定义命令时，不知道 (或不关心) 是否已存在同名命令时，或者要用模板时就用 `\def`。有可省参数的命令实际上就是用模板定义的。

`\gdef \edef \xdef`

是 `\def` 的变形；第一条命令给出一个全局定义，所得命令即使在当前环境或者 `{..}` 括号对外面也仍然有效；第二条命令是一个展开的定义，其中任何命令都具有自己本身的意义，而并不是把命令插入在定义中；最后那条是前面两条的组合，即展开的全局性定义。

`\noexpand` `\expandafter`

控制命令在定义和执行时的展开。在 `\edef` 中定义部分的任何命令都要被展开(插入其含义)，除非其前缀 `\noexpand` 。相反的效果可以用 `\expandafter` 得到，这条命令跳过后接命令，展开下一条命令，然后执行被跳过的命令。这就是相当深奥的 TEX技术了，我们最好还是用上面那条 `\Exp` 命令来演示一下结果。

```
\newcommand{\mynums}{1.1(4)} \expandafter\Exp\mynums
```

就等价于 `\Exp1.1(4)` ，而 `\Exp\mynums` 并不等价；在执行 `\Exp` 之前先把 `\mynums` 展开成 `1.1(4)` 。

`\let\` 命令一 `=\` 命令二 或 `\let\` 命令一 `\` 命令二

使得 `\`命令一 取 `\`命令二 当前含义。这通常用来在重定义命令前保存其原来定义，从而可以同时使用其原来定义。

`\relax`

绝对不做任何事，但通常用来插在应该有些什么东西的地方，但我们不想有内容的地方。

`\if` 条件 真 `\else` 假 `\fi`

就是 TEX中条件语句的形式。有许多不同形式的条件，我们这里就不一一介绍了，但常见的应用就是等价于 LATEX 的 boolean 开关命令：

```
\newif\if 标志          = \newboolean{ 标志 }
\ 标志 true             = \setboolean{ 标志 }{true}
\ 标志 false            = \setboolean{ 标志 }{false}
\if 标志 ..\else..\fi = \ifthenelse{\boolean{ 标志 }}{..}{..}
```

对那些经常用这些语句的人而言， TEX形式要紧凑一些，但它并不与一般的LATEX行事方式一致。

`ifcase` 数 文本 0 `\or` 文本 1 `\or...\fi`

会根据 数 的值来决定执行哪个 文本 。

`\endinput`

终止对当前文件的输入。这并不是必需的方式，但所有的文件都用它结束不失为一个好的程序设计习惯。在主文档中不必用它，因为 `\end{document}` 可以得到同样的效果。

§C.3 宏包示例

我们下面给出几个示范性宏包，以说明前面一节给出的程序设计命令的使用方法。这些宏包并不是微不足道的，从某种角度来看，它们都是相当有用的。

§C.3.1 修改文本尺寸

在标准LaTeX类中，要根据 \documentclass 中指定的尺寸选项（如选项 a4paper 或 legalpaper 来设置文本尺寸参数 \textwidth 和 \textheight 。而且同时调整页边距，使得文本水平和竖直居中。\textwidth 的值是有限制的，每行上最多有 60–70 个字符，这主要是出于排版中达到最佳视觉效果的考虑。

而有时候我们想去掉这个限制，充分利用纸张的最大幅度。只有通过多次尝试，我们才有可能找到恰当的参数组合，匹配特定的纸张尺寸。如果有一个宏包可以帮你做到这一点，那就太好了。

下面给出的 fullpage 宏包利用了 \paperwidth 和 \paperheight 的值（这两个值是要根据纸张尺寸选项进行设置的），以生成两边只有一英寸页边的页面。它甚至可以考虑到当前页面样式，从而为必要的页眉和页脚留下空间。如果通过选项，可以得到只有 1.5cm 窄的页边。

在继续下面的设计之前，我们有必要先复习一下在 31 页和 ?? 页插图中的页面格式参数。

这个宏包的开头部分就声明自己需要 LaTeX 2_ε ，然后标明自己的身份。宏包信息由预定义格式的日期，版本号以及作者姓名大写首字母组成。由于需要条件判断，接着就上载了 ifthen 宏包。

```
\NeedsTeXFormat[LaTeX2e}
\ProvidesPackage{fullpage}[1994/02/15 1.0 (PWD)]
\RequirePackage{ifthen}
```

下面就准备好选项。选项有 in 和 cm 分别相应于 1 英寸和 1.5 厘米（ 1 厘米就太窄了）的页边。要用一个特殊长度存贮页边长度值。这个长度是私有的内部命令，遵从 295 页上的约定。

```
\newlength{\FP@margin}
\DeclareOption{in}{\setlength{\FP@margin}{1in}}
\DeclareOption{cm}{\setlength{\FP@margin}{1.5cm}}
```

另外，四个标准的页面样式也做为选项名。它们将会设置两个内部 boolean 开关和页面样式。

```
\newboolean{FP@plain}
\newboolean{FP@empty}
\DeclareOption{plain}{\setboolean{FP@plain}{true}
                      \setboolean{FP@empty}{false}
        \pagestyle{plain}}
\DeclareOption{empty}{\setboolean{FP@plain}{true}
                      \setboolean{FP@empty}{true}
```

```
        \pagestyle{empty}}
\DeclareOption{headings}{\setboolean{FP@plain}{false}
                         \setboolean{FP@empty}{false}
        \pagestyle{headings}}
\DeclareOption{myheadings}{\setboolean{FP@plain}{false}
                         \setboolean{FP@empty}{false}
        \pagestyle{myheadings}}
```

最后，就执行选项的缺省集，再处理选定的选项，在这种情形中，是按指定的顺序进行的。只所以这样做，就是如果给出了不只一个页面样式，最后那个起作用。

```
\ExecuteOptions{in,plain}
\ProcessOptions*
```

下面开始进行计算。首先对于 plain 和 empty 样式，没有页眉行，因此把相应的参数取值为零。这时 FP@plain 为 ⟨真⟩ 。然后把为页脚行保留的空间设为零 (此时 FP@empty 为 ⟨真⟩)。对于 headings 和 myheadings ，这些空间维持不变，因为在这些样式中通常第一页就是 plain 页 (有页脚)。

```
\ifthenelse{\boolean{FP@plain}}
   {\setlength{\headheight}{0pt}
    \setlength{\headsep}{0pt}}{}
\ifthenelse{\boolean{FP@empty}}
   {\setlength{\footskip}{0pt}}{}
```

在页边，页眉和页脚空间已设置好后，就可以进行实际的计算。注意打印驱动程序要在左边和上边留下 1 英寸的边界，因此要从 \topmargin 和 \oddsidemargin 中减去这个长度。

```
\setlength{\textwidth}{\paperwidth}
\addtolength{\textwidth}{-2\FP@margin}
\setlength{\oddsidemargin}{\FP@margin}
\addtolength{\oddsidemargin}{-1in}
\setlength{\evensidemargin}{\oddsidemargin}

\setlength{\textheight}{\paperheight}
\addtolength{\textheight}{-\headheight}
\addtolength{\textheight}{-\headsep}
\addtolength{\textheight}{-\footskip}
\addtolength{\textheight}{-2\FP@margin}
\setlength{\topmargin}{\FP@margin}
\addtolength{\topmargin}{-1in}
```

注意 \paperheight 和 \paperwidth 是被纸张尺寸选项设置成恰好等于纸张的完全尺寸的。如果这种指定是错误的，那么当然最后的结果也不会正确。

至此就完成了宏包文件 fullpage.sty 的所有代码。下面是调用它的一个示例：

```
\documentclass[a4paper,12pt]{article}
\usepackage[headings]{fullpage}
. . . . . .
```

§C.3.2 重新设计页眉和页脚

LaTeX用户经常要做的一件事，可能就是调整每页上页眉和页脚的显示了。标准的 LaTeX虽然提供了有限数目的 页面样式(3.2 节)，但是通常是不够用的。它们中间最有弹性的就是 myheadings 页面样式了，它可以让作者利用 \markright 和 \markboth 来确定活动页眉的文本内容。然而，包括字体样式和页码位置等一般性格式仍然由 LaTeX决定。

我们下面给出一个例子，说明如何为一个大工程文档修改页眉，使得其包含特定的信息。除了标题和节的简写形式，章节号与页码外，还可以有文档序列号，日期，以及版本号。如果能设计一个广义文档页面样式，可以在页面定义外面指定这些条目，那就非常方便了。

由于这里的编码只适用于 article 类，因此我们就创建一个新类，名称为 dochead，它输入 article.cls，然后定义一个新的页面样式。实际上这个类中可以包含更多的其它特殊功能，我们的新页面样式只是其中一个。

我们还是首先声明所需要的 TeX版本，并标明类文件。

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{dochead}[1994/09/28 1.0 (PWD)]
```

这个类没有自己的新选项，只是把指定的选项传递给后台的 article 类。然而，它自己自动选择 12pt 和 a4paper 选项。

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article|
\ProcessOptions
\LoadClass[12pt,a4paper]{article}
```

接下来就是用来输入页眉中四部分信息(短标题，日期，序列号，以及版本)的命令。其中每条命令都把自己的参数存贮在一条内部的 \DH@ 命令里，这几条内部命令起初设成空。最后，声明这些输入命令只能用在导言中，因为如果在正文已开始后再调用它们，就会导致灾难性的后果。

```
\newcommand{\DocTitle}[1]  {\renewcommand{\DH@title}{#1}}
\newcommand{\DocDate}[1]   {\renewcommand{\DH@date}{#1}}
\newcommand{\DocNumber}[1] {\renewcommand{\DH@number}{#1}}
\newcommand{\DocVersion}[1]{\renewcommand{\DH@version}{#1}}
```

```
\newcommand{\DH@title}{}   \newcommand{\DH@date}{}
\newcommand{\DH@number}{}  \newcommand{\DH@version}{}
\@onlypreamble{\DocTitle}  \@onlypreamble{\DocDate}
\@onlypreamble{\DocNumber} \@onlypreamble{\DocVersion}
```

\@onlypreamble 是一条 LaTeX的内部命令。

我们下面定义新的页面样式 dochead，也就是说我们必须创建一个叫 \ps@dochead 的命令，它由 \pagestyle{dochead} 执行。这条命令要做的事情就是重定义四条内部命令 \@oddhead, \@evenhead, \@oddfoot, \@evenfoot。我们把页设成空的，奇偶页眉一样。页眉是一个小页，用的是页面宽度，由一个表格组成，表格中就是相关的文档信息。

```
\newcommand{\ps@dochead}{%
 \renewcommand{\@oddhead}{%
    \begin{minipage}{\textwidth}\normalfont
      \begin{tabular*}{\textwidth}{@{}l@{\extracolsep{\fill}}%
                                   l@{\extracolsep{0pt}:~}l@{}}%
       \DH@number      & Version & \DH@version \\
       \DH@title       & Section & \thesection \\
       Date:~\DH@date  & Page    & \thepage
      \end{tabular*}\vspace{0.5ex} \rule{\textwidth}{0.6pt}%
    \end{minipage}}
 \renewcommand{\@evenhead}{\@oddhead}
 \renewcommand{\@oddfoot}{}
 \renewcommand{\@evenfoot}{}
}
\pagestyle{dochead}
```

最后那行就激活了新的页面样式。

还需要增加 \headheight 和 \headsep 的尺寸，因为我们这里的页眉要比通常的高很多。我们选取的高度是通常行距的 3.5 倍。

```
\setlength{\headheight}{3.5\baselineskip}
\setlength{\headsep}{3em}
```

我们下面进一步对上述定义进行修改。类似于上面这样的文档，它通常希望页码是在一节内部进行编号的。因此，我们要重定义 \thepage 命令，使其从页码记数器中显示页码 (7.1.4 节)，而且修改 \section 命令，使得它开始一个新页，并重设页码记数器。这要首先保存 \section 的当前定义，然后才进行重定义。

```
\let\DH@section=\section
\renewcommand{\thepage}{\thesection-\arabic{page}}
```

```
\renewcommand{\section}{\newpage\setcounter{page}{1}\DH@section}
```

注意这里新的 \section 命令要调用保存在内部命令 \DH@section 中的原来定义。

如果上述代码保存在一个叫 dochead.cls 的文件中，那么下面这样的主文件就可以调用它：

```
\documentclass{dochead}
\DocTitle{Spacecraft Cleanliness}
\DocNumber{ESA--XY--123}
\DocDate{1995 Feb 26}
\DocVersion{3.1}
\begin{document}
. . . . . .
```

那么在第 3 节中第 4 页上的页眉就如下所示

ESA–XY–123 Version: 3.1
Spacecraft Cleanliness Section: 3
Date: 1995 Feb 26 Page : 3-4

以这个例子为榜样，要为其它应用创建不同条目的页眉就应该不会是一件困难的事情。

§C.3.3 为其它语言改编 LaTeX

标准 LaTeX是基于英文设计的，因此在很多地方会自动显示出英文单词，如 'Abstract' 或 'Contents' 。虽然确实可以显示出有重音的字母，但是对于一长节文本而言，这样的输入方法就太复杂了。现在对于英文单词的断词也可以有带重音的字母。因此，如果要为其它语言改编 LaTeX ，那么应该进行三方面的修改。为了降低复杂性，它们是：

1. 用另外语言的翻译替换原来的英文单词，
2. 为重音或特殊标点符号定义简化的命令，
3. 改变断词式样。

D.1.3 节处理的就是这种多语言 LaTeX情形。我们这里只是创建一个宏包，它可以处理一种非英文语言和英文本身。

在 D.1.4 节和 D.4 节讨论了第 3 点。如果 TeX的版本要早于 3.0 ，只可能保存一组断词式样，从而不可能进行真正的语言切换。

利用 TeX相当较近的版本，可以很好地做到第 1 点。起初，类似于 'Abstract' 和 'Contents' 这样的单词是显式包含在生成这些章节的命令中。在欧洲，根据 Technical University of Vienna 的 H. Partl 的建议，来自于 Darmstadt 的 J. Schrod 建立起 LaTeX一个修正版本，在这个版本中，上面所有英文单词都被类似于 \abstractname 和 \contentsname 这样的命令所代替。因此现在

就非常容易进行相应于某语言的修改，因为只要改变这些名称命令的定义就可以，而不用重定义整个 \abstract 或 \tableofcontents 命令。这些名称命令自从 1991 年 12 月 1 日开始已成为正式 LaTeX的一部分。

相应于世界语的改编

我们选取一个示例，来说明把 LaTeX改编到国际语言 — 世界语中的原则。上面所列的三个问题都要加以考虑；解决方法并不是显而易见的，但相对于改编到法语和德语中而言，复杂性则要低很多。而且，通过选择 *la internacia lingvo* ，这里并没有语言学上的分组，这是一个特别的好处，这也是世界语的基本哲学。

有两个可用的世界语宏包，一个叫 espo.sty，它没有署名，也没有日期，相当初等，另一个叫 esperant.sty，由德国 Mainz 大学的 Jörg Knappen 设计，注明日期为 1991 年 12 月 10 日。作者保证上面这个宏包以及 german.sty 的功能是稳定的。（还有第三个宏包，名称也是 esperant.sty，它是 babel 系统的一部分，我们在 D.1.3 节中描述它。）

世界语使用有 28 个字母的字母表，其包括通常拉丁字母表中除 q, w, x 和 y 外的所有字母，并增加了特殊字母 $\hat{c}$, $\hat{g}$, $\hbar$, $\hat{j}$, $\hat{s}$ 和 $\breve{u}$，分别对应于英语中的 *ch*, *j*，德语中的 *ch*，法语中的 *j*，英语中的 *sh*, *w* 。这些加了重音的字母看成是单独的字符，并不像法语中认为是被装饰的字母，或者德语中认为其它字母组合的取代。抑扬重音在 LaTeX中是用 \^ 得到的，而二全音符是用 \u 生成的。但这里要稍微复杂一些，因为在 h 上的符号位置要有点儿改变，而 j 的点要去掉。而且用 \u{u} 得到 ŭ 要输入五个键。为了给用户简化这种操作， Esperanto 样式生成了一个新的单字符命令 ^，对所有的特殊字母都有效。因此 ^c ^g ^h ^j ^s ^u 的结果就是 $\hat{c}$ $\hat{g}$ $\hbar$ $\hat{j}$ $\hat{s}$ $\breve{u}$ 。

然而，要想得到上面那样的字符，只是把 ^ 改变成主动（命令）字符，并把它定义成 \^ 是不够的。不但要把这条命令设成牢固的 (2.6 节)，而且对于包含 ^ 的单词在断词的时候，语言之间的切换，以及最重要的利用名称命令对英文标题进行翻译的时候，都有问题要解决。

我们这里所给出的例子是 Jörg Knappen 对 LaTeX 2_ε进行改编而设计的宏包 esperant.sty，它全部解决了上面提出的几个问题。这个宏包原来必须在 Plain TeX下执行，做为 LaTeX2.09 的一个样式选项。它利用相当多的 TeX技巧，我们在此不做详细解释，只说明一下最后的结果是什么。

做为一个宏包文件，其开始也是鉴定：

```
\NeedsTeXFormat{LaTeX2e}[1994/06/01]
\ProvidesPackage{esperant}
      [1994/06/08 1.1a2e Daly's adaption of Knappen's]
```

这是对 LaTeX 2ε的一个极小改编，因此没有定义任何选项。我们就直接进入主体，其中第一项利用一个 TeX技巧使得有重音的单词可以有断词。做法是加入一个空的单词断点，但不允许断行。

```
\newcommand{\allowhyphens}{\penalty\@M \hskip\z@skip}
```

第二条语句把字符 ^ 加到某个特殊列表中，而且必须把源文本原样输出。不必知道如何做到这点，只要接受它就可以了。

```
\begingroup
  \def\do{\noexpand\do\noexpand}%
  \xdef\dospecials{\dospecials\do\^}%
  \def\@makeother{\noexpand\@makeother\noexpand}%
  \xdef\@sanitize{\@sanitize\@makeother\^}%
\endgroup
```

然后，定义两条命令，让抑扬字符 ^ 的功能在做为上标运算符的普通角色与做为主动字符，即单字符命令之间切换。

```
\newcommand{\modifiedhaton}{\catcode`\^\active}
\newocmmand{\modifiedhatoff}{\catcode`\^7 }
```

第三步，定义抑扬命令，然后把它变成主动字符，并把它定义成有一个参数的牢固命令。在数学模式 (\ifmmode) 中就调用通常的上标函数 \sp，否则就在处理之前检测参数值是否是特殊字符。如果参数是 U 和 u，就显示二全音符重音；如果是 j，就用没有点的相应字符；如果是 h，就调用重叠函数。

```
\modifiedhaton
\DeclareRobustCommand{^}[1]{\ifmmode\sp{#1}
        \else\ifx#1u\u u\allowhyphens
\else\ifx#1U\u U\allowhyphens
        \else\ifx#1h{h\llap{\^{}}}\allowhyphens
        \else\ifx#1j{\^\j}\allowhyphens
        \else\ifx#1|\discretionary{-}{}{}\allowphyens
        \else{\^{#1}}\allowphyens
\fi\fi\fi\fi\fi\fi}
\modifiedhatoff
```

注意这里又为 ^ 增加了一项功能： ^| 加入一个自由连字符，即建议的单词断点，利用这种方法，单词余下的部分仍可以自动断词。通常在 TeX中，单词断点只能出现在有自由连字符的地方。

第四步，建立三条命令，以定义世界语，美国英语和英国英语中的日期。在解释 \hodiau 和 \hodiaun 这两条命令时需要一点世界语的语法：表示今天的单词是 *hodiaŭ*，例如 *Hodiaŭ estas la 15a de novembro, 1994*，当做为一

封信的日期时，日期通常必须是宾格的 (*la 15an de novembro, 1994*) 。我们对这两种可能都做了考虑。

我们不采用直接重定义 \today 的方法，而是生成几个重定义的命令，我们可以利用它们在世界语、美国英语和英国英语之间来回切换。后两者是用 \providecommand 定义的，主要是为了以防已有宏包对它进行了定义。

```
\newcommand{\dateesperanto}{\renewcommand{\today}
  {la \number\day -an de \ifcase\month\or
  januaro\or februaro\or marto\or aprilo\or majo\or junio\or
  julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
  decembro\fi, \space \number\year}%
 \let\hodiaun=\today}
\newcommand{\hodiau}{la \number\day -a de \ifcase\month\or
  januaro\or februaro\or marto\or aprilo\or majo\or junio\or
  julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
  decembro\fi, \space \number\year}

\providecommand{\dateUSenglish}{\renewcommand{\today}{%
  \ifcase\month\or January\or February\or March\or April\or
  May\or June\or July\or August\or September\or October\or
  November\or December\fi \space \number\day, \number\year}}

\providecommand{\dateenglish}{\renewcommand{\today}{%
  \ifcase\day\or 1st\or 2nd\or 3rd\or 4th\or 5th\or 6th\or 7th\or
  8th\or 9th\or 10th\or 11th\or 12th\or 13th\or 14th\or 15th\or
  16th\or 17th\or 18th\or 19th\or 20th\or 21th\or 22th\or
  23th\or 24th\or 25th\or 26th\or 27th\or 28th\or 29th\or
  30th\or 31st\fi~\ifcase\month\or
  January\or February\or March\or April\or May\or June\or
  July\or August\or September\or October\or November\or
  December\fi \space \number\year}}
```

接下来要用 LaTeX(即 1991 年 12 月 1 日以后发行的 LaTeX版本) 中的名称命令对章节标题进行翻译。同前面处理日期时一样，我们也不是直接进行，而是用 \captionsesperanto 和 \captionsenglish 进行我们的翻译工作。（注意，下面是把两组标题并列摆放，它们实际上是一个接在另一个后面的。）定义是用 TeX的 \def 命令进行的，因为它并不在意所定义的命令是否已经存在。

```
\def\captionsesperanto{%
\def\contentsname{Enhavo}%
\def\listfigurename{Listo
    de figuroj}%
\def\listtablename{Listo
    de tabeloj}%
\def\abstractname{Resumo}%
\def\partname{Parto}%
\def\chaptername{\^Capitro}%
\def\prefacename{%
     Anta\u{u}parolo}%
\def\appendixname{Apendico}%
\def\refname{Cita\^{\j}oj}%
\def\bibname{Bibliografio}%
\def\indexname{Indekso}%
\def\figurename{Figuro}%
\def\tablename{Tabelo}%
\def\pagename{Pa\^go}%
\def\seename{vidu}%
\def\alsoname{vidu anka\u{u}}%
\def\notesname{Rimarkoj}%
\def\enclname{Aldono(j)}%
\def\ccname{Kopie al}%
\def\headtoname{Al}%
\def\subjectname{Temo}%
}
```

```
\def\captionsenglish{%
\def\contentsname{Contents}%
\def\listfigurename{List
    of Figures}%
\def\listtablename{List
    of Tables}%
\def\abstractname{Abstract}%
\def\partname{Part}%
\def\chaptername{Chapter}%
\def\prefacename{Preface}%

\def\appendixname{Appendix}%
\def\refname{References}%
\def\bibname{Bibliography}%
\def\indexname{Index}%
\def\figurename{Figures}%
\def\tablename{Table}%
\def\pagename{Page}%
\def\seename{see}%
\def\alsoname{see also}%
\def\notesname{Notes}%
\def\enclname{encl}%
\def\ccname{cc}%
\def\headtoname{To}%
\def\subjectname{Subject}%
}
```

这些名称只会出现在特定的类中，而且有些也不是标准的名称。例如，\refname 只能用在 article 类中，在 book 和 report 类中其被 \bibname 所代替，而 \abstractname 也只会用在 article 和 report 类中。 \pagename, \enclname, \ccname 和 \headtoname 只会出现在 letter 类中， \seename 是为 makeidx 宏包准备的。其它命令都是非标准的，只用于特殊用途或改编，它们有： \prefacename, \notesname, \alsoname （ 为 makeidx 准备的) 以及 \subjectname (为 letter 准备的) 。

正如上面所指出的，在世界语中的重音字母是看成单独字符的。这也就是说在某些情况况下以字母为序进行排列时，其应显示为 'A B C Ĉ ...' 或 'a b c ĉ ...' 。为了做到这一点，要定义两个新的记数器类型： \Esper 和 \esper ，其功能类似于 \Alph 和 \alph 。

```
\newcommand{\esper}[1]{\@esper{\value{#1}}}
\newcommand{\Esper}[1]{\@Esper{\vaule{#1}}}
\newcommand{\@esper}[1]{\ifcase#1\or a\or b\or c\or \^c\or d\else
   \@iesper{#1}\fi}
\newcommand{\@iesper}[1]{\ifcase#1\or \or \or \or \or \or
   \or e\or f\or g\or \^g\or h\or h\llap\^{}\or i\or j\or \^\j\or
   k \or l\or m\or n\or o\or p\or r\or s\or \^s\or t\or u\or
   \u{u}\or v\or z\else\@ctrerr\fi}
\newcommand{\@Esper}[1]{\ifcase#1\or A\or B\or C\or \^C\or D\else
   \@Iesper{#1}\fi}
\newcommand{\@Iesper}[1]{\ifcase#1\or \or \or \or \or \or \or E\or
   F\or G\or \^G\or H\or \^H\or I\or J\or \^J\or K\or L\or M\or
   N\or O\or P\or R\or S\or \^S\or T\or U\or \u{U}\or V\or
   Z\else\@ctrerr\fi}
```

下面定义几个语言记数器以跟踪当前语言。其只对 3.0 版本以后的 TEX 才有效，这时把 \language 设置为一个数值，以激活存贮在该语言编号下面的断词式样。这里我们假定英语断词式样保存在零号语言中，而世界语式样保存在一号语言中。（在 D.1.3 节介绍的 babel 系统使用了一种更安全的方法进行语言编号赋值与选择。）\selectlanguage 命令通过调用日期和标题定义命令来打开选定的语言。

```
\newcounter{USenglish}   \setcounter{USenglish}{0}
\newcounter{esperanto}   \setcounter{esperanto}{1}
\newcounter{english}     \setcounter{english}{0}

\providecommand{\selectlanguage}{}
\renewcommand{\selectlanguage}[1]{%
  \ifcase \vaule{#1}%
  \dataUSenglish    \captionsenglish   \or
  \dateesperanto    \captionsesperanto \or
  \dateenglish      \captonsenglish    \fi
  \language=\value{#1}}
```

最后，定义切换世界语的开关，并用来在文档开头处打开世界语。

```
\newcommand{\EsperantoOff}{\modifiedhatoff}
\newcommand{\EsperantoOn}{\modifiedhaton}
\AtBeginDocument{\EsperantoOn\selectlanguage{esperanto}}
```

§C.3.4 作者–年代 引用

在 B.3.1 节我们指出标准的 LaTeX不能处理引用是由作者和年代组成的参考文献样式，而只能处理引用是数字型的。我们在此给出一个宏包和参考文献样式，以解决这个问题。

宏包文件

我们给宏包命名为 authyear 。这是相当广泛的 作者–年代 引用样式宏包 natbib(由 Patrick W. Daly 完成) 的一个非常简化的版本。但它确实演示了最重要的功能。

在 作者–年代 引用机制中，我们需要放在括号内的引用和平铺直述的引用。区分这两种样式的方法是采用放在 \cite 命令中的可省注释参数值来实现的。假设在 thebibliography 中有如下一项条目：

```
\bibitem[{\it Jones et~al.}(1990)]{jone90}
Jones, F. B., Smith, T. E., and Harris, R. E. ...
```

jone90 就是 \cite 命令和标签 (即 {\it jones et~al.}(1990)) 的关键词。对于这种情形，标签中年代两边的括号就相当于一个定界符，从而把作者与年代分开。然后如何处理它们，那就要看如何用 \cite 命令了：

\cite{jone90}	结果为	*Jones et al. [1990]*
\cite[]{jone90}	结果为	[*Jones et al., 1990*]
\cite[page~30]{jone90}	结果为	[*Jones et al., 1990, page 30*]

包围引用的括号可以是圆括号，也可以是方括号；在引用的中标点符号可以是逗号，也可以是分号。

宏包文件的开头仍是鉴别命令。它也需要 ifthen 宏包，因此就紧接着调用了该宏包。

```
\NeedsTeXFormat{LaTeX2e}[1995/06/01]
\ProvidesPackage{authyear}[1995/10/09 2.1 (PWD)]

\RequirePackage{ifthen}
```

下面就准备并执行选项。这些选项将设置各种命令以保存标点符号和括号类型。由于这些存贮命令并不是独有的内部命令，因此用户也可以在任何时候进行重定义。然而，标准选项应该可以满足通常的需要。首先定义存贮命令，然后用选项进行设置，并选定缺省选项 round 和 colon ，最后就执行所选定的选项。

```
\newcommand{\citebegin}{} \newcommand{\citeend}{}
\newcommand{\citesep}{}
\newcommand{\auyrsep}{,}
```

```
\DeclareOption{round}{\renewcommand{\citebegin}{(}
                      \renewcommand{\citeend}{)}}
\DeclareOption{square}{\renewcommand{\citebegin}{[}
                      \renewcommand{\citeend}{]}}
\DeclareOption{comma}{\renewcommand{\citesep}{,}}
\DeclareOption{colon}{\renewcommand{\citesep}{;}}
\ExecuteOptions{round,colon}
\ProcessOptions
```

如果用下面定义的命令对内部命令 \AY@cite 进行了格式化，那么它就会显示出实际的引用。其第一个参数就是有格式的引用，第二个就是 \cite 的可省参数值，即可省的补注。如果没有给出这个补注， #2 就是句号 (见下面的 \cite)，引用显示时并不放在括号内 (即平铺直述的引用)；否则，要把它以及补注包装在括号内 (放在括号内的引用)。

```
\newcommand{\AY@cite}[2]
   {\ifthenelse{\equal{.}{#2}}
     {#1}
     {\citebegin#1%
      \ifthenelse{\equal{#2}{}}
        {}
        {, #2}%
      \citeend}}
```

实际上主要工作是由 \cite 完成的。这里所发生的事情实际上是基于 LaTeX 2$_\varepsilon$对 \cite 的定义这一 LaTeX内部技术。新的 \cite 命令有两个参数值，第一个就是可省的补注。然而，有补注就意味着要使用放在括号内的引用。补注实际上可以是空的，这得到一个放在括号内的没有补注的引用。为了做到这一点，(可省的) 第一个参数值的缺省值为句号。它表示调用 \cite 时根本没有可省参数值，更不用说是一个空参数值了。

主要参数值 (#2) 就是一串引用关键词。它们必须一个一个地用 \@for \do 进行处理。并把 \@citeb 相继设成列表中的关键词。 \@iden 所在的行从 \@citeb 中去掉了前导空格；然后在辅助文件中为 BibTeX生成一项条目；进行一次测试，看看关键词是否有相应的转换 (保存在 \b@\@citeb 中)，如果没有的话就显示出一条警告消息。此时，就有了新的功能：调用 \AY@parse 对关键词进行解释，把作者部分放到 \AY@author 中，年代部分放到 \AY@year 中；在平铺直述引用 (#1=.) 中，要把这两部分放在一起，年代放在括号内。最后，调用引用显示命令 \AY@cite，第一个参数值为一串转换后的引用，第二个参数是 \cite 的可省参数值。注意在列表中每项引用前面加上了 \AY@sep；它起初是空的，但在第一项引用后变成 \citesep。这条命令可以在引用之间

插入逗号或分号。

```
\newcommand{\AY@sep}{}

\renewcommand{\cite}[2][.]{%
  \renewcommand{\AY@sep}{}%
  \AY@cite{\@for\@citeb:=#2\do
    {\edef\@citeb{\expandafter\@iden\@citeb}%
      \ifthenelse{\boolean{@filesw}}
        {\immediate\write\@auxout{\string\citation{\@citeb}}}
        {}%
      \@ifundefined{b@\@citeb}
      {\AY@sep{\reset@font\bfseries ?}\G@refundefinedtrue
       \@latex@warning
        {Citation `\@citeb' on page \thepage \space undefined}}
      {\AY@parse{\@citeb}%
        \ifthenelse{\equal{.}{#1}}
   {\AY@sep{\AY@author} \citebegin\AY@year
    \renewcommand{\AY@sep}{\citeend\citesep\ }}
   {\AY@sep{\AY@author}\auyrsep\ \AY@year
    \renewcommand{\AY@sep}{\citesep\ }}}}% ends \do
\ifthenelse{\equal{.}{#1}}
    {\citeend}
    {}%
  }{#1}}
```

下面的命令对引用关键词转换后的结果进行解析，提取出作者和年代消息，它做为关键词调用时的参数。展开 (转换) 后的结果用做 `\AY@split` 的参数值。在此必须加上一个没有内容的 `()` 以防转换后在括号内并不包含年代。`\let\protect=` 是为了得到牢固命令的特殊处理，用它是为了防止在此处就被展开，从而导致严重破坏。

```
\newcommand{\AY@parse}[1]{{\let\protect=\@unexpandable@protect
     \xdef\@tempa{\@nameuse{b@#1}}%
     \expandafter\AY@split\@tempa()@}}
\def\AY@split#1(#2)#3@{\gdef\AY@author{#1}\gdef\AY@year{#2}}
```

这就完成了对 `\cite` 及相关内部命令的重定义。我们现在来修补环境 `thebibliography` 。不幸的是，这意味着需要重定义整个环境，因为这里并不存在便利的内部命令可以修改。而且，我们需要间断是否处在 `article` 类中，即有没有 `\chapter` 命令，因为这确定了索引清单是否在 `\section*` 或

\chapter* 中。

```
\@ifundefined{chapter}
  {\newcommand{\AY@bibsect}{\section*{\refname
   \@mkboth{\MakeUppercase{\refname}}{\MakeUppercase{\refname}}}}}
  {\newcommand{\AY@bibsect}{\chapter*{\bibname
   \@mkboth{\MakeUppercase{\bibname}}{\MakeUppercase{\bibname}}}}}
\providecommand{\refname}{References}
\providecommand{\bibname}{Bibliography}
```

注意: \MakeUppercase 命令 (以及这里没有用的 \MakeLowercase 命令) 可以改变其参数值的大小写，相比于在 LaTeX2.09 中所用的 TeX命令而言，这两条命令要更好一些。

我们现在来重定义 thebibliography 环境，使得标签并不会显示出来，第一行左对齐，所有后面的行向右缩进 1 em 。我们这里也要利用上面定义的 \AY@bibsect 命令。

```
\renewenvironment{thebibliography}[1]
{\AY@bibsect
  \setlength{\parindent}{0pt}\list{}
  {\setlength{\leftmargin}{1em}%
   \setlength{\itemindent}{-\leftmargin}}
  \ifthenelse{\boolean{@openbib}}
    {\renewcommand\newblock{\newline}}
    {\renewcommand\newblock{\hskip .11em plus.33em minus.07em}}
  \sloppy\clubpenalty4000\widowpenalty4000
  \frenchspacing}
{\renewcommand{\@noitemerr}{\@latex@warning
      {Empty `thebibliography' environment}}%
  \endlist\vspace{-\lastskip}}
```

最后要做的一件事就是中止 \bibitem 命令的功能，不让它显示出标签。这里我们只需修改一条内部命令，并忽略其参数。

```
\renewcommand{\@biblabel}[1]{\hfill}
```

参考文献样式文件

上面完成了 authyear.sty 宏包文件。然而，只有当 thebibliography 环境中的 \bibitem 命令具有本节开始所示形式时才有效。此时参考文献可以手工得到，也可以利用 BibTeX，但必须存在一个正确格式的参考文献样式文件 (.bst) 才可以。在标准 BibTeX宏包中没有这样的样式文件，但只要对 alpha.bst 文件稍做一点儿改变就可以了。我们下面只是列出做到这点的必

要代码，而并不详加解释，因为BibTeX用的是与众不同的语言。

把alpha.bst文件复制到一个叫authyear.bst的新文件中，并在其中寻找文本FUNCTION {output.bibitem}。这个函数的第五行是"]{" write$；把它改成")]{" write$，即在右方括号前面加上右小括号。

然后，找到FUNCTION {format.date}，并在这个函数最后那个右大括号前面加上extra.label *，即现在为

```
  if$
  extra.label *
}
```

现在寻找文本FUNCTION {format.lab.name}。把这个函数的文本(从开头到FUNCTION {author.key.label}之前的所有内容)替换为下面的代码

```
FUNCTION {format.lab.names}
{ 's :=
  s #1 "{vv~}{ll}" format.name$
  s num.names$ duplicate$
  #2 >
    { pop$ " et~al." * }
    { #2 <
        'skip$
        { s #2 "{ff }{vv }{ll}{ jj}" format.name$ "others" =
            { " et~al." * }
            { " and " * s #2 "{vv~}{ll}" format.name$ * }
          if$
        }
      if$
    }
  if$
}
```

最后，查找FUNCTION {calc.label}，并定位到包含duplicate$的那行。在其后加上emphasize "(" *。在接下来的一行中，把#2改成#4。因此这些行及相邻行现在应该为

```
  if$
  duplicate$
  emphasize "(" *
  year field.or.null purify$ #-1 #4 substring$
  *
  'label :=
```

(上面的 emphasize 指的是引用中的作者，而不是索引中的作者要用斜体显示。如果不希望这样，那就忽掉它。)

注意：这个新参考文献样式只在 0.99 或以后版本的 BibTeX中才可以用。

为了演示如何使用这一样式，假定在参考文献数据文件 mylit.bib (见 B.2 节) 中有一项条目：

```
@ARTICLE{jone90,
   AUTHOR = {Fred B. Jones and
             Thomas Elwood Smith and
             Richard E. Harris},
   TITLE = {Activity in the Night-Side Ionosphere},
   YEAR = {1990},
   JOURNAL = {J. Geophys. Res.},
   VOLUME = {101},
   PAGES = {1234-1254}  }
```

如果在 LaTeX文件中现在包含如下文本行

```
\documentclass{article}
\usepackage[square]{authyear}
\begin{document}
. . . as shown by \cite{jone90}, it is possible . . .
. . . has been measured \cite[]{jone90}. This is . . .
\bibliographystyle{authyear}
\bibliography{mylit}
\end{document}
```

所得的输出会是

. . . as shown by *Jones et al.* [1990], it is possible . . .

. . . has been measured [*Jones et al.*, 1990]. This is . . .

Reference

Fred B. Jones, Thomas Elwood Smith, and Richard E. Harris. Activity in the night-side ionosphere. *J. Geophys. Res.*, 101:1234–1254, 1990.

©TEXGuru, August 16, 1999

附录D　扩展 LaTeX

LaTeX之所以变得像现在这样普及，就是因为能对它进行重新开发：如果基本安装 (核心) 中缺少了某些功能，那么经验丰富的用户就可以通过写一个宏包来弥补这一点。在 LaTeX2.09 流行的年代里，借助于计算机文件服务器，大批的宏包 (那时称为 *样式选项*) 得以普及。其中绝大多数现在与 LaTeX 2ε还是兼容的，其余的被进行了转化，当然还有更多的正在编写过程中。

本书的目标只限于标准 LaTeX宏包，并不想过多涉及其它宏包内容；可以在 Goosens 等人的 *The LaTeX Companion*(1994 年) 一书中找到对其它一些宏包的介绍。我们在此只是对几方面的扩展进行大致的描述，包括多语言 LaTeX，PostScript 字体, 图形的插入和彩色的使用。然后我们讲解一些基本安装的高级功能，例如宏包的档案集成化，某些半正式扩展，最后是 LaTeX 2ε的安装方法。

所有本附录列出的 LaTeX宏包都可以在 D.5 节列出的地点中找到。

§D.1　国际化的 LaTeX

在 LaTeX起初的版本中, 有些命令中显式包含一些英文单词, 例如 'Figure' 和 'Bibliography'。这一事实与程序设计的一条良好规则发生冲突，即永不要显式地做任何事情。在欧洲，LaTeX用户们很快就进行了改编，使其适合于自己的母语。相对于德语的改编，即宏包 `german.sty`，就是一组由维也纳技术大学的 H. Partl 收集的宏指令，这些宏指令是由不同的人员开发的，现在这个宏包已成为那些讲德语的 TeX用户组 (DANTE) 的标准。这个宏包中也包含一些处理其它语言 (如法语和英语) 的功能。下一节给出这部分的详情。

`german.sty` 宏包的关键就是对标准 LaTeX文档样式 (类) 进行更新，输出中的显式英文单词已被名称命令代替。为了适用于所有语言，在欧洲用户间这些名称已经被标准化了。来自 Darmstadt 的 J. Schrod 对 LaTeX版本进行了修改，得到了著名的 IaTeX，即国际化的 LaTeX。可以在 316 页上找到这些名称及其标准英文值。

到了 1991 年 12 月 1 日，这些名称特征成为 LaTeX标准的一部分。它们代表良好程序开发的习惯，因为通过这种方法，即使是英语用户也可以很容易地改变某些标题，例如，把 'Abstract' 改成 'Summary'，或者把 'Contents' 改成 Table of Contents'。

另外一种要应于多语言用法的重要工具是 TeX的 `\language` 记数器，通过它在 initex上载的格式中可以保存不只一组的断词式样 (D.4 节)。通过把 `\language` 设成不同的值，可以激活不同的式样。只有 3.0 或更高版本的 TeX才具有这种功能。

§D.1.1 german.sty 文件

在 1987 年 10 月召开的第六届德语 TeX用户大会上，通过了一组标准的 TeX和 LaTeX命令以应用于德语文档中。宏包 german.sty 是由维也纳技术大学的 H. Partl 建立起来的, 随后有很多人对它进行了扩展。最新版本是 2.5b , 发布日期是 1995 年 1 月 20 日，现在是由 Stuttgart 大学的 Bernd Raichle 负责它的维护工作。这个宏包不仅打算在 LaTeX 2_ε下工作正常，而且在 Plain TeX和 LaTeX2.09 , 标准的字体编码以及 Cork 编码中都能发挥其正常功能。

我们在此并不打算对 german.sty 文件进行完整的说明, 因为在 C.3.3 节对 esperant.sty 的介绍已几乎包含了所有一般性语言改编的要点。事实上, esperant.sty 就是从 german.sty 中演化而来的，可以认为是它的一个小型衍生宏包。在本附录中，我们演示的是其中包含的相应于德语的其它功能。在下一节，我们对 french.sty 宏包做同样的事情。

由于 german.sty 的开发是受国际化 LaTeX和多语言系统 babel 的启发, 因此在这一点上我们要额外注意。

在 german.sty 出现之前就存在着一个相应于德语的比较原始的小改编, 它由三条命令组成:

```
\catcode‘\"=\active \def"{\"} \def\3{\ss}
```

上述指令使得双引号 " 成为命令 (主动字符) , 并把它定义成变音重音 (\") , 把 \3 定义成德语的双写 *s* , 即 *eszet*, ß 。因此上面的指令只是简化了德语文本的输入。

双引号命令符号

现在的 german.sty 精心改进了双引号命令的定义，使得

- 在计算机现代字体中的变音离字母的位置要比通常 LaTeX中的近，例如 ö("o) 和 ö(\"o) ;
- 双引号可以用在 \verb 命令和 verbatim 环境中;
- 有变音的单词也可以进行断词;
- 也可以用 "s 显示出 *eszet* ; 但出于一致性的考虑, 原来的 \3 仍然有效, 但不鼓励这种用法。

双引号符号也可以实现德语中其它特殊功能，特别是单词的断开:

"ck 在德语中，有一个断词的规则，那就是如果要在 *ck* 间断开，那它就成为 *k-k* 。这可以通过输入 "ck 来实现，例如 Dru"cker 。

"mm 另外一条规则就是当两个单词合写时, 那么至多只能有两个重复字母 (例如， *Schwimm* + *Meister* = *Schwimmeister*) , 可是如果单词断点恰恰就在在它们之间时, 省掉的字母就又要被加上 (*Schwimm-meister*) 。在文本中可以输入 Schwi"mmeister) 做到这一点。当然也有可能存在其它的二 -

三字母情形。

"- 建议断词点是用 "- 表示的，这表示允许在此处进行自动断词，有时候通常的 LaTeX命令 \- 没法做到这一点。

"| 最后提一点，如果连写是不恰当的，可以用 "| 断开，例如两个相邻单词间的字母 (*Auf* + *Lage* = *Auflage*，输入为 `Auf"|lage`，否则结果为 *Auflage*)。

引用记号

德语不使用英语中的‘引号’，它用的是 „鹅脚“ 形式的 ‚单双引号‘。生成这两组符号的命令分别是：\glqq 表示左双引号 („)，\grqq 表示右双引号 (“)，\glq 表示左单引号 (‚)，\grq 表示右单引号 (‘)。两个双引号命令可以缩写为 "‘ 和 "’。

也提供了法语中的 ≪ 引号 ≫，它是利用 \flqq 和 \frqq 命令输入的。可以缩写为 "< 和 ">。而 ‹ 单引号 › 是利用命令 \flq 和 \frq 生成的。这两者都经常出现在德语文本中。

标题和日期

国际化 LaTeX中的名称命令也可用类似 316 页上的世界语方式，也就是利用命令 \captionsgerman，在德语中激活。可以用 \captionsenglish 命令切换回原来的模式；在 german.sty 中还有一条可用的命令 \captionsfrench，但是可以利用 french.sty(D.1.2 节) 得到更好的法语改编。

\today 会显示出当前日期，可以用声明 \dategerman, \dateaustrian, \dateUSenglish, \dateenglish 和 \datefrench 等进行重定义。其中两个英文形式的定义与 316 页上的一样。根据所选择的语言不同，\today 的结果形式也就不同。

语言选择

即使上载了 german.sty 宏包，也可以用利用 \selectlanguage 命令选择法语和英语，这条命令的参数值为 german, french, english, USenglish 或 austrian 中的一个。它通过把 \language 设成不同的值，而激活相应的断词式样。

§D.1.2 french.sty 文件

法语样式文件 french.sty 起初是由一批用法语编写文档的用户独立于 german.sty 而开发的。随后由 Bernard Gaulle 对它进行扩展与维护，Gaulle 把自己的全副精力奉献于使用法语的 TeX用户组 (GUIenberg) 中。当前版本是 3.32，发行日期为 1994 年 8 月 9 日。

我们在此讲解一些法语样式文件中可用的特殊命令，它们既可以用在TEX中，也可以用在 LATEX中。同德语时的情形一样，这样做的目的在于说明必要修改的实质以及问题的复杂性。

在德语样式 german.sty 中我们的重心是放在如何简化变音重音上，以及调整德语断词规则上，而与德语的情形不同，在 french.sty 中并没有特别的重音命令，而要考虑法语的断词以及缩写。这种修改可以组织起来，分为五种不同类型，每种类型都有两条命令，分别具有激活和关闭功能。概括命令 \french 会调用所有 5 条激活命令。

断词

利用 \frenchhyphenation，可以使 TEX切换到法语断词式样中，它是用 initex上载到格式中的。这也就是说单词的断开是按照法语的规则进行的。利用相反的命令 \nofrenchhyphenation，就又切换到标准的式样中，即第 0 号语言，这通常就是英语。

标题和日期

\frenchtranslation 命令会调用 \captionsfrench，以重定义在国际化 LATEX中所有的名称命令。它也会把 \today 设成法语的形式。另外，还增加了一些名称命令，主要是针对于非标准信件类而引入的。另外还有一条与 \abstract 相似的命令 \resume，因为许多法语文章需要以英语和法语形式给出摘要 (*résumé*).

命令 \nofrenchhyphenation 会关闭法语翻译。

标点符号

在法语的排版规则中，要求‘标点符号 ! ? ; :’前面有多余的空档。这个空档要比通常的单词间隔小；在源文本中，这些字符的前面应该是一个完整的空格 (但请看下面的 \untypedspaces)，当用了 \frenchtypography 命令后，这些空格是会自动减小的。TEX命令 \frenchspacing 也同时被激活。

在法语中只有唯一的引号，在 TEX中 ≪ 形状 ≫ 与 LATEX中 ≺ 形状 ≻ 不同。它是用 << 和 >> 生成的，这与在 german.sty 中用 \flqq 和 \frqq 生成的引号有点儿差别。它们会在被包围文本两边插上额外的空档。被引用的文本也可以放在 guillemets 环境中。这样会在引用中每个段落的开始加上左引号。

另外还有许多可用的子选项，其中几个如下：

\untypedspace 即使在源文本中少掉了 ! ? ; : 前面的空格，也会在它们前面插入额外的空档。

\noTeXdots 把 \ldots 和 \dots 命令改成显示法语模式的省略号 (...)，即三个紧相邻的句号，而不用英文中的方式 (. . .)。

\todayguillemets 在第二层次引用中的段落，或者 guillemets 环境中的段落，其前面都有左引号，即与第一层次段落一样。到了今天，这也是缺省的标准方式。

\ancientguillemets 在第二层次引用中的段落开始用的是右引号，而不是左引号。这是古法语的风格。

\noenglishquote 把引号字符 ‘ 和 ’ 转化成重音符号 ` 和 ´ 。

\noenglishdoublequotes 把双引号 “和” 转化成左右引号。

\idotless 为重音 $\hat{\imath}$ 和 $\ddot{\imath}$ 生成没有点的 i 。

命令 \nofrenchtypography 就会关闭这些特殊的标点符号规则及子选项。

布局

利用 \frenchlayout 命令，可以在 LaTeX格式中进行如下几方面的改变。

- 所有的段落都要进行缩近，即使一节开始的第一段也不例外。在 LaTeX中通常第一段不进行缩进。
- 在 itemize 环境中每一层的 \item 标签都是横线。
- 每当开始新的部分，都重设所有的章节命令。
- 定义了一个新的环境 order ，其类似于 enumerate ，但项的编号为 1° 2° 3° . . . 。

可以用 \nofrenchlayout 命令去掉这些格式修改。

缩写

当调用了 \frenchmacros 时，就会定义许多新的命令，这些命令有助于显示常见的法语缩写。

\ier 1\ier 显示为 1er 。

\iere 1\iere 显示为 1re 。

\ieme 可以显示其它的基数，例如 3\ieme 结果为 3^{e} 。

\at 显示 @ 。

\bv 利用 \tt 字体显示 | 。

\chap 显示 ^ 。

\boi 利用 \tt 字体显示反斜杠。

\tilde 显示 ~ 。

\Numero 显示 N^{o} 。

\degres 显示度数符号 ° 。

\fup{χ} 提升并缩小 χ ，例如 M\fup{me} 结果为 Mme 。

\primo \secundo \tertio \quatro 显示为 1° 2° 3° 4° 。但后接右括号时，圆圈会变得更小。

\quando=n 可以按上面的方式显示任何数值，例如 \quando={10} 的结果为 10° 。

\minMAJ{oe} 在目录表和页眉中会自动根据大小写要求显示为 œ或 Œ 。也可以取其它的参数，例如 ae 和 i 。

可以用 \nofrenchmacros 关闭这些特殊命令。

语言选择

利用 french.sty 中的程序也可以切换回英语，命令 \english 就可以做到这一点。随同的 english.sty 文件中包含了英文形式的标题名称以及相应的切换命令。

要使用已有断词式样的其它语言，可以用 \NouveauLangage[kaishu 数]{语言} 进行定义。这条命令就会创建一条新命令 \语言 ，它通过把 \language 记数器设成给定的 数 而切换到给定 语言 。这时也会执行一条叫 \语言TeX 的命令，以激活相应于该语言的其它功能，必须另外定义这条命令。

§D.1.3 多语言 LaTeX—babel 系统

在 german.sty 和 french.sty 宏包中都存在同样的一个问题，虽然它们允许加进其它的语言，但采用的都是不兼容的方式。因此在一个文档中就没法同时使用这两种样式。

丹麦 PTT 研究实验室的 Johannes Braams 开发了 babel 系统。其主要目的就是提供一种标准方法，在语言间进行切换，所采用的是一种有弹性的方法，来上载断词模式。已针对于 TeX(2.x 和 3.x 版本) 和 LaTeX(只有国际化版本和英文版本) 编写了这个系统。

在 312 页上我们列出了进行语言改编需要满足的三条要求：对显式的英文标题进行翻译，给出在相应语言中实现特定功能的简化命令，选择相应的断词模式。为此， babel 系统增加了可以卸载特殊命令的功能，以及对当前语言的测试功能。与某种语言有关的文件必须同 babel 选择命令的结构相匹配。在这个系统中也包含 “方言” 的概念，也就是说某两种 “语言” 拥有同样的断词模式。例如德语与奥地利语，以及英式英语与美式英语，它们只是在日期格式上有点儿差别 (见 327 页) 。

目前，在 babel 宏包中包含如下语言的定义文件：
世界语 (Esperanto),
荷兰语 (Dutch), 英语 (English), 德语 (German),
法语 (French), 意大利语 (Italian), 葡萄牙语 (Portuguese), 西班牙语 (Spanish), 加泰罗尼牙语 (Catalan), 加利西亚语 (Galician), 罗马尼亚语 (Romanian),

丹麦语 (Danish), 挪威语 (Norwegian), 瑞典语 (Swedish),
法国布里多尼语 (Breton), 爱尔兰语 (Irish), 苏格兰语 (Scottish),
芬兰语 (Finnish), 匈牙利语 (Hungarian), 爱沙尼亚语 (Estonian),
捷克语 (Czech), 斯洛伐克语 (Slovak), 克罗地亚语 (Croatian), 斯洛文尼亚语 (Slovene), Sorbian(大写与小写), 俄语 (Russian)(计划中), 波兰语 (Polish),
土耳其语 (Turkish),
Bahasa.

在安装版本中会同时给出必需的定义文件。然而，不同语言的断词模式却必须从另外的的渠道取得。其中绝大多数文件，并没有处理重音或拼写的特殊命令，只是重定义了国际化 LaTeX中的名称命令。

在 babel 的开发过程中， german 宏包具有很大的启示作用。然而，由于 \selectlanguage 命令的用法稍有点儿不同，因此必须为 babel 创建一个新的 german.sty 文件，这个文件由同样的特殊命令集组成，只是相对于 babel 稍做了点儿增加。同样地，在 babel 中的法语文件被命名为 francais.sty ，以与 french.sty 区分开。

babel 文件

在 babel 安装中包含如下文件，以实现所需要的目标：

babel.def 由运行 babel 所需要的一些基本宏组成，必须被读入的第一个语言文件上载进来；其余的宏就要么包含在格式文件中 (由 hyphen.cfg 上载) ，要么从 switch.def 文件中读入； babel.def 确定当前状况，并给出相应的反应；

switch.def 由其它的 babel 宏组成，以备如果在 LaTeX格式中没有定义这些宏时，把它们读入进来；

hyphen.cfg 由与 switch.def 中相同的宏定义组成, 还包含一些运行 initex时需要的宏定义；如果在创建 LaTeX 2ε格式 (D.4 节) 时这个文件可用，那么这些宏就会内建到格式中，在运行时就不再需要 switch.def ；

babel.sty 是主要的宏包文件，它上载以选项形式给出的语言文件；

language.dat 由一组语言及相应的断词模式文件名组成；在运行 initex时，这些文件会被 hyphen.cfg 文件读入进来；为了特定的安装需要 (D.1.4 节) ，这是唯一可以 (必须 !) 进行编辑的文件；

esperant.ldf... 语言定义文件。

为了与原来的版本兼容，还有一组 .sty 文件，它们除了读入相应的 .ldf 文件外，再不做任何事情。

调用 babel

在 LaTeX 2ε中， babel 宏包是用指定的语言做为选项来上载的：

```
\usepackage[english,esperanto]{babel}
```

与之相应地，语言名称也可以是全局选项，如果有其它宏包也用语言做为选项，这就是一种值得推荐的方法，例如，

```
\documentclass[english,esperanto]{article}
\usepackage{babel,varioref}
```

在这两种情形中，最后给出的名称相应的是马上被激活的语言。

在 1995 年 7 月 5 日发布的 3.5 版本 babel 系统中，可以识别的做为选项的语言名称有：

american	danish	hungarian	scottish
austiran	dutch	italian	spanish
bahasa	english	lowersorbian	slovak
brazil	esperanto	magyar	slovene
brazilian	finnish	norsk	swedish
breton	francais	nynorsk	turkish
british	french	polish	uppersorbian
catalan	galician	portuges	
croatian	german	portuguese	
czech	germanb	romanian	

如果上载了 babel 宏包，那么它所做的事情就是读入进来指定语言的宏包；然后它们依次读入 babel.def 以及可能用到的 switch.def (如果在格式中没有相应的宏) 文件。

语言切换命令

通常用户不需要对 babel 内部操作有太多的了解。新高级用户命令是：

```
\selectlanguage{ 语言 }
\foreignlanguage{ 语言 }{ 文本 }
\iflanguage{ 语言 }{是 文本 }{否 文本 }
```

第一条命令把 语言 切换为当前语言，而第二条命令用选定的 语言 设置一块 文本 ，在第三条命令中，如果当前用的是指定 语言 ，则执行 是文本 ，否则执行 否文本 。

另外，在 \languagename 中包含当前选定语言的名称。最后，任何语言名称都可以做为一个环境，这样可以设置不同于正文语言的一大段文本。

当然, 每个语言定义文件也可以拥有自己的特殊命令, 以简化输入操作, 见 D.1.1 和 D.1.2 节中相应于德语和法语中的示例。

语言定义文件的内容

虽然并不需要知道切换机制的工作原理，我们这里还是做一简单刻划，供有兴趣者参考。

增加的两条 babel 内部命令是：

\addlanguage{ 语言编号 } 以及

\adddialect{ 语言编号 1}{ 语言编号 2}

其中 语言编号 是一个内部的编号，它用来区分断词模式集合。\addlanguage 命令把其参数值设置成下一个可用的语言编号。在 babel 中 语言编号 的形式为 \l@语言；例如，\l@english 。在 initex执行时，会执行列在 language.dat 中每个语言名称对应的该类型命令。 \adddialect 命令把一个参数的值设成与第二个参数的值相同，这样两者就可以使用同样的断词模式集合。例如，在 english.ldf 中就有一条命令 \adddialect{\l@american}{\l@english} 。

语言定义文件中必须提供下述四条命令：

\captions语言 来重新定义类似于 \tablename 的名称命令；

\date语言 定义 \today ；

\extras语言 定义所有与语言有关的命令；

\noextras语言 去掉所有与语言有关的命令。

如果这里的 语言 并不是具有预存贮在当前格式文件中的断词模式的一种语言 (也就是说并没有定义 \l@语言) ，那就把它设成相应于第 0 号语言的 “方言” 。

最后再提一点，语言定义文件会调用 \selectlanguage{语言} 以激活该语言。其实现方法如下：

- 调用 \language\l@语言 以选择断词模式集合，
- 为了去掉可能存在的与语言有关的命令，调用 \originalTeX ，
- 激活 \caption语言 ， \date语言 ，以及 \extras语言 命令，以调用与语言有关的名称，日期和命令，
- 把 \originalTeX 重定义为 \noextras语言 ，这样下次语言切换时就会去掉这里与 语言 有关的所有功能。

§D.1.4 使用 \language 命令

前面已提到过，在 TeX的 3.0 版本或以后版本，可以在一个格式文件中保存不只一组断词模式。通过给 \language 记数器设成不同的值，可以在其中进行切换。

然而，还没有标准规定哪一种语言相应于哪一个编号。在 C.3.3 节中的 esperant 宏包， D.1.1 节中的 german 宏包都假定了某种顺序，但是在其它情形中就有可能行不通。 babel 系统调用了一个相当可靠的过程，这个过程也被 french 所使用。

在格式创建时读入的文件 language.dat ，由一组语言后加断词模式文件名组成。举例来说，如果 language.dat 包含

```
english   hyphen.tex
germanb   ghyphen.tex
francais  fhyphen.tex
```

那么在第 0, 1 和 2 号语言中就会分别上载保存在 hyphen.tex, ghyphen.tex 和 fhyphen.tex 中的断词模式，而 \l@english, \l@germanb, \l@francais 就分别定义成 0, 1, 2 号，以供 \selectlanguage 命令使用。因此 francais 语言就直接与法语断词联系在一起。在使用法语的研究所中，有可能用的是 1 号语言，而不是 2 号，但这是无关紧要的。关键在于编号保存在格式中，并且可以通过标准的名称来引用。

在 initex运行 (D.4 节) 时，通过提供 hyphen.cfg 文件，可以生成这样的格式。为了适应局部安装，所需要定制的唯一文件是 language.dat 。

§D.2 LaTeX与 PostScript

做为打印和绘图语言而引入的 PostScript ，已成为了一种打印质量良好的文件标准, 这样的文件拥有足够的弹性, 而且便于以电子版本传递到别处。现在的许多 DVI 驱动程序都可以把 TeX的 .dvi 输出转化成这种更具普遍性的输出格式。其中最出名的当属 dvips 了，这是一个由 Tomas Rokicki 开发的非经营性程序，可以从 CTAN 文件服务器上获取 (D.5 节) 。

PostScript 可以为 LaTeX提供许多额外的功能，例如引入外部图形，彩色, 放缩以及旋转。在 dvips 的安装中包含许多宏包以实现这些功能。然而，它们只在这种驱动程序中才可以使用；其它程序用的是不同宏包和命令。做为对 LaTeX 2ε的一部分正式扩展，为此已经开发了一组标准命令，而把与驱动程序有关的代码隐藏在 .def 文件中。通过这种方法，对所有驱动程序，高级用户命令具有相当的一致性，而只需要利用 graphics 或 color 宏包的选项来指定驱动程序。

由于这些功能并不只限于 PostScript ，因此我们在稍后的 D.3.3 节中再加以介绍。

§D.2.1 调用 PostScript 字体

利用 PostScript 驱动程序来输出 LaTeX文档，也就意味着可以使用许多 PostScript 字体，特别是现在 NFSS 成为 LaTeX 2ε的一部分后更是如此。

为了简化对这些字体的选择, 要找来一组宏包。在 CTAN 服务器 (见 353 页上的图表) 中 psnfss 目录中可以找到这些宏包。其由宏包文件以及所需的字体定义文件 .fd 组成，这些定义文件相应于 35 种标准 PostScript 字体，以

表 D.1: psnfss 宏包及相应字体

宏包	\rmfamily	\sffamily	\ttfamily
times.sty	TimesRoman	Helvetica	Courier
palatino.sty	Palatino	Helvetica	Courier
newcent.sty	NewCenturySchoolbook	AvantGarde	Courier
bookman.sty	Bookman	AvantGarde	Courier
avant.sty		AvantGarde	
helvet.sty		Helvetica	

及其它一些可用的商品化字体。这些宏包的用法相当简单；例如，times.sty 文件只是由三行文本组成，在 212 页上列出了其内容。所有这些宏包要做的事情就是重定义三个字体族 \rmdefault, \sfdefault 以及 \ttdefault 。

我们在表 D.1 中列出了相应于 35 种标准字体的宏包，以及它们做出的三个赋值。 (这 35 种字体包含了自己的斜体以及黑体变体。)

在 psnfss 目录中没有必需的字体尺寸文件 .tfm 。这些文件位于 CTAN 的 font/metrics 目录中，每种字体相应于一个子目录。例如， TimesRoman 字体的尺寸文件在 ptm 子目录中。

这些字体利用了 TeX的 *虚拟字体* 机制，即 TeX看到的实际上只是假字体，这种字体并不存在。由于 TeX只是读入 .tfm 文件，因此这是完全行得通的。驱动程序接着就读入 .vf 文件，而不再读像素文件 .pk 或 .pxl 。在虚拟字体文件中的指令告诉驱动程序如何生成每个字符：可以是绘制出来，也可以得自其它字体，或者进行变形。这也就是为什么当某一字体的 slanted 字体和小体大写变体并不存在时，可以用 “粗糙” 的方式得到 PostScript 变体。

甚至虚拟字体的字体布局也可以被重新设计。起初的 PostScript 字体所采用的编码框架并不遵从 CM 和 DC 布局 (见 E.6 节)，而虚拟字体则遵从这一布局。在计算机现代字体中，前 11 个位置是大写的希腊字母 (见 368 页上的布局 1)，而只有在 PostScript 符号字体中才能找到这些字母。 ptmr 虚拟字体为了遵从这一点，采用的方法是从原始的 TimesRoman 和 Symbol 字体中选取这些字符。

为了正确地打印出来结果，我们必须从尺寸目录中得到所有的 .tfm 和 .vf 文件。

§D.3 其它的 LaTeX附件

§D.3.1 DocStrip工具

我们知道 TeX程序既是一个文本格式化语言，也可以进行程序设计，根

据这一点，我们可以利用它，开发一组实用程序进行文件操作。有了这样的程序，那么就会非常方便了：如果你有 TeX，那么你就可以执行这些程序。DocStrip就是这样的一个程序，它由 Frank Mittelbach 开发，该程序是把文件中的注释行去掉。

粗粗一看，你可能会问，为什么要做这样一件事呢？因为注释总是被忽略的。在 LaTeX的原始发行版本中， Leslie Lamport 所给出的支持文件中有相当丰富的注释，其甚至比代码还要长几倍。在当时，存贮空间和速度都是非常珍贵的，因此为了实用的需要，他同时提供了没有注释的同样文件。为了以后参考，可以把有注释的版本存贮在其它地方。 Mittelbach 的想法就是创建一个程序，为了紧致存贮和加快上载速度，而精简掉那些注释行。

根据这个简单的概念，我们要对 DocStrip进行扩展，给它增加另外两个功能：根据处理时的选项，可以有选择地抑制或包含某些代码行；可以把多个文件，也称为模块，组合成单个文件。这也就是说，在一个源文件中可以存放有几个不同的 LaTeX宏包，一个宏包文件也可以利用不同的成分来构造起来。事实上，主要的 LaTeX 2ε生成文件 `latex.ltx` 就是由 30 多个单独文件构成的，而标准的类文件以及大多数公共代码就是利用其选项文件，从一个名为 `classes.dtx` 的文件中提取的。

只有利用 D.3.2 节中的集成文档系统，我们才能看到 DocStrip现在所具有的最重要应用。然而，它还有其它用途，例如在 B.3.2 节列出的通用参考文献样式文件。它有点儿类似于 Unix 系统中的 C 编译器和 `makefile` 工具，只是这里要运行在 TeX中，从而可以方便地移植到其它系统中。

交互执行

对一个文件执行 DocStrip的最简单方法就是用 TeX调用它 (也可用 LaTeX, 但 TeX速度更快)，例如，

```
tex docstrip
```

这样就会得到如下反馈：

```
**************************************************************
* This program converts documented macro-files into fast *
* loadable files by stripping off (nearly) all comments! *
**************************************************************
**************************************************
* First type the extension of your input file(s):  *
\infileext=
**************************************************
```

一种反应就是给出输入扩展名 (通常为 `.dtx`)；然后依次被要求给出输出扩展名，所需选项，最后是被处理文件的基本名。接着就开始执行。

©TeXGuru, August 16, 1999

这种方法有一些局限性，因为输入和输出文件有相同的基本名，只是扩展名不同而已，而且加到输出文件中开始与结尾时的注释是固定的。运行这个工具的更富有弹性的方法是采用批处理作业的形式。

利用批处理作业来执行

DocStrip的批处理作业就是一个文件，其由该工具识别的指令组成。例如，假设在 C.3.1 节示例的 fullpage 宏包被放到一个名为 fullpage.dtx 的档案源文件中，真正的宏包文件是 fullpage.sty，可以用选项 package 提取出来；那么批处理文件 fullpage.ins 就应该如下所示：

```
\def\batchfile{fullpage.ins}
\input docstrip

\preamble
This is a stripped version of the original file.
\endpreamble

\postamble
This is the end of the stripped file.
\endpostamble

\generateFile{fullpage.sty}{f}{\from{fullpage.dtx}{package}}
```

前两行是至关重要的： \batchfile 定义为包含指令的文件名称；接着上载 docstrip.tex(\input 命令的写法必须 **完全** 与这里给出的一样)，接着就读入并执行指定的文件，并 **停下来** ！剩下来的指令行当 DocStrip读入 \batchfile 时就被处理过了。事实上，它们可以一起位于另外一个没有这里前两行内容的文件中；唯一的要求是 \batchfile 必须精确定义为包含指令的那个文件名。

\preamble 和 \postamble 命令使我们可以在开始与结尾处包含解释性的注释。

主要命令是 \generateFile，它用输出文件名作为第一个参数值。第二个参数值为 t 或 f，告诉 DocStrip如果已经有一个与输出文件同名的文件存在时，是否给出一条警告消息。这里 f 表示 ⟨*false*⟩，不给出警告消息。第三个参数值由一条或多条 \from 命令组成，其两个参数值表示输入文件的名称以及有选择编码时的选项。

这样生成的 fullpage.sty 文件内容如下：

```
%% This is file `fullpage.sty', generated
%% on <1994/10/15> with the docstrip utility (2.2h).
```

```
%%
%% The original source files were:
%%
%% fullpage.dtx  (with options: 'package')
%% This is a stripped version of the original file.
\NeedsTeXFormat{LaTeX2e}
. . . . . .
\addtolength{\topmargin}{-1in}
%% This is the end of stripped file.
```

也可以用一个主批处理作业, 处理单个作业, 此时主作业用 \batchinput 输入小作业，而不能用 \input 。

去掉文本行的规则

\generateFile 命令对输入文件进行转化，逐行生成输出文件，所遵从的规则如下：

1. 以单个 % 开始的行都要被去掉。;
2. 而以两个 % 开始的行则保持不变;
3. 对于 %<opt>(或 %<+opt>) 开始的行，如果这里的 opt 是选定选项中的一个，就把余下的文本送到输出文件中；否则就把该行去掉;
4. %<!opt> 或 %<-opt> 开头的行，如果 opt 不在选定选项中，则把余下文本送到输出文件中；否则就把该行去掉;
5. 对于位于 %<*opt> 和 %</opt> 之间的所有行，会根据 opt 是否是选定的选项，来确定是否保留这些行。

选项可以进行逻辑组合，求反以及分组：

a&b	(a and b);
a|b	(a or b);
!a	(not a);
(a|b)&c	(c and one of a or b)

关于 DocStrip的详情，可以阅读用 LaTeX处理 docstrip.dtx 后得到的文档。

§D.3.2 建立 LaTeX代码的档案

为软件产品准备充分的档案是相当重要的，其一方面给用户提供了使用手册，另一方面也给出了程序员开发代码的细节。在起初的 LaTeX样式，以及基本的 latex.tex 文件中， Leslie Lamport 都给出了相当丰富的注释，但其中只是明了的普通文本。而 Frank Mittelbach 的 doc 宏包才第一次可以给源代码生成复杂的，集成化的档案。

©TEXGuru, August 16, 1999

所谓集成化档案,就是指在单个源文件中描述性语言与代码很好地揉合在一起。因此需要进行两次处理,一次提取出真正的代码 (D.3.1 节的 DocStrip工具),另一次显示档案 (doc 宏包)。这个宏包的基本想法为注释实际上也是通常的 LaTeX文本,它具有某些额外的功能,可以自动索引,并跟踪程序执行的记录。其自身的代码用一种特殊的 verbatim 环境形式表示。

档案是通过用 LaTeX运行一个特殊的驱动程序来生成的。相对于一个名为 fullpage.dtx 的源文件,该驱动程序的最简形式应为

```
\documentclass{article}
\usepackage{doc}
\begin{document}
   \DocInput{fullpage.dtx}
\end{document}
```

\DocInput 命令读入指定的文件,但它首先把 % 字符的功能从 "注释" 变为 "什么也不做"。这也就是说在 fullpage.dtx 中所有的注释行都变成要被处理的实际文本了。

原来,源文件的后缀为 .doc ,它是一个 .sty 文件,但用了特殊的 doc 宏包命令。我们可以把该文件重命名为 .sty ,作为宏包文件使用。随着 LaTeX 2_ε标准的出现, doc.sty 和 DocStrip成为基本安装中的一部分,这时不能再任选扩展名了。现在源文件标志为 .dtx ,充当档案驱动文件角色。也就是说我们只需要用 LaTeX处理这个文件就可以得到档案。在可以使用宏包 .sty 文件之前,必须用 DocStrip从源文件中提取出来。

我们下面相应于列在 C.3.1 节中的 fullpage 宏包,用 fullpage.dtx 源文件为例,来演示 doc 宏包是如何使得 .dtx 文件中的 "注释" 成为有用文本等功能。与其它情形一样,通过处理 doc.dtx 可以得到一个相当完整的使用手册。

描述部分

档案由两部分组成: *描述* 就是提供给用户的手册, *代码* 就是软件产品工作原理的详细介绍,由代码行组成。可以通过在导言中调用下述命令来抑制代码部分,只显示出描述部分:

```
\OnlyDescription
```

在描述部分可以使用的特殊命令有:

```
\DescribeMacro{\宏名}
\DescribeEnvironment{环境名}
```

它们要放在文本开头,演示新的高级命令 (宏) 或环境。这两条命令做两件事:把宏或环境的名称放在该处的页边注中 (阅读时便于检索),并在索引中插入一项条目。

由于在档案中经常要用到 \verb 命令以显示源文本，因此可以用下面的命令给出其缩写形式：

\MakeShortVerb{\ 字符 } 和 \DeleteShortVerb{\ 字符 }

其首先把给定 字符 变成 \verb字符 的简写，然后再恢复其原来用法。例如，在调用了 \MakeShortVerb{\|} 之后，|\mycom| 就会显示出 \mycom 。(上载了 shortvrb 宏包后，在任何文档中都可以使用这两条命令，这个宏包实际上就是从 doc 宏包提取出来的。)

如果已经关闭了注释字符 % 的功能，那么该如何向档案文本中添加注释呢？一种方法就是利用 TeX的条件语句 \iffalse 构成一个模块，即块注释，例如：

```
% \iffalse
%    These lines are ignored even when the
%    percent character is inactive
% \fi
```

另一种方法就是用 ^^A 代替 % ，这也是 doc 的一项特殊功能。

描述部分是用下述命令标志结束的：

\StopEventually{结束文本}

其中 结束文本 要位于文章的尾部；如果只显示出描述部分，那么就会紧接着显示 结束文本 ，然后结束档案。

代码部分

在代码部分应该包含的是更专门的材料，平常的用户不会对它有多大的兴趣。此处可以使用的特殊命令有

\begin{macro}{\ 宏名 } 文本与代码 \end{macro}

\begin{environment}{\ 环境名 } 文本与代码 \end{environment}

这两条命令也都会插入一条页边注，并在索引中生成一项条目。它们也组织可能存在的 \changes 命令，见下面的介绍。

在代码部分最重要的环境是 macrocode ，它以 verbatim 形式显示其内容，并且可以选择是否加上代码行编号。这个环境的形式多少有点儿特别：

␣␣␣␣\begin{macrocode}

代码行

␣␣␣␣\end{macrocode}

这里在 \end{macrocode} 前面的四个空格是必需的；而 \begin 前面的空格则是可有可无，但为了对称起见，最好还是加上。这一环境会统计其内部所有反斜杠的数目，供后面 校验和 测试使用，为找到的每条命令名称生成一项索引。因此它并单单只是一个 verbatim 环境！

代码部分的结尾部分应该是

\Finale

这条命令进行校验和测试，显示来自于 \StopEventually 中保存的 结束文本 。有可能在其后还有其它文本，只有当描述部分和代码部分都输出后才会显示它们。

宏的索引和修改的记录

在 doc 宏包中是通过上面列出的两条 \Describexxx 命令和两个环境，在索引中自动插入条目。另外，所有出现的命令都要生成索引。然而，只有在导言中使用了

\CodelineIndex 或者 \PageIndex

中一条时，索引功能才会有效。其中第一条命令用所位于的代码行编号来引用被索引命令，而第二行命令用的则是页码。对于后者，代码行没有编号，除非使用了 \CodelineNumbered 声明。

由于在代码中并不是所有命令都需要进行索引，如果处理的是标准 LaTeX 和 TeX 部分命令时更时如此，命令

\DoNotIndex{命令名称清 单 }

通常放在开头部分，而且可以重复使用，以剔除不需要生成索引的命令。

对代码中的命令自动生成索引，会显著减慢处理速度。一旦生成了索引文件 .idx ，那么后面的运行就不必再做这件事了 (除非对代码又进行了修改) 。命令

\DisableCrossrefs

会关闭生成索引的功能，但它可以被前面的 \EnableCrossrefs 否决，后一条命令就是抑制关闭命令的。

MakeIndex 程序 (8.4 节) 根据 .idx 数据文件，生成索引文本，为此必须用特殊的索引样式 gind.ist 执行：

makeindex -s gind.ist 文件名

索引样式文件 gind.ist 是从 doc.dtx 中提取出来的。

为了显示索引，命令

\PrintIndex

就放在希望索引所处的地方。其通常是 \StopEventually 中 结束文本 的一部分。只有在两次 LaTeX处理之间运行了 MakeIndex 程序，才能得到一个更新后的索引。

对软件修改的记录可以通过在档案任何地方插入下述命令来生成：

\changes{ 版本 }{ 日期 }{ 文本 }

插入的地方既可以是描述部分，也可以是代码部分。为了生成修改历史清单，可以调用命令

\RecordChanges

它必须放在导言中。这就使得修改条目放在汇总文件中，然后可以用 MakeIndex 进行处理：

```
makeindex -s gglo.ist -o 文件名.gls 文件名.glo
```

(索引样式文件 gglo.ist 也是从 doc.dtx 中提取出来的。) 这样修改历史就会显示在档案中

```
\PrintChanges
```

所位的地方，它通常也是 *结束文本* 的一部分。在 \changes 命令中的文本是有顺序的，首先是版本号，然后是其所处的宏或环境的名称，

测试完整性

如果要把源文件送到电子网络上，那么其就有可能被破坏或截短。为此可以进行两项测试。在档案靠近开头的地方 (必须是在代码部分前面)，放上

```
\CheckSum{数}
```

这样就会统计在 macrocode 环境中所有的反斜杠总数，然后 \Finale 命令就会把所得总和与这里给定的 *数* 进行比较。如果 *数*$=0$，那么真正的总和就会显示在终端上；否则，如果总和不等于 *数*，就会显示出一条错误消息，同时给出这两个值。

另外一个测试检查如果经过用不同字符集下的计算机系统传输后，字符集是否被破坏。

```
\CharacterTable
{Upper-case   \A\B\C\D\E\F\G\H. . .
. . . . .
. . . . .  Tilde   \~}
```

这里的参数值必须精确地与 doc 所要求的一样 (除有作用的 % 和多重空格外)。应当直接从 doc.sty 或者 doc.dtx 文件中复制这部分内容。

获取文件信息

在 doc 的 LaTeX 2ε版本中有一个新功能，那就是命令

```
\GetFileInfo{文件名}
```

这条命令要利用在 \Provides*xxx* 命令中给出的区别指定文件的可省发行信息来定义 \filename, \filedate, \fileversion 和 \fileinfo(C.2.1 节)。想法就是在 .dtx 文件中这些信息应该出现一次，而且只需要一次，但我们应该能知道它，以显示在文章的标题中。这些 \file*xxx* 命令就可以实现此目的。发行信息的形式必须为 *日期*，空白，*版本*，空白，*文本*。

ltxdoc 类

有一个名为 ltxdoc 的特殊类可用来帮助运行 doc 宏包。它用 doc 宏包启动 article 类，然后调用下述命令

```
\AtBeginDocument{\MakeShortVerb{\|}}
\CodelineNumbered
\DisableCrossrefs
```

并定义其它一些有用的命令，以帮助建立档案。详情见处理 ltxdoc.dtx 后的描述部分。它也有局部配置文件：如果有 ltxdoc.cfg，就把它读入进来。这可以把纸张尺寸或其它格式化选项传递给 article，用 \AtBeginDocument 方法调用 \OnlyDescription，等等。

.dtx 示例文件

为了说明上述功能的用法，我们下面列出 fullpage.dtx 源文件的一部分。开始几行确认的是在接受了恰当识别命令后，所有能从其中提取出来的文件 (.sty 宏包和 .drv 档案驱动程序)。日期和版本信息只需要出现一次，但会送到所有提取出来的文件中。

```
% \iffalse      (This is a meta-comment)
%% Copyright (c) 1994 Patrick W. Daly
\NeedsTeXFormat{LaTeX2e}
%<*dtx>
\ProvidesFile         {fullpage.dtx}
%</dtx>
%<package>\ProvidesPackage{fullpage}
%<driver>\ProvidesFile{fullpage.drv}
% \fi
%\ProvidesFile{fullpage}
                  [1994/02/15 1.0 (PWD)]
```

最后那条 \ProvidesFile{fullpage} 命令使得 \GetFileInfo 可以具有正确的作用；而前一条同样的命令没有任何内容，只是吸收 .dtx 文件被直接读入时的信息行。

接下来，给出驱动程序部分。当用 LaTeX直接处理文件时，这是它所见到的内容。

```
%\iffalse
%<*driver>
\documentclass[a4paper,11pt]{article}
\usepackage{doc}
\EnableCrossrefs
\RecordChanges
\CodelineIndex
\begin{document}
```

```
   \DocInput{fullpage.dtx}
\end{document}
%</driver>
%\fi
```

下面给出校验和以及不用给出索引的命令清单，然后就是文章的开始。

```
% \CheckSum{73}
% \DoNotIndex{\addtolength,\boolean,\ExecuteOptions,\ifthenelse}
% \DoNotIndex{\newboolean,\newlength,\ProcessOptions}
% \DoNotIndex{\RequirePackage,\setboolean,\setlength}
% \changes{1.0}{1994 Feb 15}{Initial version}

% \GetFileInfo{fullpage}
% \title{\bfseries A Package to Set Margins to Full Page}
% \author{Patrick W. Daly}
% \date{This paper describes package \texttt{\filename}\\
%       version \fileversion, from \filedate}
% \maketitle
% \MakeShortVerb{\|}
%
% \section{Purpose}
% To set a uniform margin of one inch or 1.5~cm on all four
% . . . . . .
```

下面跳到描述部分的结尾，代码部分的开头。

```
% \StopEventually{\PrintIndex\PrintChanges}
%
% \section{The Coding}
% The first thing is to read in the \texttt{ifthen} package,
% if it is not already there.
%    \begin{macrocode}
%<*package>
\RequirePackage{ifthen}
%    \end{macrocode}
%
% \begin{macro}{\DeclareOption}
% \begin{macro}{\FP@margin}
% Define the options with help of the length |\FP@margin|. The
% options |in| and |cm| select the actual margin size.
```

```
%    \begin{macrocode}
\newlength{\FP@margin}
\DeclareOption{in}{\setlength{\FP@margin}{1in}}
\DeclareOption{cm}{\setlength{\FP@margin}{1.5cm}}
%    \end{macrocode}
% \end{macro}
```

在文件的结尾部分结束了代码部分，并调用 `\Finale` 。

```
. . . . . .
\setlength{\topmargin}{\FP@margin}
\addtolength{\topmargin}{-1in}
%</package>
%    \end{macrocode}
% \end{macro}\end{macro}
% \Finale
```

§D.3.3 图形与彩色

正如在 D.2 节指出的那样， TeX的 PostScript 驱动程序的出现，使得我们可以扩展 LaTeX的能力，增加新的功能，诸如对外部图形文件的引入，放缩和旋转，以及彩色。由于这些驱动程序中的每一个都倾向于有自己的宏包和用户接口命令，因此我们必须把 LaTeX文档与某一个特定的驱动程序紧密关联在一起。

`graphics` 和 `color` 宏包为所有的驱动程序提供了一组相当一致的命令，而专门化的代码是放在特定的 `.def` 文件中，可以用这些宏包的选项来上载。因此只需要改变一下选项，就可以切换到另一个驱动程序；而正文并不需要进行修改。

它们所定义的命令可以是其它一些宏包的构造模块，这些新的宏包可以提供 (一定程度上的) 更轻松自在的语法，以实现上述功能。只要这些新的宏包是基于 `graphics` 和 `color` 的，那么它们就应当同样与所有支持的驱动程序兼容。

可以在 CTAN 服务器的 `graphics` 目录 (353 页上的图表) 中找到这些宏包。它们是由 Sebastian Rahtz 和 David Carlisle 提供的。

现在支持的驱动程序 (截止 1994 年) 有

`dvi2ps`	`dvipsone`	`dviwindo`	`textures`
`dvialw`	`dvitops`	`emtex`	
`dvilaser`	`dvitps`	`oztex`	
`dvips`	`dviwin`	`pubps`	

但是只有 `dvips` 和 `dvipsone` 才具有完备的功能，并且经过了测试。而其它

一些驱动程序可能有引入功能，但是也就只此而已。现在这一切还都在发展过程中，因此要想知道最新的情形，应查阅当前的资料。

有两个宏包可以使用，一个是比较基本的 graphics，另一个则是扩展了的 graphicx，它上载并使用前者的宏。独立的 color 宏包提供了处理颜色的宏包。这三个宏包上载时都可以有一个驱动程序名称做为选项，例如，

```
\usepackage[dvips]{graphics,color}
```

可以通过 graphics.cfg 和 color.cfg 文件进行局部配置，因为这两个文件如果存在，是会被上述调用读入的。

除了驱动程序名称外，上载 graphics 宏包时还可以使用其它一些选项：

draft(用 final 反过来) 使得不必真的进行引入，而只是留下空白；

hidescale 在要进行放缩的地方留下空白；

hiderotate 在要进行旋转的地方留下空白。

可以用这些选项得到一个草稿，尤其是如果预览程序不支持图形功能时更要如此。与此类似， color 宏包也识别选项

monochrome

为了校样的需要，它把所有的彩色命令转化成黑白色。

引入外部图形

我们的问题就是把其它某个程序生成的图形文件包含到文档中，做为插图使用。有可能需要对其进行放缩或者旋转 90° 。我们希望要利用剪刀和胶水进行的操作改由计算机实现。

基本命令为

\includegraphics[*llx,lly*][*urx,ury*]{文件名}

其中 *llx, lly* 是包围引入图形的 包络盒 的左下角坐标，而 *urx, ury* 则是其右上角坐标。也就是说，它们确定的是下剪刀的地方。可以给定单位 (如 [3cm,2in])，可是如果不给定单位，就假定是大点 (bp，每英寸 72 bp) 。如果只给出了一个可省参数，那它就是指右上角，而左下角假定为 [0,0] 。

如果没有给出包络盒的坐标,那么驱动程序要从其它途径来获取该信息，具体视图形文件的类型而定。例如，对于非常流行的 *encapsulated PostScript* 文件 (扩展名为 .eps)，包络盒信息就是从图形文件自身提取出来的。

利用 \includegraphics*，可以对图形进行剪切，只有指定范围的内容被包含进来。

放缩

在引入图形时，用的是其本来大小。为了放缩图形，可以用下面两条命令：

\scalebox{*h_scale*}[*v_scale*]{文本}

这条命令对 文本 内容应用水平和竖直放缩因子；如果没有给出 *v_scale* ，那么它就等于 *h_scale* ；

\resizebox{*h_length*}{*v_length*]{文本}

调整插图，使之具有给定的水平和竖直尺寸；如果有一个长度为 ! ，那么要对两个尺寸进行同样的放缩。星号形式的命令可以使 *v_length* 表示盒子的高度与深度之和，而不仅仅只是高度。对于上述两条命令， 文本 内容都可以是 \includegraphics 命令。

反射

盒子中的内容可以用下述命令进行水平反射：

\reflectbox{文本}

旋转

盒子中的空容相对于基线左端点进行旋转，采用的命令为

\rotatebox{角 }{ 文本 }

其中 角 以度数为单位，而旋转是反时针方向的。

graphicx 宏包

如果我们选用的是 graphicx ，而不是 graphics 宏包，那么在引入和旋转方面就可以用不同的接口。

\includegraphics[bb=*llx lly urx ury*, \angle=角, width=*h_length*,
height=*v_length*, scale=因子, clip=*true/false*,
draft=*true/false*]{文件名 }

其中的关键词顺序是无关紧要的，而且也不需要给出所有的。其绝大多数的含义是自明的，只是 clip= 等于 ⟨*true*⟩ ，表示要进行剪切；而 draft 意味着并不真的引入文件，而是为其留下适当的空白。也可以用命令的星号形式，这时 clip 的值为 ⟨*true*⟩ 。

\rotatebox 也用这些关键词进行了类似地重定义。

rotating 宏包

由 Sebastian Rahtz 和 Leonor Barroca 设计的 rotating 宏包尝试给出在某些程度上相对简化了的进行旋转的接口。它定义了

\begin{sideways} 文本 \end{sideways}

\begin{turn}{ 角 } 文本 \end{turn}

\begin{rotate}{ 角 } 文本 \end{rotate}

\turnbox{ 角 }{ 文本 }

这里 sideways 把 文本 旋转 90°， turn 则可以旋转任意角度。 rotate 环境与 \turnbox 命令是等价的：进行了旋转，但是所处的盒子为零尺寸，这样其内容会与周围文本重叠。

epsfig 宏包

由 Sebastian Rahtz 所开发的 epsfig 宏包，不但更新了原来的 (2.09) 版本，而且利用 graphics 命令重新实现了 Rokicki 的 epsf 宏包。这对于那些习惯于其语法的用户是非常有用的。在 epsf 中，语法为

\epsfysize=*y_size* 或 \epsfxsize=*x_size*

\epsf[*llx lly urx ury*]{文件名}

如果没有给出包络盒坐标，那就从引入文件中提取相应信息。

在 epsfig 宏包中定义的引入命令，也是利用关键词来输入参数：

\epsfig{file= 文件名， height= 高度， width= 宽度， clip=, silent=,
rotate= 角， bbllx=*llx*, bblly=*lly*, bburx=*urx*, bbury=*ury*}

为了与旧版本兼容，还有一条 \psfig 命令，它与上面的命令含义相同。缺省情形中 epsfig 宏包自动调用 dvips 驱动程序。

上述不同形式的旋转和引入命令表明现在没有一个定形的标准语法；然而，所有上述命令的基础都是 graphics 宏包，因此其应当在所有的驱动程序中都会正常工作。或者至少说，它们不会使驱动程序崩溃。

颜色

颜色的指定可以采用已定义的名称，也可以用如下形式

[模型]{ 定义 }

其中 模型 可以取的值为 rgb(红 (red), 绿 (green), 蓝 (blue)), cmyk(青 (cyan), 品红 (magenta), 黄 (yellow), 黑 (black)), gray, 或者 named 。而 定义 则是一串从 0 到 1 的数，表示模型中每个分量的大小。因此 [rgb]{1,0,0} 定义的是红色， [cmyk]{0,0,1,0} 定义的是黄色。而 gray 模型只有一个值。 named 模型是用于可识别颜色内部名称的驱动程序，例如 dvips 知道 68 种颜色。打开 dvips.def ，就会知道有哪些名称可以使用。

一种颜色可以如下定义

\definecolor{ 名称 }{ 模型 }{ 定义 }

这样 名称 就可以用到下面所有的颜色命令中。对所有的驱动程序，有些颜色是自动预定义好了的： red, green, blue, yellow, cyan, magenta, black, white 。

在下面的颜色命令中， 颜色定义 就是某一定义好的颜色，如 {blue}, 或者 [模型]{ 定义 } ，如 [rgb]{0,1,0} 。

\pagecolor颜色定义 为当前页和后续页设置背景颜色；

\color颜色定义 为一个声明，把后续文本设置成给定颜色；

\textcolor颜色定义{文本} 用给定颜色设置其 文本 参数；

\colorbox颜色定义{文本} 把参数放到一个盒子中，并以给定颜色做为其背景；

\fcolorbox颜色定义 1 颜色定义 2{文本} 类似于 \colorbox，只是用 颜色定义 1 做为框线的颜色，包围背景色为 颜色定义 2 的盒子；这两个颜色定义必须同为名称，或者同样的模型，对这后一种情形，只需给出一次模型名称；

\normalcolor 切换到在导言结束时激活的颜色。因此在导言中用 \color 颜色命令，可以改变整篇文档的标准颜色。其等价于字体选择中的命令 \normalfont 。

§D.3.4 其它有用的宏包

LaTeX3 项目组的成员个人也编写了许多宏包，为了便于使用，这些宏包收集在一个名为 tools 的特殊目录中 (见 353 上的图表) 。其中绝大多数是从 LaTeX2.09 时代就开始创办的，现在已更新到了 LaTeX 2_ε 。

我们这里扼要介绍一下其中某些宏包能做什么事；要想详细了解，只要用 LaTeX处理其 .dtx 文件就可以了。

array 是对标准 LaTeX中 tabular 和 array 环境 (4.8.1 节) 的重新实现，增加了许多功能。除了等价于 \parbox[t]{宽度} 的列定义 p{宽度}，还有 m{宽度}(类似于 \parbox{宽度}) 和 b{宽度}(类似于 \parbox[b]{宽度}) ，这三个定义都用给定的 宽度 做为列的宽度，只是各列相对于顶行，中间和底行对齐。也可以用 >{声明} 和 <{声明} 在一列的开始和结尾处插入声明，可以用这种方式来包含一个字体声明，作用于列中所有行，或者在数学模式中进出。也可以定义用户自己的列类型。

dcolumn 需要 array 宏包，在 tabular 表格中可以用小数点对齐。

delarray 需要 array 宏包，能在 array 环境外面加上大的括号符号，这样可以生成各种形式的矩阵。

hhline 需要 array 宏包，在表格的水平直线输入方面更富有弹性。

longtable (不需要 array 宏包，但认识其功能) 使我们可以创建长达几页的表格，中间自动进行分页。

tabularx 定义了 tabularx 环境，其类似于 tabular* ，生成具有希望宽度的表格，但只是伸展列宽，而不是伸展列间距。

afterpage 允许存贮代码，并插入在当前页的尾部。

enumerate 重新实现了 enumerate 环境，可以用一个可省参数值确定每个 \item 生成数字的样式。

fileerr.dtx 解开这个文件，会生成一组小文件，可以用来回答 TeX中文件

不存在的情形；它使得反应类似于通常的错误信息；文件名称为 h.tex, e.tex, s.tex 和 x.tex ；因此，回答 x⟨回车⟩ 会上载最后那个文件，从而结束输入。

fontsmpl 是一个打印希望了解的字体示例的宏包。

ftnright 在两列模式中把脚注放在列的右端。

indentfirst 对章节的第一段进行缩进，通常并不进行这种操作。

layout 定义了命令 \layout ，它生成当前页面格式的示意图，显示出各个参数的值。

multicol 给出了 multicol 环境，它是 \twocolumn 命令的扩充，把其文本以指定数目的列显示出来，而且在开始和结尾部分不开始新页，各列长度均衡。

showkeys 显示出所有的由 \label 和 \bibitem 定义，并由 \ref, \pageref 和 \cite 使用的交叉索引关键词；它们以边注和行间注释的形式出现，只适用于草稿，以检查交叉索引的情况。

somedefs 根据 \usepackage 中选项，在宏中只允许定义选定的命令。

theorem 推广了 \newtheorem 命令，使之成为更富有弹性的 “定理型” 环境。

varioref 定义了 \vref 和 \vpageref ，其类似于 \ref 和 \pageref ，它们会检测被引用对象是否就在前一页，如果是这样的话，就显示出类似于 “on the previous page” 这样的文本；实际显示的文本可以有两种变化。

verbatim 重新实现了 verbatim 环境，避免了长文本时的内存溢出；而且还定义了命令 \verbatiminput , 以输入一个文件, 并且按字面显示文本，还有一个 comment 环境，以生成源文件中的注释块。

xr 利用这个宏包可以用 \ref 交叉索引另外文档中的 \label 命令。

xspace 这个工具可以更正命令名称吞掉后接空格的问题; 因此如果命令 \PS 的定义为

```
\newcommand{\PS}{\PostScript\xspace}
```

那么 \PS file 用法就完全可以，没有必要用 \␣ 或 {} 结束 \PS 命令。

§D.4 LaTeX 2ε的安装

在发行的 LaTeX 2ε安装文件组中有一些操作指南，其中既有一般性的，也有针对特定操作系统的。后者所在文件的扩展名为 .txt ，而基本名就是 TeX供应者的名称，如 emtex.txt, oztex.txt ，而一般性指南放在一个名为 install.txt 的文件中。在进行操作之前，应首先仔细阅读这些指南。

所需要的文件，有可能是压缩的也有可能是未压缩的。压缩文件由大量的 .dtx 文件组成，其为源文件和档案文件的集成。通过用 initex执行批处理作业文件 unpack.ins 可以解开上述文件。这样就会得到必需的 .cls, .sty,

.clo 和其它文件。其也构造出来基础文件 latex.ltx，在生成格式文件时需要这个文件。

如果你所拥有的文件集中已经有了 latex.ltx，那就是说其已经被解开了。这就非常方便，因为视所用的计算机不同，解压缩操作可以化费 5 分钟到 3 小时的时间。但是另一方面，用网络传递压缩文件就会化费较少的时间。

接下来一步就是用 initex执行 latex.ltx 文件生成 LaTeX 2ε格式 latex。然而，在做这之前，要考虑许多结构特征。在处理过程中有多处要读入特定的文件，但是如果有基本名相同，扩展名为 .cfg 的文件存在，就会上载这个文件。这种方法可以让我们按自己的条件或意愿配置最终的格式。可能的配置文件有：

texsys.cfg 提供了为老版本的 TeX或者某些独特版本的 TeX增加相当与机器有关的调整能力；可以在特定的 .txt 或者 ltdirchk.dtx 文件中找到相应信息。

fonttext.cfg 如果这个文件存在，就会取代 fonttext.ltx 文件而被上载；其定义了处理时可用的字体；可以在 fontdef.dtx 中找到详情。

fontmath.cfg 如果这个文件存在，就会取代 fontmath.ltx 文件而被上载；其等价于数学字体中的 fonttext.cfg。

preload.cfg 如果这个文件存在，就会取代 preload.ltx 文件而被上载；其确定在格式中预先上载哪种字体；可以从 preload.dtx 中提取其它的 preload 文件，其每一个都可以重命名成 .cfg，从而实现它。

hyphen.cfg 如果这个文件存在，就会取代 hyphen.ltx 文件而被上载；其指定了断词模式以及赋值；默认情形中，在 hyphen.tex 中的模式被上载到第 0 号语言中；在 babel 系统中提供了如此的配置文件 (D.1.4 节)。

一旦建立起了一个配置文件，并且把它放在 TeX的读取位置，用 initex执行 latex.ltx 就会生成格式文件 latex.fmt。这个文件必须放在读取格式文件的位置，其它的类，选项和宏包也要放在 TeX读这些文件的地方。最后要定义 LaTeX处理命令，这样可以用 latex 格式调用 TeX，例如，

```
tex &latex
```

注意: LaTeX2.09 的格式名称为 lplain；因此格式名称自身就揭示了要被激活的 LaTeX版本。

§D.4.1 更新 LaTeX

计划一年更新两次 LaTeX，分别在 6 月份和 12 月份进行。在这些时间，就必须用新的 .dtx 文件生成 latex.ltx 或者直接得到该文件。在两次更新之间的重要修改是在 ltpatch.ltx 文件中进行补正的，这个文件要被 latex.ltx 文件读入。通过这种方法，我们需要得到补丁文件，并从已有的 latex.ltx 重新生成格式文件。

§D.5 LaTeX文件的来源

得到扩展 LaTeX软件的最简单方法就是利用各种教育机构支持的网络服务器。这些服务器也提供即时更新的标准 LaTeX安装版本， TeX扩展，在各种 PC 上运行 TeX(和 LaTeX) 的程序，不同打印机的驱动程序，其它语言的断词模式，等等。它们是 TeX和 LaTeX衷心用户的无穷宝库。

在美国、英国和德国有 TeX和 LaTeX的三个主要网络服务器，其构成了全面的 TeX文件网络 (Comprehensive TeX Archive Network, CTAN) 。它们不但随时更新，而且它们之间还要定期交换信息。这些系统都是基于 Unix 的，可以用文件传输协议 (File Transfer Protocol, FTP) 访问。

国家	IP 地址	IP 数
美国	`pip.shsu.edu`	192.92.115.10
英国	`ftp.tex.ac.uk`	134.151.79.32
德国	`ftp.dante.de`	129.206.100.192

要想连接到这些地址，只需要输入 `ftp 地址` 就可以了；经过几秒钟的时间，就可以用

```
user anonymous
```

登录，这时密码为你自己的 FTP 地址，当然任意文本都是可以的。现在主机就会给出如下提示

```
ftp>
```

表示它已经可以接受 FTP 命令了。用 `bye` 或 `quit` 命令结束登录。

可以用 `dir` 得到目录清单，这些目录通常还有自己的子目录，其名称表示其可能包含的内容。 `INDEX` 文件对每个子目录的目的进行了简单的介绍。相应于 LaTeX的子目录可以在 `macros/latex` 中找到。

在所有服务器上的目录结构是一样的。图 D.1 表示的是目录树的主要部分。

可以在 `base` 或 `unpacked` 子目录中找到最新版本的 LaTeX宏包，对于后一个目录，文件已经被解开， `latex.ltx` 是现成的。由其它用户奉献出来的 LaTeX扩展和一般样式文件位于子目录 `contrib/supported` 中，其中每个来源都放在单独一个子目录中。由 LaTeX3 项目组提供的扩展可以在 `packages` 子目录中找到，其中包含 `babel`, `graphics`, `tools` 等等。

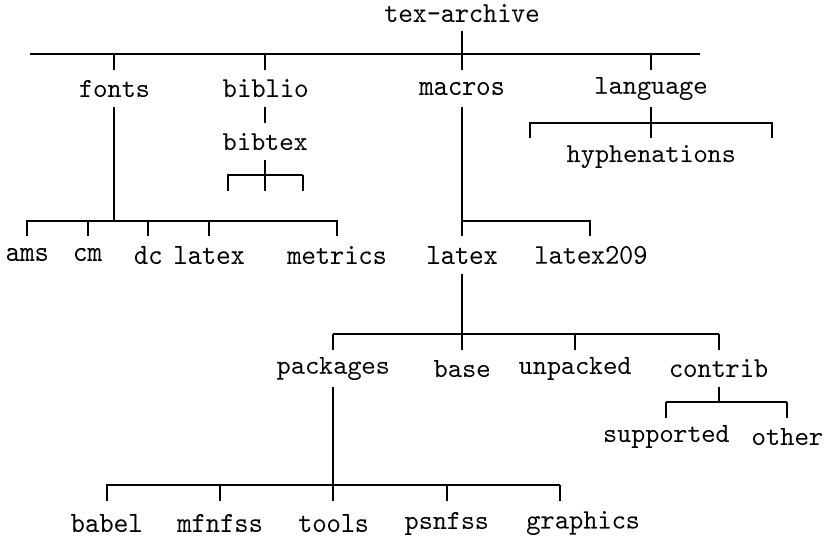
要想把文件复制到自己的计算机上，可以用 FTP 命令

```
get 文件名 . 扩展名
```

要想一次复制多个文件，就需要用 `mget` 命令，而用 `*` 做为文件指定中的通配符。

有许多文件已经用特殊的压缩程序进行了压缩，因此不能作为 ASCII 文件，而必须做为二进制文件来传输。可以用输入 FTP 命令 `binary` 来做到这

图 D.1: CTAN 服务器上的部分目录树结构



一转换，调用地方就是在相应的 get 或 mget 命令之前。对于文本文件，就必须切换回 ASCII，因为 ASCII 文件在二进制模式中通常无法正确地传输。

另外，任一目录的所有内容可以传输到一个存档 / 压缩系统中，方法就是先恰好移到该目录上方，并给出如下调用

get 目录名.tar	相应于 Unix 磁带存贮格式
get 目录名.tar.Z	相应于 Unix 的 tar，压缩后的
get 目录名.tar.gz	相应于 Unix 的 tar， GNU zip
get 目录名.zip	相应于 zip 压缩
get 目录名.zoo	相应于 zip 存档

例如，get tools.zip 会把整个 tools 目录作为 zip 文件进行转输。自然地，接受者在自己机器上必须有相应的解开程序。

§D.5.1 TeX组织

有大量的组织致力于 TeX和 LaTeX想法与程序的传播。这些用户组织也为那些无法使用网络的用户提供必须的文件。

可以从 CTAN 文件服务器上的一个名单中得到各种组织的地址。

©TEXGuru, August 16, 1999

附录E　计算机现代字体

当 Donald E. Knuth 发明 TEX程序时，他也同时给它装备了丰富的字符集或字体。 Knuth 称这些字体为 *计算机现代字体*，这样就不必依赖于所用计算机上的可用字体。在当时，打印机字体质量并不是很高，当然也不会一致。利用他所提供的字体， TEX能在所有打印机上生成相同的高质量输出。

LATEX 当然继承了这些字体，因此这些字体已经成为由 TEX 或 LATEX 生成的文档的标志。而而并不是必需如此的，因为 LATEX 并不需要与任何特定的字体集合捆绑在一起，特别是在当今新字体选择框架出现 (8.5 节) 之后，新框架极大地简化了字体安装的步骤。

但是对绝大多数 TEX/LATEX 用户而言，计算机现代字体仍然起着重要作用，因此有必要对其做一详细介绍。即使我们在 NFSS 中只是指定了一种字体属性，最终这些属性组合还是要与已有的某种字体名称联系在一起，见 8.5.8 节。对于标准安装版本，这些名称就来自于本附录余下部分描述的字体集合。

§E.1　简介

排版学就是对字样进行研究与分类的科学，而字样指的是可以上溯到五个半世纪前 Gutenberg 发明的活字 (*movable type*)(不是指中国人发明的活字印刷) 。从那时开始，人们创建了相当多的字体族，并把其分类为 Baskerville, Caramond, Univers, 等等。如此字体族中每个成员拥有同样的整体设计，或者称基本式样，但是可以有 slanted，斜体，黑体或细体等变体；当然，它们可以有不同的尺寸。

对字体族的分类通常是根据某种规则进行，主要的考虑原则是它们的用途。

Serif 字体 是一种在边缘有小水平线 (也称为衬线， serifs) 的字体，这样眼睛看起来感觉很好。经验表明这种字体适于阅读，因此通常用作正文主体。按 NFSS 的术语，这些字体称为 *罗马*(*Roman*) 字体。

Sans serif 字体 一种缺少衬线的字体。具有赤裸外形的这种字体通常用在标题或者页眉中。比较 sans serif 和 regular text 这两种字体就可以知道它们的区别了。

固定字体 是一种具有相同宽度的字体，它是从打字机字体演化而来，进而进入到计算机字体的行列中。按照正统的观点来看，在书籍印刷中是没有这种字体的用武之地的，因为这时成比例字体 (字母 i 要比 m 窄) 占主导地位。

装饰性字体 是一种由于某些不寻常之处而显得突出的字体。通常用它来吸引眼睛的注意力。这一族在形状和宽度方面还不完整，通常用在广告中。

数学字体 是数学作品中所需要的特殊符号全体。对它的进一步分类是无法与对文本字体的分类相提并论的。

书籍设计人员必须决定要使用的字样族类型。他 / 她有可能在正文用罗马 (有衬线的) 字体，页眉用 sans serif 字体，而且如果书中还有计算机程序代码，就用一种固定字体表示这部分内容。最后，如果作品中有数学内容，还需要符号字体。

在计算机现代字体集合中包含所有如此种类的族。然而，由于它们是在 NFSS 属性分类系统建立之前出现的， CM 字体命名法并不与这种框架完全一致。 NFSS 系统把用户从必须记住 CM 字体名称的细节中解脱出来，而只需要指定属性。当然， NSFF 安装中必须有字体描述文件 `.fd` ，这个文件把属性组合转化为真实的字体名称，或者进行某种过得去的替代。

§E.2 TEX基本字体的分类

任何特定的字体都可以用在 LaTeX中，方法之一就是利用 `\newfont`(4.1.6 节) 命令直接给出字体的名称，另外一种方法则是用 `\usefont`(8.5.1 节) 或 `\DeclareFixedFont`(8.5.4 节) 给出其属性。

所有 TEX 字体文件的名称都是以 `cm`(表示 “Computer Modern”) 字母开头的，后接一至四个字母描述字体的样式，最后是一两个数字指定设计尺寸的点数。这也就是用在 `\newfont` 命令中的字体基本名。因此基本名的完整语法为：

`cm`*xxnn*　　*xx*= 样式， *nn*= 设计尺寸

基本字体文件的扩展名为 `.tfm` ，含义为 “TEX font metric” 。

在 `.tfm` 文件中并没有包含生成符号的信息，其只是由字符尺寸信息组成，这些尺寸包括宽度，高度和深度。对于 slanted 字体，每个字母还有 “倾斜校正”(3.5.1 节) 。而且，要为一些字母组合指定与通常不同的间距，例如 AV 与 AV 的区别，或者有可能进行连写。最后， `.tfm` 中还包含字符的斜率 (对于非 slanted 字体，这个值为零) ，标准单词间距及其伸展和收缩量 (9.5.4 节) ， `em` 和 `quad` 间距的大小，句子结尾处间距。数学和符号字体在其 `.tfm` 文件中还包含更多的信息。

`\newfont` 命令的一般语法是

```
\newfont{\ 字体 }{ 基本名 scaled 因子 }
```

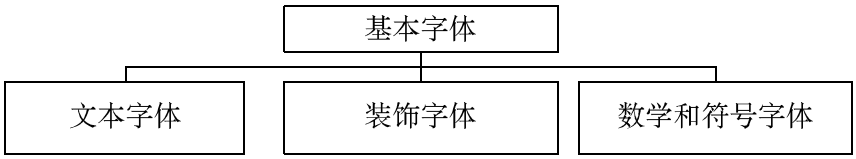
其中 *基本名* 指的是字体文件的外部基本名， *因子* 就是对字体进行放大时所采用的放缩因子的 1000 倍。这样所得的新命令 *\字体* 就是一个声明，它激活指定的字体，作用范围为当前环境结束，或者又调用了新的字体命令。如果所用的是 LaTeX ，那么 *因子* 可以取任何值，因为处理时就用 `.tfm` 文件中的数乘上这个因子。

也可以用下述命令得到同样的效果：

```
\DeclareFixedFont{\字体}{编码}{族}{序列}{尺寸}
```

这条命令自动地把 \字体 定义成适当的字体，必要时进行放缩。

下面我们给出 CM 字体的名称，并说明它们如何遵从 NFSS 属性模式。它们的组织如下：



§E.3 文本字体

用来设置普通文本的字体可以分为三族，其编码都为 OT1 。



§E.3.1 罗马字体族

罗马字体族就是有衬线的字体，间距成比例。这些字体通常充满了正文的主体。

cmr 族， OT1 编码

形状 =	n	sc	sl	it	u
序列					
m	cmr5 cmr6 cmr7 cmr8 cmr9 cmr10 cmr12 cmr17	cmcsc10	cms8 cms9 cms10 cms12	cmti7 cmti8 cmti9 cmti10 cmti12	cmu10
b	cmb10				
bx	cmbx5 cmbx6 cmbx7 cmbx8 cmbx9 cmbx10 cmbx12		cmbxsl10	cmbxti10	

直立罗马字体

直立罗马字体组包括在书籍印刷中最经常使用的标准字符集。字体样式标志为字母 r ，因此在这一族中所有文件名都以 cmr 开头，后接以点为单位的设计尺寸。设计尺寸的变化范围从 8 到 17 pt 。 cmcsc10 字体也属于该族，但形状为 sc 。这里的 csc 代表大写和小大写 (*captial and small caps*) 。因为在该族中的大写字母与 cmr10 中的一样，但小写字母是一种介于 cmr7 和 cmr8 之间尺寸的一种较小的大写字母。

cmr10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡’“”- – —*+/=( )[ ]

---

cmcsc10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
ABCDEFGHIJKLMNOPQRSTUVWXYZ ↑ ↓ ' ¡ ¿ ÆŒØSSIJ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:>?<¡’“”- – —*+/=( )[ ]

仔细查看 cmcsc10 字体，并与 cmr10 进行对比，我们会注意到所有小写字母的高度是一样的 (介于 cmr7 和 cmr8 的相应字母大小之间) ，但宽度要大一些，而且字母间距也要大些。另外连写被其它符号代替了。

cmu10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#£%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡’“”- – —*+/=( )[ ]

---

cmu10 字体实际上是一种没有倾向的斜体。这样就得到了一种直立的斜体字母。在 NFSS 框架中它占据了一种特殊的形状属性 u 。

Slanted 字体

本组的字体来源于直立罗马字体，只是对它进行斜率为 1/6 的倾斜；也就是说，这些字体向右倾斜了高度的六分之一。 *Slanted* 字体有介于 8 到 12 pt 间的四种设计尺寸。字体样式标志为 sl ，因此所有成员的文件名以 cmsl 开头，后接的 8, 9, 10 或 12 表示设计尺寸。这些字体属于形状定义 sl 。

cmsl10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&

©TEXGuru, August 16, 1999

abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡'“”- – —+/=( )[ ]*

斜体

在斜体字体组中的字体在基本形式上不同于直立和 slanted 罗马字体。它的倾斜值为 1/4，这要比 slanted 字体斜得更大。形状属性的标志为 `it`，而字体样式标志为 `ti`，表示文本斜体 (*text italic*)。因此字体文件的名称开头为 `cmti`，后接设计尺寸值。有从 7 到 12 pt 之间的五种尺寸。(数学字符字体属于另外一种斜体族，样式标志为 `mi`，代表数学斜体 (*math italic*)。)

`cmti10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#£%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡'“”- – —+/=( )[ ]*

黑体序列

这组字体还是由罗马字符组成，只不过现在字样要黑一些，线要粗一些。字体样式标志为 `bx`，代表 *bold extended*，因为这些字体要比同样高度的相应罗马字体宽一些。这也是它们的 NFSS 序列标志。提供介于 5 到 12 pt 之间的七种设计尺寸，名称分别为 `cmbx5 ... cmbx12`。对于 10 pt 的设计尺寸，同时还有一种 slanted 和斜体变体，名称为 `cmbxsl10` 和 `cmbxti10`。

在 10 pt 的设计尺寸中还包含另外一种字体 `cmb10`，这种黑体中的字母具有与普通罗马字母同样的宽度。其字体样式标志为 `b`，代表通常黑体。

`cmbx10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡'“”- – —*+/=( )[ ]

`cmbxsl10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ˙¨˝˜¸˛.,;:¿?¡¡'“”- – —*+/=( )[ ]

`cmbxti10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#£%ℭ
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
*ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ° ^ ¨ ˝ ˜ ¸ ˙ , ; : ¿ ? ¡ ¡ ’ “ ” - – — * + / = ( ) [ ]*

cmb10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ° ^ ¨ ˝ ˜ ¸ ˙ , ; : ¿ ? ¡ ¡ ’ “ ” - – — * + / = ( ) [ ]

§E.3.2 Sans serif 族

所有 sans serif 字体在其名称中都有样式标志 `ss`，因此其名称开头为 `cmss`。这也是其 NFSS 族的名称。

cmss 族， OT1 编码

形状 =	n	sl	—
序列			
m	cmss8 cmss9 cmss10 cmss12 cmss17	cmssi8 cmssi9 cmssi10 cmssi12 cmssi17	
bx	cmssbx10		
sbc	cmssdc10		
—			cmssq8 cmssqi8 cminch

直立 sans serif 字体

介于 8 到 17 pt 之间有五种设计尺寸的 sans serif 字体可以使用，名称分别为 `cmss5 ... cmss17`。还有一种字体 `cmssq8`，称为 *sans serif quotation*，其中的小写字母比通常情形按比例放大了一些。这种字体并不是按标准 NFSS 属性框架进行分类的。

cmss10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ° ^ ¨ ˝ ˜ ¸ ˙ , ; : ¿ ? ¡ ¡ ’ “ ” - – — * + / = ( ) [ ]

cmssq8

©TEXGuru, August 16, 1999

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ˚ ˆ˙¨˝˜ ¸-.,;:¿?¡¡’“”- – —*+/=( )[ ]

Slanted sans serif 字体

Slanted sans serif 字体是与同尺寸和编号的直立字体对应的。样式标志为 `ssi` 和 `ssqi` ，其中的 `i` 代表倾斜 (inclined) ，而不是斜体 (italic) 。正是因为如此，它们被赋予属性 `sl` 。倾斜大致为 1/5 。另外， quotation 字体 `cmssqi8` 并不包含在 NFSS 框架中。

`cmssi10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ˚ ˆ˙¨˝˜ ¸-.,;:¿?¡¡’“”- – —+/=( )[ ]*

`cmssqi8`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ˚ ˆ˙¨˝˜ ¸-.,;:¿?¡¡’“”- – —+/=( )[ ]*

Sans serif 黑体

只有设计尺寸为 10 pt 的 sans serif 黑体。其中一个名为 `cmssbx10` ，表示 sans serif bold extended ，其与 `cmss10` 的关系就如同 `cmbx10` 与 `cmr10` 的关系。另外一种黑体是 `cmssdc10` ，表示 sans serif demibold condensed 。

`cmssbx10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ˚ ˆ˙¨˝˜ ¸-.,;:¿?¡¡’“”- – —*+/=( )[ ]

`cmssdc10`

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ` ´ˇ˘¯ ˚ ˆ˙¨˝˜ ¸-.,;:¿?¡¡’“”- – —*+/=( )[ ]

---

cminch 字体

在 cminch 字体只有大写字母和 0,1,...,9 的数字，而再没有其它符号。其中的字符高度均为一英寸。字体的名称是一个例外，因为这时并没有用数字表示设计尺寸。对于这种情形，inch 既表示字体样式，也表示尺寸。在 NFSS 中也没有这种字体的分类。

cminch



§E.3.3 打字机字体

对于固定字体，每个字符都具有同样的宽度。甚至单词之间的距离也等于字体的宽度，而并不会进行伸展或收缩。然而，同所有 TEX 字体一样，单词间距总量并不与输入的空格数多少有关。标准的固定字体，都有一个字符表示打字机样式，它们是：

cmtt 族， OT1 编码

形状 =	n	sc	sl	it	—
序列					
m	cmtt8 cmtt9 cmtt10 cmtt12	cmtcsc10	cmsltt10	cmitt10	
—					cmtex8 cmtex9 cmtex10 cmvtt10

直立打字机字体

直立打字机字体有介于 8 到 12 pt 之间的四种设计尺寸。样式标志 tt 表示打字机类型 (*typewriter type*) 。因此文件基本名为 cmtt8 ... cmtt12 。

cmtt10

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%& ΓΔΘΛΞΠΣΥΦΨΩ
abcdefghijklmnopqrstuvwxyz æœø ßıȷ¡¿ ()[]{}\|/↑↓
0123456789 -+*<=> ¸_␣.,:;!?‘’˜`´˘˘¯˚^ˇ˝¨
```

大写与小大写

打字机大写与小大写字体 cmtcsc10 生成 10 pt 打字机字体的大写字母，而小写字母则为 8 pt 的打字机大写字母。

cmtcsc10

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%& ΓΔΘΛΞΠΣΥΦΨΩ
ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ SSIJ¡¿ ()[]{}\|/↑↓
0123456789 -+*<=> ¸_␣.,:;!?‘’˜`´˘˘¯˚^ˇ˝¨
```

有斜度的打字机字体

只在 10 pt 的设计尺寸时才有两种有斜度的打字机字体。 Slanted 的打字机字体类型 cmsltt10 与直立 tt 字体一样，只是具有斜率 1/6 。而另一方面，斜体打字机字体 cmitt10 用的则是斜体设计的字母，斜率为 1/4 。

cmsltt10

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%& ΓΔΘΛΞΠΣΥΦΨΩ
abcdefghijklmnopqrstuvwxyz æœø ßıȷ¡¿ ()[]{}\|/↑↓
0123456789 -+*<=> ¸_␣.,:;!?‘’˜`´˘˘¯˚^ˇ˝¨
```

---

cmitt10

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#£%& ΓΔΘΛΞΠΣΥΦΨΩ
abcdefghijklmnopqrstuvwxyz æœø ßıȷ¡¿ ()[]{}\|/↑↓
0123456789 -+*<=> ¸_␣.,:;!?‘’˜`´˘˘¯˚^ˇ˝¨
```

数学公式中的打字机字体

在数学打字机字体中的字母和数字与相应的直立 tt 字体一样。而其它字符都被特殊的数学符号代替。

这些字符集的设计尺寸有 8, 9 和 10 pt ，文件基本名为 cmtex8, cmtex9 和 cmtex10 ，表示 *typewriter extension* 。

这些字体也不适合于 NFSS 框架。实际上，它们是一组相当独特的群体，只是数学字体 cmex 的打字机类型版本。在任何 LaTeX版本中其都没有任何作

用。

cmtex10

·↓αβΛ¬∈πλγδ↑±⊕ϖ∂⊂⊃∩∪∀∃⊗↯↔→≠◇≤≥≡∨ !"#$%&’()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ [\]^_‘abcdefghijklmnopqrstuvwxyz{|}~∫

可变打字机字体

另外一种在 NFSS 属性系统中没有立足之地的独特字体是 cmvtt10，这种字体中的字符形状与打字机字体一样，但是间距是变化的。

cmvtt10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ¨˝˜ ¸˙,;:¿?¡¡’“”- – —*+/=( )[ ]

§E.4 装饰性字体族

有三族装饰性字体或者称特殊字体，它们只有唯一的设计尺寸和有效的属性。

	杂类族， OT1 编码， m 序列		
	形状 =	n	it
族	名称		
cmfr	Funny	cmff10	cmfi0
cmfib	Fibonacci	cmfib8	
cmdh	Dunhill	cmdunh10	

cmfr *Funny Roman*，有两种字体组成，一种是 cmff10，它具有负的倾斜度 -0.1，也就是说向左而不是向右倾斜；另一种是 cmfi10，它具有正的斜率 0.1。小写字体也要比通常的高。

cmfib *Fibonacci*，来源于数的 Fibonacci 序列。严格地讲，它是一种没有增宽的黑体，但却具有中间序列属性。

cmdunh *Dunhill*，其中的小写字母与 10 pt 时字符高度相同，但是大写字母，数字和 b, d, f, h, k, l 等有支柱的字母高度有些儿夸张。

cmff10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ`´ˇ˘¯°ˆ¨˝˜ ¸˙,;:¿?¡¡’“”- – —*+/=( )[ ]

©TEXGuru, August 16, 1999

cmfi10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#£%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ ˋˊˇ˘˜¯˚˙¨˝˛¸·,;:¿?¡!’“”- – —+/=( )[ ]*

cmfib8

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ ˋˊˇ˘˜¯˚˙¨˝˛¸·,;:¿?¡!’“”- – —*+/=( )[ ]

cmdunh10

ABCDEFGHIJKLMNOPQRSTUVWXYZ ÆŒØ @#$%&
abcdefghijklmnopqrstuvwxyz ff fi fl ffi ffl æœøßıȷ 0123456789
ΓΔΘΛΞΠΣΥΦΨΩ ˋˊˇ˘˜¯˚˙¨˝˛¸·,;:¿?¡!’“”- – —*+/=( )[ ]

§E.5 数学与符号字体

数学与符号字体就是由数学公式中需要的特殊字符组成。另外，在 LaTeX 的 picture 环境中也需要用到特殊符号，例如斜线，箭头，都可以在 LaTeX符号字体中找到。最后提一下，一些特殊记号字体也包括其中。

§E.5.1 数学字体族

数学字体所具有的编码框架与普通文本中的字体没有任何关联。正是出于这个原因，每一种字体都具有自己的编码和族定义。

	数学字体族		
编码 =	OML	OMS	OMX
族 =	cmm	cmsy	cmex
形状 =	it	n	n
序列			
m	cmmi5 cmmi6 cmmi7 cmmi8 cmmi9 cmmi10 cmmi12	cmsy5 cmsy6 cmsy7 cmsy8 cmsy9 cmsy10	cmex10
b	cmmib10	cmsyb10	

数学文本字体

这些字体由大写和小写的拉丁字母与希腊字母组成，还包括其它一些符号。由于在公式中有各种名称的斜体，而这些是基本的斜体字体，因此其字体样式标志为 `mi` ，表示数学斜体 (*mathematical italic*) ，以与通常的斜体区分开。字体有介于 5 到 12 pt 之间的七种设计尺寸。对于 10 pt 的设计尺寸，还有一种数学斜黑字体 `mib10` 。

cmmi10

$\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega\alpha\beta\gamma\delta\epsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\pi\rho\sigma\tau\upsilon\phi\chi\psi\omega\varepsilon\vartheta\varpi\varrho\varsigma\varphi\leftharpoonup\leftharpoondown\rightharpoonup\rightharpoondown\triangleright\triangleleft$
$0123456789.,</>\star\partial ABCDEFGHIJKLMNOPQRSTUVWXYZ$
$\flat\natural\sharp\smile\frown\ell abcdefghijklmnopqrstuvwxyz\imath\jmath\wp\vec{}\hat{}$

cmmib10

$\boldsymbol{\Gamma\Delta\Theta\Lambda\Xi\Pi\Sigma\Upsilon\Phi\Psi\Omega\alpha\beta\gamma\delta\epsilon\zeta\eta\theta\iota\kappa\lambda\mu\nu\xi\pi\rho\sigma\tau\upsilon\phi\chi\psi\omega\varepsilon\vartheta\varpi\varrho\varsigma\varphi\leftharpoonup\leftharpoondown\rightharpoonup\rightharpoondown\triangleright\triangleleft}$
$\boldsymbol{0123456789.,</>\star\partial ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
$\boldsymbol{\flat\natural\sharp\smile\frown\ell abcdefghijklmnopqrstuvwxyz\imath\jmath\wp\vec{}\hat{}}$

数学符号

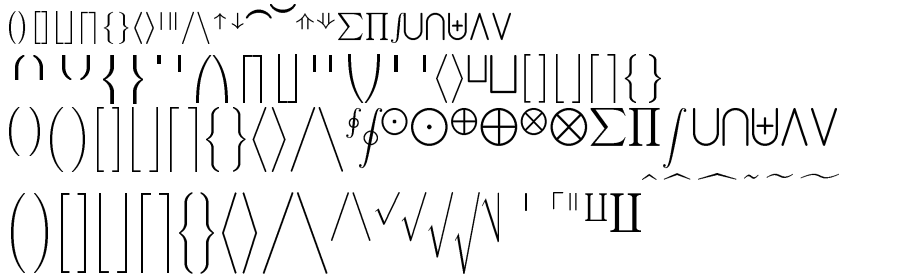
这些符号字体给出的是公式中其它数学符号，但尺寸可以变化的除外。字体样式代码为 `sy` ，表示符号 (*symbol*) 。有介于 5 到 10 pt 之间的六种设计尺寸。在 10 pt 时还有一种黑体符号字体 `cmbsy10` 。这些符号字体的文件基本名为 `cmsy5` ... `cmsy10` 。

cmsy10

$-\cdot\times*\div\diamond\pm\mp\oplus\ominus\otimes\oslash\odot\bigcirc\circ\bullet\asymp\equiv\subseteq\supseteq\leq\geq\preceq\succeq\sim\approx\subset\supset\ll\gg\prec\succ$
$\leftarrow\rightarrow\uparrow\downarrow\leftrightarrow\nearrow\searrow\simeq\Leftarrow\Rightarrow\Uparrow\Downarrow\Leftrightarrow\nwarrow\swarrow\propto\prime\infty\in\ni\triangle\triangledown/\!\!\!

这里 ex 表示 extension，其只有 10 pt 的设计尺寸。

cmex10



§E.5.2 其它字符字体

前面给出的 75 种字体是 TEX实现所提供的标准字体。也可以从一些商业机构或者公开来源得到其它的字体。要想知道那底有哪些字体可以使用，需要咨询一下你所在的计算机中心，或者查阅使用手册。

在 LATEX宏包中还有其它的一些字体，其中包括一些相当特殊的符号以及 picture 环境的构造元素，例如斜线和箭头。

还有一些记号字符字体，也包含在 TEX实现中，在 Donald E. Knuth 的书籍 *The TEXbook* 和 *The METAFONTbook* 中用到了它们。

LATEX的 lasy 字体

作为对数学符号字体 cmsy 的推广，LATEX提供了从 5 到 10 pt 六种设计尺寸的其它一些符号。这些字体文件的基本名为 lasy5 ... lasy10 。其中每个包含 16 个符号： ‹›⌃⌄◁⊴▷⊵℧⋈□◇∼⇝⊏⊐ 。

在 LATEX 2_ε中，只有用了 latexsym 宏包，这些字体才有定义。

生成图形的字体

在 LATEX的 picture 环境中所使用的特殊图形元素被保存在名称分别为 line10, lcircle10, linew10 和 lcirclew10 的字体文件中。前两个用来生成 \thinlines(6.5.1 节) 起作用时的直线、卵形线和圆。斜线和箭头 (6.4.3 和 6.4.4 节) 可以在 line10 中找到，而圆与卵形线片断 (6.4.5 和 6.4.6 节) 可以在 lcircle10 中找到。后面那对字体名称中增加了 w，表示粗线，这是用在 \thicklines 起作用的情形中。

记号字体

在记号字体 logo8, logo9, logo10, logosl10 和 logobf10 中只有七个字母 A, E, F, M, N, O, T，用来生成记号

METAFONT *METAFONT* **METAFONT**

§E.6 字体中的字符对应

除 cminch 外所有标准 TEX字体都是由 128 个字符组成，在 TEX内部给它们赋为 0 到 127 的数值。在字体中字符的布局见下面的表格。这里用的是八进制，而相应的十进制数放在对应符号旁边。

字体布局 1 表示的是罗马字体 cmr10 的字符对应关系，这是真正的 OT1 编码框架，是绝大多数具有同样符号的字体代表，因为可能在不同的样式中，这些符号具有相同的位置。文本斜体只是稍微有点儿区别。而大写及小大写字体 cmcsc10 则偏离得就有点儿多了，因为这时没有了连写。打字机字体则表明了其与布局 1 中的 OT1 标准顺序差得更远了。最后，数学和符号字体则具有截然不同的对应关系，因为其内容与文本字体并没有任何关联。所有这些有偏差的字体模式演示在布局 2–8 中。

注意：将来 TEX3.0 的字体中应包含 256 个字符。它们要遵从 T1 对应关系。

	0	1	2	3	4	5	6	7
’00x	Γ 0	Δ 1	Θ 2	Λ 3	Ξ 4	Π 5	Σ 6	Υ 7
’01x	Φ 8	Ψ 9	Ω 10	ff 11	fi 12	fl 13	ffi 14	ffl 15
’02x	ı 16	ȷ 17	` 18	´ 19	ˇ 20	˘ 21	¯ 22	° 23
’03x	¸ 24	ß 25	æ 26	œ 27	ø 28	Æ 29	Œ 30	Ø 31
’04x	˙ 32	! 33	” 34	# 35	$ 36	% 37	& 38	’ 39
’05x	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
’06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
’07x	8 56	9 57	: 58	; 59	¡ 60	= 61	¿ 62	? 63
’10x	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71
’11x	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
’12x	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87
’13x	X 88	Y 89	Z 90	[91	“ 92	] 93	^ 94	˙ 95
’14x	‘ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103
’15x	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
’16x	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119
’17x	x 120	y 121	z 122	– 123	— 124	˝ 125	˜ 126	¨ 127

字体布局 1 字符字体为 cmr10。这里给出的是字符与相应数值的标准对应关系。在下面的布局中给出的是非标准赋值。

©TEXGuru, August 16, 1999

	0	1	2	3	4	5	6	7
'00x	Γ 0	Δ 1	Θ 2	Λ 3	Ξ 4	Π 5	Σ 6	Υ 7
'01x	Φ 8	Ψ 9	Ω 10	↑ 11	↓ 12	' 13	¡ 14	¿ 15
'02x	ı 16	ȷ 17	` 18	´ 19	ˇ 20	˘ 21	¯ 22	° 23
'03x	¸ 24	ß 25	æ 26	œ 27	ø 28	Æ 29	Œ 30	Ø 31
'04x	- 32	! 33	” 34	# 35	$ 36	% 37	& 38	’ 39
'05x	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
'06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
'07x	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
'10x	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71
'11x	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
'12x	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87
'13x	X 88	Y 89	Z 90	[91	“ 92	] 93	^ 94	˙ 95
'14x	‘ 96	A 97	B 98	C 99	D 100	E 101	F 102	G 103
'15x	H 104	I 105	J 106	K 107	L 108	M 109	N 110	O 111
'16x	P 112	Q 113	R 114	S 115	T 116	U 117	V 118	W 119
'17x	X 120	Y 121	Z 122	– 123	— 124	˝ 125	~ 126	¨ 127

字体布局 2 字符字体为 `cmcsc10` 。与布局 1 的差别在于 11–15, 60 和 62 号符号。通常位于 11–15 号的连写被其它一些符号所取代。

	0	1	2	3	4	5	6	7
'00x	*Γ* 0	*Δ* 1	*Θ* 2	*Λ* 3	*Ξ* 4	*Π* 5	*Σ* 6	*Υ* 7
'01x	*Φ* 8	*Ψ* 9	*Ω* 10	*ff* 11	*fi* 12	*fl* 13	*ffi* 14	*ffl* 15
'02x	*ı* 16	*ȷ* 17	` 18	´ 19	ˇ 20	˘ 21	¯ 22	° 23
'03x	¸ 24	*ß* 25	*æ* 26	*œ* 27	*ø* 28	*Æ* 29	*Œ* 30	*Ø* 31
'04x	- 32	*!* 33	” 34	# 35	*£* 36	% 37	*&* 38	’ 39
'05x	*(* 40	*)* 41	* 42	+ 43	, 44	- 45	. 46	/ 47
'06x	*0* 48	*1* 49	*2* 50	*3* 51	*4* 52	*5* 53	*6* 54	*7* 55
'07x	*8* 56	*9* 57	*:* 58	*;* 59	*¡* 60	= 61	*¿* 62	*?* 63
'10x	@ 64	*A* 65	*B* 66	*C* 67	*D* 68	*E* 69	*F* 70	*G* 71
'11x	*H* 72	*I* 73	*J* 74	*K* 75	*L* 76	*M* 77	*N* 78	*O* 79
'12x	*P* 80	*Q* 81	*R* 82	*S* 83	*T* 84	*U* 85	*V* 86	*W* 87
'13x	*X* 88	*Y* 89	*Z* 90	*[* 91	“ 92	*]* 93	^ 94	˙ 95
'14x	‘ 96	*a* 97	*b* 98	*c* 99	*d* 100	*e* 101	*f* 102	*g* 103
'15x	*h* 104	*i* 105	*j* 106	*k* 107	*l* 108	*m* 109	*n* 110	*o* 111
'16x	*p* 112	*q* 113	*r* 114	*s* 115	*t* 116	*u* 117	*v* 118	*w* 119
'17x	*x* 120	*y* 121	*z* 122	– 123	— 124	˝ 125	~ 126	¨ 127

字体布局 3 字符字体为 `cmti10` 。与布局 1 的差别在于第 36 号 (*£* 代替了 $) 和第 38 号 (*&* 代替了 &) 符号。所有其它文本斜体都具有与之相同的模式。

	0	1	2	3	4	5	6	7
’00x	Γ 0	Δ 1	Θ 2	Λ 3	Ξ 4	Π 5	Σ 6	Υ 7
’01x	Φ 8	Ψ 9	Ω 10	↑ 11	↓ 12	' 13	¡ 14	¿ 15
’02x	ı 16	ȷ 17	` 18	´ 19	˘ 20	¯ 21	˚ 22	¸ 23
’03x	ß 24	æ 25	œ 26	ø 27	Æ 28	Œ 29	Ø 30	Å 31
’04x	␣ 32	! 33	" 34	# 35	$ 36	% 37	& 38	’ 39
’05x	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
’06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
’07x	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
’10x	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71
’11x	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
’12x	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87
’13x	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
’14x	‘ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103
’15x	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
’16x	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119
’17x	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	¨ 127

字体布局 4 字符字体为 cmtt10 。所有的 tt 字体都具有同样的模式，只是 cmvtt10 例外，其模式与布局 1 一致。差别在于 11–15, 60, 62, 92, 123, 124 号。而且斜体打字机字体 cmit 用斜体字体 £ 和 ℘ 取代了第 36, 38 号符号。 cmtcsc10 字体在 97–122 号是更小的大写字母。

	0	1	2	3	4	5	6	7
’00x	· 0	↓ 1	α 2	β 3	∧ 4	¬ 5	∈ 6	π 7
’01x	λ 8	γ 9	δ 10	↑ 11	± 12	⊕ 13	ϖ 14	∂ 15
’02x	⊂ 16	⊃ 17	∩ 18	∪ 19	∀ 20	∃ 21	⊗ 22	↹ 23
’03x	← 24	→ 25	≠ 26	◇ 27	≤ 28	≥ 29	≡ 30	∨ 31
’04x	32	! 33	" 34	# 35	$ 36	% 37	& 38	’ 39
’05x	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
’06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
’07x	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
’10x	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71
’11x	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
’12x	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87
’13x	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
’14x	‘ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103
’15x	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
’16x	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119
’17x	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	$\int$ 127

字体布局 5 字符字体为 cmtex10 。与布局 4 的差别在于 0–31 号和 127 号字符，现在这里包含的是特殊数学符号。

©TEXGuru, August 16, 1999

	0	1	2	3	4	5	6	7
’00x	Γ 0	Δ 1	Θ 2	Λ 3	Ξ 4	Π 5	Σ 6	Υ 7
’01x	Φ 8	Ψ 9	Ω 10	α 11	β 12	γ 13	δ 14	ϵ 15
’02x	ζ 16	η 17	θ 18	ι 19	κ 20	λ 21	μ 22	ν 23
’03x	ξ 24	π 25	ρ 26	σ 27	τ 28	υ 29	ϕ 30	χ 31
’04x	ψ 32	ω 33	ε 34	ϑ 35	ϖ 36	ϱ 37	ς 38	φ 39
’05x	$\leftharpoonup$ 40	$\leftharpoondown$ 41	$\rightharpoonup$ 42	$\rightharpoondown$ 43	$\triangleleft$ 44	$\triangleright$ 45	$\triangleright$ 46	$\triangleleft$ 47
’06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
’07x	8 56	9 57	. 58	, 59	< 60	/ 61	> 62	$\star$ 63
’10x	∂ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71
’11x	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
’12x	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87
’13x	X 88	Y 89	Z 90	$\flat$ 91	$\natural$ 92	$\sharp$ 93	$\smile$ 94	$\frown$ 95
’14x	ℓ 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103
’15x	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
’16x	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119
’17x	x 120	y 121	z 122	$\imath$ 123	$\jmath$ 124	$\wp$ 125	$\vec{}$ 126	$\frown$ 127

字体布局 6 字符字体为 cmmi10 。包括 cmmib10 在内的 cmmi 族字体， 11–39 号上是小写希腊字母， 40–47, 60–64 和 123–127 号上是一些数学符号。

	0	1	2	3	4	5	6	7
’00x	$-$ 0	$\cdot$ 1	$\times$ 2	$*$ 3	$\div$ 4	$\diamond$ 5	$\pm$ 6	$\mp$ 7
’01x	$\oplus$ 8	$\ominus$ 9	$\otimes$ 10	$\oslash$ 11	$\odot$ 12	$\bigcirc$ 13	$\circ$ 14	$\bullet$ 15
’02x	$\asymp$ 16	$\equiv$ 17	$\subseteq$ 18	$\supseteq$ 19	$\leq$ 20	$\geq$ 21	$\preceq$ 22	$\succeq$ 23
’03x	$\sim$ 24	$\approx$ 25	$\subset$ 26	$\supset$ 27	$\ll$ 28	$\gg$ 29	$\prec$ 30	$\succ$ 31
’04x	$\leftarrow$ 32	$\rightarrow$ 33	$\uparrow$ 34	$\downarrow$ 35	$\leftrightarrow$ 36	$\nearrow$ 37	$\searrow$ 38	$\simeq$ 39
’05x	$\Leftarrow$ 40	$\Rightarrow$ 41	$\Uparrow$ 42	$\Downarrow$ 43	$\Leftrightarrow$ 44	$\nwarrow$ 45	$\swarrow$ 46	$\propto$ 47
’06x	$\prime$ 48	∞ 49	$\in$ 50	$\ni$ 51	$\triangle$ 52	$\triangledown$ 53	/ 54	$\mid$ 55
’07x	$\forall$ 56	$\exists$ 57	$\neg$ 58	$\emptyset$ 59	$\Re$ 60	$\Im$ 61	$\top$ 62	$\bot$ 63
’10x	$\aleph$ 64	$\mathcal{A}$ 65	$\mathcal{B}$ 66	$\mathcal{C}$ 67	$\mathcal{D}$ 68	$\mathcal{E}$ 69	$\mathcal{F}$ 70	$\mathcal{G}$ 71
’11x	$\mathcal{H}$ 72	$\mathcal{I}$ 73	$\mathcal{J}$ 74	$\mathcal{K}$ 75	$\mathcal{L}$ 76	$\mathcal{M}$ 77	$\mathcal{N}$ 78	$\mathcal{O}$ 79
’12x	$\mathcal{P}$ 80	$\mathcal{Q}$ 81	$\mathcal{R}$ 82	$\mathcal{S}$ 83	$\mathcal{T}$ 84	$\mathcal{U}$ 85	$\mathcal{V}$ 86	$\mathcal{W}$ 87
’13x	$\mathcal{X}$ 88	$\mathcal{Y}$ 89	$\mathcal{Z}$ 90	$\cup$ 91	$\cap$ 92	$\uplus$ 93	$\wedge$ 94	$\vee$ 95
’14x	$\vdash$ 96	$\dashv$ 97	$\lfloor$ 98	$\rfloor$ 99	$\lceil$ 100	$\rceil$ 101	$\{$ 102	$\}$ 103
’15x	$\langle$ 104	$\rangle$ 105	$\mid$ 106	$\|$ 107	$\updownarrow$ 108	$\Updownarrow$ 109	$\backslash$ 110	$\wr$ 111
’16x	$\surd$ 112	$\amalg$ 113	∇ 114	$\int$ 115	$\sqcup$ 116	$\sqcap$ 117	$\sqsubseteq$ 118	$\sqsupseteq$ 119
’17x	§ 120	† 121	‡ 122	¶ 123	$\clubsuit$ 124	$\diamondsuit$ 125	$\heartsuit$ 126	$\spadesuit$ 127

字体布局 7 字符字体为 cmsy10 。在 cmsy 字体中有许多数学符号，以及 65–90 号位置上是花体字母 $\mathcal{A}\ldots\mathcal{Z}$ 。黑体版本 cmbsy10 具有完全一样的模式。

	0	1	2	3	4	5	6	7
'00x	0	1	2	3	4	5	6	7
'01x	8	9	10	11	12	13	14	15
'02x	16	17	18	19	20	21	22	23
'03x	24	25	26	27	28	29	30	31
'04x	32	33	34	35	36	37	38	39
'05x	40	41	42	43	44	45	46	47
'06x	48	49	50	51	52	53	54	55
'07x	56	57	58	59	60	61	62	63
'10x	64	65	66	67	68	69	70	71
'11x	72	73	74	75	76	77	78	79
'12x	80	81	82	83	84	85	86	87
'13x	88	89	90	91	92	93	94	95
'14x	96	97	98	99	100	101	102	103
'15x	104	105	106	107	108	109	110	111
'16x	112	113	114	115	116	117	118	119
'17x	120	121	122	123	124	125	126	127

字体布局 8 字符字体为 cmex10，由外观尺寸可以变化的各种数学符号组成。

©TEXGuru, August 16, 1999

	0	1	2	3	4	5	6	7
’00x	Њ 0	Љ 1	Ц 2	Э 3	І 4	Є 5	Ъ 6	Ћ 7
’01x	њ 8	љ 9	ц 10	э 11	і 12	є 13	ђ 14	ћ 15
’02x	Ю 16	Ж 17	Й 18	Ё 19	Ѵ 20	Ѳ 21	S 22	Я 23
’03x	ю 24	ж 25	й 26	ё 27	ѵ 28	ѳ 29	s 30	я 31
’04x	¨ 32	! 33	” 34	Ђ 35	◡ 36	% 37	´ 38	’ 39
’05x	(40	) 41	* 42	ђ 43	, 44	- 45	. 46	/ 47
’06x	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55
’07x	8 56	9 57	: 58	; 59	« 60	ı 61	» 62	? 63
’10x	˘ 64	А 65	Б 66	Ц 67	Д 68	Е 69	Ф 70	Г 71
’11x	Х 72	И 73	Ј 74	К 75	Л 76	М 77	Н 78	О 79
’12x	П 80	Ч 81	Р 82	С 83	Т 84	У 85	В 86	Щ 87
’13x	Ш 88	Ы 89	З 90	[91	“ 92	] 93	Ь 94	Ъ 95
’14x	‘ 96	а 97	б 98	ц 99	д 100	е 101	ф 102	г 103
’15x	х 104	и 105	ј 106	к 107	л 108	м 109	н 110	о 111
’16x	п 112	ч 113	р 114	с 115	т 116	у 117	в 118	щ 119
’17x	ш 120	ы 121	з 122	– 123	— 124	№ 125	ь 126	ъ 127

字体布局 9 字符字体为 `wncyr10`，这是一种由华盛顿大学提供的 Cyrillic 字体。

§E.7 Cyrillic 字体

包含 Cyrillic 字母表的字体并不是通常 TEX/LATEX宏包的一部分。在 1980 年美国数学会设计了一组该类字体，用于综述以俄语和其它斯拉夫语出版的书籍，因为这时需要用原来的语言输出标题。在 1988 年，华盛顿大学人文与艺术计算中心对 $\mathcal{AMS}$的 Cyrillic 字体进行了重新设计，以适应通常斯拉夫语研究的需要，给其增加了革新前的以及有重音的字母。字母的全局外观也大大改善 (见字体布局 9) 。

华盛顿大学的 Cyrillic 字体名都是前缀 `wncy`, 后接样式标志 `r`(直立), `b`(黑体), `i`(斜体), `sc`(小大写), 或者 `ss`(直立 sans serif 字体)，然后是设计尺寸的点数。

可以如通常那样用 `\newfont` 定义 Cyrillic 字体命令。例如,

```
\newfont{\cyr}{wncyr10}
```

就会把 `\cyr` 定义成一个声明，激活 10 pt 的直立 Cyrillic 字体。也可以用其它任何字体样式或者尺寸来调用这条命令。

Cyrillic 字体的布局经过了精心选择，这样输入文本时就可以直接输入通常的英文翻译。因此，在 83 号位置上的 Cyrillic C 就是通常拉丁字母 S 所在的地方。当所用字体为 Cyrillic 时，输入 S 就会输出等价的 C 。数学和标点符号也可以在标准位置上找到，因此可以如常输入。 “Санкт-Петербург 10?”

就是用输入 {\cyr Sankt-Peterburg 10?} 生成的。

激活 Cyrillic 字体的更好方法就是在 NFSS 下只要选择字体编码 OT2 就可以了，见 206 页上 8.5.2 节的演示。相应于 Cyrillic 字体的 .fd(字体定义)文件在建立时所采取的方式也与拉丁字体完全一致。这也就是说， wncyr10 具有与 cmr10 完全一样的属性，差别就在于后者的编码： cmr 族， n 形状， m 序列。(当然，如果 cm 字体不是当前标准字体，那么所需要做的也就只是选择 OT2 编码以激活 Cyrillic 字体。)

由于在 Cyrillic 字母表中拥有比拉丁字母表多的字母，其中有些字母就必须翻译成多个字母的组合。这一点是利用 TEX的连写系统来自动用程序实现的。例如， Ch 就看成是 81 号符号 Ч 的连写，这就如同在拉丁字体中把 fi 看成连写 *fi* 一样。也就是说只要简单地输入多字母翻译就可以了。生成 “Хрущев” 的输入就是 {\cyr Khrushchev}，其中 Kh → Х， shch → щ 。这种翻译连写框架是针对于英语的；其它语言也可以用自己的系统来复制这种拼写。例如， “Горбачёв” 在德语中就是 *Gorbatschow*，在意大利语中是 *Gorbaciov*，而在英语中是 *Gorbachev* 。其它框架在这种字体中并不会有作用。

并不是所有的字母都能如此自动生成 (例如 Горбачёв 中的 ё)，因此 $\mathcal{AMS}$提供了一个名为 cyracc.def 的文件，由重音字母和其它特殊功能 (例如软硬符号) 的宏定义组成。当在拉丁字体中调用这些宏，就会出现另外的翻译连写符号。

不幸的是， cyracc.def 中有些重音命令在 LATEX 2_ε下无法使用。其需要一个类似于 OT1enc.def 和 T1enc.def 的编码定义文件 OT2enc.def，利用 8.5.9 节给出的特殊命令定义那些与编码有关的命令。

§E.8 字体文件

§E.8.1 基本名

在每个 TEX实现中都应该包含列在 E.3 至 E.5 节中各种不同设计尺寸的多达 75 种标准字体。这些字体的基本名汇总如下：

cmr5	cmti9	cmssq8	cmu10	cmmi5
cmr6	cmti10	cmss8	cmff10	cmmi6
cmr7	cmti12	cmss9	cmfi10	cmmi7
cmr8	cmbx5	cmss10	cmdunh10	cmmi8
cmr9	cmbx6	cmss12	cmtt8	cmmi9
cmr10	cmbx7	cmss17	cmtt9	cmmi10
cmr12	cmbx8	cmssqi8	cmtt10	cmmi12
cmr17	cmbx9	cmssi8	cmtt12	cmmib10
cmcsc10	cmbx10	cmssi9	cmtcsc10	cmsy5

```
cmsl8   cmbx12    cmssi10   cmsltt10  cmsy6
cmsl9   cmb10     cmssi12   cmitt10   cmsy7
cmsl10  cmfib8    cmssi17   cmtex8    cmsy8
cmsl12  cmbxsl10  cmssdc10  cmtex9    cmsy9
cmti7   cmbxti10  cmssbx10  cmtex10   cmsy10
cmti8   cminch    cmvtt10   cmex10    cmbsy10
```

$\mathcal{AMS}$提供了黑体数学斜体 `cmmib` 以及黑体符号字体 `cmbsy`，设计尺寸为 5–9 pt。安装 LaTeX 2ε时，就假定这些字体是存在的，虽然如果没有它们的话，也没有什么坏处。

```
cmmib5  cmmib6  cmmib7  cmmib8  cmmib9
cmbsy5  cmbsy6  cmbsy7  cmbsy8  cmbsy9
```

除了标准 TeX字体外，还有一些特殊的记号字体以及 LaTeX字体：

```
lasy5  lasy8   lasyb10    linew10     logo8   logobf10
lasy6  lasy9   line10     lcirclew10  logo9
lasy7  lasy10  lcircle10  logosl10    logo10
```

在基本名末尾的数字表示设计尺寸的点数。对这些字体中的每一个，都存在一个以此为基本名，扩展名为 `.tfm` 的文件，例如 `cmr10.tfm`。在 TeX和 LaTeX处理文本文件的过程中，只用到这些 `.tfm` 文件。其由尺寸信息组成，还有字体中的其它字符以及符号，见 356 页上的说明。但 `.tfm` 文件并没有说明符号到底是什么样子的。

只有要用驱动程序把输出送到打印机上才会需要如何构造这些字母和符号的信息。这些信息存贮在其它的文件中，除各种基本尺寸外，还可以进行不同的放大。

§E.8.2 字体放大

TeX字体通常的放大幅度是 1.2 的幂次以及 $\sqrt{1.2}$。在 LaTeX中，字体尺寸的选择是利用 4.1.2 节的命令实现的，这些命令会检测是否有要求尺寸的相应字体，如果没有，就会对另外一种设计尺寸进行适当的放大。LaTeX2.09 是从 `lfonts.tex` 文件中接受字体信息的，而 LaTeX 2ε用的则是字体定义文件 (`.fd`, 8.5.8 节)。对字体的放大也可以用下述命令来选择：

```
\newfont{\ 字体命令 }{ 基本名  scaled  放缩因子 }
```

这样就会把 `\`字体命令 定义成激活放大 *放缩因子*/1000 倍的 *基本名* 字体的命令。也就是说 *放缩因子* 是等于放大倍数 1000 倍的整数。如果用的就是设计尺寸，那该数就是 1000；对第一个放大幅度 $\sqrt{1.2}$，其值为 1095；然后接下来就是 1200, 1440, ...。TeX命令 `\magstep`n 可以生成 1.2^n，而 `\magstephalf` 就是 $\sqrt{1.2}$。

10 pt 的字体被放大 1.2 倍并不与设计尺寸为 12 pt 的字体相同，虽然这种差别是很难发现的，例如，

12pt unscaled 10pt scaled by 1200

当放大倍数是 1.2^3，即放缩因子为 1728 时差别就比较明显了：

17pt unscaled 10pt scaled by 1728

放大的字体虽然可能具有与原始字体同样的高度，但字符显得更黑，并且要宽一些。这是因为小设计尺寸字体在字母高度和宽度以及线粗等方面的关系与大尺寸字体的不同。然而，当把它们放大到同样高度时，所有其它度量都同比例地放大。当字体以设计尺寸显示时是最好看的。

包含字体像素式样的文件完整名称与打印机分辨率以及代码系统有关。这些文件的基本名仍然是前面给出的，但扩展名要反映放缩因子或者基本分辨率。对于 300 dpi(每英寸小点数) 的比较常见的激光打印机，扩展名为：

放大倍数	放缩因子	像素代码 源 300pxl	像素代码 源 200pxl	压缩代码
1.0	1000	1000pxl	1500pxl	300pk
$\sqrt{1.2}$	1095	1095pxl	1643pxl	329pk
1.2	1200	1200pxl	1800pxl	360pk
1.2^2	1440	1440pxl	2160pxl	432pk
1.2^3	1728	1728pxl	2592pxl	519pk
1.2^4	2074	2074pxl	3110pxl	622pk
1.2^5	2488	2488pxl	3732pxl	746pk
1.2^6	2986	2986pxl	4479pxl	896pk
1.2^7	3583	3583pxl	5373pxl	1075pk

对于 10 pt 罗马字体的字符集，在放大 1.2^2 倍后的像素文件的名称应为 `cmr10.1440pxl`, `cmr10.2160pxl` 或者 `cmr10.432pk` 。前两者的差别在于：所有像素文件都是针对于特定分辨率设计的，如果用的是 300 dpi，那么就会得到正确的输出。在 `pxl` 前面的数字就是放缩因子。然而，如果像素文件是针对于 200 dpi 的打印机创建的，那么乘上 2/3 因子后的输出结果就会太小。为了校正这个问题，需要再进行 1.5 倍的放大。

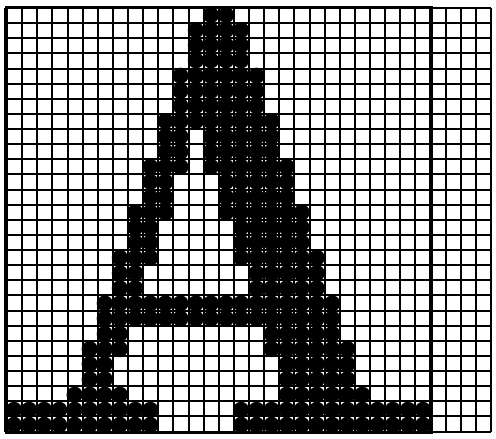
压缩代码的扩展名为 `.pk`，其前接设计尺寸的每英寸小点数。如果其为 300，输出是送到同分辨率的打印机上，那么就会得到正确的大小。如果创建的是 360 dpi 像素文件，还送到那个打印机上，结果就会比设计的大 1.2 倍，也就是说放大了这个倍数。

在有些计算机系统中，尤其是 PC 机上，像素文件的组织方式是不同的。这时它们都具有相同的扩展名，即 `.pxl` 或 `.pk`，具体与所用代码有关，但保存在不同的目录中，每个目录相应于一个放大倍数。因此目录名类似于 `360dpi` 的地方保存的就是所有放大倍数为 1.2 的像素文件 (假设打印机分辨率为 300 dpi) 。

©TEXGuru, August 16, 1999

§E.8.3 像素代码

用打印机输出每个字符, 都是用一系列的小点表示, 这些小点称为图形元素, 或者像素。像素的大小与打印机的分辨率有关。如果分辨率为 300 dpi , 那么一个像素的直径大约为 0.08 mm 。下面给出的是 10 pt 罗马字体中字母 A 的放大照。



左图每个小方框中的内容在计算机中都存贮为 0 或 1 。零表示白, 即空方框, 而 1 表示黑点。字母 A 由 28 行组成, 每行有 28 个点。顶部两行的编码为

0000000000000110000000000000

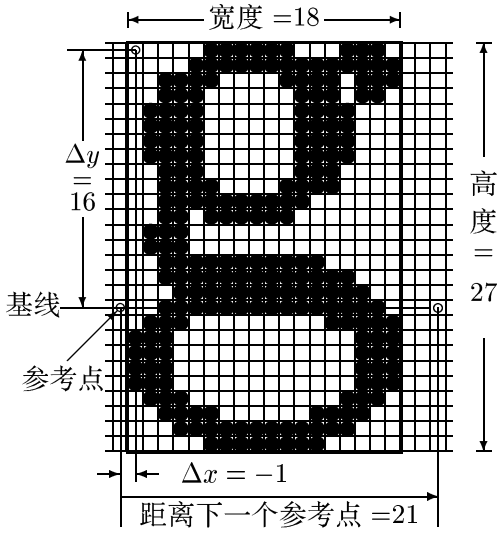
0000000000001111000000000000

而底行的编码为

1111111111000001111111111111

这就可以完整表示该字母。这样的二进制字符串不可能无穷长。计算机通常以 32 位 (即计算机中字的长度) 为一块。因此要把上面的行通过加 4 位零填补到 32 位。如果字符的宽度超过了 32 位, 那么每行就需要两组或多组 32 位。

按照计算机术语, 一组 8 位构成一个字节 (byte) 。因此 32 位的字中有 4 个字节。这里的字母 A 需要 28 个字或者说 128 个字节来存贮其式样。整个字体由 128 个字符组成, 因此完整的像素文件大概需要 14 000 个字节来表示。然而, 事情并不是到此为止, 每个字符还需要其它的信息, 从下面字母 g 的演示中可见一斑:



符号的像素式样可以包围在一个极小盒子中, 这个盒子恰好容纳了所有的黑点。对于字母 g , 这个盒子有 27 点高, 18 点宽。基线到顶行的点数用 Δy 表示, 等于 16 个像素。水平对齐用的参考点位于基线上, 距最左边列的点数用 Δx 表示; 负值意味着参考点在第一列的左边。这里 $\Delta x = -1$ 。最后, 这里同时要给出下一个参考点的位置, 对于 g , 其值为右边 21 个像素。

必须为每个符号在像素文件中保存这些值。极小盒子的高度和宽度, 以

及 Δy 和 Δx 的值，以像素为单位，每个需要 16 位，即半个字长。表示到下一个参考点距离的项要以 `tfm` 为单位，即其要与 `.tfm` 文件中的字符宽度一样。用四分之一的字长指定符号的像素式样在像素文件中的开头。

用 $128 \times 4 = 512$ 个字表示附加信息，构成一种字体中 128 个字符的一类目录表，其要位于像素文件的尾部。后接另外四个字，给出一般的字体信息，包括设计尺寸和放大倍数。像素文件的第一个和最后一个字都是 1001 的标志。因此一个像素文件是由像素式样外加 518 个字构成的附加信息。进一步的详情可能只有 `.dvi` 驱动程序的编写人员会感兴趣，可以在更专业的 TEX 文献中找到。

在输出时，如果对每个字符都要把其像素式样送到打印机上，那么打印速度会显著除低。实际上，激光打印机有自己的内存，因此要被传递的式样只需进行一次就可以了。这些被存贮起来的式样可以用该字符的代码号调用，这些代码号只有一个字节。它们就是列在 E.6 节字体布局中的代码号。

§E.8.4 压缩代码

从前一节内容可知，像素文件需要相当可观的存贮空间，尤其当处理的是放大字体时更加如此。例如，在放缩因子为 1440 时，相应于 `cminch` 的像素文件占据了总数为 705 376 的字节。

出于这个原因，Stanford 大学的 Tomas Rokicki 设计了一种压缩过程，用更紧致的形式保存像素信息。这个程序的基础就是点要么为黑，要么为白，而且一个点后接的更经常是同样点，而很少是相反的点，即像素的分布并不是随机的，而是按某种有序的方式出现的。

因此现在代码变成了一列数，其指定接续的有多少个同样的点。相邻点给出的相反颜色的连续点有多少个。当然，必须给出开始像素的颜色。对于前一节的字母 g 示例，开始颜色为白，代码的开头部分为

$$5/6/3/3/\underline{5}/14/2/4/4/1/3\ldots$$

该符号以五个白色像素开始，后接六个黑色的，再是三个白和三个黑。在第一行的最后那个像素以及第二行开头四个像素是白的，因此这里给出的是五个白的，其分散在两行上。在解码时这不会导致任何问题，因为极小盒子的宽度是已知的，所以 $\underline{5}$ 个像素是会正确地分散在两行上的。

数 0 从不会出现。对像素式样的分析可知在 90% 的情形中，同色的相邻像素数目要小于 14 。这也就是系列数中保存为 4 字节数，即 0–15 之间的值。0, 14 和 15 三个值有特殊意义。零用来表示一种颜色的像素点数大于 13 ，下面那个仍旧表示同色的像素。由于分析表明，在 37% 的情形中，一行后接的是同样模式的行，数 15 就表示这种情形。最后，数 14 表示有相同式样的行超过了两行，其后接的数表示到底有多少行。

即使是诸如极小盒子的高度和宽度，参考点的位置等附加信息也可以用

压缩方式存贮。这些详情只有在 .dvi 驱动程序的解压缩过程中才需要。

上述过程的效率是非常巨大的。 cminch.432pk 文件保存的是放大倍数为 1.44 的 cminch 字体，这时只需要 32 644 字节，只有等价像素文件大小的 5% 。这是一个极端例子，对 75 种标准字体有前四个放大幅度中，压缩形式所需的空间大约只有像素格式所需空间的 20% 。当放大倍数更大时，节省的比率就越大。

时至今日，已经很少能见到 pxl 代码了。其已经完全被 .pk 代码格式所代替。

§E.9 对 METAFONT的注释

METAFONT是一个设计和开发字符字体的程序，由 TEX 程序的发明者 Donald E. Knuth 编写。如果在你所用的计算机上有 METAFONT ，那么可以使用任何想要的 (更大或更小) 放大倍数，来重新生成已有的字体。

也可以用它创建诸如阿拉伯文或者希伯来文等全新的字体集合，虽然只有专家级人物才可能做到，因为其需要对 METAFONT的内部语言有相当深入的了解。本书并不是要讲解 METAFONT的，因此不会给出更多的信息。这里只是描写如何利用已存在的像素字体生成新的像素文件。

对 METAFONT程序的调用方式与所用的计算机操作系统有关。一旦启动了该程序，其就会在屏幕上显示出版本号以及上载的其它文件。然后就等待用户的反应，提示符为 ** 。反应必须是下面的形式：

\mode=localfont; mag=*nn*; input 文件

后接 ⟨回车⟩ 。这里 mag= 的值 *nn* 表示放大倍数；如果忽略该值，或者设为 1 ，那么用的就是设计尺寸。 input 的 文件 条目就是 E.8.1 节 75 种标准字体之一的基本名。做为 METAFONT基本实现的一部分，应该有 75 个同基本名，扩展名为 .mf 的文件。这些文件由具有 62 个参数的字符定义组成，这里不对其细节过多涉及。其它诸如相应于 Cyrillic 或特殊字符集的 .mf 文件也可以输入在 文件 所处的地方。

现在 METAFONT就会处理并生成选定字体在指定放大倍数下的像素文件，并在屏幕上显示字体名称和符号代码数。最后，程序显示出 * 并等待新的用户指令。反应 end⟨回车⟩ 就会结束该程序，把控制交还给操作系统。

METAFONT运行的结果是生成两个新文件，一个是字体的 .tfm 文件，另一个则具有相同的基本名，扩展名为 .xxxgf ，其中 gf 表示普通字体 (generic font) 。 *xxx* 反应了选定的放大倍数：如果放大倍数为 1 ，那么 *xxx* 就是以每英寸点数 (dpi) 表示的打印机分辨率。 METAFONT是从 \mode=localfont 中获取打印机信息及其分辨率的。在原始的版本中，这个分辨率应为 200 dpi ，但在任何安装中，应把 localfont 改成适应于所用打印机。

一个 .tfm 文件可以用于所有同基本名，放大倍数不同的字体。在有些地方， localfont 被进行了修改，只生成设计尺寸的 .tfm 文件，如同选定的是 mag=1 。

现在还需要把普通字体代码转化成驱动程序 (如上面描述的压缩代码) 能识别的形式。这是利用附属程序 gftopk 实现的。假设我们已经利用 METAFONT为 300 dpi 打印机生成了放大倍数为 1.44 的 cmr10 字体，这样我们就拥有了 cmr10.432gf 文件。

```
gftopk cmr10.432gf
```

的调用就进行了转化，生成压缩代码文件 cmr10.432pk 。

对于那些只能用原来像素代码的老驱动程序, 可以用附属程序 gftopxl 。还是上面的那个例子,

```
gftopxl cmr10.432gf
```

就生成 cmr10.1440pxl ，遵从 376 页表格中描述的命名约定，这里假定打印机分辨率为 300 dpi 。

通常 TEX的放大幅度为 $\sqrt{1.2}$ 或者 1.2 的整数次方。在命令行 mag= 中既可以输入十进制小数表示放大倍数，也可以用 magstep n 表示 1.2^n 。半幅度 $\sqrt{1.2}$ 可以输入为 magstep0.5 。

METAFONT允许对基本字体进行相当大的变体。无畏的用户可以尽情地对其进行操作。无论进行了怎样的操作，不要改动原来的 .mf 文件，要把新生成的字体起个别的名字。列在 374 页上的 75 种 TEX字体应保持为标准。

关于如何使用与开发 METAFONT程序的详情, 可以阅读 Knuth 的书 *The METAFONTbook* 。警告：玩耍 METAFONT会上瘾，并且耗时巨大。

为了适应于 TEX3.0 中多语言应用的发展和每种字体有 256 个字符，Donald E. Knuth 把 METAFONT扩展到了 2.0 版。现在他不再希望对这两个程序进行新改进了，所做的只是对已有错误进行修正。为了强调这一决定，从现在开始，他让 TEX的版本号收敛到 $\pi(3.14159\ldots)$ ， METAFONT的版本号收敛到 $e(2.71828\ldots)$ 。现在 TEX版本为 3.1415 ， METAFONT版本为 2.71 。这个决定的后果就是，如果有用户组织对这两个程序进行了大发展，那么就必须给它起个新名称，因为 Donald E. Knuth 保留对这些程序名称的版权。

©TEXGuru, August 16, 1999