

TEX *for the* Impatient

Paul W. Abrahams

with

Karl Berry

Kathryn A. Hargreaves

TeX 急就帖

2014 年 12 月 1 日

‘**TEX**’ is a trademark of the American Mathematical Society.

‘**METAFONT**’ is a trademark of Addison-Wesley Publishing Company.

This book, *TEX 急就帖*, contains both tutorial and reference information on all features of both plain and primitive **TEX**.

Copyright © 2003, 2013 Paul W. Abrahams, Kathryn A. Hargreaves, and Karl Berry.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover texts, and with no Back-Cover texts. A copy of the license is included in the chapter entitled “GNU Free Documentation License”.

Under the terms of the GFDL, anyone is allowed to modify and redistribute this material, and it is our hope that others will find it useful to do so. That includes translations, either to other natural languages, or to other computer source or output formats.

In our interpretation of the GFDL, you may also extract text from this book for use in a new document, as long as the new document is also under the GFDL, and as long as proper credit is given (as provided for in the license).

致 *Jodi.*

—P.W.A.

忆信任我的父亲.

—K.A.H.

致 *Dan.*

—K.B.

前言

Donald Knuth 开发的计算机排版系统 TeX，提供了几乎所有排版高质量数学符号和普通文本的功能。它因为它独有的易用性，超群的分词处理和美观的断行而闻名天下。因为这些非凡的功能，TeX 已经成为数学、自然科学和工程领域领先的排版系统，并且被美国数学学会定为标准。同时，它的成对软件，METAFONT，可以用来设计任意的字体，尤其是数学排版所需要的符号。TeX 和 METAFONT 在科学和工程领域有很广泛的应用，也被移植到各种不同的计算机架构上。TeX 当然不是全能的，它缺少对于图形的完整支持，同时一些功能，比如修订线，在 TeX 中实现起来也比较麻烦。但是，这些缺点和它的优点比较起来，是微不足道的。

《TeX 急就帖》是为科学工作者、数学工作者和专业排印者而写的。对于这些人而言，TeX 不是一个兴趣爱好，而是一个非常有用的工具。本书同时也面向那些对 TeX 有很强烈兴趣的计算机行业工作者。我们希望不管是新手们还是熟悉 TeX 的人，都能从本书获益。我们假定我们的读者群体已经熟悉了基本的计算机操作，并且他们希望用最快的速度得到他们想要的信息。因此，我们的目标是提供简明的信息，并且让读者能够方便地获取它们。

因此，本书给读者提供了探索和使用 TeX 所需要的明亮的探照灯，结识的手杖，以及详细的地图。他能让你通过查询和饰演，来快速掌握 TeX，但它不会牵着你的首带领你走过整个 TeX 系统的荆棘地。我们的方法就是，提供给你一本 TeX 的手册，来让你更方便地得到你所需要的信息。我们会同时详细介绍 TeX 的命令，以及命令背后的原理。因此你不会费时费力地阅读你不需要看的信息。

如果你从未用过 TeX，在开头的章节中，我们给予充分的指导，来让你尽快上手。我们假定你手头上有一个能用的 TeX 系统，并且知道如何用一个文本编辑器，对你其它的背景知识不做要求。由于本书是按照参考手册编排的，因此即使你熟悉了 TeX 以后，依然会发现本书很有

用。如果你希望看一个手把手的教程，我们建议你先阅读 *The T_EXbook* (见第 18 页部分的引用)，跳过所有的有“危险符号”的部分，然后再在开始使用 T_EX 的时候参考本书来获得更多的信息。*(The T_EXbook 中标明危险符号的部分讲述了更高阶的内容。)*

T_EX 的结构是非常简单的：一个 T_EX 输入文档，是一些包括指令的普通文本，这些命令来指导 T_EX 如何来排印你的文档。许多地方，比如数学公式，会包括大量的这些指令，而一般的解释性文本很少包括这些指令。

学习 T_EX 最花时间的地方就是学习命令和命令背后的原理。因此我们把本书大部分的篇幅都在介绍命令的定义，和解释命令的原理上。我们也提供了一些例子，来展示 T_EX 排版的输出，以及其相关的输入，解决常见问题的小提示，以及错误信息的有关信息等等。我们提供了非常详尽的页码交叉引用和完整的索引列表。

我们把命令和其解释进行了详细分类排列，因此你能够通过其功能或者字母顺序快速查找到它们。功能排列顺序，能够让你快速找到你能用什么命令来达到你的目的。而字母排列顺序，能够让你快速找到一个不理解的命令的实际功能。

我们必须提醒读者，我们并没有尝试提供完整的 T_EX 行为的定义。如果你需要了解这些信息，你应该看 T_EX 的经典著作——*The T_EXbook*。*The T_EXbook* 同时还提供了很多如何更好使用 T_EX 做出漂亮文档的提示，特别是讲精调数学公式的那部分。我们强烈推荐读者阅读它。

在 1989 年，Knuth 对 T_EX 系统进行了一次大的修订，以使得它能够支持 8 位的字符，因此使得排版除了英语以外的其他一些语言成为了可能。本书对于 T_EX 的介绍，包括了这次修订的内容 (见第 ‘newtex’ 页)。

你可能正在使用一个专门的 T_EX 封装形式，比如 L^AT_EX 或者 A_MS-T_EX (见 第 18 页)。虽然这些封装形式是完整的，你仍可能为更好地控制 T_EX 而去使用 T_EX 本身独有的一些功能，你可以使用这本书来学习你想要知道的这些功能，而把其它你不感兴趣的东西扔在一边。

在准备本书写作时，我们作者中的两位 (K.A.H. 和 K.B.) 获得波士顿麻省大学的鼎力帮助，尤其是 Rick Martin 在此期间，保证计算机的正常运行，以及 Robert A. Morris 和 Betty O’Neil 能让我们使用这些计算机设备。Paul English of Interleaf 帮助我们做出了封面设计的校样。

我们需要感谢本书的这些审稿人：马里兰大学的 Richard Furuta, Arbortext 公司的 John Gourlay, 电子设备公司的 Jill Carter Knuth 和 Richard Rubinstein。我们把他们对我们原稿的观点和批评铭记在心，本书从他们的洞见中获益良多。

我们感谢我们的编辑，Addison-Wesley 出版公司的 Peter Gordon。出版本书其实是他的主意，在本书的写作过程中，他也鼓励我们并

且提供有价值的建议。我们感谢他在 Addison-Wesley 的助理，Helen Goldstein，因为她在很多方面帮助我们。我们感谢 Addison-Wesley 的 Loren Stevens 在指导本书出版过程中的发挥的激情和技艺。如果没有我们的技术编辑 Janice Byer，很多小却恼人的错误可能还留在这本书中。我们感谢她在区分需要和不需要改动的地方的敏感和品位。最后，我们需要感谢 Prometheus 有限公司的 Jim Byrnes，他介绍我们互相认识，使我们能够有机会进行合作。

*Deerfield, Massachusetts
Manomet, Massachusetts*

P. W. A.
K. A. H., K. B.

自由版本前言：这本书本来是 Addison-Wesley 在 1990 年出版的。在 2003 年售空，Addison-Wesley 慷慨地把所有的权力归还给我们作者。没有社区，也不会有本书的存在，所以作为我们对社区的贡献，我们打算在 GNU 的自由文档许可协议下把这本书开源，更多的关于许可协议的信息，见本书的版权页。

原本本书包括了一些插图，不过在自由版本中，我们没有把插图包括进来。此外我们还改变了一些字体，只使用可自由获取的字体。我们把剪裁线和长条校样保留在页面上以方便识别。本书使用老版本的 Eplain 宏包编写，更多信息见 `eplain.tex` 文件。

除了修正报告给我们的错误，以及可能加入原书的插图以外，我们不打算修改或者增加本书的内容。

我们把本书发布在 `ftp://tug.org/tex/impatient` 上。你可以通过电子邮件 `impatient@tug.org` 来和我们取得联系。

中文翻译说明：本书的翻译由 CTeX 论坛上多人合作完成。在 2008 年，yulewang 组织 firingme、LiYanrui、Neals、sde、snoopyzhao 和 zpxing 等进行本书的翻译，直到 2009 年翻译完全停止时，完成了不到一半。

在 2014 年，由 zoho 继续整理并完成全书的翻译，此外还给文档添加了书签和插图，生成了交叉引用和中文索引。中文翻译的最新源码和错误反馈地址分别为：

<https://bitbucket.org/zohooo/impatient>
<http://bbs.ctex.org/forum.php?mod=viewthread&tid=78586>

在完善中文翻译的过程中，qingkuan 提供了帮助，特此致谢。

考虑到此书需要和 *The TeXbook* 一起阅读，书中概念的翻译尽量保持和 *The TeXbook* 的翻译一致或接近。

简要

1 \ 使用本书	■ 1
2 \ 使用 TeX	■ 8
3 \ 例子	■ 22
4 \ 概念	■ 46
5 \ 组段命令	■ 96
6 \ 组页命令	■ 136
7 \ 水平和竖直模式命令	■ 156
8 \ 数学公式命令	■ 196
9 \ 一般操作命令	■ 234
10 \ 建议和技巧	■ 284
11 \ 理解错误信息	■ 304
12 \ 实用宏集概述	■ 314
13 \ 命令速查表	■ 340
GNU 自由文档许可证	■ 372
索引	■ 385

目录

1 \ 使用本书 ■ 1

语法约定 □ 2

命令描述 □ 3

2 \ 使用 TeX ■ 8

从键盘输入变到油墨 □ 8

所需的程序和文件 · 8

运行 TeX · 9

准备输入文件 □ 10

命令和控制序列 · 10

参量 · 12

参数 · 12

空格 · 12

注释 · 13

标点 · 14

特殊字符 · 15

编组 · 15

数学公式 · 16

TeX 如何工作 □ 16

新 TeX 和老 TeX □ 17

资源 □ 18

3 \ 例子 ■ 22

输入简单文本 □ 23

缩进 □ 25

字体和特殊字符 □ 27

行间距	□ 29
间隔、标线和盒子	□ 31
杂项	□ 33
使用其他来源的字体	□ 35
标线表格	□ 37
排版数学公式	□ 39
更多数学内容	□ 41

4 \ 概念 ■ 46

5 \ 组段命令 ■ 96

字符和重音符	□ 97
欧洲语言字母和连写	· 97
专用符号	· 98
任意字符	· 99
重音符号	· 100
处理页边连写	· 101
选择字体	□ 102
特定字体	· 102
字体风格	· 103
大写和小写	□ 103
单词间距	□ 104
行的居中和平均分布	□ 108
塑段	□ 110
段落的开始、结束和缩进	· 110
形成段落	· 113
断行	□ 121
鼓励或阻碍断行	· 121
断行参数	· 123
连字	· 127
分节、列表和定理	□ 130

6 \ 组页命令 ■ 136

行间距和段间距	□ 136
分页	□ 139
鼓励或阻碍分页	· 139
分页参数	· 141

页面布局	□ 143
页面描述参数	· 143
页码	· 145
页眉和页脚	· 146
标记	· 147
插入项	□ 148
脚注	· 148
一般插入项	· 149
修改输出例行程序	□ 151
分割竖直列表	□ 152

7 \ 水平和竖直模式命令 ■ 156

产生间隔	□ 156
固定宽度水平间隔	· 156
固定长度竖直间隔	· 158
可变尺寸间隔	· 159
操作盒子	□ 164
构造 hbox 和 vbox	· 164
设置和获取盒子的内容	· 169
移动盒子	· 171
盒子寄存器的尺寸	· 172
支架、幻影和空盒子	· 172
有关畸形盒子的参数	· 175
从列表中获取最后的项目	□ 177
标线与指引线	□ 178
阵列	□ 182
制表阵列	· 182
常规阵列	· 184

8 \ 数学公式命令 ■ 196

简单公式排版	□ 196
希腊字母	· 196
各种普通数学符号	· 197
二元运算符	· 198
关系符号	· 200
左右定界符	· 201
箭头	· 202
已命名的数学函数	· 203
巨算符	· 204
标点	· 206

上标和下标	□ 206
选用样式	· 207
复合符号	□ 209
数学重音	· 209
分式和其他堆叠运算	· 210
圆点	· 213
定界符	· 214
矩阵	· 216
根号与根数	· 217
方程编号	□ 217
多行陈列公式	□ 218
数学公式字体	□ 220
构造数学符号	□ 222
增大定界符	· 222
大符号的一部分	· 223
对齐部分公式	□ 224
对齐数学重音	· 224
竖直对齐素材	· 225
生成间隔	□ 225
固定宽度数学间隔	· 225
可变宽度数学间隔	· 227
陈列公式的间隔参数	· 228
其他的数学间隔参数	· 229
分类数学结构	□ 230
特殊处理数学公式	□ 230

9 \ 一般操作命令 ■ 234

命名及修改字体	□ 234
把信息转为记号	□ 238
数值	· 238
环境信息	· 239
变量的值	· 240
编组	□ 241
宏	□ 245
定义宏	· 245
其它定义方法	· 247
控制展开	· 249
条件测试	· 250
重复操作	· 256
什么也不做	· 257

寄存器	□ 258
使用寄存器	· 258
命名和预留寄存器等	· 260
寄存器中的算术运算	· 261
结束任务	□ 263
输入和输出	□ 263
操作输入文件	· 263
操作输出文件	· 265
解释输入字符	· 267
控制与 <i>TEX</i> 的交互	□ 269
帮助调试	□ 270
显示内部数据	· 270
指定追踪的内容	· 273
传送信息	· 278
初始化 <i>TEX</i>	□ 280

10 \ 建议和技巧 ■ 284

纠正不良分页	□ 284
保留页尾	□ 286
保留页首空白	□ 286
纠正不良断行	□ 287
纠正溢出或未满盒子	□ 287
恢复丢失的单词间距	□ 288
避免多余的单词间空白	□ 289
避免陈列公式周围的额外空白	□ 290
避免段后的额外空白	□ 290
改变段落形状	□ 290
把段落放入盒子	□ 291
画线	□ 291
创建多行的页眉页脚	□ 292
找出匹配错误的花括号	□ 293
设置尺寸	□ 294
创建混合字体	□ 294
原文呈现	□ 295

- 使用外部宏 □ 297
- 改变类别码 □ 299
- 使宏文档易读 □ 299

11 \ 理解错误信息 ■ 304

- 第一个例子 □ 304
- 第二个例子 □ 306
- 第三个例子 □ 307
- 第四个例子 □ 307
- 第五个例子 □ 308

12 \ 实用宏集概述 ■ 314

- 预备定义 □ 314
- 陈列公式 □ 318
- 日期时间 □ 321
- 列表 □ 322
- 原文呈现 □ 324
- 目录 □ 324
- 交叉引用 □ 326
- 环境 □ 329
- 对齐 □ 331
- 表格 □ 332
- 脚注 □ 333
- 双栏 □ 334
- 收尾 □ 336

13 \ 命令速查表 ■ 340

GNU 自由文档许可证	■ 372
<i>PREAMBLE</i>	□ 372
<i>APPLICABILITY AND DEFINITIONS</i>	□ 373
<i>VERBATIM COPYING</i>	□ 375
<i>COPYING IN QUANTITY</i>	□ 375
<i>MODIFICATIONS</i>	□ 376
<i>COMBINING DOCUMENTS</i>	□ 378
<i>COLLECTIONS OF DOCUMENTS</i>	□ 379
<i>AGGREGATION WITH INDEPENDENT WORKS</i>	□ 379
<i>TRANSLATION</i>	□ 380
<i>TERMINATION</i>	□ 380
<i>FUTURE REVISIONS OF THIS LICENSE</i>	□ 381
索引	■ 385

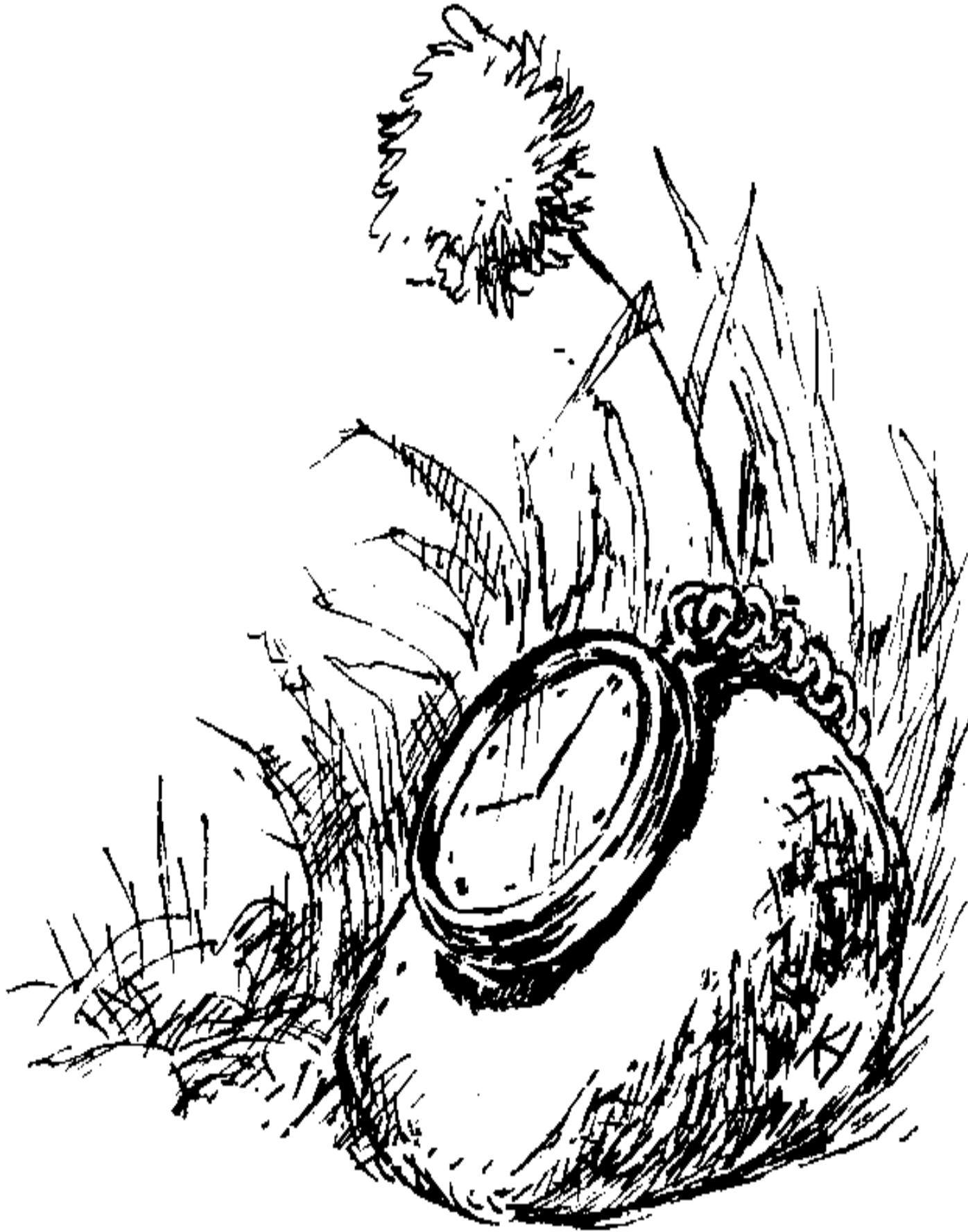
阅读指南

如果你刚学习 TeX:

- 首先阅读第 1–2 章。
- 查看第 3 章中与你要做的事情相似的例子。查找第 13 章，“命令速查表”中相关的命令。利用页码索引找到这些命令及类似命令的更完整的描述。
- 查找第 4 章，“概念”中不熟悉的词语，使用书背内页的概念列表快速找到这些词语的解释。
- 反复尝试和探索。

如果你已经熟悉 TeX，或者你正在编辑或修改其他人创建的 TeX 文档：

- 要得到某个命令用法的快速提示，看一下第 13 章，“命令速查表”。该表按照字母顺序排列，并给出了到命令的更完整介绍的页码索引。
- 利用命令描述中的功能组，找到与你知道的某个命令相关的其他命令，或者找到某个有特殊用处的命令。
- 利用第 4 章，“概念”，找到任何你不理解的，或想理解得更准确的，或已经遗忘的，概念的解释。用书背内页上的概念列表可以快速找到一个概念。



1 | 使用本书

本书是一本介绍 TeX 的自学手册。在本章中，我们将指导你如何更好地使用本书。

我们建议你先阅读或浏览 1 到 3 章，这些地方会告诉你如何上手 TeX。如果你已经使用过 TeX，这些地方仍会值得一看，至少你可以知道这些章节介绍了些什么内容。其余部分，主要是 4-10 章，可以有选择地翻阅。当然，如果你是一个很喜爱阅读参考手册的人，你会发现从头到尾地把这本书阅读一遍是完全可行的，当然你会为此在开头学习时绕些弯路。

在第 2 章，“使用 TeX”，我们解释如何把一个 TeX 输入文件转为一个 TeX 文档。我们还讲述了准备输入文件的一些习惯，简要地解释了 TeX 是如何工作的，告诉你可以获取的其它资源信息。阅读了这个章节，会有助于你理解下一个章节的例子。

第 3 章，“例子”，包括了一系列的例子来描述 TeX 的功用。每个例子包括了一页的输出内容，也包括了输出该页内容所输入的源代码。这些例子会指导你且帮助你找到你所需要的更详细的资料。通过观察输入使用了什么命令，你会知道在哪里能够找到输出效果所使用方法的更详细的信息。这些例子也可以作为简单的文档的模版，当然我们必需提醒你，因为我们尝试把很多的 TeX 命令放在这几页纸中，这些例子不可能描述好的或者完整的文档设计。

当你看这些命令的解释时，你会遇到很多不熟悉的技术术语。在第 4 章，“概念”中，我们定义并解释了这些术语。我们同时还讨论了本书其它地方没有涉及到的话题。本书书背内页列出了所有的概念名称以及它们所出现的位置。我们建议你把书背内页复印下来，放在旁边，这样你就能在遇到不熟悉的概念时马上查找到它们。

\TeX 的命令是它的主要词汇，本书最主要的部分就是用来解释这些命令。从第 5 章到第 9 章，我们解释了这些命令。你可以在第 3 页找到命令描述的描述惯例。这些命令根据功能分门别类，而不是按字典顺序，所以如果你知道你想做什么，但不知那该用什么命令，你可以使用目录来引导你找到正确的命令组。我们给我们认为常用且容易理解的命令标上了一个手形符号 (☞).

第 13 章为“命令速查表”，这个专门的索引与第 5–9 章的更完整描述相辅相成。在这一章中按字母顺序列出了各个 \TeX 命令及其简要解释，并给出该命令的详细描述所在的页码。此速查表可以让你快速知道某个命令的作用。

\TeX 是个复杂的程序，偶尔地会以诡异的方式工作。在第 10 章，“建议和技巧”中，对一系列你时不时会遇到的特定的问题，我们提供了一些建议。而若你被 \TeX 的错误信息难住了，第 11 章，“理解错误信息”将给你提供帮助。

页边的灰色标签可以帮你快速定位到本书各个部分。它们将本书划分为下面几个主要部分：

- 1) 总体的说明和例子
- 2) 概念
- 3) 命令描述（五个短标签）
- 4) 建议，错误信息和 `eplain.tex` 宏集
- 5) 命令速查表
- 6) 索引

在很多地方我们都给出了到 *The \TeXbook* 的页码索引（要引用该书可以见第 18 页）。这些索引适用于 *The \TeXbook* 第十七次印刷版。对于其它版本，有的索引可能相差一两页。

语法约定

在任何涉及到为计算机准备输入的书籍中，都得明确区分哪些字符是需要逐字输入的，而哪些字符属于解释性文本。我们用计算机现代打字机字体显示文字输入，比如 `literal input`，以及 \TeX 命令名。在可能导致混淆的地方，我们将 \TeX 命令用单引号围起来，比如 ‘`like this`’。然而，在表示单个字符时，我们偶尔也会用括号将它围起来，比如 (‘) (其原因你容易明白)。

通常我们按照你输入空格的方式显示空格，以免让你看花眼。但为了强调某些空格，我们用‘`\`’字符表示它。这个字符很自然地被称为可见空格。

命令描述

第 5–9 章描述了几乎每个 TeX 命令，包括原始命令和 plain TeX 命令。原始命令是那些内建于 TeX 程序中的命令，而 plain TeX 命令是那些在标准辅助定义文件中定义的命令（见第 87 页）。我们仅仅忽略那些只在 plain TeX (*The TeXbook* 附录 B) 定义中用到的局部命令。这些命令分为如下几部分：

- 第 5 章，“组段命令”，介绍字符、单词、文本行和整个段落。
- 第 6 章，“组页命令”，介绍页面及其组成部分，以及输出例行程序。
- 第 7 章，“水平和竖直模式命令”，介绍水平模式（段落和水平盒子）和竖直模式（页面和竖直盒子）中对应或等同的命令。这些命令提供了盒子、间隔、标线、指引线和对齐。
- 第 8 章，“数学公式命令”，介绍构造数学公式的命令。
- 第 9 章，“一般操作命令”，介绍 TeX 的编程功能以及不适合放在其他部分的命令。

你应该将这些分类视为提示性的而不是严格的，因为把命令都归入这些（或者其他）类别并不完全合适。

各章的命令描述按照其功能组织。当几个命令密切相关时，它们合起来介绍；否则，每个命令分别解释。命令的描述中包含一个或多个例子，可能还有例子的输出（对有些例子这不可行）。若某节内容涉及到功能上相关的其他命令时，务必看看该节末尾的“参见”项，那里给出了其它地方介绍到的相关命令所在的页码。

有些命令与特定的概念密切相关。例如，`\halign` 和 `\valign` 命令与“对齐”的概念相关，`\def` 命令与“宏”相关，而 `\hbox` 和 `\vbox` 命令与“盒子”相关。对此种情形，在命令这里我们通常只给出粗略介绍，而将对基本原理的解释放在相关概念中。

命令相关的例子在排版时设定了段落缩进 `\parindent` 为零，因而其中段落一般是不缩进的；这使得例子更加容易阅读。对有些例子段落缩进是必不可少的，我们显式设置缩进为非零值。

在一个或一组命令前面的手形符号，表示我们认为这个或这组命令是常用且容易理解的。

某些命令需要特定类型的参量（第 12 页）。命令的参量提供 T_EX 执行该命令时所需的额外信息。每个参量都放在尖括号中，并用意大利体标明它的类型：

$\langle argument \rangle$	单个记号或者放在花括号中的一些文本
$\langle charcode \rangle$	字符码，即在 0 到 255 之间的整数
$\langle dimen \rangle$	尺寸，即长度
$\langle glue \rangle$	粘连（有可选的伸展值和收缩值）
$\langle number \rangle$	可带符号的整型值（整数）
$\langle register \rangle$	寄存器编号，在 0 到 255 之间

所有这些术语会在第 4 章中进行详述。除了这些以外，我们会使用一些不言自明的或者在后文进行讲述的术语，比如 $\langle token\ list \rangle$ 。有些命令有自己独特的格式，比如需要括弧或者特定的字词。它们会使用和命令标题相同的黑体标出。

有些命令是参数（第 12 页）或表格项。这会在命令的列表中指示出来。你可以将这个参数作为一个命令的参量，也可以给它赋值。表格项也是如此。我们使用“参数”这个术语来指代这些条目，比如本是寄存器但可当参数使用的 `\pageno`。



不

2 | 使用 TeX

从键盘输入变到油墨

■ 所需的程序和文件

为了制作一个 TeX 文档，你必需运行 TeX 及其相关软件。你同时也需要一些 TeX 及其相关软件所用到的辅助文件。在这本书中，我们仅仅谈论 TeX 和一些通用的软件及辅助文件，不过我们不会介绍其它的，因为它们仅仅和你自己的 TeX 环境相关。为你提供 TeX 的人可以为你提供我们所说的本地信息。这些本地信息可以告诉你如何去启动 TeX，如何去使用相关的程序，以及如何去访问这些所用到的辅助文件。

你可以使用一个文本编辑器来编写一个可以输入到 TeX 的普通文本文件。一个 TeX 输入文件和一个文字处理软件的输入文件不同，它通常并不存在任何不可见的控制字符。任何 TeX 读到的字符对你来说，都是你可以看到的。

你的输入文件，可以仅仅是一个引用其它输入文件的框架。TeX 用户往往把大的文档，比如说这本书，组织成这种形式。你可以使用 `\input` 命令（第 263 页）来把一个输入文件嵌入到另一个中。特别地，你可以使用 `\input` 来调入包含宏定义的文件，宏定义是一些辅助的定义，它可以来增强 TeX 的功能。如果你的 TeX 系统中包含任何的宏文件，和 TeX 相关的本地信息会告诉你如何得到这些宏文件，以及它们的功能。TeX 的标准安装形式，也就是在这本书中所描述的，包括了一个被称为 plain TeX（第 87 页）的宏集。

当 TeX 处理你的文档时，它会产生一种叫做 .dvi 文件的文件。其中，“dvi”的全称为“device independent (设备无关)”。之所以选用这个缩写，是因为 .dvi 文件中所包含的信息和你的打印或显示设备无关。

当你需要打印或使用浏览器查看你的文档时，你需要使用相应的设备驱动来处理 .dvi 文件。(浏览器是一个程序，它能把和打印出来的结果差不多的排版内容显示在屏幕上。) 不同的输出设备往往需要不同的设备驱动。在运行设备驱动以后，你还需要把设备驱动输出的内容传输到打印机或者其它输出设备上。和 TeX 相关的本地信息会告诉你如何得到正确的设备驱动和如何使用它。

因为 TeX 内部并没有任何关于一个特定字体的任何信息，它使用字体文件 来得到你文档中所使用的字体的信息。字体文件也是你本地 TeX 环境的一个组成部分。一个字体一般对应两个文件：一个文件(字体度量文件)包括了字体中每个字符的大小信息，另一个文件(字体轮廓文件)则描述了字体中字符的形状。一个字体的缩放版本使用同样的字体度量文件，不过并不使用相同的字体轮廓文件。字体度量文件往往用来指称 .tfm 文件，而字体轮廓文件则往往用来指称 .pk 文件，.pxl 文件和 .gf 文件等不同的种类。这些名称和 TeX 以及相关软件所使用的文件的文件名所对应。比如，cmr10.tfm 为 cmr10 字体的字体度量文件(10-点的计算机现代字体)。

TeX 本身仅使用字体度量文件，因为它只关心这个字体的字符占用了多大的空间，而并不关心这个字体的外形是什么样子的。设备驱动则往往需要使用字体轮廓文件，因为它需要负责把每个排出的字符变成印出的图形。某些设备驱动也需要使用字体度量文件。一些设备驱动可以使用打印机中包含的字体，所以可以不需要这些字体的字体轮廓文件。

■ 运行 TeX

你可以通过输入类似 ‘run tex’ 或 ‘tex’ (查看你的本地信息) 来运行 TeX 来处理一个输入文件 screed.tex. TeX 会作出如下显示

```
This is TeX, Version 3.0 (preloaded format=plain 90.4.23)
**
```

在这里，“preloaded format” (已预先载入的格式) 指的是 TeX 提供的 plain TeX 宏集的一个已预先产生的形式。现在你可以输入 ‘screed’ 来让 TeX 来处理你的文件。当处理结束时，你可以在终端或者打印机印出的记录上 (如果你不使用终端) 得到类似下面的输出：

```
(screed.tex [1] [2] [3] )
```

```
Output written on screed.dvi (3 pages, 400 bytes).
Transcript written on screed.log.
```

这个输出大部分是不言自明的。在括号中的数字是 TeX 在把每页内容输入 .dvi 文件时所显示的当前页码。当你提供的名件名不含扩展名时, TeX 往往会把你的文件扩展名当作 ‘.tex’。在某些 TeX 环境中, 你可能可以直接执行类似下面的语句:

```
tex screed
```

来直接处理文档。

TeX 不仅可以读取文件输入, 也可以直接读取你在终端中的输入: 只需将 ‘**’ 提示后的输入从 ‘screed’ 改为 ‘\relax’。TeX 将在你输入的各行前面给出 ‘*’ 提示, 并且逐行解释你的输入。输入类似 ‘\bye’ 这样的命令可以结束输入。在试验 TeX 时直接输入有时更方便。

当你的输入文件包含其他嵌入的输入文件时, 显示的信息中表明了 TeX 何时开始和结束处理每个嵌入文件。当 TeX 开始处理一个文件时它显示一个左圆括号和该文件名, 而结束时显示一个对应的右圆括号。如果在所显示的输出中有任何错误信息, 你就能从最接近的未配对左圆括号中确定是哪个文件引起的。

要得到如何运行 TeX 的更完整解释, 可以见 *The TeXbook* 第 6 章以及你的本地信息。

准备输入文件

在这一节中, 我们解释准备 TeX 输入文件时必须遵循的一些约定。其中有些信息同样出现在本书第 3 章的例子中。

■ 命令和控制序列

TeX 的输入由一系列指引 TeX 如何排版文档的命令组成。大部分字符如同一种特别简单的命令: “排版我”。比如, 字母 ‘a’ 是作为命令将排版出 ‘a’。但有另一种命令——控制序列 (control sequence)——将给 TeX 提供更细致的指令。控制序列通常以反斜杠 (\) 开头, 但必要时你也可以更改此约定。例如, 下列输入:

```
She plunged a dagger (\dag) into the villain's heart.
```

包含一个控制序列 `\dag`；它排版出下列的输出：

She plunged a dagger (†) into the villain's heart.

在这个例子中，除 `\dag` 和空格之外的每个字符都如同一个“排版我”命令。在第 12 页中我们将更详细地解释空格的作用。

控制序列可分为控制词 (control word) 和控制符 (control symbol) 这两种类型：

- 控制词由反斜杠后跟一个或多个字母组成，例如 ‘`\dag`’。其后用一个非字母字符结束该控制词。
- 控制符由反斜杠后跟一个单个的非字母字符组成，例如 ‘`\$`’。该非字母字符可以是空格符，甚或是行结束符（这也是合乎规则的字符）。

控制词后的任何空格或行结束符将被忽略（但控制符后不会这样）。如果你真要在控制词后得到一个空格，可以在该控制词后键入一个控制空格 (`\u0020`) 或者 ‘`\{`’。因而这样输入：

```
The wonders of \TeX\u0020shall never cease!
```

或者这样输入：

```
The wonders of \TeX{} shall never cease!
```

将得到同样的结果：

The wonders of `\TeX` shall never cease!

作为对比，下面的结果：

The wonders of `\TeX`shall never cease!

是你去掉上面输入中的 ‘`\u0020`’ 或者 ‘`\{`’ 后得到的。

不要将控制词和其后的文本连在一起---这样 `\TeX` 无法知道控制词在哪里结束。比如，控制词 `\c` 给其后的字符加上软音符。要得到法文单词 *garçon*，你必须输入 ‘`gar\c{c}on`’，而不是 ‘`gar\ccon`’；若你输入后者，`\TeX` 将抱怨有未定义的控制序列 `\ccon`。

另一方面，控制符后面任何一个字符都不会被忽略。因此，要得到 ‘`\$13.56`’ 你必须键入 ‘`\$13.56`’，而不是 ‘`\$13.56`’；后者将生成 ‘`\$ 13.56`’。然而，那些作为控制符的重音命令，是以忽略其后空格的方式定义的。例如，要得到法文单词 *déshabiller*，键入 ‘`d\'eshabiller`’，或者 ‘`d\'eshabiller`’ 都可以。

每个控制序列同时也是一个命令，但反之未必。例如字母 ‘`N`’ 是一个命令，但它不是一个控制序列。在本书里面，当两者都适用时我们用“命令”而不用“控制序列”。在强调 `\TeX` 的语法层面时，我们用“控制序列”，因为此时用“命令”一般并不合适。

■ 参量

有些命令的运行依赖于其后的一个或多个参量 (argument)。例如，`\vskip` 命令告诉 TeX 在页面中往上 (或往下) 跳过一定间距，它需要一个参量来指定所跳过间距的大小。要往下跳过两英寸，你需要键入 ‘`\vskip 2in`’，其中的 `2in` 就是 `\vskip` 的参量。

不同的命令要求不同类型的参量。很多命令的参量要求是长度值，比如上面例子中的 `2in`。有些命令，特别是用宏定义的命令，其参量可以是单个字符或者放在花括号中的文本。而对其他有些命令，其参量只能放在花括号中，而不能是单个字符。在本书对各个命令的描述中，若一个命令需要参量，我们都说明了该命令所需参量的类型。在有些情形中，命令的参量所需的花括号定义了一个编组 (见第 16 页)。

■ 参数

有些命令实际上是参数 (parameter，见第 86 页)。你可以按下面两种方式使用这种参数：

- 1) 将该参数的值用作其他命令的参量。例如 `\vskip\parskip` 这个命令生成大小等于粘连参数 `\parskip` (段落间距) 的竖直间距。
- 2) 通过赋值修改该参数的值。例如 `\hbadness=200` 这个赋值使得 `\hbadness` 参数的值变为 200.

有些命令，例如 `\pageno`，虽然是寄存器但和参数的用法一样。因此我们同样使用“参数”这个术语来指代这些命令。

有些命令是表名。这些命令的用法和参数类似，只是它们需要额外的参量指定表格项。例如，`\catcode` 是类别码表格的名称 (第 54 页)。因此，`\catcode`~=13` 这个命令设置字符 ‘~’ 的类别码为 13。

■ 空格

在输入文件中你可以任意加上额外的空格。在几乎所有情形中 TeX 都将同一行的连续多个空格看成一个空格。例如，在句号后加上一个或两个空格是没有差别的。不论按照哪种写法，TeX 都会执行其句子结束操作，并在句号后加上适当 (在多数情形下) 大小的间隔。TeX 同样将输入中的行结束符视为一个空格。因此在输入时你可以在方便的

地方换行—— \TeX 按照此方式将各输入行转换为段落，而不在乎输入中的换行位置。

输入中的一个空行结束当前段落。多个空行也等同于一个空行。

\TeX 忽略数学公式中的空格（见后面）。因此，你可以加上或者省去公式中的空格—— \TeX 不在乎这个。但即使在数学公式中，你也不要将控制词和其后的字母连在一起。

在定义自己的宏时，你必须小心定义中的行结束符位置。因为一不小心该宏就在你希望的空格之外生成了不需要的空格。因为这是个有些技术性的问题，我们在其他地方也讨论到此问题；见第 289 页。

输入文件中两单词间的一个空格或其等价字符，并不简单地变成输出中的一个空格。由于输入行和输出行通常不会正好对应，输入中的这些空格有的变成输出中的行结束符，另外有些空格则变成宽度可变的间隔即“粘连”（第 66 页）；粘连有其自然宽度（它希望的宽度），但也可以伸展或者收缩。 \TeX 排版要求右页边对齐（通常如此）的段落时，将调整各行的粘连的宽度以让各行正好在页边结束。（段落的最后一行是个例外，因为一般不要求它在右页边结束。）

要阻止输入中的空格在输出中变成行结束符，你可以用带子（`~`）。比如，你不会希望 \TeX 在 ‘Fig. 8’ 的 ‘Fig.’ 和 ‘8’ 之间断行。输入 ‘Fig.~8’ 就可以禁止这样断行。

■ 注释

在 \TeX 输入中可以包含注释。 \TeX 跳过它所碰到的注释，因此注释中的内容不会以任何方式影响到排版的文档。注释用于在输入文件中提供更多的信息。例如：

```
% ===== Start of Section 'Hedgehog' =====
```

注释以百分号（%）开始，一直延续到该输入行结束。 \TeX 既忽略注释也忽略该行的结束符，因此注释有另外的重要用法：将两个输入行连接起来，使得 \TeX 看不到两行间的行结束符，从而不会在输出中生成空格或换行。例如，对如下的输入：

```
A fool with a spreadsheet%
sheet is still a fool.
```

你得到的输出为：

A fool with a spreadsheet is still a fool.

■ 标点

若‘.’，‘?’或‘!’之后有个输入空格，TeX 认为它是句末标点符号，并在其后添加额外间隔。但若该标点符号之前为大写字母，TeX 就不会添加额外间隔，因为它认为大写字母是人名的首字母。如若不然你可以强制生成额外间隔，例如：

```
A computer from IBM\null?
```

命令 \null 不生成任何输出，但它阻止 TeX 将大写字母‘M’和问号结合在一起。反过来，如果某标点符号不为句末标点，你也能够取消该标点之后的额外间隔；这可以通过在其后键入控制空格达到，例如：

```
Proc.\_Royal Acad.\_of Twits
```

得到的正确结果为：

```
Proc. Royal Acad. of Twits
```

而去掉控制空格后的结果是：

```
Proc. Royal Acad. of Twits
```

有些人更喜欢不在句末标点后留下更多间隔。利用 \frenchspacing 命令（第 106 页）你就可达成此效果。在参考文献中通常建议使用这个 \frenchspacing 命令。

对于单引号，用键盘上的左右单引号（‘ 和 ’）就可以得到。对于左双引号或右双引号，需要用两个左单引号（‘‘）或右单引号（’’）得到，不能直接用键盘上的双引号（"）。键盘上的双引号在多数字体中将得到一个右双引号，但两个右单引号是 TeX 中首选的写法。例如：

```
There is no ‘q’ in this sentence.  
“Talk, child,” said the Unicorn.  
She said, “\thinspace‘Enough!’, he said.”
```

这三行输入得到的结果为：

```
There is no ‘q’ in this sentence.  
“Talk, child,” said the Unicorn.  
She said, “‘Enough!’, he said.”
```

第三个输入行中的 \thinspace 避免单引号和双引号太过靠近。否则，你将看到三个并排的间隔相近的引号。

在 TeX 中有三种横线号 (dash) :

- 短横线即连字号 (hyphen) 如这样 (-)。键入 ‘-’ 可以得到。
- 中横线即连接号 (en-dash) 如这样 (–)。键入 ‘--’ 可以得到。
- 长横线即破折号 (em-dash) 如这样 (—)。键入 ‘---’ 可以得到。

一般地，你将用连字号表示像 “will-o'-the-wisp” (鬼火) 这样的复合词，用连接号表示像 “第 81–87 页” 这样的页码起止，而用破折号表示语气转折— 像这样。

■ 特殊字符

在 TeX 中某些字符有特殊的含义，因而不能在普通文本中使用。这些特殊字符为：

\$ # & % _ ^ ~ { } \

要在排版的文档中生成它们，你需要使用间接的方法。对前面五个字符，你需要键入：

\\$ \# \& \% _

而对另外五个字符，你需要键入更多：

\^{} \~{} \\$\{ \\$\} \\$\\\$

■ 编组

包含在配对的左右花括号 ({ 和 }) 中的内容组成一个编组。编组内部的命令，其作用范围被限制在该编组内部。例如，\bf 命令让 TeX 将某些内容用粗体显示。如果你将 \bf 放在你的输入中，而没有用任何方式取消它，\bf 之后的所有内容都将以粗体显示。如果将 \bf 包含在一个编组中，就可以将它的作用限制在此编组中。比如，如果你键入：

```
We have {\bf a few boldface words} in this sentence.
```

你将会得到如下的结果：

```
We have a few boldface words in this sentence.
```

利用编组，你可以限制 TeX 某个参数的取值的作用范围。这些参数的取值影响 TeX 对文档的排版。例如，\parindent 参数的值指定了段首缩进量，赋值 \parindent = 15pt 将设定该缩进量为 15 点。若将该

赋值放在包含几个段落的编组开始处，你就可以仅仅改变这几个段落的段首缩进。若不将该赋值包含在编组中，对缩进量的修改将作用到文档的剩余部分（或者作用到下个对 `\parindent` 的赋值为止，若还有下个赋值的话）。

并非所有配对花括号都表示一个编组。特别地，若参量不要求有花括号，则与该参量一起的花括号并不表示一个编组——它们仅用于确定该参量的界限。对于那些参量中确实需要花括号的命令，有些命令将花括号作为编组的定义，而其他命令用各自的特殊方式解释该参量。¹

■ 数学公式

数学公式可以出现在正文中（文内公式），或者出现在留出额外上下间距的单独一行中（陈列公式）。文内公式两边用单个美元符 (\$) 括起来，而陈列公式两边用双个美元符 (\$\$) 括起来。例如：

```
If $a < b$, then the relation $$e^a < e^b$$ holds.
```

此输入得到的结果是：

If $a < b$, then the relation

$$e^a < e^b$$

holds.

第 8 章描述了数学公式中常用的命令。

TeX 如何工作

为了更有效地使用 TeX，对 TeX 着手将输入转换为输出的活动有所了解是有益的。你可以将 TeX 想象为一种有“眼睛”、“嘴巴”、“食道”、“胃”以及“肠道”的生物体。该生物体的每个器官都以某种方式转换它的输入，并将转换的结果送到下个器官中。

眼睛将输入文件转换为一系列字符。嘴巴将这串字符转换为一系列记号，其中每个记号是单个字符或者一个控制序列。食道将该串记号展开为一系列原始命令记号。胃执行原始命令指定的操作，生成一些页面。

¹ 更准确地说，对于原始命令，花括号要么定义了一个编组，要么不会在 TeX 的胃中被处理。对于 `\halign` 和 `\valign` 命令，编组只有平凡的效果；这是由于编组中的一切东西要么不会到达胃里（因为它在模板中），要么包含在更深一层的编组中。

最后，肠道将每个页面转换为 .dvi 文件所需的形式并发送给它。这些活动在第 4 章的“TeX 剖析”（第 49 页）中有更详细的描述。

实际的排版活动出现在胃中。这些命令指导 TeX 用这样或那样的字体排版这个或那个字符，插入单词间空隔，结束一个段落，等等。从单个排版字符及其他简单排印元素开始，TeX 将盒子一层层放入其他盒子中（见“盒子”，第 53 页），这一整套盒子构造了一个页面。每个排版字符占用一个盒子，整个页面也一样。盒子中不仅包含更小的盒子，也包含粘连（第 66 页）和一些其他东西。粘连生成小盒子之间的间距。粘连的一个重要属性在于它可以伸展和收缩；因此通过伸缩盒子中的粘连，TeX 可以让盒子变大或变小。

粗略说来，文本行是一个包含一系列字符盒子的盒子，而页面是一个包含一系列行盒子的盒子。一行的各单词之间以及页面的各行之间都有粘连。TeX 伸缩各行中的粘连以让右页边对齐，伸缩各页中的粘连以让各页的下边距相等。其他类型的排印元素也可以出现在行内或页内，但我们不在这里讨论。

作为组装页面过程的一部分，TeX 需要将段落分为多行，将多行组成页面。事实上，TeX 的胃刚开始将段落看成一个很长的行。它插入断行点以将段落分为长度合适的一些行，通过复杂的分析以找到最佳的断行点的集合（见“断行点”，第 74 页）。TeX 的胃也执行类似但简单些的分析以将一些行组成页面。基本上，胃累加各行直到页面上放不下更多的行。然后它选择一个分页位置，将该分页位置之前的各行放在当前页面，将之后的各行留给下个页面（见“分页点”，第 84 页）。

在 TeX 组装一个列表项（盒子，粘连等）中的某个东西时，它总处于六种模式（第 81 页）的其中一种模式中。它所组装的东西的类型定义了它所在的模式。首先有两种普通模式：普通水平模式用于组装段落（在它们分为多行之前），普通竖直模式用于组装页面。其次有两种限制模式：受限水平模式用于水平地添加元素形成水平盒子，内部竖直模式用于竖直地添加元素以形成竖直盒子（非页面盒子）。最后还有两种数学模式：文内数学模式用于组装段落内部的数学公式，陈列数学模式用于组装单独一行显示的数学公式（见“数学公式”，第 16 页）。

新 TeX 和老 TeX

在 1989 年，Knuth 对 TeX 做了一次大幅修改，使得它能够处理非英语排版所需的字符集。这次修改还包括了一些额外的不干扰其它东西的小功能，这本书介绍“新 TeX”。如果你仍使用旧版本的 TeX（2.991

版或以前的版本), 你可能会想知道哪些新 TeX 的功能你是没法使用的。以下这些功能无法在旧版中使用:

- `\badness` (第 176 页)
- `\emergencystretch` (第 124 页)
- `\errorcontextlines` (第 280 页)
- `\holdinginserts` (第 152 页)
- `\language`、`\setlanguage` 和 `\newlanguage` (第 129 和 260 页)
- `\lefthyphenmin` 和 `\righthyphenmin` (第 129 页)
- `\noboundary` (第 101 页)
- `\topglue` (第 160 页)
- 十六进制的数字表达方式 `^~xy` (第 57 页)

我们建议你尽可能地使用新版本的 TeX。

资源

有很多资源可以帮助你使用 TeX, 其中 *The TeXbook* 是最可靠的 TeX 信息来源。

Knuth, Donald E., *The TeXbook*. Reading, Mass.: Addison-Wesley, 1984.

请务必使用第十七次 (1990 年 1 月) 及以后的印刷版本。早先的印刷版本不包括新 TeX 的许多功能。

L^AT_EX 是一套为了简化 TeX 的使用而设计的命令集, 它在这本书里面有介绍²:

Lamport, Leslie, *The L^AT_EX Document Preparation System*. Reading, Mass.: Addison-Wesley, 1986.

AMS-Tex 是一套美国数学学会定义的提交电子数学手稿的命令标准, 它在这里有介绍:

Spivak, Michael D., *The Joy of TeX*. Providence, R.I.: American Mathematical Society, 1986.

你可以加入 TeX 用户组织 (TUG), 这个组织出版一本名为 *TUGBoad* 的通讯。TUG 是一个很好的信息来源渠道, 它同时也提供包括 *AMS-Tex* 在内的很多 TeX 宏包资源。它的地址是:

TeX 用户组织

² 译注: 这本书中介绍的 L^AT_EX 已经过时。

c/o American Mathematical Society
P.O. Box 9506
Providence, RI 02940
U.S.A.

最后，你可以获取在第 12 章介绍的用来排版本书的宏包 `eplain.tex`。
它可以通过使用 `ftp` 匿名登录获取：

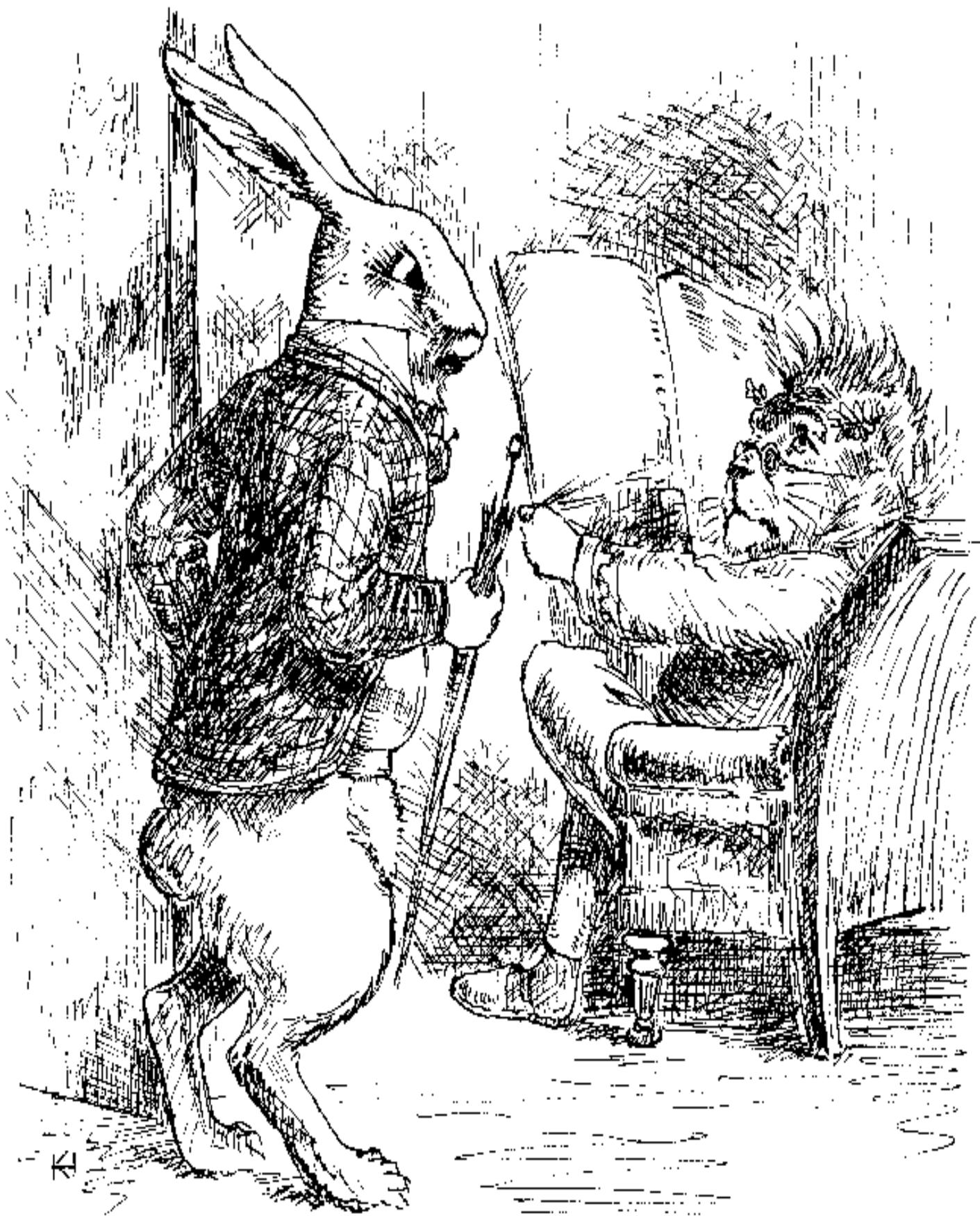
```
labrea.stanford.edu [36.8.0.47]  
ics.uci.edu [128.195.1.1]  
june.cs.washington.edu [128.95.1.4]
```

电子版本还包括了为 BIB_T_EX 程序处理输入并排印其输出的宏——
BIB_T_EX 程序是斯坦弗大学的 Oren Patashnik 编写的。如果你在这个宏
包中发现了错误，或者想改进它，你可以给 Karl 发电子邮件。Karl 的电
子邮箱地址是 `karl@cs.umb.edu`。

这个宏包还提供了 $5\frac{1}{4}$ " 或 $3\frac{1}{2}$ " 软盘，你可以向以下地址邮寄
10.00 美元购买：

Paul Abrahams
214 River Road
Deerfield, MA 01342
Email: `Abrahams%Wayne-MTS@um.cc.umich.edu`

这个地址在 1990 年 6 月是正确的；请注意在这之后它可能发生改变，尤
其是电子邮件地址。



3 | 例子

这一章中包括了一组例子，来帮助你熟悉 TeX，同时这些例子还展示了如何使用 TeX 来完成各种排版工作。每个例子都有一个放在左页的 TeX 排版的结果和放在右页的相对应的 TeX 输入文本。你可以把这些例子作为模仿的样式，也可以用来找到你想要的效果的实现命令。不过要注意的是，这些例子仅能展示 TeX 900 条左右的命令的一小部分。

这里的某些例子是其义自现的——也就是说，它们在介绍每个所排印出来的功能。这个介绍很粗略，因为没有足够的篇幅来讲术所有你想得到的信息。命令速查表（第 13 章）和索引可以帮你来找到例子中的每个 TeX 功能。

因为我们在设计这些例子时，把很多的东西放在一起描述，因此，这些例子展示了很多的排版效果。这些例子一般并不是好的排版实践模板。比如例 8 把有些公式编号放在左边，又把另一些放在了右边。你永远不会在一个实际的科学出版物中使用这样的公式编号。

除了第一个例子以外，每个例子都由一个叫 `\xmpheader` 的宏开始（见第 75 页）。我们这样做是为了节省输入文本的篇幅，否则每个例子开头你都会看到几行你先前已经看到的内容。`\xmpheader` 会排印出例子的标题和标题后的空白。你可以参见没有使用 `\xmpheader` 的第一个例子是如何实现这一点的，然后你就能模仿它了。除了 `\xmpheader`，在这里使用的每个命令都是在 Plain TeX 中定义过的。

例 1: 输入简单文本

由于 TeX 并不关心原始的输入文本中的换行，因此，我们无须关心应该给其提供何种格式化的文本。在 TeX 看来，输入文本中的换行符和空格等价。[†] 如果你不想在行尾处看到空格，可以在行尾处写一个百分号（注释符号）。TeX 忽略行首的空格，对多个一个的空格当做单独的空格看待 — 哪怕这些空格位于句号后面。你可以用一个或者多个空白行表示一个新的段落的开始。

当 TeX 看见一个句号后面紧接着一个空格（或者是一个行结束符，这两者等价）时，它通常会认为此句号代表一句话的结束，并在句号的后面加上一点小小的空格。同样的处理还适用于问号和惊叹号。

有时需要微调 TeX 处理句号的方式。当句号前面紧跟着标题字母（大写字母）时，TeX 会认为这个句号并不做为句子的结束符，因此在这种情况下，你需要作一些小改动。例如：我们这样写 DNA. 如果想将多个单词绑定到一起，中间不分行，则可以将它们放入一个引用中，例如：“见 Fig. 8”；或者名字中，例如：V. I. Lenin；或者省略号 ...。在这样的情况下，TeX 不会在它们之间断行（三个点表示省略号）。

为得到正确的左右双引号，你需要用两个左右单“引号”。“在相邻的单引号和双引号之间，要插入一个‘小间隔’”。连接号--可以这样写，而破折号---可以这样写。

[†] TeX 将 Tab 也当作空格处理，就如你在这个脚注所见到的一样。

```
% TeX 忽略一行中% 之后的任何内容
% 下面两行定义标题所用的字体
\font\xmplbx = cmbx10 scaled \magstephalf
\font\xmplbxti = cmbxti10 scaled \magstephalf
% 现在是标题行
\leftline{\xmplbx 例 1:\quad\xmplbxti 输入简单文本}
\vglue .5\baselineskip % skip an extra half line
```

由于 \TeX\ 并不关心原始的输入文本中的换行，因此，我们无须关心应该给其提供何种格式化的文本。在 \TeX\ 看来，输入文本中的换行符和空格等价。%

\footnote \dag{\TeX\ 将 Tab 也当作空格处理，就如你在这个 {\it 脚注} 所见到的一样。} 如果你不希望在行尾处看到空格，可以在行尾处写一个百分号（注释符号）。\TeX\ 忽略行首的空格，对多个空格当做一个单独的空格看待 --- 哪怕这些空格位于句号后面。你可以用一个或者多个空白行表示一个新的段落的开始。

当 \TeX\ 看见一个句号后面紧接着一个空格（或者是一个行结束符，这两者等价）时，它通常会认为此句号代表一句话的结束，并在句号的后面加上一点小小的空格。同样的处理还适用于问号和感叹号。

有时需要微调 \TeX\ 处理句号的方式。当句号前面紧跟着标题字母（大写字母）时，\TeX\ 会认为这个句号并不做为句子的结束符，因此在这种情况下，你需要作一些小改动。例如：我们这样写 DNA\null. % \null 可以阻止 \TeX\ 感知句号前面的标题字母 ‘A’。如果想将多个单词绑定到一起，中间不分行，则可以将它们放入一个引用中，例如：“见 Fig.\~8”；或者名字中，例如：V.\~I\null. Lenin；或者省略号 \\$\ldots\$. 在这样的情况下，\TeX\ 不会在它们之间断行（三个点表示省略号）。

为得到正确的左右双引号，你需要用两个左右单“引号”。‘‘在相邻的单引号和双引号之间，要插入一个‘小间隔’\thinspace’’。连接号--可以这样写，而破折号---可以这样写。

\bye % 结束此文档

例 2: 缩进

现在让我们来看看如何控制缩进。如果一个普通的文字处理程序都能处理缩进的话，那 T_EX 一定也能。请注意，本段并未缩进。

通常来说，你会想在段落的起始处安排缩进，在段落之间安排一些额外的空白。由于我们在此尚未做任何特殊处理，因此本段是缩进的。

让我们对这两个段落作一些特殊处理，这两段将不缩进，并且两段之间有 6pt 宽的空白。

这是第二段不缩进的文字。如果不在这两段之间插入空白的话，将很难分辨段与段之间的结束和起始。

你也能在段落的两边同时缩进，下面的 3 段文字演示了双边缩进的效果：

“我们对这一段作了双边缩进的处理。在长段引用时，通常都会这么做。”

“如果你愿意的话，可以同时对多个段落作如此处理。这是第二个轻微缩进的段落。”

如果需要，你也可以让段落缩进双倍长度。这就是一个双倍缩进段落的例子。

在这一段，如你所见，我们返回到正常的边距状态。我们尽量让这一段稍微长一点，这样你可以更清楚地看到边距的效果。

现在这一段，我们将其调整为左边距为半英尺，而右边距保持原样。

最后，我们将本段调整为右边距半英尺，而左边距保持原样。

\xmpheader 2/{缩进}% 见 第 22 页

\noindent 现在让我们来看看如何控制缩进。如果一个普通的文字处理程序都能处理缩进的话，那 \TeX\ 一定也能。请注意，本段并未缩进。

通常来说，你会想在段落的起始处安排缩进，在段落之间安排一些额外的空白。由于我们在此尚未做任何特殊处理，因此本段是缩进的。

```
{\parindent = 0pt \parskip = 6pt
% 这里的左括弧包裹了一组不缩进的文本
```

让我们对这两个段落作一些特殊处理，这两段将不缩进，并且两段之间有 6pt 宽的空白。

这是第二段不缩进的文字。如果不在这两段之间插入空白的话，将很难分辨段与段之间的结束和起始。

```
\par % The paragraph *must* be ended within the group.
} % The right brace ends the group containing unindented text.
```

你也能在段落的两边同时缩进，下面的 3 段文字演示了双边缩进的效果：

```
\smallskip % Provide a little extra space here.
% Skips like this and \vskip below end a paragraph.
{\narrower
“我们对这一段作了双边缩进的处理。在长段引用时，通常都会这么做。”
```

“如果你愿意的话，可以同时对多个段落作如此处理。

这是第二个轻微缩进的段落。’’\par}

```
{\narrower \narrower 如果需要，你也可以让段落缩进双倍长度。
这就是一个双倍缩进段落的例子。 \par}
```

\vskip 1pc % Skip down one pica for visual separation.

在这一段，如你所见，我们返回到正常的边距状态。我们尽量让这一段稍微长一点，这样你可以更清楚地看到边距的效果。

```
{\leftskip .5in 现在这一段，我们将其调整为左边距为半英尺，而
右边距保持原样。 \par}
```

{\rightskip .5in 最后，我们将本段调整为右边距半英尺，而左边距保持原样。 \par}

\bye % 结束此文档

例 3: 字体和特殊字符

这是意大利体单词 *italic words*, 粗体单词 **bold words**, 而这里是意大利体、粗体和罗马体这几种字体混用的效果: *put bold and roman inside italic*。当倾斜字体后面紧跟非倾斜字体时, 我们在其间插入一个“倾斜校正” ($\backslash\textit{V}$), 使得中间的空白看起来较舒适。这是一些更小的字词---可是标准的 TeX 字体并不会给出小于如此大小的字体。

如果你需要以下十个字符任意之一:

\\$ & # - % ^ ~ { } \

你需要通过特殊的方式来写他们。参照对开页, 学习正确的记录这些特殊字符的方法。

TeX 拥有一些语音和字母, 可以用于某些法语词, 例如: *rôle* 以及 *élève*, 或者德语词, 例如 *Schüß*, 或者某些别的什么语言。你可以在 page 100 和 page 97 处找到 TeX 的欧洲语音和字母表。

你也可以写一些希腊字母, 例如数学中的 “ α ” 以及 “ Ω ”, 扑克牌中的 “ \spadesuit ” 以及 “ \diamondsuit ”, 一些音乐符号, 例如: “ \sharp ” 和 “ \flat ”, 以及一些你可以在 page 197 找到的各种符号。TeX 仅在“数学模式”下接受这些符号, 因此, 你需要将它们放在两个 ‘\$’ 字符中间。

```
\xmpheader 3/{字体和特殊字符}% 见 第 22 页
\chardef \\ = '\\" % Let \\ denote a backslash.
这是意大利体单词 {\it italic words}, 粗体单词 {\bf bold words},
而这里是意大利体、粗体和罗马体这几种字体混用的效果:
{\it put {\bf bold} and {\rm roman} inside italic}。
当倾斜字体后面紧跟非倾斜字体时, 我们在其间插入一个 “倾斜校正”,
({\tt \\}), 使得中间的空白看起来较舒适。
这是一些 {\sevenrm 更小的} 字词---可是标准的 \TeX\
字体并不会给出小于 {\fiverm 如此大小} 的字体。
```

如果你需要以下十个字符任意之一:

```
\medskip
\centerline{\$ \quad \& \quad # \quad _ \quad \% \quad
\char '^' \quad \char '^' \quad \$\$ \quad
\$\$ \quad \$\backslash\$}
% The \quad inserts an em space between characters.
```

\medskip
\noindent 你需要通过特殊的方式来写他们。参照对开页,
学习正确的记录这些特殊字符的方法。

\TeX\ 拥有一些语音和字母, 可以用于某些法语词,
例如: {\it r\^ole\}/} 以及 {\it 'el`eve\}/},
或者德语词, 例如 {\it Schu\ss\}/}, 或者某些别的什么语言。
你可以在 page 100 和 page 97 处
找到 \TeX\ 的欧洲语音和字母表。

你也可以写一些希腊字母, 例如数学中的 “\$\alpha\$” 以及
“\$\Omega\$”，扑克牌中的
“\$\spadesuit\$” 以及 “\$\diamondsuit\$”，一些音乐符号, 例如:
“\$\sharp\$” 和 “\$\flat\$”，以及一些你可以在 page 197
找到的各种符号。 \TeX\ 仅在 “数学模式” 下接受这些符号, 因此,
你需要将它们放在两个 ‘{\tt \\$}’ 字符中间。
\bye % 结束此文档

例 4: 行间距

有的时候，你会想在文档的文字行之间增加一些空白。例如：议会的某些讨论稿需要这样做，使得议院们能够在其上作一些标记。书商们也基于同样的理由，要求作者提交的手稿具有双倍的行间距。当然，对最终的出版物来说，双倍行间距的情形极其少见。

基线是一条想象出来的线，其作用类似于带横格标记的纸张上横格线的作用。你可以通过控制基线间的距离来控制两行文字之间的距离——打印机管这个距离叫做“起始空白”。具体的做法可以参看这段的源文件。你可以用 1.5 取代 2 获得 1 1/2 空白的效果。（或者可以写成更好看的 $1\frac{1}{2}$ 形式。）

做为举例，我们这里增加了段落的缩进，并且在两段之间增加了一个空行。

```
\xmpheader 4/{行间距}%
\baselineskip = 2\baselineskip % double spacing
\parskip = \baselineskip % Skip a line between paragraphs.
\parindent = 3em % Increase indentation of paragraphs.

% The following macro definition gives us nice inline
% fractions. You'll find it in our eplain macros.
\def\frac#1/#2{\leavevmode
  \kern.1em \raise .5ex \hbox{\the\scriptfont0 #1}%
  \kern-.1em $/$%
  \kern-.15em \lower .25ex \hbox{\the\scriptfont0 #2}%
}%
```

有的时候，你会想在文档的文字行之间增加一些空白。例如：议会的某些讨论稿需要这样做，使得议院们能够在其上作一些标记。书商们也基于同样的理由，要求作者提交的手稿具有双倍的行间距。当然，对最终的出版物来说，双倍行间距的情形极其少见。

基线是一条想象出来的线，其作用类似于带横格标记的纸张上横格线的作用。你可以通过控制基线间的距离来控制两行文字之间的距离 --- 打印机管这个距离叫做“起始空白”。具体的做法可以参看这段的源文件。你可以用 {\tt 1.5} 取代 {\tt 2} 获得 $$1\frac{1}{2}$$ 空白的效果。（或者可以写成更好看的 $$1\frac{1}{2}$ 形式。）

```
% Here we've used the macro definition given above.
```

做为举例，我们这里增加了段落的缩进，并且在两段之间增加了一个空行。

```
\bye % 结束此文档
```

例 5：间隔、标线和盒子

这里展示的是“描述列表”的实例。实际使用时，最好将这些重复结构定义成宏，并且保证子标题的宽度足够宽。

Queen of Hearts An ill-tempered woman, prone to saying “Off with his head!” at the slightest provocation.

Cheshire Cat A cat with an enormous smile that Alice found in a tree.

Mock Turtle A lachrymose creature, quite a storyteller, who was a companion to the Gryphon. Reputedly the principal ingredient of Mock Turtle Soup.

这个例子在一个标线盒子中放入一些词句，它们是路易斯·卡罗所写的：

谁不是最想尝一尝，
两便士一碗的好汤？

* * * * *

| 我们给此段落内容添加了修订线效果。修订线用于表示内容改动。

\xmpheader 5/{间隔、标线和盒子}% 见 第 22 页
 这里展示的是“‘描述列表’”的实例。实际使用时，最好将这些重复结构定义成宏，并且保证子标题的宽度足够宽。

```
\bigskip
% 用 \descindent 表示描述列表的缩进量，并设定它的值为 8 派卡。
\newdimen\descindent \descindent = 8pc
% 整个段落缩进 \descindent。段落间增加半个行距的间距。
{\noindent \leftskip = \descindent \parskip = .5\baselineskip
% 将描述放在段落的左边。
\llap{\hbox to \descindent{\bf Queen of Hearts\hfil}}%
An ill-tempered woman, prone to saying “Off with his
head!”\ at the slightest provocation.\par
\noindent\llap{\hbox to \descindent{\bf Cheshire Cat\hfil}}%
A cat with an enormous smile that Alice found
in a tree.\par
\noindent\llap{\hbox to \descindent{\bf Mock Turtle\hfil}}%
A lachrymose creature, quite a storyteller, who was a
companion to the Gryphon. Reputedly the principal ingredient
of Mock Turtle Soup.
\par}
\bigskip\hrule\bigskip % A line with vertical space around it.
这个例子在一个标线盒子中放入一些词句，它们是路易斯·卡罗所写的：
\bigskip
% 在文本和四周标线之间留有 8pt 宽的间隔。
\hbox{\vrule\vbox{\hrule
    \hbox spread 8pt{\hfil\vbox spread 8pt{\vfil
        \hbox{谁不是最想尝一尝，}%
        \hbox{两便士一碗的好汤？}%
    \vfil}\hfil}
\hrule}\vrule}%
\bigskip\line{\hfil\hbox to 3in{\leaders\hbox{ * }\hfil}\hfil}
\bigskip
\line{\hskip -4pt\vrule\hfil\vbox{
我们给此段落内容添加了修订线效果。修订线用于表示内容改动。}}
\bye % 结束此文档
```

例 6: 杂项

TeX 懂得如何将单词连字化，但它并非绝对可靠的。若你在讨论化学 5-[p-(Fluorosulfonyl)benzoyl]-1,N⁶-ethenoadenosine 时，TeX 给出关于“水平盒子溢出”的警告，你可以尝试插入某些“自定连字符”。“‐”记号告诉 TeX 一个自定连字符位置；在此位置 TeX 本来是不会插入连字符的。

你可以不对齐地排版文字，即让右页边不对齐。在过去，文字处理软件还未普及，因为没有其他方便的选择，排版的文档都是不对齐的。有些人更喜欢不对齐地排版文字，因为此时单词间距是一致的。大部分书籍都设置为页边对齐，但并非都是。

断言 27. 有很简单的办法排版断言、引理、定理等的标题。

这个例子显示如何排版两层的有序列表。如果需要更多层级，唉，你得自己编程。

1. 这是第一个列表项。

2. 这是第二个列表项。它由两个段落组成。为让你看清楚第二个段落从哪里开始，我们缩进了该段落。

第二个段落下边有三个子列表项。

- (a) 这是第一个子列表项。
- (b) 这是第二个子列表项。
- (c) 这是第三个子列表项。

• 这是一个看起来很奇怪的列表项，因为它和其他列表项不同。

这是一个左对齐的行。⇐

⇒ 这是一个右对齐的行。

⇒ 这是一个居中的行。⇐

```
\xmpheader 6/{杂项}% 见 第 22 页
\chardef \\ = ‘\\ % 用 \\ 表示反斜杠。
\footline{\hfil{\tenit - \folio -}\hfil}
% \footline 给出页脚行；此处为居中的意大利体页码。
\TeX\ 懂得如何将单词连字化，但它并非绝对可靠的。若你在讨论化学
${\it 5}]-[p-(Flouro\-\sul\-\fonyl)ben\-\zoyl]-1,%
$N^6$-ethe\-\no\-\adeno\-\sine 时，
\TeX\ 给出关于 ‘‘水平盒子溢出’’ 的警告，  

你可以尝试插入某些 ‘‘自定连字符’’。  

‘{\tt \\-}’ 记号告诉 \TeX\ 一个自定连字符位置；  

在此位置 \TeX\ 本来是不会插入连字符的。
\medskip
{\raggedright 你可以不对齐地排版文字，即让右页边不对齐。  

在过去，文字处理软件还未普及，  

因为没有其他方便的选择，排版的文档都是不对齐的。  

有些人更喜欢不对齐地排版文字，因为此时单词间距是一致的。  

大部分书籍都设置为页边对齐，但并非都是。 \par}
```

\proclaim 断言 27. 有很简单的方法排版断言、引理、定理等的标题。

这个例子显示如何排版两层的有序列表。
如果需要更多层级，唉，你得自己编程。
\smallskip
\item {1.} 这是第一个列表项。
\item {2.} 这是第二个列表项。它由两个段落组成。
为让你看清楚第二个段落从哪里开始，我们缩进了该段落。

```
\item{} \indent 第二个段落下边有三个子列表项。  

\itemitem {(a)} 这是第一个子列表项。  

\itemitem {(b)} 这是第二个子列表项。  

\itemitem {(c)} 这是第三个子列表项。  

\item {$\bullet$} 这是一个看起来很奇怪的列表项，  

因为它和其他列表项不同。  

\smallskip  

\leftline{这是一个左对齐的行。 $\Leftarrow$}  

\rightline{$\Rightarrow$ 这是一个右对齐的行。 }  

\centerline{$\Rightarrow$ 这是一个居中的行。 $\Leftarrow$}  

% 不要在段落内部使用这些命令。  

\bye % 结束此文档
```

例 7: 使用其他来源的字体

你并不局限于只能使用随 TeX 发行的 Computer Modern 系列字体。从其他很多渠道你都可以得到一些你可能会更喜欢的字体。例如我们当前页面就被设置为 10pt 大小的 Palatino Roman 字体。Palatino 是一种由 Hermann Zapf 所设计的字体，它被认为是二十世纪最伟大的字体设计之一。从本页的输出可以获得关于这个字体的一些直观印象（由于译本的缘故，本页的字体并未被设为 Palatino 字体）。

字体既可以是轮廓字体，也可以是位图字体。轮廓字体的意思是它描述的是各个字符的形状，而位图字体则是标识了每个字符形状所占据的像素点。轮廓字体可用来产生同一种字体的多种不同大小。随 TeX 而发行的 `metafont` 程序是一个产生位图字体的强大工具，而且它并不是唯一的工具。

由于单一的轮廓字体文件既可生成多种不同大小的字体，因此很多数字字体提供商倾向于仅提供字体的单一文件——例如 Palatino Roman 即是如此。这也许是个经济的决定，可它却同时是个美学上的重大牺牲。字体无法做到在不损害其显示质量的前提下，线性地放大或者缩小。一个字符的大字体版本，通常来说，与小字体的版本应该有不同的比例——而如果我们坚持这么做的话，其结果看起来将不会那么好看。例如：如果将一个字体线性缩小的话，其各笔画间的间距将会显得过窄，而它的 `x-height` 值也会显得过小。

字体的设计者可以为不同大小的字体设计不同的轮廓形状，从而对这个问题进行弥补，可这同时需要付出额外的代价。使用 `Metafont` 程序的一个巨大的好处就是可以将字符符号形状的描述参数化。这样在改变字体大小的时候，`Metafont` 可以视情况做一些自动处理，从而得到较好的效果。

```
\xmpheader 7/{使用其他来源的字体}%
\font\tenrm = pplr % Palatino
% Define a macro for invoking Palatino.
\def\pal{\let\rm = \tenrm \baselineskip=12.5pt \rm}
\pal % Use Palatino from now on.
```

你并不局限于只能使用随 \TeX 发行的 Computer Modern 系列字体。从其他很多渠道你都可以得到一些你可能会更喜欢的字体。例如我们当前页面就被设置为 10p 大小的 Palatino Roman 字体。Palatino 是一种由 Hermann Zapf 所设计的字体，它被认为是二十世纪最伟大的字体设计之一。从本页的输出可以获得关于这个字体的一些直观印象（由于译本的缘故，本页的字体并未被设为 Palatino 字体）。

字体既可以是轮廓字体，也可以是位图字体。轮廓字体的意思是它描述的是各个字符的形状，而位图字体则是标识了每个字符形状所占据的像素点。轮廓字体可用来产生同一种字体的多种不同大小。随 \TeX\ 而发行的 `metafont` 程序是一个产生位图字体的强大工具，而且它并不是唯一的工具。

由于单一的轮廓字体文件既可生成多种不同大小的字体，因此很多数字字体提供商倾向于仅提供字体的单一文件——例如 Palatino Roman 即是如此。这也许是个经济的决定，可它却同时是个美学上的重大牺牲。字体无法做到在不损害其显示质量的前提下，线性地放大或者缩小。一个字符的大字体版本，通常来说，与小字体的版本应该有不同的比例——而如果我们坚持这么做的话，其结果看起来将不会那么好看。例如：如果将一个字体线性缩小的话，其各笔画间的间距将会显得过窄，而它的 `x-height` 值也会显得过小。

字体的设计者可以为不同大小的字体设计不同的轮廓形状，从而对这个问题进行弥补，可这同时需要付出额外的代价。使用 Metafont 程序的一个巨大的好处就是可以将字符符号形状的描述参数化。这样在改变字体大小的时候，Metafont 可以视情况做一些自动处理，从而得到较好的效果。

\bye % 结束此文档

例 8: 标线表格

一些上等的食用蘑菇

学名	常用名	识别特征
<i>Pleurotus ostreatus</i>	平菇	Grows in shelflike clusters on stumps or logs, pink-gray oyster-shaped caps, stem short or absent.
<i>Lactarius hygrophoroides</i>	稀褶乳菇	Butterscotch-brown cap and stem, copious white latex, often on ground in woods near streams.
<i>Morchella esculenta</i>	羊肚菌	Conical cap with black pits and white ridges; no gills. Often found near old apple trees and dying elms in the spring.
<i>Boletus edulis</i>	美味牛肝菌	Reddish-brown to tan cap with yellow pores (white when young), bulbous stem, often near conifers, birch, or aspen.

```

\xmpheader 8/{标线表格}% 见 第 22 页
\bigskip
\offinterlineskip % 让竖直标线连接起来
% \tablerule 构造横穿表格的细标线
\def\tablerule{\noalign{\hrule}}
% \tableskip 在单元格之间生成 9pt 的空隙
\def\tableskip{\omit&height 9pt&&&\omit\cr}
% 用 & 分开模版各列。TeX 将把 # 替换为单元格的文本。
% 在表格每行都要有一个支架 (strut)；
% 否则盒子将不能良好接合，标线也不能连接起来。
\halign{\tabskip = .7em plus 1em % 各列之间的粘连
% 用 \vtop 得到表格第一列中的多行单元格。
% 让多行文本右边不对齐，且不连字化。
\vtop{\hsize=6pc\pretolerance = 10000\hbadness = 10000
      \normalbaselines\noindent\it#\strut}%
&\vrule ##\hfil &\vrule ## 标线和中间各列
% 用 \vtop 得到表格最后一列中的段落。
&\vtop{\hsize=11pc \parindent=0pt \normalbaselineskip=12pt
      \normalbaselines \rightskip=3pt plus2em #}\cr
% 表格各行从这里开始。
\noalign{\hrule height2pt depth2pt \vskip3pt}
% 表头行横跨各列。
\multispan5\bf 一些上等的食用蘑菇 \hfil\strut\cr
\noalign{\vskip3pt} \tablerule
\omit&height 3pt&\omit&&\omit\cr
\bf 学名 &&\bf 常用名 &&\omit \bf 识别特征 \hfil\cr
\tableskip Pleurotus ostreatus&& 平菇 &&
Grows in shelf\kern 1pt like clusters on stumps or logs,
% 若不加上 \kern , ‘f’ 和 ‘l’ 将挨得太近
pink-gray oyster-shaped caps, stem short or absent.\cr
\tableskip Lactarius hygrophoroides&& 稀褶乳菇 &&
Butterscotch-brown cap and stem, copious white latex,
often on ground in woods near streams.\cr
\tableskip Morchella esculenta&& 羊肚菌 &&Conical cap
with black pits and white ridges; no gills. Often found
near old apple trees and dying elms in the spring.\cr
\tableskip Boletus edulis&& 美味牛肝菌 &&Reddish-brown to
tan cap with yellow pores (white when young),
bulbous stem, often near conifers, birch, or~aspen.\cr
\tableskip \tablerule \noalign{\vskip 2pt} \tablerule
}\bye

```

例 9: 排版数学公式

设球面三角形的三条边长度分别为 a , b 和 c , 它们的对角分别为 α , β 和 γ , 我们有:

$$\cos \alpha = -\cos \beta \cos \gamma + \sin \beta \sin \gamma \cos \alpha \quad (\text{余弦定理})$$

且有:

$$\tan \frac{\alpha}{2} = \sqrt{\frac{-\cos \sigma \cdot \cos(\sigma - \alpha)}{\cos(\sigma - \beta) \cdot \cos(\sigma - \gamma)}}, \quad \text{其中 } \sigma = \frac{1}{2}(a + b + c)$$

我们还有:

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

且有:

$$\int_0^\infty \frac{\sin ax \sin bx}{x^2} dx = \frac{\pi a}{2}, \quad \text{如果 } a < b$$

从 n 个物品中任取 r 个的组合数 $_nC_r$ 为:

$$C(n, r) = {}_nC_r = \binom{n}{r} = \frac{n(n-1)\cdots(n-r+1)}{r(r-1)\cdots(1)} = \frac{n!}{r!(n-r)!}$$

n 阶行列式 D 的值:

$$D = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

定义为下面 $n!$ 项的和:

$$\sum (\pm) a_{1i} a_{2j} \dots a_{nk}$$

其中 i, j, \dots, k 取遍 1 到 n 的所有可能值, 且当排列 i, j, \dots, k 为偶置换时乘积取 + 号, 否则取 - 号。此外:

$$Q(\xi) = \lambda_1 y_1^2 \sum_{i=2}^n \sum_{j=2}^n y_i b_{ij} y_j, \quad B = \|b_{ij}\| = B'$$

\xmpheader 9/{排版数学公式}% 见 第 22 页

设球面三角形的三条边长度分别为 \$a\$, \$b\$ 和 \$c\$,

它们的对角分别为 \$\alpha\$, \$\beta\$ 和 \$\gamma\$, 我们有:

```
$$\cos \alpha = -\cos \beta \cos \gamma +  
 \sin \beta \sin \gamma \cos \alpha \quad  
 \hbox{(余弦定理)}$$
```

且有:

```
$$\tan {\alpha \over 2} = \sqrt{  
 {- \cos \sigma \cdot \cos(\sigma - \alpha) \over  
 {\cos(\sigma - \beta) \cdot \cos(\sigma - \gamma)}}}, \quad  
 \hbox{其中 } \sigma = {1 \over 2}(a+b+c)$$
```

我们还有: \$\$\sin x = {{e^{ix} - e^{-ix}} \over 2i}

且有:

```
$$\int_0^\infty {{\sin ax \sin bx} \over x^2} dx  
 % 上面的 \, 生成一个小间隔  
 = {\pi a \over 2}, \quad \hbox{如果 } a < b
```

\noindent 从 \$n\$ 个物品中任取 \$r\$ 个的组合数 \${}_n C_r\$ 为:

```
$$C(n, r) = {}_n C_r = {n \choose r} =  
 {{n(n-1) \cdots (n-r+1)} \over {r(r-1) \cdots (1)}} =  
 {{n!} \over {r!(n-r)!}}
```

\noindent

\$n\$ 阶行列式 \$D\$ 的值:

```
$$D = \left| \begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{matrix} \right|
```

定义为下面 \$n!\$ 项的和:

\$\$\sum_{\pm} a_{1i} a_{2j} \cdots a_{nk}\$\$

其中 \$i\$, \$j\$, \$\dots\$, \$k\$ 取遍 \$1\$ 到 \$n\$ 的所有可能值,

且当排列 \$i\$, \$j\$, \$\dots\$, \$k\$ 为偶置换时乘积取 \$+\$ 号,

否则取 \$-\$ 号。此外:

\$\$Q(x_i) = \lambda y_1^2 \sum_{i=2}^n \sum_{j=2}^n y_i

\$b_{ij} y_j, \quad \hbox{qquad } B = \begin{pmatrix} b_{ij} \end{pmatrix} = B'\$\$

\bye

例 10: 更多数学内容

X 的绝对值 $|x|$ 定义为:

$$|x| = \begin{cases} x, & \text{if } x \geq 0; \\ -x, & \text{otherwise.} \end{cases}$$

现在写一些标号了的公式。在 $k \geq 0$ 时, 我们有:

$$x^{k^2} = \overbrace{x x \cdots x}^{2k \text{ times}} \quad (1)$$

这是一个展示空格间距调整和公式左编号的例子:

$$(2a) \quad [u][v][w] [x] [y] [z]$$

从做到右公式中的项和括号的距离逐步增大。(我们把前两项的距离设定得比正常距离少。有时候,

$$(2b) \quad u'_1 + t u''_2 = u'_2 + t u''_1$$

$$(2c) \quad \begin{aligned} \hat{i} &\neq \hat{j} \\ \vec{a} &\approx \vec{b} \end{aligned}$$

结果为 $O(n \log \log n)$. 因此

$$\sum_{i=1}^n x_i = x_1 + x_2 + \cdots + x_n = \text{Sum}(x_1, x_2, \dots, x_n). \quad (3)$$

且

$$dx dy = r dr d\theta. \quad (4)$$

满足 $q \leq 0$ 的所有的 q 可以被写作:

$$\{ q \mid q \leq 0 \}$$

因此

$$\forall x \exists y P(x, y) \Rightarrow \exists x \forall y P(x, y)$$

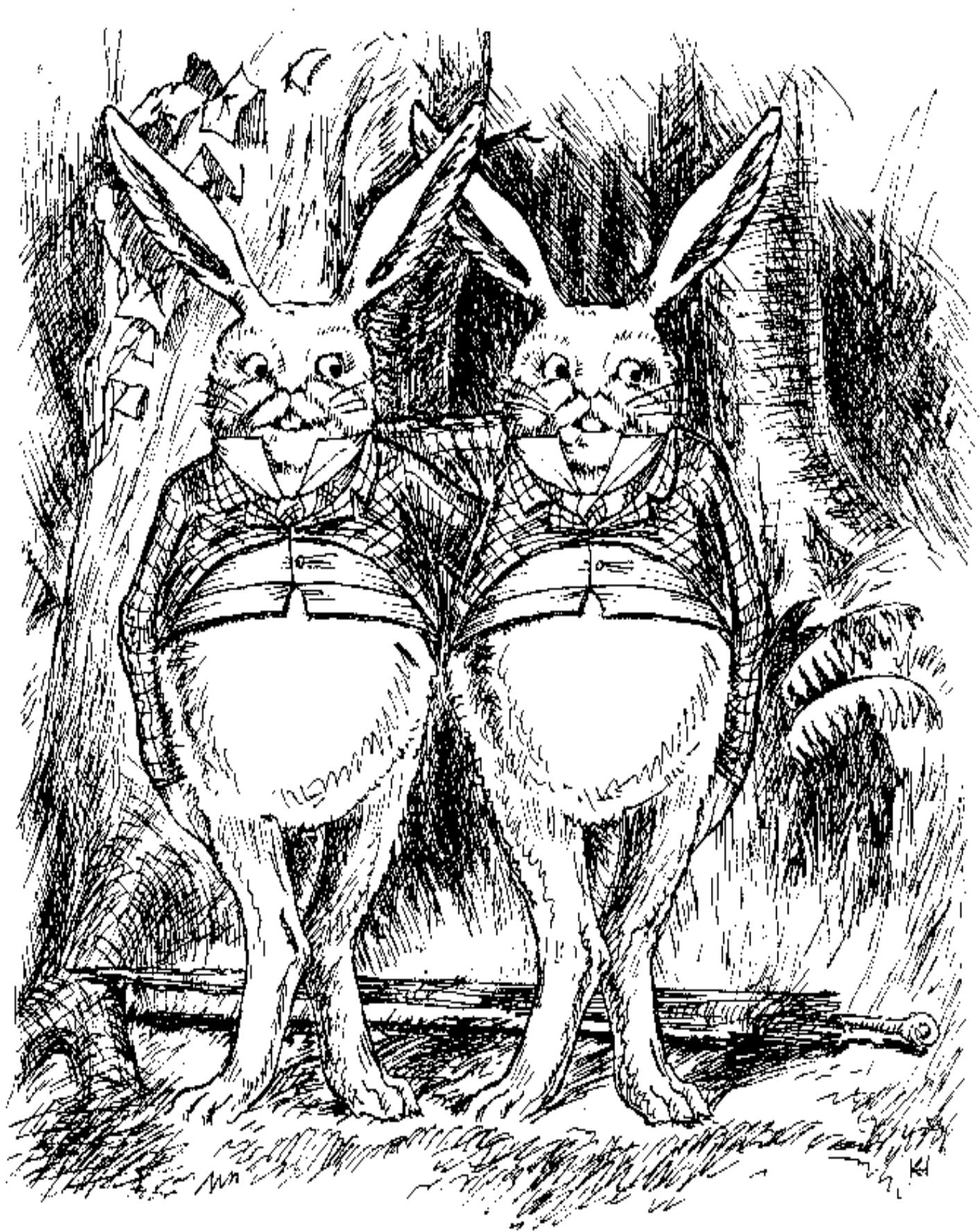
其中

$$P(x, y) \stackrel{\text{def}}{=} \text{任何一个能够得出 } x \text{ 和 } y.$$

```
\xmpheader 10/{更多数学内容}% 见 第 22 页
%^{math}
^{数学}
$X$ 的绝对值 $|x|$ 定义为:
$$|x| = \cases{x, & if $x \geq 0$; \\ -x, & otherwise.}$$
现在写一些标号了的公式。
在 $k \geq 0$ 时, 我们有:
$$x^{k^2}=\overbrace{x>x>\cdots>x}^{2k \text{ \rm times}} \eqno (1)$$
```

这是一个展示空格间距调整和公式左编号的例子:

```
$$[u]![v][w]\!, [x]>[y]\!; [z]\leqno(2a)$$
从做到右公式中的项和括号的距离逐步增大。
(我们把前两项的距离设定得比正常距离 {\it 少 \}。)
有时候, $$\leqalignno{
u'_1 + tu''_2 &= u'_2 + tu''_1 && (2b) \\
\hat{\imath} &\neq \hat{\jmath} && (2c) \\
\vec{vphantom{b}}a &\approx \vec{b} && \\
\% \vphantom{b} 是一个不可见的和 “b” 一样高的盒子。
结果为 $O(n \log \log n)$. 因此
$$\sum_{i=1}^n x_i = x_1+x_2+\cdots+x_n
= \{\rm Sum\}(x_1,x_2,\ldots,x_n). \eqno(3)$$
且
$$dx\!, dy = r\!, dr\!, d\theta\!. \eqno(4)$$
满足 $q \leq 0$ 的所有的 $q$ 可以被写作:
$$\{ \!, q \mid q \leq 0 \!, \}$$
因此
$$\forall x \exists y; P(x,y) \Rightarrow
\exists x \exists y; P(x,y)$$
其中
$$P(x,y) \text{ buildrel } \rm def \over \equiv
\hbox{\rm 任何一个能够得出 $x$ 和 $y$} . \quad
\bye
```

4 | 概念

本书的这部分包含了使用 TeX 时所需要了解的概念。它包括我们在解释命令时所使用的技术词汇以及重点内容，这些内容并不适合放在本书的其它地方。

这些概念使用英文字母顺序进行排列，书背内页包含了一份详细的在此提及的概念列表，以及他们被提及所在的页数。我们建议你把书背内页复印下来，放在旁边，这样你就能在遇到不熟悉的概念时马上查找到它们。我们尽可能地使用和 *The TeXbook* 相一致的术语。

活动字符. 活动字符 (active character) 是一个有定义的字符。比如一个活动字符可能表示着一个宏定义。你可以把活动字符理解为一种控制序列。当 TeX 碰到了一个活动字符，它就执行这个字符所表示的定义。如果 TeX 碰到的活动字符并没有表示一个定义，则它会报错说这是一个没有定义的控制序列。

一个活动字符的类别码为 13 (`\active` 的值)。当你定义一个活动字符时，你必须先使用 `\catcode` 命令 (第 267 页) 来使这个字符活动，然后再使用类似 `\def`, `\let` 或者 `\chardef` 之类的命令指定这个字符的定义。定义活动字符和定义控制序列所使用的方法是相同的。如因你尝试先对字符指定定义后把它设置成活动的，TeX 会报错说这是一个不明的控制序列。

比如，在 Plain TeX 中，波浪号 (~) 被定义成一个活动字符。它可以在所连接的两个单词当中产生一个空格，并且不让 TeX 把这个空格转为断行。Plain TeX 通过以下方式来定义 ‘~’：

```
\catcode ‘~ = \active \def~{\penalty10000\ }{}
```

(这里的 `\penalty` 可以阻止断行，‘\ ’可以插入一个空格。)

阵列。阵列 (alignment) 是一个把素材有序排印的结构，就如一张有行有列的表格。当你需要构造一个阵列时，你要 (a) 描述行和列的布局，并且 (b) 告诉 TeX 各行或各列中的内容。制表阵列和水平阵列是由一系列的行组成；竖直阵列则由一系列的列组成。我们首先讲述制表阵列和水平阵列，然后简要地介绍一下竖直阵列。

制表阵列是在 Plain TeX 中定义的。它很简单，因此不如水平阵列那样灵活。制表阵列和水平阵列的主要区别在于你如何定义它们的布局。

要构造一个制表阵列，你首先需要用 `\settabs` 命令（第 182 页）确定 TeX 如何将可用空白水平地分为多列。然后你就可以逐行地填充表格的各行。每行由控制序列 `\+`（第 182 页）及其后的一排用 & 隔开的“条目”组成（条目是行和列的交叉部分），并以 `\cr` 结束。如果某行的条目数目比该阵列的列数少，TeX 就用空白条目填充它们。

你可以将制表阵列的行放在文档的任何地方，只要前面有 `\settabs` 命令。特别地，你可以在制表阵列的各行之间添加其他内容，或者用同一个 `\settabs` 描述多个制表阵列。这里有一个制表阵列的例子：

```
{\hsize = 1.7 in \settabs 2 \columns  
 \+cattle&herd\cr  
 \+fish&school\cr  
 \+lions&pride\cr}
```

这个例子中的 `\settabs 2 \columns` 命令（第 182 页）让 TeX 生成总宽度为 1.7 英寸的等宽两列。所排版出的阵列如下：

cattle	herd
fish	school
lions	pride

制表阵列还有另一种用模板指明各列宽度的用法。模板的各列宽度决定了该阵列其后的各列宽度：

```
{\settabs \+cattle\quad&school\cr  
 \+cattle&herd\cr  
 \+fish&school\cr  
 \+lions&pride\cr}
```

这是上面例子生成的结果：

cattle	herd
fish	school
lions	pride

水平阵列用 `\halign`（第 184 页）命令构造。TeX 将根据各列内容调整水平阵列的列宽。当 TeX 碰到 `\halign` 命令要开始一个水平阵列

时，它首先检查该阵列的各行，看看各条目都有多宽。然后设定各列宽度以容纳该列中最宽的条目。

由 `\halign` 命令构造的水平阵列包含一个“导言”，该导言说明了随后各行的布局。

- 导言由一系列模板组成，一个模板对应一列，指明该列的文本应该如何排版。每个模板中都必须包含一个 # 字符，用于给出 TeX 将条目的文本放入模板的位置。各个模板之间用 & 分隔，并以 `\cr` 结束该导言。利用合适的模板你可以让某列居中，左对齐或右对齐，或者使用某个特定的字体。
- 除了要省略各行开头的 `\+`，各行的写法都和制表阵列一样：一行的各条目之间用 & 分隔，并以 `\cr` 结束该行。TeX 将各个条目都看作一个编组，因此对某列模板的字体设定或其他赋值都只对该列各个条目有效。

导言和各行都必须放在 `\halign` 后的花括号里面。每个 `\halign` 阵列都必须包含各自的导言。

例如，下面这个水平阵列：

```
\tabskip=2pc
\halign{\hfil#\hfil &\hfil#\hfil &\hfil#\hfil \cr
&&\it Table\cr
\noalign{\kern -2pt}
\it Creature&\it Vinctual&\it Position\cr
\noalign{\kern 2pt}
Alice&crumpet&left\cr
Dormouse&muffin&middle\cr
Hatter&tea&right\cr}
```

生成下面的结果：

		<i>Table</i>
<i>Creature</i>	<i>Vinctual</i>	<i>Position</i>
Alice	crumpet	left
Dormouse	muffin	middle
Hatter	tea	right

这个例子中的 `\tabskip` (第 190 页) 命令让 TeX 在各列间插入 2pc 的粘连。`\noalign` (第 189 页) 命令让 TeX 在两行间插入竖直模式素材。在此例子中我们用 `\noalign` 生成标题行和数据行之间的额外间距，还让“Table”和“Position”紧挨在一起。(在首行之前或者末行之后你同样可以使用 `\noalign` 命令。)

用 `\valign` 命令（第 185 页）可以构造一个竖直阵列。竖直阵列按照列而不按照行组织。它遵循和水平阵列相同的法则，只是此时行和列的地位交换了。例如，下面的竖直阵列：

```
{\hsize=0.6in \parindent=0pt  
\valign{#&\strut&#&\strut&#&\strut\cr  
one&two&three\cr  
four&five&six\cr  
seven&eight&nine\cr  
ten&eleven\cr}}
```

得到的结果为：

one	four	seven	ten
two	five	eight	eleven
three	six	nine	

模板中的 `\strut` 命令（第 172 页）是必不可少的，因为它让一行的各条目合适地排成一行，即让它们有共同的基线，并让基线间的距离保持一致。

TeX 剖析. *The TeXbook* 将 TeX 处理输入的过程用 TeX 的“消化道”来描述，包括“眼睛”，“嘴巴”，“食道”，“胃”和“肠道”。知道这个处理过程如何运转，将有助于你理解 TeX 消化文档时的行为细节。

- 使用“眼睛”，TeX 从输入文件中读取字符并传给它的嘴巴。由于输入文件中可能包含 `\input` 命令（第 263 页），TeX 实际上能够从一个文件“转移视线”到另一个文件。
- 使用“嘴巴”，TeX 将字符组合为记号并传给它的食道。每个记号要么是一个控制序列要么是单个字符，其中控制序列总是以转义符开始。注意空格符以及行尾符自身也是字符，只是 TeX 将输入中的多个空格合并为一个空格记号。在 *The TeXbook* 第 46–47 页中介绍了 TeX 将字符组合为记号的规则。
- 使用“食道”，TeX 展开任何宏、条件句，以及它找到的类似结构（见 *The TeXbook* 第 212–216 页，并将得到的一系列记号传给 TeX 的胃。展开一个记号也许会得到本身也需要展开的其他记号。TeX 按照从左到右的顺序执行记号展开，除非你用类似 `\expandafter`（第 249 页）的命令改变其顺序。换言之，TeX 的食道总是展开未送入胃中的记号中最左边的未展开记号。
- 使用“胃”，TeX 一组一组地处理记号。每个分组里面包含一个原始命令及可能跟随着的命令参量。大多数命令属于“排版此字符”

这种，因此它们的分组中仅包含一个记号。按照命令的指示， \TeX 的胃从字符到页面逐步组装出越来越大的部件，并将生成的页面传给 \TeX 的肠道。 \TeX 的胃执行断行任务——即将每个段落分为若干行，以及分页任务——即将连续若干行和其他竖直模式素材分为多个页面。

- 使用“肠道”， \TeX 将它的胃生成的页面转换为适合其他程序处理的形式，并将转换出的结果送到 .dvi 文件。

多数时候你都可以认为在 \TeX 的眼睛，嘴巴，食道，胃和肠道中的操作是一个接一个进行的。但事实真相是，在 \TeX 的胃中执行的命令可能会影响前面步骤的消化过程。举例来说，当 \TeX 的胃碰到 \input 命令（第 263 页）时，它的眼睛将开始读取另一个文件；当 \TeX 的胃碰到指定字符 c 的类别码的 \catcode 命令（第 267 页）时， \TeX 的嘴巴对 c 的解释就会受影响；而当 \TeX 的胃碰到宏定义时，在 \TeX 的食道进行的宏展开也会受到影响。

为更好地理解各个器官的相互影响，你可以把各个器官都想象为急性子，前任器官一有输出，后面的器官就急切地吞食该输出。比如， \TeX 的胃一看到 \input 命令的文件名的最后一个字符，它的眼睛就立即转移视线到所指定输入文件的第一个字符。

参量. 命令的参量 (argument) 包含传递给命令的文本，它补全了该命令正常运行所需的信息。这里的命令可以是原始命令或者宏。

原始命令的参量有各自规定的形式。例如，下面的一系列记号：

```
\hskip 3pc plus 1em
```

由 ‘\hskip’ 命令及其参量 ‘3pc plus 1em’ 组成。但若你这样写：

```
\count11 3pc plus 1em
```

你将得到完全不同的结果。 \TeX 将认为 ‘\count11’ 为命令，其参量为 ‘3’，而后面的 ‘pc plus 1em’ 是普通的文本记号（因为计数寄存器要求一个整数值作为参量）——这多半不是你想要的。顺便说一下，该命令将给第 11 号计数寄存器赋值 3（见 \count 这里的讨论，第 258 页）。

另一方面，所有宏都遵循相同的参量约定。传递给宏的每个参量都对应宏定义中的一个参数。宏的参数可以是“定界的”或是“非定界的”。在宏的定义中确定了宏参数的个数和类型，从而确定了宏参量的个数和类型。

定界参量和非定界参量的区别在于， \TeX 用不同方式确定该参量的结束位置。

- 定界参量由该参量开始处和作为该参量定界子的特定记号序列之间的所有记号组成，但不包含该定界子序列。其中的定界子是在宏的

定义中规定的。因此要给出定界参量，你需要写上该参量并在其后加上定界子。定界参量也可以是空的，即不包含任何文本。定界参量里面的任何花括号都必须配好对，即每个左花括号必须有对应的右花括号，反之亦然。

- 非定界参量由单个记号或者包含在花括号的一系列记号组成，比如 ‘{Here is {the} text.}’。外面的一对花括号，虽然看起来像，实际上并不组成一个编组——*TeX* 只是用它们确定参量的范围。而参量内部的花括号，比如 ‘the’ 两边的，必须正确配对。若你误将太多右花括号放在里面，*TeX* 将报错说有多余的右花括号。若左花括号太多，*TeX* 同样会报错；但这个错误可能出现在该参量所期望的结束位置之后很远的地方（见 第 293 页）。

请参阅“宏”（第 75 页）这里的更多关于参数和参量的内容。在 *The TeXbook* 第 203–204 页你能找到定界参量和非定界参量的精确规则。

ASCII. ASCII 是“美国信息交换标准编码”（American Standard Code for Information Interchange）的缩写。总共有 256 个 ASCII 字符，每个都有自己的编码，但只有前面 128 个是标准编码。在 ASCII “编码表”，比如 *The TeXbook* 第 367 页里面，你可以找到这些编码的含义。编码在 32–126 之间的字符为“可打印字符”，比如字母，数字和标点符号。其余字符为“控制字符”，通常用于（在计算机工业中，而不是 *TeX* 中）控制输入/输出和数据交换设备。比如，ASCII 码 84 对应于字符 ‘T’，而 ASCII 码 12 对应于“换页”功能（大多数打印机都将它解释为“开始新页”）。虽然 ASCII 标准指定了控制字符的含义，但很多设备，比如调制解调器和打印机，都将控制字符用于标准规定之外的目的。

TeX 中字符的含义通常和标准 ASCII 中的含义一致。而且在包含 ASCII 可打印字符的字体中，这些字符所在的位置也和 ASCII 中的一样。但有些字体，特别是数学字体，将 ASCII 可打印字符替换为与 ASCII 无关的字符。比如，计算机现代数学字体 *cmsy10* 在 ASCII 数字 ‘8’ 的位置上是数学符号 ‘\forall’。

赋值. 赋值（assignment）是一种为 *TeX* 的寄存器，内部参数，内部表格项，或者控制序列分配值的结构。赋值的一些例子如下：

```
\tolerance = 2000
\advance\count12 by 17
\lineskip = 4pt plus 2pt
\everycr = {\hskip 3pt \relax}
\catcode`\@ = 11
\let\graf = \par
```

```
\font\myfont = cmbx12
```

第一个赋值要求 \TeX 分配数值 2000 给数值参数 \tolerance , 即让 \tolerance 的值等于 2000。其他赋值类似。赋值中的 ‘=’ 和空格是可省略的, 因此你也可以把第一个赋值改写为下面更紧凑的形式:

```
\tolerance2000
```

在 *The \TeXbook* 第 276–277 页中对赋值语法有详细介绍。

劣度. 文本行的劣度 (badness) 值测量该行的字间隔偏离它们的自然值多远, 其中自然值就是文本行所用字体中指定的值。偏离越大, 劣度就越大。类似地, 页面的劣度测量页面盒子的间隔偏离它们的理想值多远。(一般地, 这些盒子大多数是段落的文本行。)

更准确地说, 劣度测量这些间隔对应的粘连要伸缩多少才能正好填满文本行或页面。 \TeX 让劣度近似等于, 为组成所需尺寸的行或页, 粘连必须伸缩的比例的立方的 100 倍。例如, 粘连伸长了给定伸长量的两倍得到的比例为 2 而劣度为 800; 粘连伸长了给定伸长量的一半得到的比例为 .5 而劣度为 13。 \TeX 将超过 10000 的劣度视为等于 10000。

\TeX 分段为行时使用行劣度 (见 “断行点”, 第 74 页)。它在下面两个步骤中使用行劣度:

- 1) 在选择断行点时, \TeX 要求各行劣度小于或等于 \tolerance 值 (第 123 页)。如果不得不放置一个劣度超过此值的行, \TeX 将把它设定为未满或过满水平盒子。 \TeX 仅在万不得已时才会放置未满或过满水平盒子, 即仅在没有其他方法分段为行时。
- 2) 假设各行都相当糟糕, \TeX 用行劣度评价不同的分段为行方式。在评价过程中它对每个可能的行结合一个 “缺陷”。劣度会增加缺陷值。然后 \TeX 选择让段落总缺陷最小的方式来分段为行。在多数情况下, \TeX 使用让最糟糕行的劣度最小的方式安排段落。请参阅 *The \TeXbook* 第 97–98 页中 \TeX 如何分段为行的细节。

\TeX 将一系列文本行和其他竖直模式素材组装为页面的过程与断行的过程类似。但是组装页面没那么复杂, 这是由于 \TeX 寻找分页点时每次只考虑一个页面, 因此只需要决定在哪里结束当前页面。与此相反, 当选择断行点时 \TeX 要同时考虑多行。(大多数文字处理器每次只选择一个断行点, 因而断行做得不如 \TeX 。) 请参阅 *The \TeXbook* 第 111–113 页中 \TeX 如何选择分页点的细节。

基线. 盒子的基线 (baseline) 是一条想象中的横穿盒子的线。在将水平列表的盒子组装为更大的盒子时, \TeX 将列表中各盒子对齐, 以让它们的基线重合。打个比方, 想象你在标线上写字, 你所写的每个字母

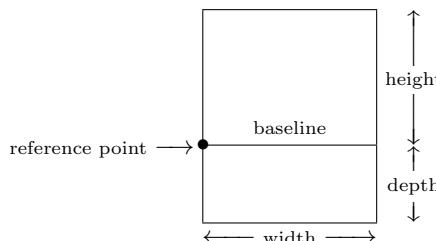
都有一条隐含基线。为让各字母水平对齐，你会让它们的基线和本子上印刷的浅色提示线重合。

盒子可以而且也经常会延伸到基线下边。比如，字母‘g’就延伸到其盒子的基线下边，因为它有一个降部（‘g’底部弯曲部分）。

盒子。盒子（box）是要排版的矩阵素材。单个字符自身就是一个盒子，整个页面同样也是一个盒子。 \TeX 用盒子套盒子套盒子以形成页面。最外层盒子就是页面自身，最内层盒子多半是单个字符，而单个文本行的盒子在中间某个地方。

\TeX 在构造段落和页面时隐式执行大多数盒子构建活动。利用一些 \TeX 命令，比如 \hbox （第 164 页）、 \vbox （第 166 页）和 \vtop （第 166 页），你可以显式构造盒子。 \hbox 命令通过从左到右水平添加较小的盒子构造一个盒子；它对一个水平列表进行操作以生成一个水平盒子（horizontal box）。 \vbox 和 \vtop 命令通过从上到下竖直添加较小的盒子构造一个盒子；它们对一个竖直列表进行操作以生成一个竖直盒子（vertical box）。这些水平列表和竖直列表不仅可以包含较小的盒子，还可以包含其他几种实体，比如粘连和紧排。

盒子有高度、深度和宽度，如下所示



基线（baseline）如同标线本上的浅色提示线。像‘g’这样的字母盒子延伸到基线下边；而像‘h’这样的字母盒子却不会。盒子的高度是盒子在基线上边的延伸距离，而它的深度是在基线下边的延伸距离。盒子的基准点（reference point）是它的基线与左边缘的交点。

在从水平列表构建水平盒子 H 时， \TeX 先设定 H 的基准点，然后从左到右逐个添加列表中的项目到 H 中。列表中的每个盒子按照其基线与 H 的基线重合的方式放置，即让组成盒子水平排成一排。¹ H 的高度等于列表中最高盒子的高度，而深度等于列表中最深盒子的深度。 H 的宽度等于列表中所有项目的宽度之和。如果这些项目中有的是粘连， \TeX 需要伸缩粘连，从而 H 的宽度将相应地变大或变小。请参阅 *The \TeXbook* 第 77 页的细节描述。

¹ 若盒子被 \raise 或 \lower 升高或降低了， \TeX 放置时使用它移动之前的基准点。

类似地，在从竖直列表构建竖直盒子 V 时， \TeX 先设定 H 的临时基准点，然后从上到下逐个添加列表中的项目到 V 中。列表中的每个盒子按照其基准点与 V 的基准点竖直对齐的方式放置。² 在第一个除外的每个盒子加到 V 时， \TeX 在它之前放置行间粘连。（在水平盒子中没有和行间粘连对应的东西。） V 的宽度等于列表中最宽盒子的宽度，而 V 的竖直长度（高度加深度）等于列表中所有项目的竖直长度之和。

\vbox 和 \vtop 的区别在于它们如何将 V 的竖直长度划分为高度和深度。选定了 V 的基准点就决定了划分方式。

- 对于 \vbox ， \TeX 将基准点放在 V 的最后一个组成盒子或标线的基准点所在的水平线上，除非最后一个盒子（或标线）之后还有粘连或紧排，此时 \TeX 将基准点放在 V 的最底端。³
- 对于 \vtop ， \TeX 将基准点放在 V 的第一个组成盒子或标线的基准点所在的水平线上，除非第一个盒子（或标线）之前还有粘连或紧排，此时 \TeX 将基准点放在 V 的最顶端。

粗略地说， \vbox 将基准点放在靠近竖直盒子底部处，而 \vtop 将基准点放在靠近顶部处。在你想对齐一行竖直盒子，让它们的顶部水平排成一排时，你通常应该使用 \vtop 而不是 \vbox 。请参阅 *The \TeXbook* 第 78 和 80–81 页中 \TeX 如何构建竖直盒子的详细信息。

在构造盒子时你有极大的自由度。盒子中排版的素材可以超出盒子的边界，就像某些字母那样（多半是那些意大利体或斜体字母）。大盒子的组成盒子可以相互重叠。盒子可以有负的宽度、深度或高度，尽管这种盒子不常用到。

你可以把一个盒子保存在盒子寄存器中并在稍后取回它。在使用盒子寄存器之前，你应当用 \newbox 命令（第 260 页）预留它并给它命名。请参阅“寄存器”（第 87 页）中的关于盒子寄存器的更多信息。

类别码. 字符的类别码 (category code) 决定它在 \TeX 中扮演的角色。例如， \TeX 赋予某个角色给字母，另一个角色给空格符，诸如此类。 \TeX 给读到的每个字符都附加一个代码。例如， \TeX 读到字母 ‘r’ 时附加类别码 11 (字母符) 给它。在简单地使用 \TeX 时你无需考虑到类别码，但当你试图达到特别效果时它们就变得很重要。

类别码只附加到 \TeX 从输入文件中读取的字符中。只要字符到达 \TeX 的食道（见“ \TeX 剖析”，第 49 页）并被解释完，它的类别码就不再改变。用 \char 命令（第 99 页）生成的字符没有类别码，因为 \char 命令是一个让 \TeX 生成特定字体中的某个特定字符的指令。例如，‘\’

² 如果盒子被 \moveleft 或 \moveright 左移或右移了， \TeX 放置时使用它移动之前的基准点。

³ 深度还受 \boxmaxdepth 参数（第 168 页）限制。

(通常的转义字符) 的 ASCII 码是 92。如果你键入 ‘\char92 grok’，却并不等同于 \grok。实际上它让 TeX 排版出 ‘cgrok’，其中 c 是当前字体的编码表中位置为 92 的字符。

你可以用 `\catcode` 命令 (第 267 页) 重新设定任何字符的类别码。改变了类别码就改变了各种字符所扮演的角色。例如，如果你键入 ‘\catcode‘@ = 11’，@ 符号的类别码就被设定为“字母符”。这样你就可以在控制序列名称中使用 ‘@’。

这里给出 plain TeX 定义的类别码列表，以及每个类别中的字符 (第 57 页有 ^^ 记法的解释)：

编码	意义	字符
0	转义符	\
1	编组开始符	{
2	编组结束符	}
3	数学环境符	\$
4	阵列制表符	&
5	行结束符	^^M ≡ ASCII <return>
6	宏参数符	#
7	上标符	^ 和 ^^K
8	下标符	_ 和 ^^A
9	可忽略符	^^@ ≡ ASCII <null>
10	空格符	□ 和 ^^I ≡ ASCII <horizontal tab>
11	字母符	A … Z 和 a … z
12	其他字符	(不属于其他类别的字符)
13	活动字符	~ 和 ^^L ≡ ASCII <form feed>
14	注释符	%
15	无效字符	^^? ≡ ASCII <delete>

除了第 11–13 类别，一个类别中的所有字符产生的效果都相同。例如，假设你键入：

```
\catcode‘[ = 1 \catcode‘] = 2
```

则左方括号和右方括号分别就变成编组开始符和编组结束符，即它们分别和左花括号和右花括号等价。在这些定义之下，‘[a b]’ 是一个有效的编组，‘[a b]’ 和 ‘{a b}’ 同样也是。

类别为 11 (字母符) 或 12 (其他字符) 的字符成为一个命令，它表示“用当前字体排版这个字符并放在一个盒子中”。字母符和“其他字符”的惟一区别是，字母符可以出现在控制词中而“其他字符”不能。

类别为 13 (活动字符) 的字符单独作为控制序列。如果 TeX 碰到一个未定义的活动字符，它将会报错。

如果 TeX 在输入中碰到一个无效字符（类别为 15），它也会报错。

字符 ‘^K’ 和 ‘^A’ 被包含在类别 7（上标符）和 8（下标符）中，即使它们的意义并不符合标准 ASCII 中的解释。这是因为某些键盘，尤其是 TeX 起源地哈佛大学的键盘，用上方向键和下方向键生成这两个字符。

在 TeX 分配类别码的方式中有个细微之处，如果没注意到它你就会犯错误。TeX 在初步扫描时有时候需要查看一个字符两次：第一次找到前面的控制序列等结构的结束位置，第二次才将该字符转换为记号。TeX 在第二次查看该字符时才会给它分配类别码。例如：

```
\def\foo{\catcode`\$ = 11 } % 将 \$ 变成字母符。
\foo\$ % 生成一个 ‘$’。
\foo\$ % 未定义的控制序列 ‘foo$’。
```

这段 TeX 代码在输出中生成 ‘\$’。在第二行，当 TeX 首次查看 ‘\$’ 时，它是在寻找控制序列名称的结束位置。由于 ‘\$’ 还不是字母符，它表示 ‘\foo’ 已结束。接下来，TeX 展开 ‘\foo’ 宏并将 ‘\$’ 的类别码改为 11（字母符）。然后 TeX 才“真正地”读取 ‘\$’。由于 ‘\$’ 已经是字母符，TeX 用当前字体生成包含 ‘\$’ 的盒子。当 TeX 看到第三行时，它将 ‘\$’ 视为字母符，因此认为它是控制序列名称的一部分。结果它抱怨 \foo\\$ 是一个未定义的控制序列。

即使当结束符号是行尾符时，TeX 也是这样处理的。举个例子，假设你用 \fum 宏激活行尾符。那么，如果 \fum 单独出现在第 ℓ 行中，TeX 将首先将第 ℓ 行的行尾符解释为 \fum 控制序列的结束，然后将该行尾符重新解释为活动字符。

字符。TeX 在两个方面用到字符（character）：作为读取的输入字符，以及作为排版的输出字符。TeX 将大多数输入字符转变成输出字符以显示它们。例如，它通常将输入字母 ‘h’ 转换为用当前字体排版的字母 ‘h’。但是，对于类似 ‘\$’ 这些有特别意义的字符，却不是这么回事。

TeX 通过读取输入文件（或终端）和展开宏得到输入字符。它们是 TeX 获取输入字符的仅有的途径。每个输入字符有一个编码，它表示该字符在 ASCII 码表的位置。例如，字母 ‘T’ 的 ASCII 编码为 84。

在读取字符时，TeX 给它附加一个类别码。类别码影响 TeX 读取字符时对该字符的解释。在读取宏定义时，TeX 判定（并记住）宏里面的字符的类别码。在用它的眼睛读取字符时（见“TeX 剖析”，第 49 页），TeX 执行一些“过滤”，比如将连续多个空格压缩为一个。在 *The TeXbook* 第 46–48 页中描述了过滤的细节。

ASCII “控制字符”的编码为 0–31 和 127–255。倘若你试图在终端中显示它们，在多数终端中它们将显示不出来或导致奇怪行为。即便如

此，有时候它们还是需要在 TeX 输入中出现，因此对这些字符 TeX 给出了一种特别的表示法。如果你键入 ‘`^c`’，其中 `c` 为任何字符，你将得到一个比 `c` 的 ASCII 码大 64 或小 64 的字符。这种表示法得到的编码不能超过 127，因此它是无歧义的。这种表示法的三个最常见例子是 ‘`^M`’ (ASCII `\langle return \rangle` 字符)，‘`^J`’ (ASCII `\langle line feed \rangle` 字符) 和 ‘`^I`’ (ASCII `\langle horizontal tab \rangle` 字符)。

TeX 还有另一种指明 ASCII 编码的表示法，它能够表示从 0 到 255 的所有字符。如果你键入 ‘`^xy`’，其中 `x` 和 `y` 是任何十六进制数字 ‘`0123456789abcdef`’，你就得到该编码对应的字符（这里只能使用小写字符。）TeX 优先选择“十六进制数字”这种解释，因此在类似 ‘`^a`’ 的字符后面不要跟着一个小写十六进制数字——这样将会得到错误的解释。在需要使用这种表示法时，你会发现有个 ASCII 编码表更方便些。

输出字符就是待排版的字符。生成输出字符的命令有这样的意义：“生成一个包含当前字体第 `n` 个字符的盒子”，其中 `n` 由该命令给出。TeX 将这些盒子和其他排版元素组合起来并在页面上排好，以生成你排版的文档。

类别码为 11 (字母) 或 12 (其他) 的输入字符，是一个生成对应输出字符的命令。另外，你也可以用 ‘`\char n`’ (第 99 页) 让 TeX 生成字符 `n`，其中 `n` 是介于 0 和 255 之间的数。命令 ‘`h`’，`\char 'h` 和 `\char104` 的结果相同 (`104` 是 ‘`h`’ 的 ASCII 编码。)

类. 字符所属的类 (class) 指明该字符在数学公式中的角色。字符的类记录在它的数学码中。例如，等号 ‘`=`’ 属于第 3 类 (关系符号)。TeX 利用字符类的知识确定数学公式的各部分的间隔的大小。例如，下面这个公式首先用正常方式显示，然后用随机修改各字符类的方式显示：

$$a + (b - a) = a \quad a + (b - a) = a$$

参见本书第 230 页中各种类的列表，以及 *The TeXbook* 第 154 页中各种类的含义。

命令. 命令 (command) 指导 TeX 执行某种动作。到达 TeX 的胃 (见 “TeX 剖析”，第 49 页) 的每个记号都作为一个命令，那些作为其他命令的参量 (见下面) 的记号除外。命令可以由控制序列，由活动字符，或者由普通字符产生。TeX 将普通字符视为命令这事看来有些奇怪，但实际上它是这样做的：当 TeX 看到一个普通字符时，它构造一个用当前字体排版该字符的盒子。

命令可以有参量。命令的参量为单个记号或一组记号，它们完成对命令该如何执行的说明。例如，命令 ‘`\vskip 1in`’ 告诉 TeX 竖直跳过 1 英寸。它有一个参量 ‘`1in`’ 包含三个记号。如果不指明跳过的距离，

对 `\vskip` 如何执行的说明就是不完整的。这些出现在命令参量中的记号不被看成命令。

各种类型的 TeX 命令的一些例子如下：

- 普通字符，比如 ‘W’ 指导 TeX 生成包含排版好的 ‘W’ 的盒子
- 字体设置命令，比如 `\bf` 开始粗体形式
- 重音符，比如 `\`` 生成一个如同 ‘é’ 的钝音符
- 特殊符号和连写，比如 `\P` (¶) 和 `\ae` (æ)
- 参数，比如 `\parskip` 说明 TeX 在段落间添加的粘连的大小
- 数学符号，比如 `\alpha` (α) 和 `\in` (\in)
- 数学运算符，比如 `\over` 生成一个分式

条件测试。条件测试 (conditional test) 是一个命令，它测试某个条件是否成立，并依此让 TeX 展开或忽略某些文本。条件测试的一般形式是：

`\if α ⟨true text⟩\else⟨false text⟩\fi`

或者：

`\if α ⟨true text⟩\fi`

其中 α 指定某个特定的测试。例如，`\ifvmode` 测试 TeX 是否位于竖直模式中。如果条件成立，TeX 展开 ⟨true text⟩。如果条件不成立，TeX 展开 ⟨false text⟩ (如果它存在)。条件测试在 TeX 的食道 (见 “TeX 剖析”，第 49 页) 中解释，因此在解释出的文本中，任何可展开记号都被展开了。在“条件测试”(第 250 页)中解释了各种条件测试。

控制序列。控制序列 (control sequence) 是 TeX 命令的名称。控制序列总是以转义符开头，即以反斜杠 (\) 开头。控制序列必为如下两种形式中的一种：

- 控制词是由转义符紧跟一个或多个字母组成的控制序列。控制词在 TeX 碰到非字母符时结束。例如，当 TeX 读取 ‘`\hfill\, the`’ 时，它看到六个记号：控制序列 ‘`\hfill`’、逗号、空格、‘t’、‘h’、‘e’。在 TeX 扫描控制序列时，‘`\hfill`’ 之后的空格结束该控制序列并被吸收掉。(另一方面，对于 ‘`\hfill, the`’，逗号不仅结束控制序列，而且自身也算作一个字符。)
- 控制符是由转义符紧跟单个非字母符组成的控制序列——非字母符也可以是空格符或者行尾符。控制符是自定界的，即 TeX 无需读取后面的字符就知道它在哪里结束。控制符之后的字符永远不会被吸收掉。

你可以参考第 12 页，其中有对控制序列后的空格的介绍。

\TeX 提供了许多预定义的控制序列。原始的控制序列内建于 \TeX 程序中，因此可以在各种 \TeX 形式中使用。其他预定义的控制序列由 plain \TeX (即本书描述的 \TeX 形式) 提供。

你可以用自己的控制序列扩充预定义的控制序列，只需用 \def 和 \let 等命令定义它们。本书第 12 章包含了一批实用的控制序列定义。此外，你的计算设备也可能提供一批本地开发的 \TeX 宏。

控制符. 控制符 (control symbol) 是由转义符紧跟单个非字母符组成的控制序列——非字母符也可以是空格符或者行尾符。

控制词. 控制词 (control word) 是由转义符紧跟一个或多个字母组成的控制序列。⁴ \TeX 忽略控制词之后的空格符或行尾符，仅仅用它们表示控制词的结束。

小数. 见“数”(第 81 页)。

定界符. 定界符 (delimiter) 是一个作为数学公式的可见边界的字符。 \TeX 可以根据子公式的竖直尺寸 (高度加深度) 调整定界符大小，这就是定界符的本质特性。然而，要让 \TeX 对定界符作调整，定界符必须出现在“定界环境”中，即作为 \left 、 \right 、 \overwithdelims 、 \atopwithdelims 或 \abovewithdelims (见 211, 214 页) 这些命令之一的参数。定界环境也包括在定界环境中使用参数的宏的任何参数。

举例说，左圆括号和右圆括号是定界符。如果你在定界环境中用圆括号围住一个公式， \TeX 将增加圆括号的大小以让它们框住包含该公式的盒子 (只要你所使用的字体有足够大的圆括号)。例如：

```
 $$ \text{\left( a } \text{\over b } \text{\right)} $$
```

得到下面的结果：

$$\left(\frac{a}{b}\right)$$

这里 \TeX 已经调整了圆括号的大小以框住分数。但是如果你改为这样写：

```
 $$\{a } \text{\over b}\}$$
```

将会得到下面的结果：

$$\left(\frac{a}{b}\right)$$

由于圆括号不在定界环境中， \TeX 不会增加它们的大小。

⁴ 其中“字母”的确切含义是指类别码为 11 的字符。

定界符必须成对出现：一个开定界符放在子公式左边，另一个闭定界符放在子公式右边。你可以用 `\bigl`, `\bigr` 及其他相关命令（第 222 页）直接选择较大号的定界符。⁵ 例如，要得到下面的陈列公式：

$$(f(x) - x)(f(y) - y)$$

其中外层的圆括号比内层的稍微大一点，你应该这样写：

```
$$\bigl( f(x) - x \bigr) \bigl( f(y) - y \bigr)$$
```

Plain TeX 中有 22 个定界符，用它们的自然尺寸显示如下：

`() [] { } [] < > / \ | \| \uparrow \downarrow \updownarrow \upuparrows \downdownarrows`

而用 plain TeX 提供的最大尺寸（`\Biggl`, `\Biggr` 形式）显示如下：

`() [] { } [] < > / \ | \| \uparrow \downarrow \updownarrow \upuparrows \downdownarrows`

这些定界符（‘(’，‘)’ 和 ‘/’ 除外）都出现在第 201–202 页的符号列表中。它们也都出现在 *The TeXbook* 第 146 页的列表中。

定界符可以属于任何类。对于用 `\bigl` 和 `\bigr` 等命令增加大小的定界符，它所属的类由该命令决定：l-命令属于“开符号”，r-命令属于“闭符号”，m-命令属于“关系符号”，g-命令比如 `\Big` 属于“普通符号”。

你可以用两种不同方法得到一个定界符：

- 1) 通过用 `\delcode` 命令（第 268 页）给一个字符赋予非负定界码（见下面），你可以将这个字符变成定界符。这样，如果你在定界环境中使用这个字符，它就和定界符一样。⁶
- 2) 你可以用 `\delimiter` 命令（第 215 页）直接生成一个定界符，这与你用 `\char` 命令生成一个普通字符或者用 `\mathchar` 命令生成一个数学字符类似。`\delimiter` 命令使用的定界码和 `\delcode` 表格项用到的一样，但前面多了一个数字用于标示所属的类。除了在宏定义中，其他地方很少用到 `\delimiter` 命令。

定界码 (delimiter code) 告诉 TeX 如何搜索合适的输出字符来表示一个定界符。这些搜索规则相当复杂（见 *The TeXbook* 第 156 和 442 页。要完全理解这些规则需要熟悉字体度量文件的结构，而这个主题不仅超出本书范围，也超出 *The TeXbook* 的范围。

本质上搜索过程是这样运行的。定界码给出了字体族和字体位置，从而指定了一个“小号”输出字符和一个“大号”输出字符（见

⁵ Plain TeX 定义了各种 `\big` 命令，其中用 `\left` 和 `\right` 提供定界环境。它通过构造所需尺寸的空公式来设定尺寸。

⁶ 在非定界环境中使用定界码非负的字符也是可以的。此时 TeX 将不会执行搜索，而仅仅以普通方式使用该字符（见 *The TeXbook* 第 156 页）。

第 268 页)。利用这些信息, TeX 能够找到(或者构造)定界符的越来越大的版本。TeX 首先尝试“小号”字体的“小号”字符的不同尺寸(从小到大), 然后是“大号”字体的“大号”字符的不同尺寸(同样是从小到大), 寻找一个高度加上深度足够大的字符。如果所找到的字符都不够大, 它就用最大那个。也有可能小号字符或大号字符, 或两者都没指定(即定界码的对应部分为零)。如果只指定了一个字符, TeX 使用那一个。如果两个字符都没指定, 它就将定界符替换为宽度为 `\nulldelimiterspace` 间隔。

缺陷. TeX 用缺陷(demerits)度量它分段为行时文本行的不良度(见“断行点”, 第 74 页)。文本行的缺陷同时受该行的劣度和它带有的惩罚影响。TeX 的目标是合理地分配各行文本, 以让段落的总缺陷最小, 其中总缺陷由各行的缺陷加起来。参见 *The TeXbook* 第 97–98 页以了解 TeX 分段为行的细节。TeX 选择分页点时不使用缺陷; 它使用的是被称为分页“代价”的类似概念。

深度. 盒子的深度(depth)是盒子在基线之下的距离。

尺寸. 尺寸(dimension)指定了一个距离, 即对空间的长度测量。你用尺寸指定事物的大小, 比如文本行的长度。英语国家的打印机习惯用点(point)和派卡(pica)来测量距离, 而欧洲大陆的打印机习惯用迪多点(didot points)和西塞罗(cicero)。你可以使用这些单位, 或者其他你更熟悉的单位, 比如英寸。TeX 能够识别的与字体无关的度量单位有如下这些:

<code>pt</code>	point	点	72.27 points = 1 inch
<code>pc</code>	pica	派卡	1 pica = 12 points
<code>bp</code>	big point	大点	72 big points = 1 inch
<code>in</code>	inch	英寸	
<code>cm</code>	centimeter	厘米	2.54 centimeters = 1 inch
<code>mm</code>	millimeter	毫米	10 millimeters = 1 centimeter
<code>dd</code>	didôt point	迪多点	1157 didôt points = 1238 points
<code>cc</code>	cicero	西塞罗	1 cicero = 12 didôt points
<code>sp</code>	scaled point	缩点	65536 scaled points = 1 point

还有两个与字体相关的度量单位: ‘`ex`’是一个纵向尺寸, 通常与字体中字母‘x’的高度相关; ‘`em`’是一个横向尺寸, 通常等于字体的大小, 与字体中字母‘M’的宽度相关。最后, TeX 还提供了三个“无限的”度量单位: ‘`fil`’, ‘`fill`’和‘`filll`’, 它们的强度的阶依次增大。

尺寸可以用一个因子，即乘数，后面加上度量单位来表示。因子可以是一个整数，或是一个带有小数点或小数逗号的小数。因子之前可以带有加号或减号，因此尺寸可正可负。即使数值为零，度量单位也必须写上。数值和度量单位之间可以有也可以没有空格。在 *The TeXbook* 第 270 页中有尺寸的精确定义。下面这些是尺寸的例子：

```
5.9in    0pt    -2,5 pc    2fil
```

其中最后一个尺寸表示一个一阶无限距离。

无限距离远远大于任何有限距离或者任何低阶的无限距离。如果你给 .001fil 加上 10in，你得到 .001fil；如果你给 -1fill 加上 2fil，你得到 -1fill，依此类推。只有在指定粘连的伸长量和收缩量时，才可以使用无限距离。

TeX 对文档中的所有尺寸都乘以一个放大率因子 $f/1000$ ，其中 f 是 `\mag` 参数的值。由于 `\mag` 的默认值为 1000，在一般情形下文档是照常排版的。你可以指定一个与放大率无关的最终文档尺寸，只要在度量单位前加上 ‘true’。例如，‘`\kern 8 true pt`’ 生成了一个与放大率无关的 8 点大小的紧排。

陈列公式. 陈列公式 (display math) 表示在单独一行显示的公式；TeX 在陈列公式上边和下边添加额外间隔，以隔开周围文本。陈列公式两边用 ‘\$\$’ 括起来。TeX 在陈列数学模式中读取陈列公式。

转义符. 转义符 (escape character) 引入一个控制序列。Plain TeX 的转义符是反斜杠 (\)。利用 `\catcode` 命令（第 267 页）重新定义 c_1 和 c_2 的类别码，你可以将转义符从 c_1 改为 c_2 。类似地你可以定义额外的转义符。如果你想排版包含字面转义符的材料，你必须或者 (a) 定义一个表示打印转义符的控制序列，或者 (b) 使用第 2 页介绍的方法，改变转义符的类别码以临时取消转义。下述定义：

```
\def\\{\$\\backslash\$}
```

是一种生成表示 ‘\’ 的控制序列的方法（用数学字体排版反斜杠）。

对于合成控制序列，比如那些由 `\string` 和 `\message` 生成的控制序列，你可以用 `\escapechar` 参数（第 240 页）指定其中的转义符用哪个字符表示。

族. 族 (family) 是用于数学模式的，由三个相关字体组成的集合。在数学模式之外，族没有任何作用。一个族的三个字体分别用于正常符号（文本尺寸），上标和下标（标号尺寸），以及次上标和次下标等（小标号尺寸）。例如，数字 ‘2’ 用这三个字体分别显示为 ‘2’、‘2’ 和 ‘ ${}_2$ ’（在

plain TeX 中)。通常你会将一个族的三个字体设定为相同字型且不同点数, 但没人阻止你将这三个字体也设定为不同字型, 或者将其中两个字体设定为完全一样。

TeX 提供多达十六个字体族, 它们用 0–15 编号。比如在 plain TeX 中, 第 0 族由用于文本尺寸的 10 点罗马字体, 用于标号尺寸的 7 点罗马字体, 以及用于小标号尺寸的 5 点罗马字体组成。Plain TeX 还将第 1 族定义为由数学意大利字体组成, 并将第 2 族和第 3 族分别保留给特殊符号和数学扩展符号。⁷ ⁸ 如果你需要自己定义字体族, 你应该使用 `\newfam` 命令 (第 260 页) 获取未使用的族编号, 再用 `\textfont`、`\scriptfont` 和 `\scriptscriptfont` 命令 (第 221 页) 分别设定该族的各个字体。

文件. 文件 (file) 是 TeX 解释或者创建的信息流。文件由监督 TeX 运行的操作系统管理。TeX 在四种不同的背景下用到文件:

- 1) “源码文件”是 TeX “眼睛”读取 (见“TeX 剖析”, 第 49 页), 并根据它的一般法则解释的文件。你的主要输入文件——即调用 TeX 时在 ‘**’ 之后或者在命令行中指定的文件——是一个源码文件, 你用 `\input` 命令 (第 263 页) 请求的任何文件同样也是。
- 2) “结果文件”是包含 TeX 运行结果的文件。TeX 运行后产生两个结果文件: `.dvi` 文件和日志文件。`.dvi` 文件包含打印文档所需的信息, 而日志文件包含运行记录, 包括 TeX 生成的任何错误信息。若主要输入文件的名称是 `screed.tex`, 则 `.dvi` 文件和日志文件的名称分别是 `screed.dvi` 和 `screed.log`。⁹
- 3) 要用 `\read` 命令 (第 264 页) 读取一个文件, 你需要将这个文件和一个输入流关联起来。你最多可以有 16 个同时活动的输入流, 它们以 0–15 编号。`\read` 命令读取一行并把它作为指定控制序列的值, 因此用 `\read` 读取和用 `\input` 读取是大不相同的 (后者读入整个文件)。TeX 将编号不在 0 和 15 之间的输入流视为终端, 因此, 比如说 ‘`\read16`’, 将读取你在终端中键入的下一行文本。
- 4) 要用 `\write` 命令 (第 265 页) 写入一个文件, 你需要将这个文件和一个输出流关联起来。你最多可以有 16 个同时活动的输出流, 它们以 0–15 编号。输入流和输出流是相互独立的。任何送到编号为负数的输出流的东西, 将被写到日志文件中; 而任何送到编号大于 15 的输出流的东西, 将同时被写到日志文件和终端中。因此, 比如说 ‘`\write16`’, 将在终端中写上一行文本并将它送到日志文件中。

⁷ 第 2 族和第 3 族的特殊之处在于, 它们的字体度量文件必须包含数学间隔参数。

⁸ 译注: 第 4–7 族分别定义为 `\itfam`、`\slfam`、`\bffam` 和 `\ttfam`。

⁹ 这是通常的约定, 但特定的 TeX 实现可以自由地修改它。

在使用流文件之前，你必须先打开它。输入流文件用 `\openin` 命令（第 264 页）打开，而输出流文件用 `\openout` 命令（第 265 页）打开。为保持整洁，你应该在完成后关闭流文件；然而即使你不这么做， \TeX 在运行结束时也会帮你关闭它。用于关闭流文件的两个命令分别是 `\closein`（第 264 页）和 `\closeout`（第 265 页）。在完成后关闭流文件的优点是，你可以把它重新用到另一个文件去。在读取一长串文件时，这样做是很有必要的。

虽然你可以自己给输入流和输出流编号，但最好还是用 `\newread` 和 `\newwrite`（第 260 页）命令处理流编号。一个特定文件可以关联到多个流，但除非所有流都是输入流，否则你将会得到（也许是原因不明的）垃圾。当你想把同个输入文件用于两个不同目的时，一个输入文件关联多个流就很有用。

一般地， \TeX 会推迟输出流的打开、写入和关闭操作，直到它用 `\shipout` 送出一个页面（详情可见 *The \TeXbook* 第 227 页。甚至用 `\write` 输出消息到终端时也是这样的。然而，通过在操作命令之前加上 `\immediate`（第 266 页），你可以让 \TeX 立即执行输出流操作。比如：

```
\immediate\write16{Do not pass GO! Do not collect $200!}
```

文件名。文件名（file name）是文件在监督 \TeX 运行的操作系统中的名称。文件名的语法并不遵循 \TeX 语法的一般规则，而且实际上在不同的 \TeX 实现中它们各不相同。特别地，大多数 \TeX 实现都认为文件名在空格符或行尾符处结束。因此 \TeX 有可能曲解 ‘{\input chapter2}’，将右括号视为名称的一部分。一般来说，你应该在文件名后面添加一个空格符或行尾符，就像 ‘{\input chapter2\ }’ 这样。

字体。在 \TeX 中，字体（font）是由不超过 256 个输出字符组成的集合，这些字符通常有相同的设计、样式（罗马体、意大利体、粗体、窄体等）和点数。¹⁰ \TeX 自带的计算机现代字体一般只有 128 个字符。在本书最后面的书尾页中描述了排版本书所用的字体。

举个例子，这里是 10 点 Palatino 罗马字体的字母表：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz
```

而这里是 12 点计算机现代粗体扩展字体的字母表：

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ  
abcdefghijklmnopqrstuvwxyz
```

¹⁰ Plain \TeX 用于构造数学符号的特殊字体包含点数不同的字符。在排版标识（logo）等应用中也经常使用其他的特殊字体。

字体中的字符都是编好号的。对于在 ASCII 字符表中存在的字符，它们的编号一般与其 ASCII 编码一致。每个字体的编码表指明该字体的第 n 个字符是怎样的。有些字体，比如那些用于数学符号的字体，根本不包含任何字母字符。你可以用 ‘\char n ’（第 99 页）生成包含第 n 个字符的盒子，该字符用当前字体排版。

要在文档中使用某个字体，你必须先用控制序列给它命名再载入它。此后，在需要用到它的地方，你就可以键入该控制序列来选择它。Plain TeX 提供了若干已经命名和载入的字体。

你可以用像 ‘\font\twelvebf=cmbx12’ 这样的命令在同一个操作中命名并载入字体。这里的 ‘\twelvebf’ 是用于命名该字体的控制序列，而 ‘cmbx12’ 表示计算机文件系统中的字体度量文件 `cmbx12.tfm`。现在你可以通过键入 ‘\twelvebf’ 开始使用这个字体。在此之后，这个字体将一直生效，直到 (a) 你选择了另一个字体，或者 (b) 你结束了编组，如果在开始字体时就位于这个编组中。例如下面的输入：

```
{\twelvebf white rabbits like carrots}
```

将使得 `cmbx12` 字体只对文本 ‘white rabbits like carrots’ 生效。

在 TeX 中你可以使用与计算机现代不同的字体（见第 35 页的例子以及页眉）。这些字体的文件需要安装在计算机文件系统的某个 TeX 能找到的位置中。对每个字体，TeX 及相关程序一般需要两个文件：一个给出各字符的度量（比如 `cmbx12.tfm`），另一个给出各字符的形状（比如 `cmbx12.pk`）。TeX 本身仅用到度量文件。另一个程序，设备驱动程序，把 TeX 生成的 `.dvi` 文件转换为能被打印机或其他输出设备处理的格式。设备驱动程序要用到形状文件（如果该文件存在）。

字体度量文件包含 TeX 对每个排版字符所分配空间的信息。即它包含各个字符的尺寸，对相邻字符的连字和紧排，等等。字体度量文件并不包含关于其字符的形状的任何信息。

形状（像素）文件可以有多种格式。文件的扩展名（点号之后部分）告诉驱动文件该形状文件的格式。例如，`cmbx12` 字体的形状文件可以是压缩格式的 `cmbx12.pk`，也可以是一般格式的 `cmbx12.gf`。对存在于输出设备中的字体，其形状文件也许不是必需的。

页脚. 页脚 (footer) 是放在每个页面底部，位于正文之下的素材。在 plain TeX 中默认页脚是居中的页码。通常页脚只包含一行文本，你可以通过给 `\footline`（第 146 页）指定一个记号列来设定页脚。请参阅第 292 页中介绍的生成多行页脚的方法。

格式文件. 格式文件 (format file) 是一个包含 TeX 的内存映像的文件，该文件以一种能够快速重新载入的形式存储。格式文件可以用 `\dump` 命

令（第 280 页）建立。映像文件包含转储发生时存在的（字体和宏等）定义的完整记录。然后利用 `virtex`, `TeX` 的一种特别的“原生”形式，你可以快速地重新载入该格式文件，并从 `TeX` 转储时所在的状态中继续。相对于包含相同信息的普通输入文件，格式文件的优点是 `TeX` 能够更快地载入它。

格式文件仅可以用 `initex` 建立，`initex` 是专用于此目的的另一种 `TeX` 形式。除了内建于 `TeX` 本身的原始命令，`virtex` 和 `initex` 均不包含其他任何命令。

`TeX` 的预载入形式是一种已载入某个格式文件，并准备接收用户输入的形式。称为 `tex` 的 `TeX` 形式通常都预载入了 plain `TeX` 定义。（Plain `TeX` 通常还以另外两种形式出现：作为格式文件与作为 `TeX` 源码文件。在某些环境中，`tex` 等价于调用 `virtex` 然后载入 `plain`。）仅用 `TeX` 本身无法建立 `TeX` 的预载入形式，它需要一个特别的程序。

全局的。一个全局的（global）定义在文档结束前始终有效，即使它出现在一个编组里面（除非它被其他定义覆盖）。即全局定义不受编组范围的影响。要让任何定义成为全局的，你可以在它前面加上 `\global` 命令（第 243 页），除非 `\globaldefs`（第 243 页）为负值。

还有种特殊方法将一个宏定义为全局的。通常你用 `\def` 命令或者 `\edef` 命令（第 245 页）来定义一个宏。如果你将 `\def` 和 `\edef` 分别换为 `\gdef` 和 `\xdef`，这个宏定义就会是全局的。也就是说，‘`\gdef`’等价于‘`\global\def`’，而‘`\xdef`’等价于‘`\global\edef`’。

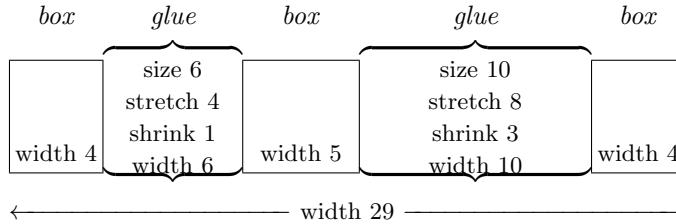
粘连。粘连（glue）是可以伸长或者收缩的空白间距。粘连给 `TeX` 提供了生成漂亮文档所需的灵活性。粘连有两种类型：水平粘连和竖直粘连。水平粘连出现在水平列表中，而竖直粘连出现在竖直列表中。你可以隐式地生成一个粘连项，比如用单词间空格，或者显式地生成一个粘连项，比如用 `\hskip` 命令。在排版文档时 `TeX` 本身也生成很多粘连项。这里我们只描述水平粘连——竖直粘连与之类似。

在 `TeX` 将一系列盒子和粘连组装成一个较大单元时，它调整各个粘连的尺寸以适合较大单元的空间要求。例如，为了让页面的右边距保持一致，`TeX` 调整了各行的水平粘连。类似地，为了让各页的下边距保持相同，它调整了各个文本块比如段落和陈列公式之间的粘连（这种修改是最不可能引人注意的）。

每个粘连项有它的自然间距——即它“所期望的”尺寸。粘连还有另外两个属性：它的伸长量（stretch）和它的收缩量（shrink）。你可以用 `\hskip` 命令（第 159 页）生成特定大小的水平粘连。命令 `\hskip 6pt plus 2pt minus 3pt` 生成一个水平粘连，其自然尺寸为 6 点，伸长量

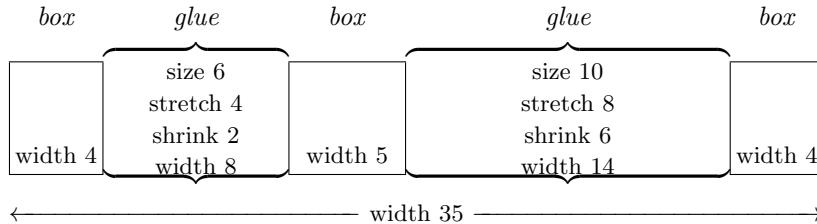
为 2 点，收缩量为 3 点。类似地，你可以用 `\vskip` 命令（第 159 页）生成特定大小的竖直粘连。

要理解什么是伸长量和收缩量，最好方法是看粘连的一个实际例子。假设你要用三个盒子和两个粘连项构造一个水平盒子项目，如下图：



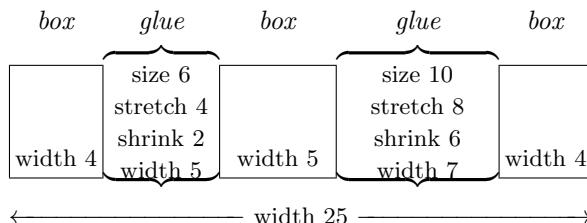
这里的度量单位可以是点，毫米，或其他任何单位。如果水平盒子可以呈现它的自然宽度，那么盒子内的每个粘连项也呈现它的自然宽度。从而水平盒子的总宽度就是它个各部分宽度之和，即 29 点。

接下来，假设这个水平盒子需要比 29 单位还宽，比方说 35 单位。这是可能出现的，比如水平盒子可能需要占据行宽为 35 单位的整行。由于 TeX 不可能改变盒子的宽度，它通过增加粘连项的宽度生成所需的额外间距。现在的图形看起来是这样的：



各粘连项增加的宽度并不相等；增加的宽度与它们的伸长量成比例。由于第二个粘连项的伸长量是第一个的两倍，它的宽度增加了四个单位，而后者只增加了两个单位。粘连允许任意伸长，尽管 TeX 有时不愿意超出定义中给出的伸长量。

最后，假设这个水平盒子需要比 29 单位还窄，比如说 25 单位。此时 TeX 将减少粘连项的宽度。图形看起来是这样的：



各粘连项减少的宽度于它们的收缩量成比例。第一个粘连项的宽度减少了一个单位，而第二个粘连项减少了三个单位。粘连收缩时不能超出定义中给出的收缩量，而伸长时却不受限制。从这个重要意义上说，收缩和伸长有不同的表现。

有一个不错的经验法则可用于设定粘连，即取它的自然尺寸为文档最美观时的间距大小，取它的伸长量为文档开始变难看之前 TeX 能添加的最大间距，取它的收缩量为文档开始变难看之前 TeX 能去掉的最大间距。你也许需要通过试验确定这些值。

以 ‘fil’，‘fill’，或 ‘filll’ 为单位设定粘连的伸长量，就可以生成可无限伸长的粘连。以 ‘fill’ 为单位的粘连相比以 ‘fil’ 为单位的粘连有更加无限的伸长能力，以 ‘filll’ 为单位的粘连相比以 ‘fill’ 为单位的粘连有更加无限的伸长能力。你很少需要使用 ‘filll’ 粘连。伸长量为 `2fil` 的粘连，其伸长能力是伸长量为 `1fil` 的粘连的两倍，对其他类型的可无限伸长粘连情形类似。

在 TeX 给各个粘连项分配额外间距时，如果存在可无限伸长的粘连，它们将分到全部间距。可无限伸长的粘连在设定文本左对齐，右对齐或者居中时特别有用：

- 要让文本左对齐，可以在它右边放上一个可无限伸长的水平粘连。这个粘连将占用该行所有可用的多余间距。你可以用 `\leftline` 命令（第 108 页）或 `\raggedright` 命令（第 116 页）达到此目的。
- 要让文本右对齐，可以在它左边放上一个可无限伸长的水平粘连。如同上面，这个粘连将占用该行所有可用的多余间距。你可以用 `\rightline` 命令（第 108 页）达到此目的。
- 要设定居中文本，可以在它两边各放上一个同样的可无限伸长的水平粘连。这两个粘连项将平分该行的多余间距。你可以用 `\centerline` 命令（第 108 页）达到此目的。

类似地，你也可以指定可无限收缩的粘连。可无限收缩粘连可以作为负间距。注意 `fil` 等只可用于指定粘连的伸长量和收缩量——它们不能用于指定自然尺寸。

编组. 编组 (group) 是文稿中 TeX 视为一个单元的一部分。要表示一个编组，你可以用括号 ‘{’ 和 ‘}’（或者类别码合适的其他字符）将它括起来。

编组的重要特性在于，当一个编组结束时，在其中所作的任何非全局的定义或赋值都消失了。假如你这样写：

```
Please don't pour {\it any} more tea into my hat.
```

\it 控制序列让 TeX 设定 ‘any’ 为意大利体，但它不会影响其他文本。再举一个例子，如果你在编组中用 \hsize 命令（第 113 页）改变行宽，在 TeX 完成这个编组后，行宽将恢复为之前的取值。

编组也可以作为一种控制间距的方法。举个例子，如果你这样写：

```
\TeX for the Impatient and the Outpatient too.
```

你将会得到下列结果：

TeXfor the Impatient and the Outpatient too.

这是由于控制序列 \TeX (它生成 TeX 标识) 吸收了其后的空格。你所需要的也许是这样：

TeX for the Impatient and the Outpatient too.

要得到这种结果，其中一种方法就是将 ‘\TeX’ 括到一个编组里面：

```
{\TeX} for the Impatient and the Outpatient too.
```

其中的右花括号阻止了控制序列吸收空格。

水平盒子. 水平盒子 (hbox, horizontal box) 是 TeX 从左到右逐个放置水平列表的项目而构造出来的盒子。水平盒子，作为一个单元，既不是本质上水平的也不是本质上竖直的，即它可以出现在水平列表或竖直列表中。你可以用 \hbox 命令（第 164 页）构造水平盒子。

页眉. 页眉 (header) 是放在每个页面顶部，位于正文之上的素材。一个简单报告的页眉可能由左侧的标题和右侧的‘第 n 页’组成。通常页眉只包含一行文本，你可以通过给 \headline (第 146 页) 指定一个记号来设定页眉。在 plain TeX 中默认页眉是空白的。你也可以生成多行页眉；请参阅第 292 页中介绍的方法。

高度. 盒子的高度 (height) 是盒子在基线之上的距离。

水平列表. 水平列表 (horizontal list) 是 TeX 位于某种水平模式时 (即组装段落或水平盒子时) 生成的一列项目。见下面的“水平模式”。

水平模式. 在组装段落或者水平盒子时，TeX 位于两种水平模式 (horizontal mode) 之一：普通水平模式用于组装段落，而受限水平模式用于组装水平盒子。只要 TeX 位于水平模式中，它的胃 (见“TeX 剖析”，第 49 页) 就在构造项目 (盒子、粘连、惩罚等) 的水平列表。TeX 逐个排版列表中的项目，从左到右。

水平列表不能包含任何用本质上的竖直命令，比如 `\vskip`，生成的项目。

- 如果 TeX 正在普通水平模式中组装水平列表，并碰到一个本质上的竖直命令，它将结束当前段落并进入竖直模式。
- 如果 TeX 正在受限水平模式中组装水平列表，并碰到一个本质上的竖直命令，它将报错。

这两个命令你也许会认为是本质上水平的，实际上却是本质上竖直的：`\halign`（第 184 页）和 `\hrule`（第 178 页）。在 *The TeXbook* 第 286 页中列出了所有本质上的竖直命令。

对于受限水平模式，你要知道这个微小但是重要的特性：你无法从受限水平模式进入普通水平模式。在实践上，这意味着在组装水平盒子时 TeX 无法处理类似段落的文本，即需要断行的文本。在水平盒子内部，通过把类似段落的文本放入一个竖直盒子中，你就可以绕过此限制。其他类似方法也可行，比如将一个水平阵列放在水平盒子内部。

连字。在处理文档时，TeX 自动将单词连字化（hyphenation）。TeX 并不急于插入连字符，它更喜欢通过调整单词间隔找到合适断行点，从而将单词用一行移动到另一行。TeX 足够聪明，它可以理解已经出现在单词中的连字符。

你可以用如下几种方式控制 TeX 的连字：

- 你可以用 `\-` 命令（第 127 页）插入自定连字符，以允许 TeX 在特定位置连字化。
- 你可以用 `\hyphenation` 命令（第 128 页）告诉 TeX 在整个文档中如何对某个单词连字化。
- 你可以将单词放入一个水平盒子中，以阻止 TeX 将它连字化。
- 你可以设定一些惩罚值，比如 `\hyphenpenalty`（第 126 页）。

如果单词中包含显式或自定连字符，TeX 将不会在其他位置断开。

输入流。见“文件”（第 63 页）。

插入项。插入项（insertion）是一个竖直列表，它包含即将在页面建造完成后插入到该页面的素材。¹¹ 脚注和图片就是插入项的例子。在 plain

¹¹ TeX 本身并不插入这些素材——它仅提供这些素材给输出例行程序，输出例行程序负责将它们移动到排版出的页面中。`\insert` 命令（第 150 页）仅有的直接效果是，改变了 TeX 的分页计算公式以给插入素材留出空间。稍后，当 TeX 实际分出了页面时，它将插入素材分为两组：当前页能够放下的和无法放下的。当前页能够放下的素材被放入盒子寄存器中，每个插入项放入一个寄存器，而无法放下的素材则保留到下一页中。此过程允许 TeX 将长脚注分到连续多个页面中。

\TeX 中，用于创建插入项的命令有 \footnote , \topinsert , \midinsert 和 \pageinsert ，以及原始命令 \insert 本身（见 148–151 页）。 \TeX 插入项的处理机制是相当复杂的；你可以在 *The $\text{\TeX}book$* 第 122–125 页中看到详细介绍。

行间粘连。行间粘连（interline glue）是 \TeX 在竖直列表的每个盒子（第一个盒子除外）之前插入的粘连。通常指定行间粘连以让各盒子基线间保持相同距离。行间粘连的值由 \baselineskip , \lineskip 和 \lineskiplimit 这三个参数合起来确定（第 136 页）。

项目。项目（item）这个术语常用来表示水平、竖直或数学列表（即 \TeX 在水平、竖直或数学模式中建造的项目列表）的一个组成元素。

对齐文本。对齐文本（justified text）是两边对齐排版的文本。而非对齐文本是单边或两边“不对齐”排版的文本。老式打印机排版的文档几乎总是左对齐的。虽然 \TeX 默认生成两边对齐的文档，如果你需要也可以生成左对齐或右对齐的文档（或一些文本行）。你也可以让 \TeX 将一些文本行居中，从而让两边都不对齐。用 \leftskip , \rightskip 和 \raggedright 命令（第 115, 116 页）可以达到这些目的。

在生成对齐文本时， \TeX 通常需要伸缩各行的粘连以让两边对齐。而在生成非对齐文本时， \TeX 通常让各行的粘连保持自然尺寸。很多排印工更喜欢非对齐文本，因为它的字间距更加一致。

紧排。紧排（kern）表示对竖直或水平列中的项目间的距离作一定调整。紧排可正可负。将正紧排放入两个项目间，将以该紧排的大小拉开两个项目。而将负紧排放入两个项目间，将以该紧排的大小拉近两个项目。例如，下列文本：

```
11\quad 1\kern1pt 1\quad 1\kern-.75pt 1
```

将生成类似下面的几个字符对：

```
11 11 11
```

在竖直模式中可以用紧排改变特定两行的间距。

大小为 d 的紧排与大小为 d 且无伸缩的粘连非常相似。紧排和粘连两者都在相邻项目间插入或删除一定间距。本质区别在于，若两盒子间只有紧排， \TeX 将认为它们是合在一起的。¹² 这就是说， \TeX 不会在紧排处断行或者分页，除非其后跟着一个粘连。当你决定用紧排或粘连实现某种目的时，务必记得这两者的区别。

¹² 译注：本书将“kern”翻译为“紧排”正由于这个原因，“紧”应该理解为“坚固”。

TEX 在某些相邻的字符对之间自动插入紧排，从而改变这些字符的间距，使得排版出的文档更加美观。例如，在计算机现代字体的 10 点罗马字体中，包含一个字符对 ‘To’ 的紧排，将字母 ‘o’ 的左边缘拉到字母 ‘T’ 下边。没有这个紧排，你将得到 “Top” 而不是 “Top” ——两者的差别是微小的但却是明显的。**TEX** 用某个字体排版文本时自动插入的紧排的位置和大小，是在该字体的度量文件 (**.tfm** 文件) 中指定的。

指引线。利用指引线 (leaders)，你可以用某个模式的复本填充空间，比如在目录中的用重复的点填充标题和页码之间的空间。指引体 (leader) 是模式的单个复本。要指定一个指引线，你需要给出三方面的信息：

- 1) 单个指引体是怎样的
 - 2) 要填充的空间有多大
 - 3) 模式的副本在空间内应该如何组织

`\TeX` 有三个用于指定指引线的命令: `\leaders`, `\cleaders` 和 `\xleaders` (第 179 页)。各命令的参量指定了指引体。命令之后必须是一个粘连; 粘连的大小指定了要填充空间的大小。命令的选择决定了指引体在空间内如何组织。

这里有个例子展示了 \leaders 的工作方式：

```
\def\dotting{\leaders\hbox to 1em{\hfil.\hfil}\hfil}
\line{The Political Process\dotting 18}
\line{Bail Bonds\dotting 26}
```

其中我们将指引线及相关粘连放在一个宏定义中，以方便我们在两个地方使用它。这个例子的输出结果是：

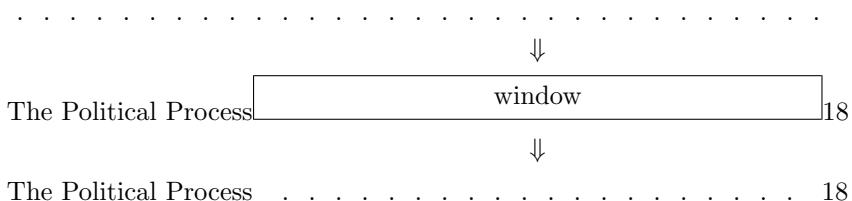
The Political Process	18
Bail Bonds	26

\leaders 之后的水平盒子指定了指引体，即包含居中点的 1em 宽的水平盒子。空间将用这个盒子的副本填充，实际上是用中心相隔 1em 的点填充。随后的 \hfil（宏定义末尾那个）是一个粘连，它给出所要填充空间的大小。在这个例子中，它就是填满文本行所需的空间。我们选择 \leaders 而不是 \cleaders 或 \xleaders，是为了让不同行的点相互对齐。

一般地，所要填充的空间就作为包含指引体的多个复本的窗体。**TeX**插入尽可能多的复本，但通常会有些空间剩下——或者是由于指引体在窗体内的位置问题，或者是由于窗体的宽度不是指引体宽度的整数倍。这三个命令的差异在于，在窗体内如何安排指引体和分配剩余空间：

- 对于 `\leaders`, TeX 首先生成一行指引体复本。然后该行复本的起始位置与包含 `\leaders` 命令的最内侧盒子 B 的左侧对齐。在上面

的两行例子中，*B* 就是由 `\line` 生成的盒子。整个落入窗体的指引体将被放入 *B* 中，而左右两边剩下的空间将空着。图形如下：



这个过程确保在上面的两行例子中，两行中的各个点是竖直对齐的（由于用 `\line` 生成的水平盒子的基准点是竖直对齐的）。

- 对于 `\cleaders`，*T_EX* 平分窗体两边的剩余空间以让指引线在窗体中居中。剩余空间总是小于单个指引体的宽度。
- 对于 `\xleaders`，*T_EX* 将剩余空间在窗体内均匀分布。也就是说，如果剩余空间是 *w*，指引体被重复 *n* 次，*T_EX* 将在相邻指引体中间及指引线两端填充宽度为 *w/(n + 1)* 的空间。其效果通常是将指引体散开一点。和 `\cleaders` 类似，`\xleaders` 的剩余空间也总是小于单个指引体的宽度。

到目前为止，我们都假定指引线由横向排列的水平盒子组成的。实际上指引线还有两个可能的变种：

- 1) 你可以用标线代替水平盒子作为指引体。*T_EX* 将让标线尽可能宽以让它填满该粘连（从而这三个命令是等价的）。
- 2) 你可以不在水平列表而在竖直列表中使用它，以在页面中生成向下排列的竖直指引线。在这种情形中指引线后面需要给出竖直粘连。

请参阅 *The T_EXbook* 第 223–225 页，其中给出了 *T_EX* 用于排版指引线的精确规则。

连写. 连写 (ligature) 是排版的文档中替换特定的相邻字符串的单个字符。例如，在高质量的排版系统中，单词 ‘*office*’ 将被排版为 “*office*”，而不是 “*office*”。连写信息已经嵌入在你使用的字体中，因此什么也不用做你就可以让 *T_EX* 生成它们。（通过将输入写成 ‘*of{f}ice*’ 可以取消 “*office*” 中的连写，正如我们刚才所做的。）利用此连写机制，对单词的开头或者末尾字母，*T_EX* 也可以使用与它在单词中间时不同的方式排版。用 `\noboundary` 命令（第 101 页）你可以取消这种效果（如果你曾经碰到它）。

有时候你会需要欧洲语言的连写。除非你使用一个针对该语言设计的字体，否则 *T_EX* 不会自动生成这些连写。有一些连写，比如 ‘*Æ*’，可以用命令得到（见“欧洲语言字母和连写”，第 97 页）。

断行点。断行点 (line break) 是文档中 TeX 结束一行并开始新行的位置。在处理你的文档时, TeX 将一个段落的内容收集到一个水平列表中。收集完整个段落之后, 它分析该列表以找到它认为最佳的断行点。TeX 附加“缺陷”值给各种形态的丑陋断行点——单词间隔太大或太小的行, 连续以连字符结尾的行, 诸如此类。然后它选择让总缺陷值最小的断行点。请参阅 *The TeXbook* 第 96–101 页中的 TeX 的断行规则的完整说明。

你可以用下面几种方式控制 TeX 对断行点的选择:

- 你可以在 TeX 形成段落时构建的水平列表的某处插入惩罚项 (第 122 页)。正惩罚值阻碍 TeX 在该处断行, 而负惩罚值——即奖励值——鼓励 TeX 在该处断行。大于或等于 10000 的惩罚只阻止断行, 而小于或等于 -10000 强制断行。用 `\break` 和 `\nobreak` 命令 (第 121, 121 页) 可以达到相同效果。
- 你可以用 `\-` 命令 (第 127 页) 插入自定连字符, 以允许 TeX 在特定位置连字化。你也可以控制 TeX 在你的文档中如何连字化 (见“连字”, 第 70 页)。
- 你可以在两个单词间用 `\slash` (第 123 页) 插入斜线符号 (/), 并允许 TeX 在斜线后断行, 比如 ‘furlongs\slash fortnight’。
- 你可以通过在两个特定单词间插入带子 (~), 以禁止 TeX 在该处断行。
- 你可以给 TeX 的断行参数赋予不同的值, 以调整断行时附加的惩罚值。
- 你可以将一个单词或一列单词放在一个水平盒子中, 从而阻止 TeX 盒子里面的任何位置断行。

知道在什么地方 TeX 可以断行也是有益的:

- 在粘连处, 只要:
 - 1) 粘连之前是下列项目之一: 盒子、自定项目 (比如自定连字符)、公式结尾、小玩意或者用 `\mark` 或 `\vadjust` 或 `\insert` 生成的竖直素材
 - 2) 粘连不是数学公式的一部分
- 在粘连处断行时, TeX 让断行点位于粘连的左边缘, 并忽略粘连的其他部分。
- 在紧排处, 只要其后直接跟着粘连, 且该紧排不在数学公式中
- 在数学公式结尾处, 只要其后直接跟着粘连
- 在惩罚项处, 即使该惩罚项在数学公式中
- 在自定可断点处

在断行时, TeX 丢弃断点之后的任意多个粘连、紧排和惩罚项。如果这些项目之后还是一个数学公式起点, 它同样丢弃公式起点生成的任何紧排。

列表. 列表 (list) 是一列用于组成水平列表、竖直列表或数学公式的项目 (盒子、粘连、紧排等)。请参阅 “水平列表” (第 69 页), “竖直列表” (第 92 页)。

日志文件. 见 “文件” (第 63 页)。

宏. 宏 (macro) 是一个定义, 它给出了 TeX 输入文本的一种模式的名称。¹³ 这个名称可以是一个控制序列或一个活动字符。其中的模式称为“替换文本”。用于定义宏的原始命令是 `\def` 控制序列。

举个简单例子, 假设你有一个文档, 其中多次出现 ‘ $\cos \theta + i \sin \theta$ ’ 这串字符。为避免每次都写一遍, 你可以给它定义一个宏:

```
\def\arctheta{\cos \theta + i \sin \theta}
```

这样每次需要这串字符时, 用 ‘`\arctheta`’ 命令 “调用” 这个宏就可以得到它。比如, ‘`$e^{\arctheta}`’ 将给出 ‘ $e^{\cos \theta + i \sin \theta}$ ’。

但宏的真正威力在于它可以带有参数。在调用一个带有参数的宏时, 你给出用于替换这些参数的参量。例如, 假设你这样写:

```
\def\arc#1{\cos #1 + i \sin #1}
```

记号 #1 表示宏的第一个参数, 在这个例子中宏只有一个参数。现在你可以用 ‘`\arc {2t}`’ 得到 ‘ $\cos 2t + i \sin 2t$ ’。

更一般地, 一个宏最多可以有九个参数, 在宏定义中它们分别用 ‘#1’, ‘#2’ 等表示。TeX 提供两种类型的宏参数: 定界参数和非定界参数。简单来说, 定界参数对应一个由特定记号序列 (即定界子) 定界 (即结束) 的参量, 而非定界参数对应一个无需定界子结束的参量。我们先解释只有非定界参数时宏如何运作, 然后解释包含定界参数时宏如何运作。

当宏只有非定界参数时, 这些参数必须依次出现在宏定义中, 在各个参数之间, 以及最后一个参数和替换文本前的左花括号之间, 不能有任何东西。对这种宏的调用由该宏的名称, 以及后面跟随的各个参量组成。每个参量对应一个参数, 参量可以是

- 不为左或右花括号的单个记号, 或者

¹³ 更准确地说, 该定义给出了一个记号序列的名称。

- 一个包含在配对的左右花括号之间的记号序列。¹⁴

TEX 在食道中展开它所遇到的宏（见“TEX 剖析”，第 49 页），方法是将替换文本中的每个参数替换为它所对应的参量。宏展开后所得的文本中可能还包含其他宏。当 TEX 遇到这种嵌套的宏调用时，它将立即展开此宏调用，完成后再查看后续内容。¹⁵ 当 TEX 的食道得到无法再展开的原始命令时，TEX 就将该命令送入 TEX 的胃。宏展开的先后顺序有时候是至关重要的，为帮助你理解这个问题，下面我们将给出一个例子。

假设你给 TEX 提供如下输入：

```
\def\ a#1#2{\b#2#1\kern 2pt #1}
\def\b{bb}
\def\c{\char49 cc}
\def\d{dd}
\ a\c\{e\}\d} % Call on \a.
```

则与 #1 对应的参量为 `\c`，与 #2 对应的参量为 `e\c`。TEX 按照如下步骤展开宏调用：

```
\b e\c\kern 2pt \c
bbe\c\kern 2pt \c
\c\kern 2pt \c ('b', 'b', 'e' 送到胃里)
dd\c\kern 2pt \c
\c\kern 2pt \c ('d', 'd' 送到胃里)
\char49 cc\kern 2pt \c
\c ('char', '4', '9', 'c', 'c', '\kern', '2', 'p', 't' 送到胃里)
\char49 cc
('char49', 'c', 'c' 送到胃里)
```

注意字母 ‘b’，‘c’，‘d’ 和 ‘e’，以及控制序列 ‘\kern’ 和 ‘\char’ 都是无法再展开的原始命令。

宏也可以包含“定界参数”，它们可以和非定界参数混合使用。定界参数的想法是，TEX 通过寻找标记参量结束的特定记号序列（即定界子）确定对应的参量。也就是说，在查找这种参量时，TEX 选取从当前位置到定界子之间的所有记号，不包含该定界子。

¹⁴ 参量内部也可以有嵌套的配对花括号，其中每对花括号表示一个编组，或者表示一个另外的宏参量。

¹⁵ 用计算机科学中的术语，宏展开是“深度优先的”，而不是“广度优先的”。注意用类似 `\expandafter` 的命令你可以修改宏展开的顺序。

要指定一个定界参数，你可以在 ‘`#n`’ (*n* 必须在 1 和 9 之间) 后面加上一个或多个充当定界子的记号。定界子一直延续到下一个 ‘`#`’ 或 ‘`\``’ 为止——这是有道理的，因为 ‘`#`’ 开始另一个参数，而 ‘`\``’ 开始替换文本。

定界子不能是 ‘`#`’ 或者 ‘`\``’，因而你可以从后面的字符辨别出定界参数和非定界参数。

如果参数之后的字符为 ‘`#`’ 或 ‘`\``’，你得到一个非定界参数，否则你得到一个定界参数。注意这两种参数的参量形式不同——非定界参数匹配单个记号或者围在花括号中的记号序列，而定界参数匹配任意多个记号，即便是零个也可以。

下面的例子用到了两个定界参数：

```
\def\diet#1 #2.{On #1 we eat #2!}
```

其中第一个参数用单个空格定界，而第二个参数用英文句号定界。如果你这样写：

```
\diet Tuesday turnips.
```

你将得到文本 “On Tuesday we eat turnips!”。但若将定界记号放在编组中，*TeX* 就不会将它们认作定界记号。因此如果你这样写：

```
\diet {Sunday mornings} pancakes.
```

即使在 ‘`Sunday`’ 和 ‘`mornings`’ 之间有一个空格，你也将得到文本 ‘On Sunday mornings we eat pancakes!’。当你将空格符作为定界子时，行尾符一般同样定界了对应的参量；这是因为 *TeX* 在宏机制运作之前就将行尾符都转换为空格了。

偶尔在你所定义的宏里面需要将 ‘`#`’ 用做一个有意义的字符。当你在宏定义中实际上又定义了第二个宏时，你多半需要这样做。怎样处理第二个宏的参数，以让 *TeX* 不会混淆它们俩呢？答案是，在第一个宏展开后需要一个 ‘`#`’ 的地方写上两个 ‘`#`’。假如你写出下面的宏定义：

```
\def\first#1{\def\second##1{#1/###1}}
```

则对 ‘`\first{One}`’ 的调用将把 ‘`\second`’ 定义为：

```
\def\second#1{One/#1}
```

而随后的调用 ‘`\second{Two}`’ 将得到文本 ‘`One/Two`’。

还有一些命令提供了宏的其他定义方式（见第 245–257 页）。在 *The TeXbook* 第 20 章中有关于宏的完整规则。

放大率。*TeX* 排版文档时将把所有尺寸乘以一个放大 (magnification) 因子 $f/1000$ ，其中 f 是 `\mag` 参数（第 237 页）的值。由于 `\mag` 的默

认值为 1000，在一般情形下文档是照常排版的。若以后需要缩印文档，增加文档的放大率通常就比较有用。

你也可以缩放一个字体，以得到比它的“设计大小”更小或更大的字体。对每个使用的缩放字体，你需要提供它的字形文件给设备驱动程序（见“字体”，第 64 页）——除非该字体已经内嵌在打印机中，而设备驱动程序也知道它。在用 `\font` 命令（第 234 页）定义一个字体时，你可以用单词 ‘`scaled`’ 指定放大率。比如：

```
\font\largerbold = cmbx10 scaled 2000
```

将 ‘`\largerbold`’ 定义为两倍大小的 `cmbx10` 字体（计算机现代粗体扩展的 10 点字体），它的各个字符形状也统一地放大了两倍。

为方便起见，很多计算机中心提供了按 1.2 比例放大的字体，这些字体对应的放大率为 1200, 1440, 等等。在 TeX 中这些放大率有特别的名称：‘`\magstep1`’ 表示 1200, ‘`\magstep2`’ 表示 1440, 依此类推直到 ‘`\magstep5`’。特殊值 ‘`\magstephalf`’ 对应 $\sqrt{1.2}$ 倍的放大率，它看上去位于 ‘`\magstep0`’（不放大）和 ‘`\magstep1`’ 的正中间。例如：

```
\font\bigbold = cmbx10 scaled \magstephalf
```

若想指定一个最终文档中的尺寸，让它不受放大率影响，你可以在单位前面加上 ‘`true`’。例如，‘`\kern 8 true pt`’ 将生成 8 点大小的紧排，不管放大率为多少。

边距. 页面的边距 (margin) 确定了一个矩形边框，该边框通常包含页面的印刷内容。你也可以让 TeX 在这个边框之外印刷内容，但你必须用某种明确方法将内容移动到那里。TeX 将页眉和页脚放在边框外面。

这个矩形边框通过它的左上角位置，宽度，以及深度来定义。左上角位置用 `\hoffset` 和 `\voffset`（第 143 页）参数来定义。默认将这个位置放在离页面顶端和左端各一英寸的地方，这对应于 `\hoffset` 和 `\voffset` 都取值为零。¹⁶ 边框的宽度用 `\hsize` 给出，而深度用 `\vsize` 给出。

这些规定给出的结果是：

- 左边距等于 `\hoffset + 1in`。
- 右边距等于页面宽度减去 `\hoffset + 1in + \hsize`。
- 上边距等于 `\voffset + 1in`。
- 下边距等于页面长度减去 `\voffset + 1in + \vsize`。

从这些信息中你可以知道，要改变页面边距应该修改哪些参数。

¹⁶ 这个约定看起来比较奇怪。让 `\hoffset` 和 `\voffset` 的 (0,0) 点对应页面的左上角，而将它们的默认值设置为一英寸，将会更自然一些。

对 `\hoffset`, `\voffset` 或者 `\vsize` 的修改在 TeX 开始新页面时才生效。也就是说，如果你在页面中间改变它们，这些改动将只影响后面的页面。然而，对 `\hsize` 的修改却会立即生效。

标记。标记 (mark) 是可以插入水平、竖直或者数学列表中，并在输出例行程序中还原的项目。标记可用于记录即将出现在页眉中的内容等。每个标记带有一列记号——即“标记文本”。`\mark` 命令（第 147 页）以这样的记号列作为参数，并将包含这个记号列（在展开之后）的项目添加到 TeX 当前构造的列表中。`\topmark`, `\firstmark` 和 `\botmark` 命令（第 147 页）用于取回页面上的各种标记。这些命令经常在页眉和页脚中用到。

这里有个简化的例子。假设你定义了节标题宏如下：

```
\def\section#1{\medskip{\bf#1}\smallskip\mark{#1}}
% #1 is the name of the section
```

调用这个宏将生成粗体的节标题，并将标题名记录为一个标记。现在你可以对每个打印页定义页眉如下：

```
\headline = {\ifodd\pageno \hfil\botmark\quad\folio
\else \folio\quad\firstmark\hfil \fi}
```

这样，每个偶数页（左手页）页眉是页码加上该页第一个标题名，而每个奇数页（右手页）页眉是该页最后一个标题名加上页码。特殊情形，比如该页没有节标题，通常也会按照 `\firstmark` 和 `\botmark` 的规定正常显示。

在用 `\vsplit` 命令（第 152 页）分割页面时，用 `\splitfirstmark` 和 `\splitbotmark` 命令（第 147 页）就可以取得分割的这部分中的第一个和最后一个标记的标记文本。

在 *The TeXbook* 第 258–260 页中，对如何生成和取回标记作了更准确的解释。

数学模式。数学模式 (math mode) 是 TeX 构造数学公式时所在的模式。TeX 有两种数学模式：文内数学模式用于构造放在文本行中的公式，而陈列数学模式用于构造单独一行显示的公式。括在 \$ 里面的公式表示一个文内数学公式，而括在 \$\$ 里面的公式表示一个陈列数学公式。这两种数学模式有个重要特性，即其中的输入空格都被忽略。*The TeXbook* 第 290–293 页详细介绍了在数学模式中 TeX 如何处理各种不同命令。

数学码。数学码 (mathcode) 是一个数, TeX 把这个数用来识别和描述一个数学字符, 即一个在数学公式中有特别作用的字符。数学码表达了一个字符三个方面的信息: 它的字符位置, 它所在的字体族, 以及它所属的类。256 个可能的输入字符中每个字符都有一个数学码, 它们由 TeX 程序定义, 但也能被修改。

TeX 有十六个字体族, 它们用 0 到 15 编号。每个字体族包含三个字体: 一个用于文本尺寸, 一个用于标号尺寸, 另一个用于小标号尺寸。TeX 根据上下文选择特定字符的尺寸, 从而选择它的字体。一个字符的类指定了它在公式中扮演的角色 (见 *The TeXbook* 第 154 页。例如, 等号 ‘=’ 属于第 3 类 (关系符号)。TeX 根据字符所属的类确定数学公式各部分的间距大小。

要全面理解数学码, 最好是看 TeX 如何使用它们。因此我们将说明, 对数学公式中类别码为 11 或 12 的字符记号 t , TeX 如何处理它:

- 1) 它查找这个字符的数学码。
- 2) 它从数学码确定字体族 f 。
- 3) 它从上下文确定字符尺寸 s 。
- 4) 它选择第 f 族的尺寸为 s 的字体 F 。
- 5) 它从数学码确定字符编号 n 。
- 6) 它从字体 F 的位置 n 选出字符 c 作为要排版的字符。
- 7) 它根据 t 所属的类和上下文调整 c 周围的间距。
- 8) 它排版字符 c 。

在 (3) 和 (7) 中的上下文相关性意味着, TeX 在排版数学字符时, 要先看完该字符所在的整个公式。比如对于公式 ‘\$a\over b\$’, TeX 无法确定 ‘a’ 的尺寸, 直到它看到 \over。

字符的数学码由公式 $4096c + 256f + n$ 来表示, 其中 c 是字符所属的类, f 是它所在的字体族, 而 n 是它在该字体族中的 ASCII 编码。通过对一个输入字符的 \mathcode 表格项赋值 (第 267 页), 你可以改变数学模式中 TeX 对这个字符的解释。只有类别码为 11 (字母) 或者 12 (其它) 的字符, TeX 才会查看它的 \mathcode。

只要将一个数学字符分到第 7 类, 你可以将它的族定义为 “可变的”。当 TeX 在数学公式中遇到这种字符时, 它就将该字符的字体族取为 \fam 参数 (第 221 页) 的当前值。可变的字体族使得你可以指定数学公式中普通文本的字体。举个例子, 如果罗马字符是在第 0 族, 赋值 \fam = 0 将让数学公式的普通文本以罗马字体显示, 而不是用其他类似数学意大利体的字体显示。如果 \fam 的值不在 0 到 15 的范围内, TeX 取它的值为 0, 从而让第 0 类和第 7 类等价。每次进入数学模式时 TeX 都设定 \fam 为 -1。

数学单位。数学单位 (mathematical unit)，记为 ‘mu’，是一个用于设定数学公式中的粘连大小的距离单位。见“数学粘连”(第 81 页)。

模式。 \TeX 在胃中处理你的输入时 (见“ \TeX 剖析”，第 49 页)，它总是处于六种模式 (mode) 的其中一种：

- 普通水平模式 (组装段落)
- 受限水平模式 (组装水平盒子)
- 普通竖直模式 (组装页面)
- 内部竖直模式 (组装竖直盒子)
- 文内数学模式 (组装在文本中显示的公式)
- 陈列数学模式 (组装单独一行显示的公式)

模式描述了 \TeX 正在组装的实体的类型。

由于你可以将某种实体嵌入另一种实体中，比如将一个竖直盒子放入数学公式中， \TeX 记录的不只是一个模式，而是一整列模式 (即计算机科学中所说的“栈”)。假设 \TeX 在模式 M 中，然后遇到某些东西而进入新模式 M' 。当它完成在模式 M' 的工作时，它将继续在模式 M 中的工作。

数学粘连。数学粘连 (muglue) 是一种仅用于数学公式中的粘连，它的度量单位是 mu (数学单位)。1 mu 的大小与 $1/18 \text{ em}$ 相等，而 1 em 的大小取自第 2 族数学字体。 \TeX 根据上下文自动调整 mu 的大小。比如同样是 2mu 大小的粘连，在下标中一般就比在普通文本中小一些。要生成数学粘连，你必须使用 `\mskip` 命令。例如，‘`\mskip 4mu plus 5mu`’生成了一个自然大小为 4mu，伸长量为 5mu 的数学粘连。

数。在 \TeX 中，数 (number) 是指正整数或负整数。 \TeX 中的数可以用如下四种不同方式表示：

- 1) 用通常的十进制整数表示，比如 52
- 2) 用八进制数表示，比如 ’14
- 3) 用十六进制数表示，比如 "FF0
- 4) 用字符的 ASCII 编码表示，比如 ‘\r’ 或者 ‘\n’

上述任何一种形式都可以在前面加上 ‘+’ 或者 ‘-’。

八进制数只能包含数字 0–7。而十六进制数可以包含数字 0–9 以及字母 A–F，它们分别表示从 0 到 15 的数值。然而，你不能用小写字母来表示十六进制数。在 *The TeXbook* 43–44 页中你可以找到八进制和十六进制数的解释。

十进制、八进制或十六进制数都在第一个不能作为该数一部分的字符处结束。因此，任何字母都会结束一个十进制数，而 ‘A’ 和 ‘F’ 之间的

字母却不能结束一个十六进制数。你也可以用一个或多个空格结束一个数, TEX 通常将忽略这些空格。¹⁷

上述的第四种形式用字符的 ASCII 编码表示一个数。TEX 同样忽略此形式的数后面的空格。一个数按这种形式可以写成 ‘c’ 或者 ‘\c’。第二种写法虽然长一点, 但优点在于可以用于任何字符, 甚至是 ‘\’, ‘%’ 或者 ‘^M’。不过它确实也有一个技术上的缺点: 比如当 TEX 展开 \edef 或 \write 命令的记号序列时, 出现在数中间的能展开的 ‘\c’ 将同样被展开; 而这多半不是你想要的。

下面是十进制数 78 的全部有效的表示方式:

```
78    +078    "4E    '116   'N   '\N
```

你不能在文本中单独使用一个数, 因为它不是命令。但是, 将它放在 \number 命令 (第 238 页) 之后, 你就可以在文本中插入这个数的十进制形式; 或者将它放在 \romannumeral 命令之后, 以得到它的罗马数字形式。

你也可以用小数, 即带有小数部分的数, 来指定一个尺寸 (见 “尺寸”, 第 61 页)。小数包含一个可以在最前面的小数点。小数点的圆点也可以用逗号代替。小数之前也可以加上正号或负号。因此 ‘.5in’, ‘-3.22pt’ 和 ‘+1,5\baselineskip’ 都是有效的尺寸。但是, 除了作为尺寸的“因子”, 即乘数, 你不能在其他地方使用小数。

普通模式. 普通模式 (ordinary mode) 是 TEX 分段为行或组行为页时所在的模式。见 “水平模式” (第 69 页), “竖直模式” (第 92 页)。

外部的. 外部的 (outer) 宏是在某些 TEX 高速处理记号的环境中无法使用的宏。将一个命令定义为外部的, 目的是使 TEX 能够尽早捕获到错误。在定义宏时, 你可以用 \outer 命令 (第 246 页) 让它成为外部的。

在下列这些环境中, 你不可以使用外部宏:

- 在命令的参量中
- 在宏定义的参数文本或替换文本中
- 在阵列的导言中
- 在条件测试的跳过部分中

外部环境是你可以使用外部宏的环境, 即刚才列出的环境除外的环境。

例如, 下面的输入就是外部宏的禁止用法:

```
\leftline{\proclaim Assertion 2. That which is not inner
is outer.}
```

¹⁷ 在定义一个以数结尾的宏时, 你应该始终在该数后面加上空格; 否则到后面 TEX 可能将该数和其他字符结合在一起。

\proclaim 宏（第 131 页）在 plain T_EX 中被定义为外部的，但这里却将它放在 \leftline 的宏参量中。

输出例行程序。T_EX 收集足够填满页面的素材后，它选择分页点并将分页点之前的素材放在 \box255 中。然后 T_EX 调用输出例行程序（output routine）以加工素材并最终将它送到 .dvi 文件中。输出例行程序可以执行进一步的加工，比如插入页眉、页脚和脚注。Plain T_EX 提供了一个默认的输出例行程序，它在每个页面底部插入居中的页码。通过提供不同的输出例行程序，你可以达到诸如双栏输出的效果。你可以认为输出例行程序只有一个职责：以某种方式处理掉放在 \box255 中的素材。

当前的输出例行程序由 \output（第 151 页）的值定义，它是一个记号列。当 T_EX 准备好生成页面时，它就展开该记号列。

实际上不用修改 plain T_EX 的输出例行程序，你就能对它的行为做出一些简单改动。例如，通过赋予一个记号列给 \headline 或 \footline（第 146 页），你就可以让 T_EX 生成和通常情形不同的页眉或页脚。

输出例行程序也负责收集任何插入项；它将这些插入项和任何“装饰”（比如页眉页脚）合并到页面主要内容中，并将所有这些素材包装在一个盒子里；最后它用 \shipout 命令（第 151 页）将这个盒子送到 .dvi 文件中。虽然输出例行程序大多是这样运行的，但有特定用途的输出例行程序会有不同表现。

输出流。见“文件”（第 63 页）。

页面。T_EX 每次一页地将你的文档组装为页面（page），并将它们送往输入例行程序。在处理文档时，T_EX 维持着一个由文本行和其他要放入页面的项目组成的列表。（这些文本行实际上是水平盒子。）这个列表称为“主竖直列”。T_EX 周期性地执行称为“运行页面构建器”的操作。在运行页面构建器时添加到主竖直列的项目称为“备选内容”。

页面构建器首先检查主竖直列，看是否有必要输出页面，这可能由于主竖直列上的项目不能填满页面，或者有类似 \eject（第 140 页）的显式项目让 T_EX 结束页面。如果没必要输出页面，页面构建器这回的工作就完成了。

否则，页面构建器分析主竖直列以找到最佳的可能分页点。它在各种不美观的分页点——即可能导致单独一行出现在页面顶部或底部的分页点，或者在陈列公式前的分页点，诸如此类——添加惩罚项。然后它选择代价最小的分页点，其中分页代价由分页点的惩罚值以及所得页面的劣度加起来得到（代价计算公式可以见 *The T_EXbook* 第 111 页。如果找到多个代价相同的分页点，它选择最后一个。

一旦页面构建器选定了分页点，它就将列表中位于分页点之前的项目放入 `\box255`，并把剩余项目留给下个页面。接着它调用输出例行程序。`\box255` 充当了邮箱的角色，页面构建器为发送者，输出例行程序为接收者。输出例行程序如常处理 `\box255`，添加其他项目比如插入项和页眉页脚到该页面，然后用 `\shipout` 命令将页面输出到 `.dvi` 文件。（有些输出例行程序可能有不同的行为。）从 `TEX` 的角度看，输出例行程序输出或不输出页面都无所谓；输出例行程序的惟一职责是以某种方式处理 `\box255`。

认识到最佳分页点未必是最后的分页点是非常重要的。惩罚项和其他考虑可能导致分页点出现在前面。此外，`TEX` 是批量地而不是逐个地添加项目到主竖直列的。段落的各行是批量的例子。由于这些原因页面构建器分页时通常会剩余一些项目。这些剩余项目组成了下一页的主竖直列的开始部分（可能在批量的中间）。由于项目可能被从一个页面移动到另一个页面，你无法假定 `TEX` 处理输入时的当前页码准确表示了对应输出中显示的页码。在 *The T_EXbook* 第 110–114 页中有 `TEX` 的分页规则的完整说明。

分页点. 分页点（page break）是文档中 `TEX` 结束一个页面并（除非在文档结尾处）开始新页面的位置。在“页面”（第 83 页）中介绍了 `TEX` 选择分页点的过程。

你可以用以下几种方式控制 `TEX` 对分页点的选择：

- 你可以插入惩罚项（第 139 页）到主竖直列的两个项目之间。正惩罚值阻碍 `TEX` 在此处分页，而负惩罚值——换言之，奖励值——鼓励 `TEX` 在此处分页。大于或等于 10000 的惩罚值阻止分页，而小于或等于 -10000 的惩罚值强制分页。你可以用 `\break` 和 `\nobreak`（第 139 页）命令得到相同的结果。
- 你可以通过改变 `TEX` 的分页参数的值来调整分页惩罚值。
- 你可以将主竖直列的一系列段落或者其他项目放入竖直盒子，从而阻止 `TEX` 在它们之间的任何位置分页。

一旦 `TEX` 选定了分页点，它就将该主竖直列中位于分页点之前的内容放入 `\box255`。接着它调用当前的输出例行程序处理 `\box255`，最终输出页面内容到 `.dvi` 文件。输出例行程序还得处理 `TEX` 加工页面时积累的插入项，比如脚注。

最好能知道 `TEX` 在什么地方可以分页：

- 在粘连处，只要在粘连之前的项目是一个盒子，小玩意，标记，或者插入项。当 `TEX` 在粘连处分页时，它把分页点放在该粘连间距的顶部，并忽略该粘连的其他值。

- 在紧排处，只要该紧排之后是一个粘连。
 - 在惩罚处，这可能出现在段落的两行之间。
- TeX 分页时将丢弃分页点之后的任意多个粘连，紧排和惩罚项。

页面构建器。见“页面”（第 83 页）。

页面布局。在设计文档时，你需要决定其页面布局（page layout）：页面大小，四个方向的边距，页眉和页脚，若存在的话，还有正文和页眉页脚之间的间隔大小。对上述这些 TeX 都有默认值：它设定页面大小为 $8\frac{1}{2}$ 英寸乘 11 英寸，四个边距约为一英寸，没有页眉，页脚是居中的页码。

边距由四个参数 `\hoffset`, `\voffset`, `\hsize` 和 `\vsize` 共同确定（在第 78 页的“边距”概念中有如何调整它们的指南）。页眉是在每个页面顶部的一行内容，位于上边距区域中。通过将一个记号列表赋值给 `\headline` 参数（第 146 页），你就可以设定页眉。类似地，页脚是在每个页面底部的一行内容，位于下边距区域中。通过将一个记号列表赋值给 `\footline` 参数（第 146 页），你就可以设定页脚。例如，下面的输入：

```
\headline = {Baby's First Document\dotfill Page\folio}  
\footline = {\hfil}
```

将给每个页面生成类似下面的页眉行

Baby's First Document.....Page 19

而页脚行是空白的。

利用标记，你可以将当前主题或节标题放入页眉或页脚中。在“标记”（第 79 页）这里将详细解释如何做到这些。

段落。直观地，段落（paragraph）就是一系列输入行，并用空行或者 `\par` 命令（第 110 页），或者类似 `\vskip` 的本质上的竖直模式命令结束。更准确地说，段落就是 TeX 在普通水平模式中处理的一系列命令。TeX 收集整个段落的内容后，就通过选择断行点（见“断行点”，第 74 页）将它分为多行。得到的结果是一系列水平盒子，以及它们之间的粘连，行间惩罚和穿插的竖直素材。其中的每个水平盒子就是一个文本行，而这些粘连就是行间粘连。

当 TeX 在竖直模式中碰到本质上为水平模式的命令，它开始一个新段落。特别地，在刚结束一个段落的时候，它位于竖直模式，因此在空白输入行之后的水平素材自然就开始下一个段落。本质上为水平模式的命令有好多种，但最常见的就是普通字符，比如一个字母。

\indent 和 \noindent 命令（见 111, 111 页）也属于本质上为水平模式的命令，它们让 TeX 在段落开始缩进或不缩进。其他水平模式命令出现在竖直模式时都会导致 TeX 隐性执行 \indent。一旦 TeX 开始一个新段落，它就处于普通水平模式中。它首先执行 \everypar 中的命令，然后着手收集该段落的项目，直到它得到段落结束的信号。在段落结尾处，它重置段落形状参数 \parshape, \hangindent 和 \looseness。

TeX 一般将一个空行转换为一个 \par 命令。当它在水平模式碰到一个本质上的竖直模式命令时，它同样插入 \par 命令到输入中。因此结束段落的最终都是一个 \par 命令。

当 TeX 收到一个 \par 命令时，它首先填满¹⁸ 当前段落。接着将该段落分为多行，并将所得结果添加到所在的竖直列中，然后执行页面构建器（仅当所在的竖直列就是主竖直列时）。如果该段落是用本质上为竖直模式的命令结束的，TeX 接下来将执行该命令。

参数。参数（parameter）这个术语有两个不同的含义——它可以表示 TeX 参数或者表示宏参数。

TeX 参数是一个控制序列，它是某个值的名称。参数的值可以是数值，长度，一定大小的粘连或数学粘连，或者是一个记号列表。例如，\parindent 参数指定了 TeX 在缩进段落开头所跳过距离的大小。

你可以对 TeX 参数的控制序列取值或赋值。如果该控制序列出现在需要取值的地方，TeX 将它解释为取值；否则，解释为赋值。例如：

```
\hskip\parindent
```

生成一个自然大小为 \parindent 的水平粘连，而

```
\parindent = 2pc % (or \parindent 2pc)
```

设置 \parindent 等于两派卡的长度。而下列赋值：

```
\parindent = 1.5\parindent
```

用两种方式使用 \parindent。其结果是将 \parindent 的值乘以 1.5 倍。

你可以将 TeX 参数视为内置的寄存器。在 *The TeXbook* 第 272–275 页中你可以找到 TeX 参数的完整列表。

宏参数是一个占位符，用于插入文本到宏定义中。在“宏”（第 75 页）■这里有这类参数的更多信息。

¹⁸ 准确地说，它执行下列这些命令：

```
\unskip \penalty10000 \hskip\parfillskip
```

并将所得项目添加到当前竖直列的最后面。

惩罚. 惩罚是一个你可以放在水平、竖直或数学列表中的项目，用于阻碍或鼓励 TeX 在此处断开该列表。正惩罚值表示一个糟糕断开点，而负惩罚值表示一个良好断开点。断开普通水平列生成一个断行点，而断开普通竖直列生成一个分页点。(在限制水平模式或内部竖直模式的惩罚项无效。)

你可以用 `\penalty` 命令 (见 122, 139 页) 直接插入一个惩罚项。大于或等于 10000 的惩罚值阻止断开。而小于或等于 -10000 的惩罚值强制断开。

plain TeX. *Plain TeX* 是本书和 *The TeXbook* 中所描述的 TeX 形式。Plain TeX 包含在标准 TeX 系统中，因此仅使用 plain TeX 的文档通常很容易从一个 TeX 系统转移到另一个中。

Plain TeX 由原始命令，加上大量的宏和其他定义组成。在 *The TeXbook* 附录 B 中给出了这些附加定义。它们应该同样包含在你的计算机系统的某个 `plain.tex` 文件中。

原始的. 原始的 (primitive) 命令是在 TeX 程序内部就定义好的命令。作为对比，非原始命令或者是从宏定义得到的，或者是从 TeX 编写的其他形式的定义得到的。Plain TeX 的命令由原始命令加上用原始命令定义的其他命令组成。

基准点. 盒子的基准点 (reference point) 是盒子左边缘与基线的交点。在处理水平列表或竖直列表时，TeX 用列表中各盒子的基准点水平或竖直对齐这些盒子 (见“盒子”，第 53 页)。

寄存器. 寄存器 (register) 是用于存储数值的命名位置。它很像编程语言中的一个变量。TeX 有五种寄存器，如下表所示：

寄存器类型	存储内容
<code>box</code>	一个盒子
<code>count</code>	一个数
<code>dimen</code>	一个尺寸
<code>muskip</code>	数学粘连
<code>skip</code>	粘连
<code>toks</code>	一个记号列

每种类型的寄存器都从 0 到 255 编号。要存取第 n 个类型为 c 的寄存器，你可以用 ‘ $\text{\char}cn$ ’ 这种形式，比如 `\muskip192`。寄存器可以用在任何需要相应类型信息的地方。例如，你可以在需要一个数的地方使用 `\count12`，或者在需要粘连的地方使用 `\skip0`。

要将信息放入寄存器中，你可以给它赋值：

```
\setbox3 = \hbox{lagomorphs are not mesomorphs}
\count255 = -1
```

第一个赋值构造一个水平盒子并把它分配给 3 号盒子寄存器。此后你可以在需要盒子的地方用 ‘\box3’ 得到该盒子。¹⁹ 第二个赋值将 -1 分配给 255 号计数寄存器。

给定类型的寄存器，比如粘连寄存器，就像该类型的参数一样。你可以如同参数那样获取或赋予它的值。有些 TeX 参数，比如 \pageno，实际上就是作为寄存器实现的。

Plain TeX 用了很多寄存器来实现自己的目标，因此在需要寄存器时你不能随意选择一个来用。你应该用下列这些命令之一让 TeX 给你保留一个寄存器：\newbox、\newcount、\newdimen、\newmuskip、\newskip 或 \newtoks（第 260 页）。这些命令是外部的，因此你不能在宏定义中使用它们。要是可以这样做，每次调用宏时就会用掉一个寄存器，可能很快就用完所有寄存器。

即便如此，只需谨慎一点你就可以在编组内部临时使用寄存器，甚至是 TeX 在其他地方用到的寄存器。在执行完编组内的命令后，TeX 将每个寄存器的内容重置为开始执行编组前的内容。在编组内部使用显式编号的寄存器时，你必须确保该寄存器不会被你在编组内调用的宏所修改。如果编组内调用了非你所写的宏，在其中使用任意寄存器时需要特别小心。

TeX 将某些寄存器保留作特别用途：\count0 到 \count9 用作页面编号信息，而 \box255 用作送到输出例行程序之前的页面内容。寄存器 \dimen0-\dimen9、\skip0-\skip9、\muskip0-\muskip9、\box0-\box9 和 \box255 除外的 255 号寄存器一般用作“临时”寄存器。也就是说 plain TeX 仅提供一个用于计数的临时寄存器 \count255。请参阅 *The TeXbook* 第 122 和 346 页，其中有选择寄存器编号时要遵守的约定。

要在 TeX 运行时检查一个寄存器的内容，你可以用 \showthe 命令（第 270 页），比如 ‘\showthe\dimen0’。

受限模式. 受限模式 (restricted mode) 是 TeX 组装水平盒子或竖直盒子时所处的模式。我们使用 *The TeXbook* 中的术语“内部竖直模式”来命名我们认为的“受限竖直模式”。请参阅“水平模式”（第 69 页）和“竖直模式”（第 92 页）。

¹⁹ 但是要注意：使用盒子寄存器的同时也清空了该寄存器，从而它的内容变为空白。其他类型的寄存器不会这样。你可以用 \copy 命令（第 169 页）获取盒子寄存器的内容而不清空它。

标线。标线 (rule) 是一个黑色实心矩形。标线和盒子一样也有宽度、高度和深度。该矩形的竖直尺寸是它的高度和深度之和。普通的水平或竖直直线是标线的特殊情形。

标线可以是水平的或者竖直的。水平标线和竖直标线的区别与你如何生成标线有关，因为竖直标线可以矮而胖（从而看似水平标线），而水平标线也可以高而瘦（从而看似竖直标线）。*TeX* 所说的标线比排印工的更一般，它们认为标线就是一条线而通常不把黑色方块称为标线。

你可以用 `\hrule` 命令生成水平标线，用 `\vrule` 命令（第 178 页）生成竖直标线。例如，控制序列 `\hrule` 本身就生成一个横跨页面的细标线，如下：

命令 ‘`\vrule height .25in`’ 生成往下 .25 英寸的竖直标线如下：

|

水平标线和竖直标线有如下两个区别：

- 1) 对于水平标线，*TeX* 设定将其默认宽度为包含它的最小盒子或阵列的宽度。对于竖直标线，*TeX* 用类似方法设定其默认高度和深度。（如果你没有显式设定尺寸，默认为内容的尺寸。）
- 2) 水平标线是本质上的竖直项目，不能放在水平列表中；而竖直标线是本质上的水平项目，不能放在竖直列表中；这种行为初看也许有些奇怪，但却是有合理原因的：水平标线在外观上通常是从左到右的，因而分开了竖直列表中的项目；而竖直标线在外观上通常是从上到下的，因而分开了水平列表中的项目；

如果你构造一个有三个明确尺寸的标线，无论作为水平标线还是竖直标线它看起来都是一样的。例如，命令 ‘`\vrule height1pt depth2pt width3in`’ 生成如下看似水平方向的标线：

在 *The TeXbook* 第 221–222 页中，你可以找到对 *TeX* 如何处理标线的精确表述。

标号尺寸。 标号尺寸 (script size) 描述一族字体中三个相关字体的其中一个。标号尺寸比文本尺寸小但比小标号尺寸大。对上标，下标以及文本中分式的分子和分母，*TeX* 使用标号尺寸。

小标号尺寸。 小标号尺寸 (scriptscript size) 描述一族字体中三个相关字体的最小一个。对第二级的上标、下标、分子和分母，*TeX* 使用小标号尺寸。比如，对于下标中的下标，或者标号尺寸分子中的上标，*TeX* 将使用小标号尺寸。

收缩量。见“粘连”(第 66 页)。

间隔。要让 TeX 在两个项目间留下间隔 (space)，有如下几种方法：

- 1) 你可以写上某些 TeX 视为空格记号的东西：一个或多个空格符，行尾符 (行尾符作用和空格相似)，或任何能展开为空格记号的命令。TeX 一般将多个连续空格视为单个空格，包括其中有一个行尾符的情形。(一个空行表示段落的结束；它导致 TeX 生成一个 \par 记号。) TeX 调整此种间隔的大小以适应由上下文所需的长度。
- 2) 你可以写上一个生成指定粘连的间距命令。粘连有伸长量或收缩量，从而生成或多或少的间隔。你可以有水平粘连和竖直粘连。在断行或分页位置之后的粘连将会消失。
- 3) 你可以写上一个紧排。紧排生成固定大小的间隔，它不可伸长也不可收缩，而且在断行或分页时也不会消失 (除非其后紧跟着一个粘连)。紧排常用于给两个相邻盒子建立固定的空间关系。

粘连和紧排也可以取负值。两个相邻项目间的负粘连或负紧排将拉近它们俩的距离。

伸长量。见“粘连”(第 66 页)。

支架。支架 (strut) 一个不可见的盒子，它的宽度为零，而高度和深度比上下文中的“正常”行稍微大一点。支架常用于在通常行间隔取消时获得一致的竖直间隔，比如在数学公式中，或者在设定了 \offinterlineskip 的水平阵列中。因为支架比该行的其他东西都高和深，它确定了该行的高度和深度。你可以用 \strut 命令 (第 172 页) 或 \mathstrut 命令 (第 173 页) 生成一个支架。 \strut 可以在任何地方使用，但 \mathstrut 只能在数学模式中使用。在 plain TeX 中，支架的高度为 8.5 点，深度为 3.5 点，而数学支架的高度和深度和当前样式的左圆括号的相等 (因此在上下标中会小一点)。

这里有个例子给你展示了支架的用法：

```
\vbox{\hsize = 3in \raggedright
  \strut Here is the first of two paragraphs that we're
  setting in a much narrower line length.\strut}
\vbox{\hsize = 3in \raggedright
  \strut Here is the second of two paragraphs that we're
  setting in a much narrower line length.\strut}
```

这个输入产生下列输出：

Here is the first of two paragraphs that
we're setting in a much narrower line length.

Here is the second of two paragraphs that
we're setting in a much narrower line length.

要是没有支架，这两个盒子将太过靠近了。类似地，在数学公式中：

$\overline{x\mathstrut} \otimes \overline{t\mathstrut}$

数学支架让两个横线位于相同的高度，即使‘ x ’和‘ t ’有不同的高度：

$\bar{x} \otimes \bar{t}$

样式。数学公式中的素材依据上下文被设定为八种样式（style）的其中一种。对样式的了解，将有助于你将某部分公式设为不同类型的大小，即与 TeX 按照通用法则选择的大小不同。

其中的四种主要样式如下：

陈列样式	(用于单独一行显示的公式)
文本样式	(用于嵌入普通文本内的公式)
标号样式	(用于上标和下标)
小标号样式	(用于上标中的上标，等等)

另外四种样式被称为狭窄变体。在这些变体中，上标升高得没那么多，因而公式和原来相比占用更少的竖直间隔。在 *The TeXbook* 第 140–141 页中详细介绍了 TeX 如何选择样式。

TeX 依照样式选择尺寸类型：

- 陈列样式和文本样式设定为文本尺寸，就像‘this’这样。
- 标号样式设定为标号尺寸，就像‘this’这样。
- 小标号样式设定为小标号尺寸，就像‘this’这样。

参考“族”（第 62 页）以获得更多关于这三种尺寸的信息。

在 TeX 中没有“小小标号”样式，因为此样式对应的尺寸太小以致无法阅读。因此 TeX 用小标号样式设定三阶上下标等。

偶尔你会发现 TeX 给公式设定的样式和你想要的不同。你可以用 `\textstyle`, `\displaystyle`, `\scriptstyle` 和 `\scriptscriptstyle` 命令（第 207 页）改变 TeX 的选择。

TeX MeX. (a) 一个在中美洲国家用于数学排版的 TeX 变种。(b) 一种深受埃尔帕索的 TeX 用户欢迎的香辣菜。

文内公式。我们用文内公式 (text math) 这个术语表示放在一行文本之中的数学公式，即两边用 \$ 括起来的公式。 \TeX 在文内数学模式中排版文内公式。

文本尺寸。文本尺寸 (text size) 描述一族字体中三个相关字体的最大一个。对数学模式中的普通符号， \TeX 使用文本尺寸。

记号。一个记号 (token) 要么是一个标有类别码的字符，要么是一个控制序列。 \TeX 用它的眼睛从文件中读取字符串（见“ \TeX 剖析”，第 49 页），并用它的嘴将字符串咀嚼成一个个记号。当一个记号到达 \TeX 的胃时， \TeX 将它解释为一个命令，除非它是之前命令的参量。

度量单位。见“尺寸”（第 61 页）。

竖直盒子。竖直盒子 (vbox, vertical box) 是 \TeX 从上到下逐个放置竖直列表的项目而构造出来的盒子。竖直盒子，作为一个单元，既不是本质上水平的也不是本质上竖直的，即它可以出现在竖直列表或水平列表中。你可以用 `\vbox` 或 `\vtop` 命令（第 166 页）构造竖直盒子。两者的区别在于，对于 `\vbox`，所构造的竖直盒子的基准点来源于列表的最后一个（通常是最下面那个）组成项；而对于 `\vtop`，所构造的竖直盒子的基准点来源于列表的第一个（通常是最上面那个）组成项。

竖直列表。竖直列表 (vertical list) 是 \TeX 位于某种竖直模式时（即组装竖直盒子或页面时）生成的一列项目。见下面的“竖直模式”。

竖直模式。在组装竖直盒子或者主竖直列（输出的页面来源于此）时， \TeX 位于两种竖直模式 (vertical mode) 之一：普通竖直模式用于组装主竖直列，而内部竖直模式用于组装竖直盒子。只要 \TeX 位于竖直模式中，它的胃（见“ \TeX 剖析”，第 49 页）就在构造项目（盒子、粘连、惩罚等）的竖直列表。 \TeX 逐个排版列表中的项目，从上到下。

竖直列表不能包含任何由本质上的水平命令，比如 `\hskip` 或普通的（非空格）字符，生成的项目。²⁰

- 如果 \TeX 正在普通竖直模式中组装竖直列表，并碰到一个本质上的水平命令，它将切换到水平模式。
- 如果 \TeX 正在内部竖直模式中组装竖直列表，并碰到一个本质上的水平命令，它将报错。

²⁰ 在竖直模式中 \TeX 忽略它遇到的任何空格符。

这两个命令你也许会认为是本质上竖直的，实际上却是本质上水平的：`\valign`（第 185 页）和 `\vrule`（第 178 页）。在 *The TeXbook* 第 283 页中列出了所有本质上的水平命令。

有个特别重要的事实需要知道，即 TeX 将任何普通的非空格符视为本质上水平的。如果 TeX 在你预期之外突然开始一个新段落，这可能是由 TeX 在竖直模式中遇到的字符引起的。要让 TeX 不把该字符视为本质上水平的，你可以将它放入一个水平盒子中；这是因为，尽管看名字容易误解，`\hbox` 命令既不是本质上水平的也不是本质上竖直的。

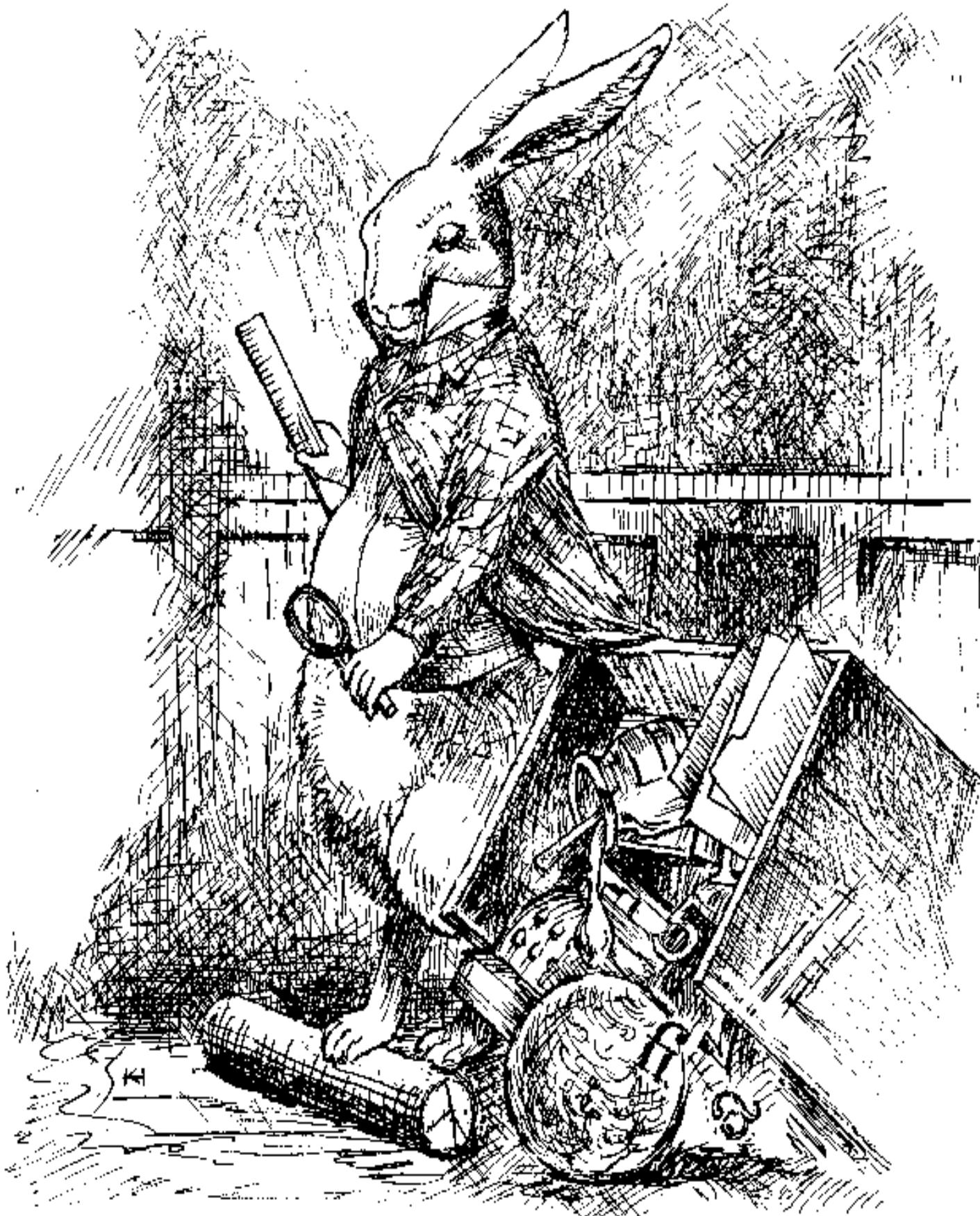
小玩意. 小玩意 (whatsit) 是一个信息项目，它用于让 TeX 执行某些无法归入普通事物体系中的操作。小玩意可以出现在水平列表或竖直列表中，就像一个盒子或粘连项那样。TeX 将小玩意排版为一个宽度、高度和深度都为零的盒子——即一个不包含任何东西也不占用任何空间的盒子。

在 TeX 中内建了三种类型的小玩意：

- `\openout`、`\closeout` 和 `\write` 命令（第 265 页）生成用于操作输出文件的小玩意。TeX 将此操作推迟到下次送出页面到 `.dvi` 文件时（除非在该操作之前加上 `\immediate`）。TeX 用小玩意表示这些命令，因为它们与当时正在排版的内容完全无关。
- `\special` 命令（第 266 页）让 TeX 直接插入某些文本到 `.dvi` 文件中。如同 `\write` 命令，TeX 也将此操作推迟到下次送出页面到 `.dvi` 文件时。`\special` 的典型用法是命名一个图片文件，以让设备驱动文件将它整合到最终输出中。
- 当你用 `\language` 或 `\setlanguage` 命令（第 129 页）改变语言时，TeX 插入一个小玩意，以引导它在以后分段为行时使用特定集合的连字规则。

TeX 的某些特别实现可能提供额外的小玩意。

宽度. 盒子的宽度 (width) 是它总共占用的水平间隔，即从它的左边缘到右边缘的距离。盒子内排版的素材可以比盒子本身还宽。



5 | 组段命令

这一章介绍了与字符，单词，行以及整段相关的命令。本章写作惯例的说明可以在“命令描述”（第 3 页）中找到。

字符和重音符

■ 欧洲语言字母和连写

\AA	斯堪地纳维亚文字符 Å
\aa	斯堪地纳维亚文字符 \aa
\AE	\AE 连写
\ae	\ae 连写
\L	波兰文字符 \L
\l	波兰文字符 \l
\O	丹麦/挪威文字符 \O
\o	丹麦/挪威文字符 \o
\OE	\OE 连写
\oe	\oe 连写
\ss	德文字符 \ss

这些命令可以产生欧洲语言的各种字符和连写。在文本中偶而出现这些语言的单词和词组时它们显得很有用。但是如果需要排印大量的欧洲语言文本时，你可能会去使用一个为这些语言定制的 TeX 版本。¹

当你在一个单词中使用这些命令时，需要在它们后面加上一个空格，这样，TeX 会把它后面的字母当作是这个词语的一部分而不是这个命令的一部分。你不需为了使用这些符号而进入数学模式。

¹ TeX 用户组织 (第 18 页) 可为你提供 TeX 的欧洲语言版本。

例子：

```
{\it les œ vres de Molière}
```

结果：

les œvres de Molière

■ 专用符号

◊	\#	英镑符号 # ²
	\\$	美元符号 \$
	\%	百分号 %
	\&	和 &
	_	下划线 _
	\lq	左引号 ‘
	\rq	右引号 ’
	\lbrack	左中括号 [
	\rbrack	右中括号]
	\dag	短剑符号 †
	\ddag	双剑符号 ‡
	\copyright	版权符号 ©
	\P	段落符号 ¶
	\S	章节符号 §

这些命令可以产生各种特殊字符和符号。前五个是必需的，因为 TEX 默认地把符号 (#, \$, %, &, _) 作特殊的用途。你不用为了使用这些符号而进入数学模式。

你可以用计算机现代意大利字体中的美元符号来得到英镑符号，例如：

例子：

\dag 在这边它只要花费你 \\$9.98，但在英格兰要 {\it \\$}24.98。

结果：

†在这边它只要花费你 \$9.98，但在英格兰要 £24.98。

² 译注：位于键盘的数字 3 之上，英式键盘为英镑符号，美式键盘为数字符号 #。

\TeX

这个符号可以得到 TeX 的标志。如果你需要在后面加上空格, 请在其后加上 `\u` 或者把它放到一个组中。

例子:

```
A book about \TeX\ is in your hands.
```

结果:

```
A book about \TeX is in your hands.
```

\dots

这个命令可以产生一个省略号, 也就是在文本中的三个小点。它必须使用在数学模式中; 如果你想在单词间加入省略号, 你应该使用 `\ldots` 命令 (第 213 页)。`\dots` 已经插入了它所应有的空白, 所以你不必在后面加上 `\u`。

例子:

```
数列 $x_1$, $x_2$, \dots, $x_\infty$
```

永远也不会结束

结果:

```
数列 $x_1$, $x_2$, \dots, $x_\infty$ 永远也不会结束
```

参阅: “各种常用数学符号” (第 197 页).

■ 任意字符

\char <charcode>

这个命令可以产生当前字体在 `<charcode>` 位置的字符。

例子:

```
{\char65} {\char 'A} {\char '\A}
```

结果:

```
A A A
```

\mathchar <mathcode>

这个命令可以产生由 `<mathcode>` 所指定的类, 族以及位置的数学字符。这个命令仅能用在数学模式中。

例子：

```
\def\digger{\mathchar "027F} % 像 plain TeX 的 \spadesuit.
% 0 类, 2 族, "7F 位置.
$\digger$
```

结果：



参阅: \delimiter (第 215 页)。

■ 重音符号

\'	锐音符 比如 é
\.	上点符 比如 ñ
\=	长音符 比如 ũ
\^	扬抑符 比如 ô
\`	钝音符 比如 è
\"	分音符 比如 ö
\~	波浪符 比如 ã
\c	软音符 比如 ç
\d	下点符 比如 ð
\H	匈牙利分音符 比如 ö
\t	连音符 比如 üü
\u	短音符 比如 ũ
\v	抑扬符 比如 ö

这些命令为普通文本产生重音记号。通常，你需要为其中以单个字母表示的音符后面留一个空格（参见“空格”，第 12 页）。

例子：

```
Add a soup\c con of \'elan to my pin\~a colada.
```

结果：

```
Add a soupçon of élán to my piñá colada.
```

```
\i  
\j
```

这些命令产生无点的字母 ‘i’ 和 ‘j’。在普通文本中，如果需要为 ‘i’ 和 ‘j’ 标记重音，就使用它们。在数学公式里，无点的 ‘i’ 和 ‘j’ 使用 `\imath` 和 `\jmath` 命令（第 197 页）。

例子：

```
long 'i' as in l\=\i fe \quad \v\j
```

结果：

```
long 'i' as in life j
```

`\accent <charcode>`

这个命令为后面的字符加上重音标记。重音是当前字体里 `<charcode>` 位置的字符。 \TeX 假定重音符号被设计的比同样字体里的字符高 1 ex。如果需要标记重音的字符过高或者过低， \TeX 自动调整重音位置。你可以在重音符号和其后的字符之间切换字体。如果重音符号被设计的很另类， \TeX 将不会产生警告；只是排印出一些稀奇古怪的结果。

例子：

```
l'H\accent94 otel des Invalides  
% Position 94 of font cmr10 has a circumflex accent.
```

结果：

```
l'Hôtel des Invalides
```

参阅：数学重音（第 209 页）。

■ 处理页边连写

`\noboundary`

某些时候， \TeX 对单词的开头或者末尾字符连写或者紧排。你可以紧接着单词前面或者后面放一个 `\noboundary` 来取消它们。有些非英文字体包含一个专门的边界字符，被 \TeX 放在单词的前后。这个边界字符不占空间，打印出来也不可见。它可以成为能被紧排或连写的一串字符的组成部分，所以有了它， \TeX 可以对单词的头尾做别致的处理。（标准的 \TeX 字体不包含此边界字符）`\noboundary` 的效果是删除存在的边界字符，从而防止 \TeX 识别连写或紧排。

选择字体

■ 特定字体

```
\fivebf   使用 5 点粗体
\fivei   使用 5 点数学意大利体
\fiverm  使用 5 点罗马体
\fivesy  使用 5 点数学符号字体
\sevenbf 使用 7 点粗体
\seveni  使用 7 点数学意大利体
\sevenrm  使用 7 点罗马体
\sevensy  使用 7 点数学符号字体
\tenbf   使用 10 点粗体
\tenex   使用 10 点数学扩展字体
\teni    使用 10 点数学意大利体
\tenrm   使用 10 点罗马体
\tensl   使用 10 点斜罗马体
\tensy   使用 10 点数学符号字体
\tenit   使用 10 点意大利体
\tentt   使用 10 点打字机字体
```

这些命令使 TeX 用指定的字体排版后面的文本。一般你可以把字体选择命令和相应的文本置于同一个编组中。编组之外使用一个字体选择命令直到文档最后。（除非中途用另一个类似的命令覆盖掉它）

例子：

```
See how I've reduced my weight---from  
120 lbs.\ to {\sevenrm 140 lbs}.
```

结果：

```
See how I've reduced my weight—from 120 lbs. to 140 lbs.
```

\nullfont

这个命令选择一种 TeX 内置的没有任何字符的空字体。在一族数学字体中，TeX 将用它取代其中没有定义的字体。

■ 字体风格

- ◊ \bf 粗体
- \it 意大利体
- \rm 罗马体
- \sl 斜体
- \tt 打字机字体

这些命令选择一种字体风格，而不改变字样或者字号。³ ⁴ 一般情况下，这些字体风格命令和相应的文本被一起包含在一个编组里。如果放在编组之外，效果将会作用于文档剩余部分（除非用另外一个这样的命令覆盖它）。

例子：

```
The Dormouse was {\it not} amused.
```

结果：

```
The Dormouse was not amused.
```

参阅：“数学公式字体”（第 220 页）。

大写和小写

```
\lccode <charcode> [ <number> table entry ]
\uccode <charcode> [ <number> table entry ]
```

\lccode 和 \uccode 的值为 256 个输入字符设定大小写的对应关系。
\lowercase 和 \uppercase 命令以及 TeX 的断词算法将会使用这些值。

TeX 这样初始化 \lccode 和 \uccode 的值：

- 小写字母的 \lccode 是它的 ASCII 代码。
- 大写字母的 \lccode 是它的小写形式的 ASCII 代码。
- 大写字母的 \uccode 是它的 ASCII 代码。
- 小写字母的 \uccode 是它的大写形式的 ASCII 代码。
- 非字母字符的 \lccode 和 \uccode 都是零。

³ 译注：注意区分字样 (typeface) 和字体 (font)，字样指一整套统一设计的字体。

⁴ TeX 并不提供预定义的命令来改变字号，比如，\eightpoint。要支持这样的命令，需要大量的字体，可能很多从来都不会用到。这样的命令曾被用来排版 The TeXbook。

在大多数时候，没有必要改变这些值，除非你所使用语言的字母比英文多。

例子：

```
\char\uccode{'s} \char\lccode{'a} \char\lccode{'M}
```

结果：

```
Sam
```

```
\lowercase {〈token list〉}
```

```
\uppercase {〈token list〉}
```

这些命令按照 `\lccode` 和 `\uccode` 的值转换 `〈token list〉` 中的字母（类别码为 11 的）为它们的大写和小写形式。非字母字符不受影响，即使它们是宏调用或其他展开成字母的命令。

例子：

```
\def\x{Cd} \lowercase{Ab\x} \uppercase{Ab\x}
```

结果：

```
abCd ABCd
```

单词间距

☆ \u

这个命令明确地产生一个单词间空格，称为“控制空格”。当一个字母出现在一个控制序列之后或者其他情况下你不想让两个记号排在一起时，就要用到控制空格。由 `\u` 产生的空白不受前面标点的影响，即其距离因子（第 107 页）为 1000。

如果你想打印出 ‘u’ 空格，可以输入 `\tt \char '\u'`.

例子：

```
The Dormouse was a \TeX\ expert, but he never let on.
```

结果：

```
The Dormouse was a TEX expert, but he never let on.
```

`\space`

这个命令等价于一个输入的空格字符。和 `\u` 不同，它的宽度会受到前面标点的影响。

例子：

```
Yes.\space No.\space Maybe.\par
Yes.\_No.\_Maybe.
```

结果：

```
Yes. No. Maybe.
Yes. No. Maybe.
```

`^^M`

这将产生行尾字符。通常有两个效果：

- 1) 类似一个命令，产生一个输入的空格（如果它出现在一个非空行的尾部）或者一个`\par`记号（如果出现在空行的尾部）。
- 2) 结束一行，使TeX忽略剩余的字符。

但是，如果以‘`\^M`’（即 control-M 的 ASCII 代码 13）出现，`^\^M` 不会结束一行。你可以修改类别码，从而赋予它新的含义。关于`^`更多的解释请参见第 57 页。

例子：

```
Hello.^MGoodbye.
Goodbye again.\par
The \char '\^M\ character.\par
% The fl ligature is at position 13 of font cmr10
\number '\^M\ is the end of line code.\par
Again, \number '\^M is the end of line code,
isn't it? % 32 is the ASCII code for a space
```

结果：

```
Hello. Goodbye again.
The fl character.
13 is the end of line code.
Again, 32isn't it?
```

☆ ~

活动字符‘`~`’称为“带子”(tie)，它产生一个正常的单词间空白，同时保证前后两个单词之间不会断行。只要断行会有歧义，就应该使

用带子。比如，在中间名之前，缩写词“Dr.”之后，或者“Fig.”之后：像“Fig. 8”。

例子：

```
P.D.Q.^Bach (1807--1742), the youngest and most
imitative son of Johann^S. Bach, composed the
{\sl Concerto for Horn and Hardart}.
```

结果：

```
P.D.Q. Bach (1807–1742), the youngest and most imitative son of
Johann S. Bach, composed the Concerto for Horn and Hardart.
```

◊ \v/

TEX 字体中每个字符都有“倾斜修正”。直立字体的倾斜修正一般就是零。当从倾斜的字体（不一定是意大利体）切换到直立的字体时，倾斜修正会增加额外的空白。这个额外的空白之所以必要，是因为倾斜的字符略微挤占了后面的空格位置，当后面跟着直立字体时，会显得中间的空格太小。所有字符的倾斜修正都包含在字体的度量文件里。

命令 \v/ 产生对前面字符的倾斜修正。当从倾斜字体切换到直立字体的时候，必须要插入倾斜修正，除非下一个字符是句点或逗号。

例子：

```
However, {\it somebody} ate {\it something}: that's clear.
```

```
However, {\it somebody\v/} ate {\it something\v/}:
that's clear.
```

结果：

```
However, somebody ate something: that's clear.
```

```
However, somebody ate something: that's clear.
```

```
\frenchspacing
\nonfrenchspacing
```

TEX一般会为了标点符号而调整单词间距。例如，一个句子的末尾会有额外的空白，句尾标点符号之后的粘连会被拉长。`\frenchspacing` 命令让 TEX调整单词间距时忽略标点符号；`\nonfrenchspacing` 命令让 TEX使用正常间距，即不使用 `\frenchspacing` 效果。

关于 TEX如何处理句尾标点，请参考第 14 页。

例子：

```
{\frenchspacing An example: two sentences. Right? No.\par}
{An example: two sentences. Right? No. \par}%
```

结果：

```
An example: two sentences. Right? No.
An example: two sentences. Right? No.
```

\obeyspaces

TeX 把连续的空格处理为一个空格。`\obeyspaces` 的作用是输入多少空格，就输出多少。但是 `\obeyspaces` 不会显示行首的空格。这时候推荐使用 `\obeywhitespace`，其定义在 `eplain.tex` 中（第 317 页）。`\obeyspaces` 主要用于以等宽字体打印计算机代码，以及显示所见即所得的内容。

`\obeylines` 命令（第 122 页）的作用是让 TeX 输出跟你所输入的一模一样的行。`\obeylines` 经常和 `\obeyspaces` 一起被使用。

例子：

```
These spaces are closed up
{\obeyspaces but these are not }.
```

结果：

```
These spaces are closed up but these are not .
```

```
\spacefactor [⟨number⟩ parameter]
\spaceskip [⟨glue⟩ parameter]
\xspaceskip [⟨glue⟩ parameter]
\sfcode ⟨charcode⟩ [⟨number⟩ table entry]
```

这些基本参数影响相邻单词之间的空白，即单词间距。默认的单词间距由当前字体决定。当 TeX 处理一个水平列表时，会监视 `\spacefactor` 间隔因子 f 。每当一个输入字符 c 被处理时， f 就会随 f_c (c 的间隔因子代码) 的值而更新。大多数的字符， f_c 是 1000，因此 TeX 设定 f 为 1000。（ f 的初始值也是 1000。）当 TeX 遇到一个单词间空格的时候，就会调整间隔的大小，给该间隔的伸长量和收缩量分别乘以 $f/1000$ 和 $1000/f$ 。所以：

- 1) 当 $f = 1000$ 时，单词间距为默认值。
- 2) 当 $f < 1000$ 时，单词间距有更少伸长量更多收缩量。
- 3) 当 $f > 1000$ 时，单词间距有更多伸长量更少收缩量。

另外，如果 $f \geq 2000$ ，当前字体的“额外空白”参数会让单词间距进一步增大。

每一个输入字符 c 对应于 `\sfcode` (间隔因子代码) 表中的一项。`\sfcode` 表项与字体无关。通常 TeX 处理完 c 之后就让 f 等于 f_c 。但是：

- 如果 f_c 为零, f 将不变。所以 plain TeX 中 f_c 为零的字符, 如 ‘)’, 对于单词间距的计算毫无影响。
- 如果 $f < 1000 < f_c$, TeX 会让 f 为 1000 而不是 f_c , 也就是, 不会让 f 快速地增加。

句号的 `\sfcode` 值通常是 3000, 所以 TeX 会在句号之后增加额外的空白。(参见上面 $f \geq 2000$ 的情形)。水平列表中的非字符, 比如竖直标线, 一般视为间隔因子为 1000 字符。

通过修改 `\spacefactor` 的值, 就可以显式地调整间隔因子。通过修改 `\xspaceskip` 或者 `\spaceskip` 的值, 你还可以覆盖默认的单词间距：

- `\xspaceskip` 设定 $f \geq 2000$ 时的粘连; 如果 `\xspaceskip` 是零, 就取默认值。
- `\spaceskip` 设定 $f < 2000$ 或者 `\xspaceskip` 为零时的粘连; 如果 `\spaceskip` 是零, 就取默认值。`\spaceskip` 粘连的伸长或收缩依 f 的值而变, 就像普通的单词间距一样。

The TeXbook 第 76 页对 TeX 计算单词间粘连的规则有一个具体的解释。*The TeXbook* 第 285–287 页详细描述了水平列表的各种项目之后 `\spacefactor` 的调整。

行的居中和平均分布

◊ `\centerline {argument}`
`\leftline {argument}`
`\rightline {argument}`

命令 `\centerline` 产生产生一个与当前行同宽的水平盒子, 并把 `{argument}` 在里面居中放置。命令 `\leftline` 和 `\rightline` 的作用类似：把 `{argument}` 放在盒子里靠左或者靠右的位置。如果想让几行内容都有同样的效果, 每一行都必须这样操作。其他的办法, 请看第 331 页。

这些命令不能用在段落里, 否则分段成行的时候, TeX 可能会遇到麻烦并警告盒子溢出了。

例子：

```
\centerline{Grand Central Station}
\leftline{left of Karl Marx}
\rightline{right of Genghis Khan}
```

结果：

left of Karl Marx	Grand Central Station	right of Genghis Khan
-------------------	-----------------------	-----------------------

◊ `\line <argument>`

这个命令产生一个 hbox 来包含 `<argument>`。hbox 的宽度和当前行宽相同，即从左边距延伸到右边距。

例子：

```
\line{ugly \hfil suburban \hfil sprawl}
% Without \hfil you'd get an ‘underfull box’ from this.
```

结果：

ugly	suburban	sprawl
------	----------	--------

`\llap <argument>`

`\rlap <argument>`

这些命令把文本覆盖到当前位置的左边或者右边。`\llap` 会向左退 `<argument>` 的宽度，然后开始排版 `<argument>`。`\rlap` 类似，但是先排版 `<argument>` 然后把后面的内容向左退。`\llap` 和 `\rlap` 主要用于把文本放在现在的页边距之外。它们的工作原理是创建一个零宽度的盒子。

`\llap` 和 `\rlap` 可以被用来以叠印的方式新建特殊字符。但是必须确保作原料的字符一般宽（等宽字体如 cmtt10 和 Computer Modern 10-point 的打字机字体是这样的）。

例子：

```
\noindent\llap{off left }\line{\vrule \$\Leftarrow\$%
left margin of examples\hfil right margin of examples
\$\$%\Rightarrow\$}\rlap{ off right}
```

结果：

off left \Leftarrow left margin of examples	right margin of examples \Rightarrow off right
---	--

参阅：`\hsize` (第 113 页)。

塑段

■ 段落的开始、结束和缩进

\par

这个命令结束一个段落，并使 TeX进入竖直模式来在页面上增加更多内容。因为 TeX把一个空行当作 \par 记号，所以不需要明确地键入 \par。

必须指出 \par 不会让 TeX开始新的一段；它仅仅告诉 TeX去结束一段。要使 TeX开始新段落，需要处于竖直模式之下（在 \par 之后已经是了），并遇到一个水平项目比如一个字母。为了开始新段落，TeX 将插入一些竖直空白，取决于 \parskip (第 144 页)；新段落缩进，取决于 \parindent (第 112 页)。

要想减小段落间 \par 产生的空白，通常可以使用命令 \vskip -\lastskip。当你需要写一个宏而让它与前面是不是空行无关的时候，这个命令就有用。

你可以使用\everypar (第 113 页) 来让 TeX在每开始一个新段落的时候做一些事。

\par 的具体效果请参阅 *The TeXbook* pages 283 and 286。

例子：

```
\parindent = 2em
‘‘Can you row?’’ the Sheep asked, handing Alice a pair of
knitting-needles as she was speaking.\par ‘‘Yes, a little%
---but not on land---and not with needles---’’ Alice was
starting to say, when suddenly the needles turned into oars.■
```

结果：

“Can you row?” the Sheep asked, handing Alice a pair of knitting-needles as she was speaking.

“Yes, a little—but not on land—and not with needles—” Alice was starting to say, when suddenly the needles turned into oars.

\endgraf

这个命令与\par 同义。有时候你重新定义了\par，但是又想取得 \par 的原始定义，这个命令就会有用。

\parfillskip [*glue* parameter]

这个参数设定 TeX 插入段落末尾的水平粘连。默认值是 0pt plus 1fil，这使得最后一行被空格填满。如果设成 0pt，TeX 就会把段落的最后一行拉伸到右边距。

☆ \indent

当 TeX 处于竖直模式时，比如段落结束以后，这个命令插入段落间粘连 \parskip，使 TeX 进入水平模式，开始新的一段，并缩进 \parindent 大小。如果 TeX 已经处于水平模式了，这个命令就只做缩进。一行中两个 \indent 产生两个缩进。

如下面的例子所示，新的段落开始之后的 \indent 是多余的。当 TeX 处在竖直模式下，遇到一个字母或其他水平模式中的命令时，就会切换到水平模式缩进 \indent，然后继续。

例子：

```
\parindent = 2em This is the first in a series of three
paragraphs that show how you can control indentation. Note
that it has the same indentation as the next paragraph.\par
\indent This is the second in a series of three paragraphs.
It has \indent an embedded indentation.\par
\indent\indent This doubly indented paragraph
is the third in the series.
```

结果：

This is the first in a series of three paragraphs that show how you can control indentation. Note that it has the same indentation as the next paragraph.

This is the second in a series of three paragraphs. It has an embedded indentation.

This doubly indented paragraph is the third in the series.

☆ \noindent

当 TeX 结束一段后处于竖直模式时，这个命令插入段落间粘连 \parskip，使 TeX 进入水平模式，然后开始新段，不要缩进。在水平模式中，比如段落中，这个命令没有效果。 \noindent 开始的新段会取消通常的 \parindent 缩进。

当新的段落是在某些独立显示的内容之后出现时，`\noindent` 通常用于取消段落第一行的缩进。

例子：

```
\parindent = 1em
Tied round the neck of the bottle was a label with the
words \smallskip \centerline{EAT ME}\smallskip
\noindent beautifully printed on it in large letters.
```

结果：

```
Tied round the neck of the bottle was a label with the words
EAT ME
beautifully printed on it in large letters.
```

`\textindent <argument>`

这个命令让 TeX 开始新段，正常缩进 `\parindent`。然后 TeX 在缩进的地方向右对齐排版 `<argument>` 并加入一个 en 大小的空白 (em 的一半)。Plain TeX 使用这个命令来排版脚注 (第 148 页) 以及列表项 (见 `\item`, 第 131 页)。

例子：

```
\parindent = 20pt \textindent{\raise 1pt\hbox{$\bullet$}}%
You are allowed to use bullets in \TeX even if
you don't join the militia, and many peace-loving
typographers do so.
```

结果：

- You are allowed to use bullets in TeX even if you don't join the militia, and many peace-loving typographers do so.

`\parindent [<dimen> parameter]`

这个参数设定段落第一行缩进的大小。如下面的例子所示，最好不要同时使 `\parindent` 和 `\parskip` 为零，否则两段的区别就不明显了。

例子：

```
\parindent = 2em This paragraph is indented by 2 ems.
\par \parindent=0pt This paragraph is not indented at all.
\par Since we haven't reset the paragraph indentation,
this paragraph isn't indented either.
```

结果：

This paragraph is indented by 2 ems.

This paragraph is not indented at all.

Since we haven't reset the paragraph indentation, this paragraph isn't indented either.

\everypar [*<token list>* parameter]

T_EX处于水平模式下，比如新段开始的时候，就会在〈记号列表〉中执行这个命令。默认的 \everypar 为空，但是你可以添加一些命令，每一段开始的时候在记号列表中执行并为 \everypar 分配记号列表。

例子：

```
\everypar = {${\Longrightarrow}\enspace}
Now pay attention!\par
I said, “Pay attention!”.\par
I'll say it again! Pay attention!
```

结果：

⇒ Now pay attention!
⇒ I said, “Pay attention!”.
⇒ I'll say it again! Pay attention!

■ 形成段落

◊ **\hsize** [*<dimen>* parameter]

这个参数设定当前的行宽，即从左边距开始算起，段落里一行的宽度。很多 T_EX命令，比如 \centerline (第 108 页) 和 \hrule (第 178 页)，都间接使用了 \hsize 的值。改变 \hsize 的值，就相应地改变了这些命令的宽度效果。

如果 \hsize 的变化发生在一个非空的竖直盒子里，盒子的宽度就是给定的 \hsize。

Plain T_EX默认 \hsize 为 6.5in。

例子：

```
{\hsize = 3.5in % Set this paragraph 3.5 inches wide.  
The hedgehog was engaged in a fight with another hedgehog,  
which seemed to Alice an excellent opportunity for  
croqueting one of them with the other.\par}%
```

结果：

The hedgehog was engaged in a fight with another hedgehog, which seemed to Alice an excellent opportunity for croqueting one of them with the other.

例子：

```
\leftline{\raggedright\vtop{\hsize = 1.5in  
Here is some text that we put into a paragraph that is  
an inch and a half wide.}\quad  
\vtop{\hsize = 1.5in Here is some more text that  
we put into another paragraph that is an inch and a  
half wide.}}
```

结果：

Here is some text that we put into a paragraph that is an inch and a half wide.	Here is some more text that we put into another paragraph that is an inch and a half wide.
--	---

☆ \narrower

这个命令产生较窄的段落，左右边距将会增加 \parindent（即当前段落缩进的大小），原因是 \leftskip 和 \rightskip 分别增加了 \parindent。通常 \narrower 被放在一个编组的开始，作用于其中的段落上。如果你忘记把 \narrower 封闭在编组里，就会发现文档所余部分全都变窄了。

\narrower 只影响其后的段落。如果包含 \narrower 的编组在段落末尾之前结束，TeX 将不会产生窄的段落。

例子：

```
{\parindent = 12pt \narrower\narrower\narrower  
This is a short paragraph. Its margins are indented  
three times as much as they would be  
had we used just one ‘‘narrower’’ command.\par}
```

结果：

This is a short paragraph. Its margins are indented three times as much as they would be had we used just one “narrower” command.

```
\leftskip [⟨glue⟩ parameter]  
\rightskip [⟨glue⟩ parameter]
```

这些参数告诉 TeX 在当前段落每一行的左右各放多大的空白。我们将只解释 `\leftskip`, 因为 `\rightskip` 与其类似。

你可以通过设置 `\leftskip` 为一个固定的非零尺寸来增加左边距。如果 `\leftskip` 是一个粘连, 就会得到右对齐的内容, 即左边距参差不齐。

通常, 你应该在一个编组里为 `\leftskip` 赋值, 来避免影响其余的文档部分。可是在一个段落内的编组里改变 `\leftskip` 的值是无意义的——只有段落末尾处 `\leftskip` 的值才会被 TeX 用来拆段成行。

例子：

```
{\leftskip = 1in The White Rabbit trotted slowly back  
again, looking anxiously about as it went, as if it had  
lost something. {\leftskip = 10in % has no effect  
It muttered to itself, ‘‘The Duchess! The Duchess! She’ll  
get me executed as sure as ferrets are ferrets!’’}\par}%
```

结果：

The White Rabbit trotted slowly back again, looking anxiously about as it went, as if it had lost something. It muttered to itself, “The Duchess! The Duchess! She’ll get me executed as sure as ferrets are ferrets!”

例子：

```
\pretolerance = 10000 % Don't hyphenate.
\rightskip = .5in plus 2em
The White Rabbit trotted slowly back again, looking
anxiously about as it went, as if it had lost something.
It muttered to itself, ‘‘The Duchess! The Duchess! She'll
get me executed as sure as ferrets are ferrets!’’
```

结果：

The White Rabbit trotted slowly back again, looking
 anxiously about as it went, as if it had lost something. It
 muttered to itself, “The Duchess! The Duchess! She'll get
 me executed as sure as ferrets are ferrets!”

☆ \raggedright
 \ttraggedright

这些命令让 TeX 以“左对齐”的方式排版文档，其中单词间距保持正常，即间距一样，不需伸缩。因此右边距是不均匀的。而 TeX 默认的则是均匀对齐，即左右边距一样。均匀对齐的文本单词间距将会有伸缩以产生均匀的右边距。有些排版者偏好使用左对齐。因为左对齐避免了页面上的空白参差不齐。

当使用等宽字体排版的时候，你需要用 \ttraggedright 命令；而使用其他字体时，需要用 \raggedright 命令。

通常你会希望对整篇文档使用这些命令，但是通过编组就可以把效果局限在部分文本中。

例子：

```
\raggedright ‘‘You couldn't have it if you {\it did\}/}
want it,’’ the Queen said. ‘‘The rule is, jam tomorrow
and jam yesterday---but never jam {\it today\}/.’’
‘‘It {\it must\}/} come sometimes to ‘jam today,%
thinspace’’ Alice objected. ‘‘No, it can't’’, said the
Queen. ‘‘It's jam every {\it other\}/} day: today isn't
any {\it other\}/} day.’’
```

结果：

“You couldn’t have it if you *did* want it,” the Queen said. “The rule is, jam tomorrow and jam yesterday—but never jam *today*.”

“It *must* come sometimes to ‘jam today,’ ” Alice objected. “No, it can’t”, said the Queen. “It’s jam every *other* day: today isn’t any *other* day.”

\hang

此命令使段落第二行开始各行也缩进 \parindent 大小 (第 112 页)。因为第一行已经缩进了 \parindent (除非你使用 \noindent 取消了缩进)，所以整个段落呈现悬挂缩进。

例子：

```
\parindent=24pt \hang ``I said you {\it looked} like an
egg, Sir,’’ Alice gently explained to Humpty Dumpty. ‘‘And
some eggs are very pretty, you know,’’ she added.
```

结果：

“I said you *looked* like an egg, Sir,” Alice gently explained to Humpty Dumpty. “And some eggs are very pretty, you know,” she added.

\hangafter [⟨number⟩ parameter]

\hangindent [⟨dimen⟩ parameter]

这两个参数一起决定段落的“悬挂缩进”。悬挂缩进使 TeX 产生某些行缩进而其他行依然正常的效果。 \hangafter 决定哪些行将被缩进； \hangindent 决定缩进的大小以及位置是左边还是右边：

- 设 \hangafter 的值为 n 。如果 $n < 0$, 开始的 $-n$ 行将被缩进。如果 $n \geq 0$, 除开始的 n 行以外的各行将被缩进。
- 设 \hangindent 的值为 x 。如果 $x \geq 0$, 左边将缩进 x ; 如果 $x < 0$, 右边将缩进 $-x$ 。

当你设置悬挂缩进的时候，仅对下一段（如果处于竖直模式）或者当前段（如果处于水平模式）有效。TeX 拆段成行时，使用的是段落末尾的 \hangafter 和 \hangindent 的值。

与其他塑段参数不同的是，\hangafter 和 \hangindent 在每一段的开始都被重设为其默认值，即 \hangafter 为 1, \hangindent 为 0。想要排版许多悬挂缩进的段落，可以使用 \everypar (第 113 页)。如

果你同时设定 `\hangafter`, `\hangindent` 以及`\parshape`, TeX将忽略 `\hangafter` 和 `\hangindent`。

例子：

```
\hangindent=6pc \hangafter=-2
This is an example of a paragraph with hanging indentation.■
In this case, the first two lines are indented on the left,■
but after that we return to unindented text.
```

结果：

This is an example of a paragraph with hanging indentation. In this case, the first two lines are indented on the left, but after that we return to unindented text.

例子：

```
\hangindent=-6pc \hangafter=1
This is another example of a paragraph with hanging
indentation. Here, all lines after the first have been
indented on the right. The first line, on the other
hand, has been left unindented.
```

结果：

This is another example of a paragraph with hanging indentation. Here, all lines after the first have been indented on the right. The first line, on the other hand, has been left unindented.

`\parshape n i1l1 i2l2 ... inln`

此命令指定一个段落前 n 行的形状。如果处于水平模式，该命令作用于当前段落；如果处于竖直模式，则作用于下一段落。其中的各个 i 和 l 都是尺寸。第 1 行的缩进为 i_1 ，长度为 l_1 ；第 2 行的缩进为 i_2 ，长度为 l_2 ；依此类推。若该段落超过 n 行，后面各行都使用 i_n 和 l_n 。为达到这里展示的特殊效果，你通常要反复调试，插入多个紧排，并且选择适合该形状的词语。

与 `\hangafter` 和 `\hangindent` 一样，`\parshape` 也仅对此段落有效。如果你同时指定 `\hangafter` 和 `\hangindent` 以及 `\parshape`, TeX 将忽略 `\hangafter` 和 `\hangindent`。

例子：

```
% A small font and close interline spacing make this work
\smallskip\font\sixrm=cmr6 \sixrm \baselineskip=7pt
\fontdimen3\font = 1.8pt \fontdimen4\font = 0.9pt
\noindent \hfuzz 0.1pt
\parshape 30 0pt 120pt 1pt 118pt 2pt 116pt 4pt 112pt 6pt
108pt 9pt 102pt 12pt 96pt 15pt 90pt 19pt 84pt 23pt 77pt
27pt 68pt 30.5pt 60pt 35pt 52pt 39pt 45pt 43pt 36pt 48pt
27pt 51.5pt 21pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt 53pt■
16.75pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt 53pt 16.75pt■
53pt 14.6pt 48pt 24pt 45pt 30.67pt 36.5pt 51pt 23pt 76.3pt
The wines of France and California may be the best
known, but they are not the only fine wines. Spanish
wines are often underestimated, and quite old ones may
be available at reasonable prices. For Spanish wines
the vintage is not so critical, but the climate of the
Bordeaux region varies greatly from year to year. Some
vintages are not as good as others,
so these years ought to be
s\kern -.1pt p\kern -.1pt e\kern -.1pt c\hfil ially
n\kern .1pt o\kern .1pt t\kern .1pt e\kern .1pt d\hfil:
1962, 1964, 1966. 1958, 1959, 1960, 1961, 1964,
1966 are also good California vintages.
Good luck finding them!
```

结果：

```
The wines of France and California
may be the best known, but they
are not the only fine wines. Span-
ish wines are often underestimated,
and quite old ones may be avail-
able at reasonable prices. For
Spanish wines the vintage is
not so critical, but the cli-
mate of the Bordeaux re-
gion varies greatly from
year to year. Some
vintages are not as
good as others,
so these years
ought to be
specially
noted:
1962,
1964,
1966,
1958,
1959,
1960,
1961,
1964,
1966
are also
good Cal-
ifornia vintages.
Good luck finding them!
```

`\prevgraf [⟨number⟩ parameter]`

在水平模式中，此参数表示该段落的当前行数；在竖直模式中，它表示上个段落的行数。只有在完成一些文本断行后，例如在陈列公式前或者在段落结尾处，*TEX* 才会设置 `\prevgraf`。详见 *The TEXbook* 第 103 页。

`\vadjust {⟨vertical mode material⟩}`

此命令在当前位置所在行下边插入指定的 *⟨vertical mode material⟩*。例如，你可以用它输出页面，或者在特定行添加额外空白。

例子：

```
Some of these words are \vadjust{\kern8pt\hrule} to be
found above the line and others are to be found below it.
```

结果：

```
Some of these words are to be found above the line and others are
to be found below it.
```

参阅：`\parindent` (第 112 页), `\parskip` (第 144 页), `\everypar` (第 113 页). ■

断行

■ 鼓励或阻碍断行

\break

强制在此处断行。除非用某种方法填满该行，否则你将得到“underfull hbox”的警告。此命令也能用于竖直模式。

例子：

```
Fill out this line\hfil\break and start another one.\par
% Use \hfil here to fill out the line.
This line is underfull---we ended it\break prematurely.
% This line causes an ‘underfull hbox’ complaint.
```

结果：

```
Fill out this line
and start another one.
This line is underfull—we ended it
prematurely.
```

\nobreak

阻止在此处断行。此命令也能用于竖直模式。

例子：

```
Sometimes you'll encounter a situation where
a certain space\nobreak\qquad must not get lost.
```

结果：

```
Sometimes you'll encounter a situation where a certain space must
not get lost.
```

\allowbreak

在通常不断行的地方允许 TeX 断行。它经常用于数学公式中，因为 TeX 通常不愿意在其中断行。此命令也能用于竖直模式。

例子：

```
Under most circumstances we can state with some confidence
that $2+2\allowbreak=4$, but skeptics may disagree.
\par For such moronic automata, it is not difficult to
analyze the input/\allowbreak output behavior in the limit.■
```

结果：

```
Under most circumstances we can state with some confidence that
 $2 + 2 = 4$ , but skeptics may disagree.
For such moronic automata, it is not difficult to analyze the input/
output behavior in the limit.
```

\penalty <number>

此命令生成一个惩罚 (penalty) 项。惩罚项使得 TeX 或多或少愿意在此处断行。负惩罚值，实际上是奖励值，鼓励断行；正惩罚值阻碍断行。大于或等于 10000 的惩罚值彻底阻止断行，而小于或等于 -10000 的惩罚值强制断行。此命令也能用于竖直模式。

例子：

```
\def\break{\penalty -10000 } % as in plain TeX
\def\nobreak{\penalty 10000 } % as in plain TeX
\def\allowbreak{\penalty 0 } % as in plain TeX
```

\obeylines

TeX 通常将行结束符视为空格。`\obeylines` 让 TeX 将每个行结束符都视为段落结束符，从而强制断行。在排版诗歌或者程序代码时 `\obeylines` 大有用处。然而，如果某些行的长度超过行的实际长度 (`\hsize - \parindent`)，它们将会被自动断行。

TeX 将在设定了 `\obeylines` 的各行间插入 `\parskip` (第 144 页) 粘连 (既然它认为每行都是一个段落)。因此，在使用 `\obeylines` 时一般应该设定 `\parskip` 为零。

用 `\obeyspaces` 命令 (第 107 页) 可以让 TeX 按原样处理行中的空格。我们经常一起使用 `\obeylines` 和 `\obeyspaces`。

例子：

```
\obeylines
‘‘Beware the Jabberwock, my son!
\quad The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
\quad The frumious Bandersnatch! ’’
```

结果：

```
"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"
```

◊ \slash

此命令生成一个斜线符号 (/)，并且告诉 TeX 必要时可以在该斜线后断行。

例子：

```
Her oldest cat, while apparently friendly to most people,
had a Jekyll\slash Hyde personality when it came to mice.
```

结果：

```
Her oldest cat, while apparently friendly to most people, had a
Jekyll/Hyde personality when it came to mice.
```

■ 断行参数

```
\pretolerance [⟨number⟩ parameter]
\tolerance [⟨number⟩ parameter]
```

在 TeX 选择段落断行点时，这两个参数确定对各行劣度 (badness) 的容许值 (tolerance)。劣度衡量了单词间距和理想情形差别多大⁵。其中 \pretolerance 指定在不加连字符断行时对劣度的容许值，而 \tolerance 指定加上连字符断行时对劣度的容许值。当某行排得太紧密 (单词间距过小)，或者排得太松散 (单词间距过大) 时，劣度就会超过容许值。

- 如果 TeX 将某一行排得太松散，它将给出 “underfull hbox” 的警告。
- 如果 TeX 将某一行排得太紧密，它将让该行超出右边缘，并给出 “overfull hbox” 的警告。

TeX 按照如下步骤选择断行点：

- 1) 它试图选择无需添加连字符的一系列断点。如果结果中每行的劣度都不超过 \pretolerance，这组断点是可接受的，该段落也就完成了断行。

⁵ 译注：在中文排版中，“单词空隙”实际上是字间距。

- 2) 否则, 它在允许添加连字符的情况下再次寻找断点。如果结果中每行的劣度都不超过 `\tolerance`, 这组新断点是可接受的, 该段落也就完成了断行。
- 3) 否则, 它给每行的伸长量增加 `\emergencystretch` (见下面), 然后重新尝试。
- 4) 如果上述各种尝试都无法得到可接受的断点系列, 它给该段落设置一个或多个溢出的水平盒子, 并给出警告。

Plain TeX 设置 `\tolerance` 为 200, `\pretolerance` 为 100。若你设置 `\tolerance` 为 10000, TeX 将有无穷大的容许值, 可以接受任何空隙, 不管它有多糟糕。(除非它遇到即使添加连字符也无法放在一行的单词)。因此, 通过改变 `\tolerance` 值, 你可以避免溢出的或者松散的水平盒子, 但会得到更糟糕的空隙。通过增大 `\pretolerance` 值你可以让 TeX 不添加连字符 (并且运行得更快), 但同样可能得到更糟糕的空隙。而如果你设置 `\pretolerance` 为 -1, TeX 将不再尝试不添加连字符的断行方式。

参数 `\hbadness` (第 175 页) 确定劣度达到多大时 TeX 才给出警告, 但它不影响 TeX 对文档的排版。参数 `\hfuzz` (第 176 页) 确定水平盒子宽度超出多大时, TeX 才认为是错误的。

`\emergencystretch` [$\langle\text{dimen}\rangle$ parameter]

设定此参数大于零, 可以让 TeX 更容易排版文档, 而不是生成溢出的水平盒子。这样比设定 `\tolerance=10000` 更好, 因为那样往往得到十分难看的行。如果 TeX 排版段落时无法不超过 `\tolerance` 值, 它将给每行的伸长量增加 `\emergencystretch` 然后重新尝试。伸长量的增加将会缩减各行劣度, 允许 TeX 生成比原来更宽的间距, 从而选出当前情况下尽可能好的断行点。

`\looseness` [$\langle\text{number}\rangle$ parameter]

此参数用于修改段落的行数, 相对其最佳行数。之所以称为松散度 (`\looseness`), 是因为它衡量该段落有多松散, 即包含多少额外的间距。

一般地, 若 `\looseness` 等于 0, TeX 将按通常方式选择断行点。但若 `\looseness` 等于, 比如 3, TeX 依下面步骤处理:

- 1) 按通常方式选择断行点, 得到总行数为 n 的段落。
- 2) 丢弃这些断行点, 并试着寻找总行数为 $n + 3$ 的一系列新断点。(缺少上一步, TeX 就无法知道 n 的值。)

- 3) 如果上一步尝试无法得到行劣度不超过 `\tolerance` 的结果, 它将试着将段落分为 $n+2$ 行; 如果这也不行, 试着分为 $n+1$ 行; 最终只能再次分为 n 行。

类似地, 若松散度为 $-k$, TeX 试着让段落的行数比正常情形少 k 行。让段落多出一行的最简单方法是, 将一个单词放到多出的那行。在最后两个单词之间加上一个带子 (tie, 第 105 页) 符号, 你就可以阻止此种断行方法。

设定 `\looseness` 是迫使段落占用给定行数的最好方法。想增加页面中的文本时, 你可以将它设为负值。同样地, 想减少页面中的文本时, 可以将它设为正值。

在结束段落并分段为行之后, TeX 将 `\looseness` 重置为 0。若要改动多个段落的松散度, 你必须对每个段落分别设置, 或者将改动放在 `\everypar` (第 113 页) 命令中。

`\linepenalty` [$\langle number \rangle$ parameter]

此参数设定 TeX 分段为行时各行的惩罚值。此惩罚值与断行位置无关。增大此参数将让 TeX 把段落压缩到最小的行数, 但会损害其它审美上的考虑, 比如避免过紧的单词间距。Plain TeX 设定 `\linepenalty` 为 10。

`\adjdemerits` [$\langle number \rangle$ parameter]

此参数给行断点设定额外的缺陷 (demerit), 只要该断点在“视觉不相容的”相邻两行间出现。这些相邻行使得段落看起来参差不齐。不相容性取决于各行的松紧度⁶:

- 1) 此行是过紧的, 如果其粘连至少需要收缩 50%。
- 2) 此行是适中的, 如果它的劣度小于或等于 12。
- 3) 此行是松散的, 如果其粘连至少需要伸长 50%。
- 4) 此行是空荡的, 如果其粘连需要伸长太多以致它的劣度超过 100。

如果相邻的两行分类却不相近, 比如, 松散的行后面跟着一个过紧的行, 或者空荡的行后面是一个适中的行, 则称它们为视觉不相容的。

缺陷以劣度的平方为单位, 因此只有在给定一个较大的数值时 (比如数千) 才会产生作用。Plain TeX 设定 `\adjdemerits` 为 10000。

⁶ 译注: 如果粘连调整比例大于或等于 50%, 其劣度必定大于或等于 13。

`\exhyphenpenalty [⟨number⟩ parameter]`

此参数指定 TeX 在显式连字符（比如“helter-skelter” 的连字符）处断行时所附加的惩罚值。增大此参数将阻碍 TeX 在显式连字符处换行。Plain TeX 设定 `\exhyphenpenalty` 为 50。

`\hyphenpenalty [⟨number⟩ parameter]`

此参数指定 TeX 在隐式连字符处断行时所附加的惩罚值。隐式连字符来自 TeX 的连字词典，或者来自你用 `\-` (第 127 页) 插入的任意连字符。增大此参数将阻碍 TeX 将单词连字化。Plain TeX 设定 `\hyphenpenalty` 为 50。

`\doublehyphendemerits [⟨number⟩ parameter]`

此参数指定 TeX 在导致连续两行都以连字符结尾的断点处所附加的额外缺陷值。增大此参数将阻碍 TeX 将连续两行连字化。缺陷以劣度的平方为单位，因此只有在给定一个较大的数值时（比如数千）才会产生作用。Plain TeX 设定 `\doublehyphendemerits` 为 10000。

`\finalhyphendemerits [⟨number⟩ parameter]`

此参数指定 TeX 在段落倒数第二行以连字符结尾时所附加的额外缺陷值。由于该行下面可能有短行造成的空白，这样的连字符通常认为是缺乏美感的。增大此参数将阻碍 TeX 将段落倒数第二行连字化。缺陷以劣度的平方为单位，因此只有在给定一个较大的数值时（比如数千）才会产生作用。Plain TeX 设定 `\finalhyphendemerits` 为 5000。

`\binoppenalty [⟨number⟩ parameter]`

此参数指定 TeX 在段内公式的二元运算符后断行所附加的惩罚值。Plain TeX 设定 `\binoppenalty` 为 700。

`\relpenalty [⟨number⟩ parameter]`

此参数指定 TeX 在段内公式的二元关系符后断行所附加的惩罚值。Plain TeX 设定 `\relpenalty` 为 500。

■ 连字

◊ \-

此命令在单词中加入“自定连字符”。自定连字符允许 TeX 在该处连字化。但 TeX 未必非得如此连字——它仅在必要时这样做。当某个单词在文档中出现一次或多次，而 TeX 又找不到合适连字点时，这个命令就比较有用了。

例子：

```
Alice was exceedingly reluctant to shake hands first  
with either Twee\-\dle\-\dum or Twee\-\dle\-\dee, for  
fear of hurting the other one's feelings.
```

结果：

```
Alice was exceedingly reluctant to shake hands first with either Twee-  
dledum or Tweedledee, for fear of hurting the other one's feelings.
```

```
\discretionary{<pre-break text>} {<post-break text>} {<no-break text>}
```

此命令指定一个“自定断点”，即，TeX 可以断行的位置。它还告诉 TeX 在断点前后放置的文字。

- 如果 TeX 不在该处断行，它使用 *<no-break text>*。
- 如果 TeX 确实在该处断行，它把 *<pre-break text>* 放在断点前，而把 *<post-break text>* 放在断点后。

如同 \-, 在自定断点处 TeX 未必非得断行。实际上，\- 通常等同于 \discretionary{-}{ }{ }。

TeX 有时也会加入自己的自定断点。例如，它在显式连字号或者破折号后加入 \discretionary{}{}{}。

例子：

```
% An ordinary discretionary hyphen (equivalent to \-):
\discretionary{-}{ }{ }

% A place where TeX can break a line, but should not
% insert a space if the line isn't broken there, e.g.,
% after a dash:
\discretionary{}{ }{ }

% Accounts for German usage: 'flicken', but 'flik-
% ken':
German ``fli\discretionary{k-}{k}{ck}en''
```

\hyphenation { <word> | ... | <word> }

TeX 中有个词典记录了它的连字规则的例外情形。词典中每个条目指明了某个单词应该如何连字化。这个 \hyphenation 命令用于添加单词到该词典。它的参数是用空格分开的多个单词，不区分字母大小写。单词中的连字符指明 TeX 可以在哪些地方将它连字化。一个不含连字符的单词将永远不会被连字化。然而，在特定单词中的 \- 还是优先于连字词典中对该单词的规定。你需要提供单词的各种语法形式给 TeX 处理，比如单数和复数形式。

例子：

```
\hyphenation{Gryphon my-co-phagy}
\hyphenation{man-u-script man-u-scripts piz-za}
```

\uchyph [<number> parameter]

取 \uchyph (大写单词连字化) 为正数，将允许对专有名词等以大写字母开始的单词连字化。取为零或者负数将禁止此类连字。Plain TeX 设定 \uchyph 为 1，因此 TeX 一般会试着将大写字母开始的单词连字化。

\showhyphens { <word> | ... | <word> }

此命令在文档中一般不会用到，使用它你可以在终端输出中看到 TeX 对一些单词是如何连字的。用连字符表示的单词将出现在编译记录和终端输出中。同时你也会看到关于“松散的盒子”的警告，忽略它即可。

例子：

```
\showhyphens{threshold quizzical draughts argumentative}
```

结果（在日志中）：

```
Underfull \hbox (badness 10000) detected at line 0
[] \tenrm thresh-old quizzi-cal draughts ar-gu-men-ta-tive
```

\language [⟨number⟩ parameter]

不同的语言有不同的连字规则。此参数确定 TeX 使用哪组连字规则。通过改变 **\language**，你可以让 TeX 对部分或整个文档用特定语言的连字规则连字。你所用的 TeX 会说明是否还有另外的连字规则可用（除了英语），以及适当的 **\language** 值是哪些。默认的 **\language** 值为 0。

在段落开始处，TeX 设定当前语言值为 0。在添加字符到当前段落时，TeX 会比较 **\language** 和当前语言值。如果两者不同，TeX 添加一个小玩意，表明当前语言的变化。这个小玩意提示后续处理时所用的语言规则应该改变。

\setlanguage ⟨number⟩

此命令通过添加小玩意设定当前语言值为 ⟨number⟩。该小玩意和修改 **\language** 时的一样。然而，此命令不会改动 **\language** 的值。

\lefthyphenmin [⟨number⟩ parameter]
\righthyphenmin [⟨number⟩ parameter]

这两个参数指定，对于连字单词，TeX 在连字符左右至少得有几个字母。Plain TeX 中两者的默认值分别为 2 和 3；这是英语的建议值。

\hyphenchar ⟨font⟩ [⟨number⟩ parameter]

TeX 未必总是以 ‘-’ 字符为连字符。实际上，它使用的是当前字体的 **\hyphenchar**，此字符通常是 ‘-’ 但未必总是。若某字体的 **\hyphenchar** 值为负数，TeX 将不会对该字体下的单词连字化。

注意 ⟨font⟩ 指的是该字体的控制序列名称，而不是字体文件名 ⟨font-name⟩。谨记：在编组结束时所分配的 **\hyphenchar** 不会撤销。若要局部改变 **\hyphenchar**，你必须显式地保存和恢复原有取值。

例子：

```
\hyphenchar\tenrm = '-'
  % Set hyphenation for tenrm font to '-'.
\hyphenchar\tentt = -1
  % Don't hyphenate words in font tentt.
```

`\defaulthyphenchar [⟨number⟩ parameter]`

当 TeX 碰到 `\font` 命令时，它读取该字体的度量文件，并设置该字体的 `\hyphenchar` 为 `\defaulthyphenchar`。如果载入字体时 `\defaulthyphenchar` 的值不在 0–255 范围中，TeX 将不会对该字体下的任何单词连字化；除非你以后用 `\hyphenchar` 命令设定此字体的连字符。Plain TeX 设定 `\defaulthyphenchar` 为 45，即 ‘-’ 字符的 ASCII 码。

例子：

```
\defaulthyphenchar = '-'
  % Assume ‘-’ is the hyphen, unless overridden.
\defaulthyphenchar = -1
  % Don't hyphenate, unless overridden.
```

参阅：`\pretolerance` (第 123 页)。

分节、列表和定理

◊ `\begin{section} ⟨argument⟩ \par`

此命令开始文档的一节。`⟨argument⟩` 用于表示节标题。标题将用粗体显示，靠左对齐，而且上下有额外的竖直间距。你也可以在 `⟨argument⟩` 后用一个空行得到 `\par`。

例子：

```
$\ldots$ till she had brought herself down to nine
inches high.
```

```
\begin{section} Section 6. Pig and Pepper
```

```
For a minute or two she stood looking at the house $\ldots$ ■
```

结果：

... till she had brought herself down to nine inches high.

Section 6. Pig and Pepper

For a minute or two she stood looking at the house ...

```
\item <argument>
\itemitem <argument>
```

这两个命令用于创建逐项列表。在 *<argument>* 后面的整个段落会有 *\parindent* (对于 *\item*) 或者 2*\parindent* (对于 *\itemitem*) 的缩进。(*\parindent* 的解释见第 112 页。) *<argument>* 加上 1en 的间距后被放置在段落首行文字的左边。也就是说，它总是落在由 *\parindent* 确定的段落缩进空白中。

要让一个列表项包含多个段落，可以将 *\item{}* 放在另外的段落前。

例子：

```
{\parindent = 18pt
\noindent Here is what we require:
\item{1.}Three eggs in their shells,
but with the yolks removed.
\item{2.}Two separate glass cups containing:
\itemitem{(a)}One-half cup {\it used} motor oil.
\itemitem{(b)}One cup port wine, preferably French.
\item{3.}Juice and skin of one turnip.}
```

结果：

Here is what we require:

1. Three eggs in their shells, but with the yolks removed.
2. Two separate glass cups containing:
 - (a) One-half cup *used* motor oil.
 - (b) One cup port wine, preferably French.
3. Juice and skin of one turnip.

◊ *\proclaim* <argument>. □ <general text> \par

这个命令“陈述”一个定理、引理、假设等等。它用粗体显示 *<argument>*，并用斜体显示后面的段落。*<argument>* 后面必须跟着句号和空格，以此区分开 *<argument>* 和 *<general text>*。*<general text>* 到下

一个段落之前截止。若要包含多个段落，你可以用 `\endgraf` 而不是空行或 `\par` 结束段落。

例子：

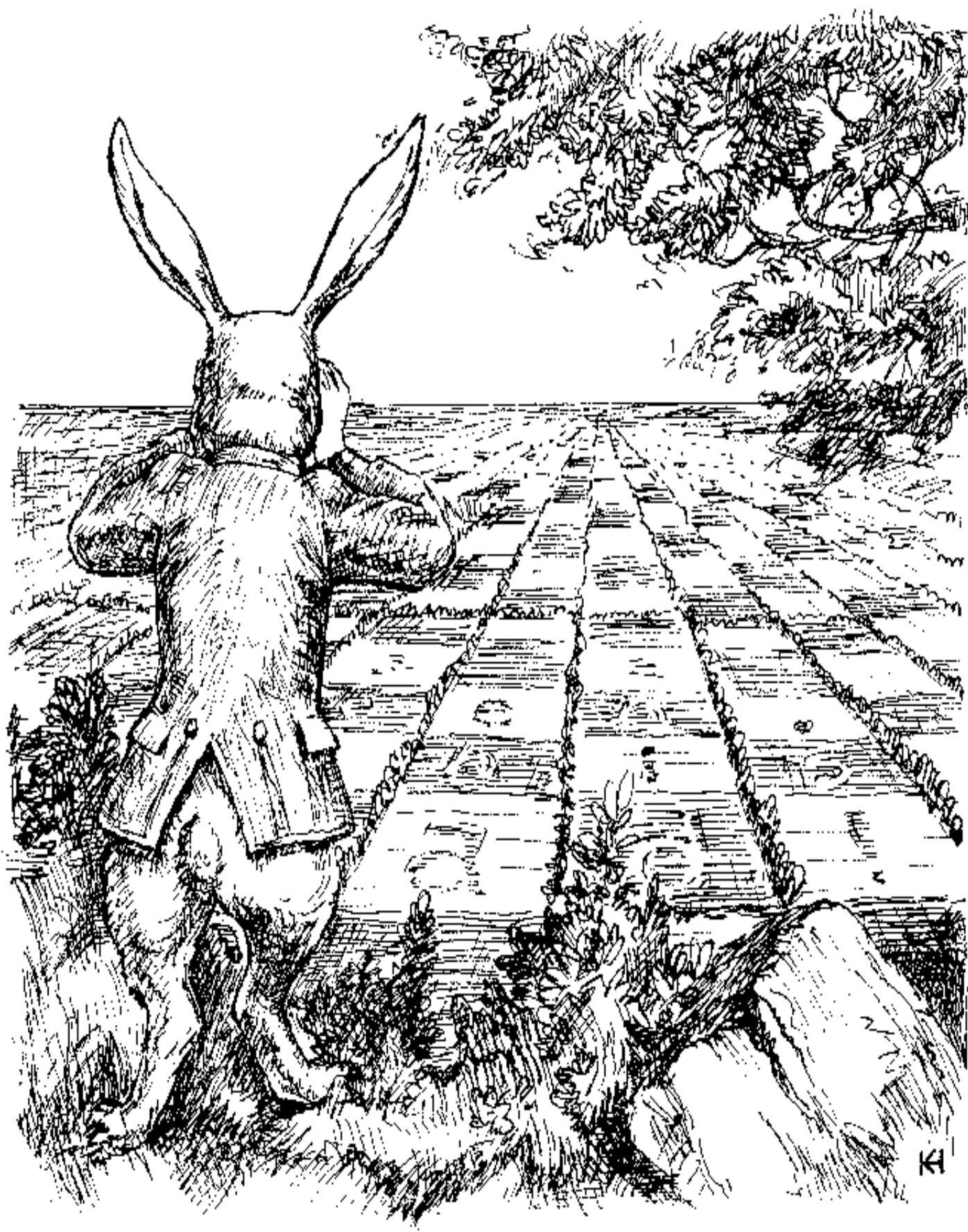
```
\proclaim Theorem 1.  
What I say is not to be believed.
```

```
\proclaim Corollary 1. Theorem 1 is false.\par
```

结果：

Theorem 1. *What I say is not to be believed.*

Corollary 1. *Theorem 1 is false.*



6 | 组页命令

这个章节介绍了与页，页的组成部分以及输出程序相关的命令。本章写作惯例的说明可以在“命令描述”（第 3 页）中找到。

行间距和段间距

```
\baselineskip    [〈glue〉 parameter]  
\lineskiplimit [〈dimen〉 parameter]  
\lineskip     [〈glue〉 parameter]
```

在一个普通竖直列（例如段落各行）中，这三个参数共同确定了 TeX 在相邻两个盒子间插入的间隔。这个间隔称为“行间粘连”。对于内部竖直模式中构造的竖直盒子，其中的组成盒子间同样会有此间隔。

在通常情况下，盒子的高度和深度不会太大，此时 TeX 让相邻盒子的两基线距离等于 `\baselineskip`。为此可以让行间粘连等于 `\baselineskip` 减去上盒子深度（由 `\prevdepth` 给出）和下盒子高度。但如果两个盒子靠得太近，即行间粘连小于 `\lineskiplimit`，TeX 将改为插入 `\lineskip` 大小的间距。¹ 详情可见 *The TeXbook* 第 79–80 页。

注意 `\baselineskip` 和 `\lineskip` 测量不同的东西：前者为两盒子基线之间的距离，而后者为上盒子底部和下盒子顶部的距离。详情可看 *The TeXbook* 第 78 页。下面第一个例子展示了 `\lineskiplimit` 的作用。

¹ TeX 实际上在竖直列开头设定 `\prevdepth` 为 -1000 点，从而可以测定每个盒子的 `\prevdepth`。若 `\prevdepth \leq -1000` 点，它就不会插入任何行间粘连。

如下面第二个例子所示，将 `\baselineskip` 加倍，就能得到双倍行距的效果。在段落结束前任何位置改变 `\baselineskip`，都会影响整个段落。

例子：

```
\baselineskip = 11pt \lineskiplimit = 1pt
\lineskip = 2pt plus .5pt
Sometimes you'll need to typeset a paragraph that has
tall material, such as a mathematical formula, embedded
within it. An example of such a formula is $n \choose k$.
Note the extra space above and below this line as
compared with the other lines.
(If the formula didn't project below the line,
we'd only get extra space above the line.)
```

结果：

Sometimes you'll need to typeset a paragraph that has tall material, such as a mathematical formula, embedded within it. An example of such a formula is $\binom{n}{k}$. Note the extra space above and below this line as compared with the other lines. (If the formula didn't project below the line, we'd only get extra space above the line.)

例子：

```
\baselineskip = 2\baselineskip % Start double spacing.
```

`\prevdepth` [⟨dimen⟩ parameter]

当 TeX 将盒子添加到竖直列时，它设定 `\prevdepth` 为该盒子的深度。在竖直列的开始，TeX 设定 `\prevdepth` 为 -1000 点，这表示不添加行间粘连。

```
\normalbaselineskip [⟨glue⟩ parameter]
\normallineskiplimit [⟨dimen⟩ parameter]
\normallineskip [⟨glue⟩ parameter]
\normalbaselines
```

前三个参数分别包含 `\baselineskip`, `\lineskip` 和 `\lineskiplimit` 的值。而 `\normalbaselines` 命令将 `\baselineskip`, `\lineskip` 和 `\lineskiplimit` 分别设定为这三个参数的值。

`\offinterlineskip`

此命令让 TeX 从此之后停止插入行间粘连。除非你希望对文档的剩余部分都生效（你多半不希望），你应该将此命令和你希望生效的文本放

在一个编组中。它的主要用处在于可以让你自己处理行间空隙，比如用支架 (struts)，而不影响 TeX 的正常行间粘连。在构造水平阵列时经常会用到 `\offinterlineskip`。

例子：

```
\def\entry#1:#2 {\strut\quad#1\quad&\quad#2\quad\cr}
\offinterlineskip \tabskip = 0pt \halign{%
\vrule\quad\hfil#\hfil\quad\vrule&
\quad\hfil#\hfil\quad\vrule\cr
\noalign{\hrule}
\vphantom{\vrule height 2pt}\cr \noalign{\hrule}
\entry \it Opera:\it Composer
\vphantom{\vrule height 2pt}\cr \noalign{\hrule}
\vphantom{\vrule height 2pt}\cr
\entry Fidelio:Beethoven
\entry Peter Grimes:Britten
\entry Don Giovanni:Mozart
\vphantom{\vrule height 2pt}\cr \noalign{\hrule}}
```

结果：

<i>Opera</i>	<i>Composer</i>
Fidelio	Beethoven
Peter Grimes	Britten
Don Giovanni	Mozart

`\nointerlineskip`

此命令让 TeX 停止在下一行插入行间粘连。它不影响后续其他行。

`\openup <dimen>`

此命令让 `\baselineskip` 增加 `<dimen>`。在段尾之前的 `\openup` 命令将影响整个段落，因此想在段落中间改变 `\baselineskip` 时不要用 `\openup` 命令。在排版表格和陈列公式时 `\openup` 非常有用——多点间隔将会增加它们的可读性。

例子：

```
Alice picked up the White King very gently, and lifted him  
across more slowly than she had lifted the Queen; but before■  
she put him on the table, she thought she might well dust  
him a little, he was so covered with ashes.  
\openup .5\baselineskip % 1.5 linespacing.
```

结果：

```
Alice picked up the White King very gently, and lifted him across  
more slowly than she had lifted the Queen; but before she put him  
on the table, she thought she might well dust him a little, he was so  
covered with ashes.
```

分页

■ 鼓励或阻碍分页

\break

此命令强制在此处分页。除非用某种方式填满该页，你将得到一个松散的 vbox。 \break 也能用在水平模式中。

\nobreak

此命令阻止在此处分页。 \nobreak 也能用在水平模式中。

\allowbreak

在通常不分页的地方允许 T_EX 分页。 \allowbreak 也能用在水平模式中。

\penalty <number>

此命令生成一个惩罚 (penalty) 项。惩罚项使得 T_EX 或多或少愿意在此处分页。负惩罚值，实际上是奖励值，鼓励分页；正惩罚值阻碍分

页。大于或等于 10000 的惩罚值彻底阻止分页，而小于或等于 -10000 的惩罚值强制分页。此命令也能用于水平模式。

例子：

```
\def\break{\penalty-10000 } % as in plain TeX
\def\nobreak{\penalty10000 } % as in plain TeX
\def\allowbreak{\penalty0 } % as in plain TeX
```

`\goodbreak`

此命令结束当前段落，并告诉 TeX 这里是一个合适的分页位置。

```
\smallbreak
\medbreak
\bigbreak
```

这些命令指明了越来越合适的分页位置。如果实际上没在此处分页，TeX 将分别插入大小为 `\smallskip`, `\medskip` 或 `\bigskip` 的间距（第 158 页）。但如果此位置之前已经有大于或等于此值的间距，TeX 就不会再插入间距。

◊ `\eject`

`\supereject`

这两个命令将结束当前段落并强制在当前位置分页。如果你不在前面加上 `\vfil`（第 161 页），TeX 将试着将伸展页面内容（很可能会警告有松散的竖直盒子）。`\supereject` 命令还让 plain TeX 输出例行程序处理剩下的插入项，例如长脚注，在处理其他输入之前生成它们。因而，`\supereject` 更适合在文档各章或主要单元结束时使用。

`\filbreak`

此命令规定了有条件的分页。它告诉 TeX，如果后面的 `\filbreak` 不能放在同一页，就在此处分页。用一对 `\filbreak` 将一个段落围起来，就可以确保 TeX 尽量将该段落放在同一页。不要在段落中间使用 `\filbreak`，因为这将让 TeX 进入竖直模式，从而结束该段落。在第 285 页中有此主题的更多建议。

```
\raggedbottom
\normalbottom
```

通常 TeX 会尽全力让每页都一样高，即让它们的下边距相等。`\raggedbottom` 命令允许 TeX 给不同页面的下边距一些差别。如果

文档中有些不能跨页的大型内容，用 `\raggedbottom` 就比较合适。用 `\normalbottom` 命令可以取消 `\raggedbottom` 的效果。

■ 分页参数

`\interlinepenalty [⟨number⟩ parameter]`

此参数设定在段落内部各行间分页时的惩罚值。将它设为 10000 将强制在段落之间分页，只要页面中有足够的伸展值让 TeX 排版。Plain TeX 设定 `\interlinepenalty` 为 0。

`\clubpenalty [⟨number⟩ parameter]`

此参数设定在段落首行之后分页的惩罚值。页面底部的单独一行称为“孤行”。Plain TeX 设定 `\clubpenalty` 为 150。

`\widowpenalty [⟨number⟩ parameter]`

此参数设定在段落尾行之前分页的惩罚值。页面顶部的单独一行称为“寡行”。Plain TeX 设定 `\widowpenalty` 为 150。

`\displaywidowpenalty [⟨number⟩ parameter]`

此参数设定在陈列公式上一行同时也是段落尾行之前分页的惩罚值。Plain TeX 设定 `\displaywidowpenalty` 为 50.

`\predisplaypenalty [⟨number⟩ parameter]`

此参数设定在陈列公式之前分页的惩罚值。Plain TeX 设定 `\predisplaypenalty` 为 10000。

`\postdisplaypenalty [⟨number⟩ parameter]`

此参数设定在陈列公式之后分页的惩罚值。Plain TeX 设定 `\postdisplaypenalty` 为 0。

`\brokenpenalty [⟨number⟩ parameter]`

此参数设定在以自定连字项（通常是一个连字符）结尾的一行之后分页的惩罚值。`\brokenpenalty` 用于分页，而 `\hyphenpenalty` (第 126 页) 用于断行。Plain TeX 设定 `\brokenpenalty` 为 100.

`\insertpenalties [⟨number⟩ parameter]`

此参数包含 TeX 放置插入项到当前页面时所累积的某些惩罚值之和。在 TeX 处理 `\insert` 命令时，如果发现前面有同类型插入项被分开，留下一部分到之后的页面，就会算上这些惩罚值。详细的计算过程可以见 *The TeXbook* 第 123–125 页。

在输出例行程序中，`\insertpenalties` 有完全不同的意义——它表示已经见到但未能放入当前页的插入项的总数（见 *The TeXbook* 第 125 页）。

`\floatingpenalty [⟨number⟩ parameter]`

在 TeX 处理插入项时，如果发现前面有同类型插入项被分开，留下一部分到之后的页面，就会添加 `\floatingpenalty` 惩罚值到 `\insertpenalties`。Plain TeX 设定 `\floatingpenalty` 为 0。

`\pagegoal [⟨dimen⟩ parameter]`

此参数设定当前页面所需的高度。当放入第一个盒子或者插入项到当前页面时，TeX 设定 `\pagegoal` 等于 `\vsize` 的当前值。改变 `\pagegoal` 的值可以让 TeX 排版时缩短该页面的高度。即使新的值比页面已有内容的高度还小也可以，此时 TeX 将把多余的内容放在下一页。但记住——在下一页 TeX 还是会重置 `\pagegoal` 为 `\vsize`。

`\pagetotal [⟨dimen⟩ parameter]`

此参数表示当前页面累积的自然高度。TeX 添加内容到竖直列时会更新 `\pagetotal` 的值。

`\pagedepth [⟨dimen⟩ parameter]`

此参数表示当前页面的深度。TeX 添加内容到竖直列时会更新 `\pagedepth` 的值。

`\pageshrink [⟨dimen⟩ parameter]`

此参数表示当前页面累积的粘连的可收缩量。TeX 添加内容到竖直列时会更新 `\pageshrink` 的值。

```
\pagestretch [⟨dimen⟩ parameter]  
\pagefilstretch [⟨dimen⟩ parameter]  
\pagefillstretch [⟨dimen⟩ parameter]  
\pagefullstretch [⟨dimen⟩ parameter]
```

这四个参数合起来表示当前页面累积的粘连的可伸长量。可伸长量总共等于 $n_0 + n_1\text{fil} + n_2\text{fill} + n_3\text{full}$ ，其中四个参数给出四个 n_i 的值。 \TeX 添加内容到竖直列时会更新这些参数的值。

页面布局

■ 页面描述参数

```
\hsize [⟨dimen⟩ parameter]
```

此参数指定当前行的长度。Plain \TeX 设定 \hsize 为 6.5in。在第 113 页有完整的解释。

```
\vsize [⟨dimen⟩ parameter]
```

此参数指定页面的竖直长度。 \TeX 仅在页面开始时检查此值。因此，在页面中间改变 \vsize 到下一页才会生效。若要在页面中间改变其竖直长度，应该修改 \pagegoal （第 142 页）的值。（若要改变当前页面开始的所有页面，你需要同时修改 \vsize 和 \pagegoal 。）Plain \TeX 设定 \vsize 为 8.9in。

```
\hoffset [⟨dimen⟩ parameter]  
\voffset [⟨dimen⟩ parameter]
```

\TeX 通常设定页面的“起点”——即它开始打印的地方——在离上页边和左页边均为一英寸的地方。² \hoffset 和 \voffset 的值分别

² \TeX 本身并不关心页面起点的位置，但设备驱动程序将 $.dvi$ 文件转换为可打印形式时要用到，此位置信息使得在不同的设备中能得到相同的结果。

给出实际起点相对该点的水平和竖直偏移量。因此，若 `\hoffset` 和 `\voffset` 均为零， \TeX 就使用正常起点。

例子：

```
\hoffset = -.3in
% Start printing .7 inches from left edge of paper.
\voffset = 1in
% Start printing 2 inches from top edge of paper.
```

`\topskip` [⟨glue⟩ parameter]

为确保页面第一个盒子的基线到上页边的距离总是 d ， \TeX 在每个页面顶部都插入粘连。`\topskip` 给出了 d 的值，以此确定该粘连大小（只要页面第一个盒子不会太高）。 d 等于 `\topskip` 粘连的自然大小。如果页面第一个盒子的高度超过 d ，计算出的粘连将为负值，此时在该页顶部 \TeX 不插入 `\topskip` 粘连。

为更好地理解这些规则，我们假设 `\topskip` 无伸长量也无收缩量，且页面第一项就是个盒子。如果该盒子的高度不大于 `\topskip`，不管高度为多少，它的基线和上页边的距离就始终等于 `\topskip`。反过来，如果该盒子的高度比 `\topskip` 大 e ，它的基线和上页边的距离就是 `\topskip + e`。在 *The TeXbook* 第 113–114 页中详细解释了 `\topskip` 的作用。Plain \TeX 设定 `\topskip` 为 10pt。

`\parskip` [⟨glue⟩ parameter]

此参数设定“段落间距”，即 \TeX 在段落开头插入的竖直粘连。在 `\par` (第 110 页) 那里介绍了 \TeX 开始新段落时所作的事情。Plain \TeX 设定 `\parskip` 为 0pt plus 0.1pt。

`\maxdepth` [⟨dimen⟩ parameter]

此参数设定页面底部盒子的最大深度。它和 `\boxmaxdepth` 参数 (第 168 页) 也有关系。如果底部盒子的深度超过 `\maxdepth`， \TeX 下移该盒子的基准点，让深度就等于 `\maxdepth`。若不作此调整，页面底部盒子可能大大超出下边距，甚至超出页面，Plain \TeX 设定 `\maxdepth` 为 4pt。

■ 页码

\pageno [⟨number⟩ parameter]

此参数以整数值记录当前页码。在前页 (front matter) 中页码通常是负数，我们用小写罗马数字而不用阿拉伯数字编号。在页面中间对页码的改动，会在该页的页眉和页脚中用到。页码事实上由 TeX 的输出例行程序打印，你可以修改它们。

Plain TeX 将页码记录在寄存器 \count0 中。（\pageno 实际上就是 \count0 的别名。）当它输出一个页面到 .dvi 文件时，TeX 在终端中显示 \count0 的值，让你知道它在排版哪一页。也可以把寄存器 \count1–\count9 用于嵌套的多级页码（但你必须自己编程）。如果这些寄存器的值非零，TeX 在终端中也显示它们的值。³

例子：

```
This explanation appears on page \number\pageno\
of our book.
```

结果：

```
This explanation appears on page 145 of our book.
```

例子：

```
\pageno = 30 % Number the next page as 30.
Don't look for this explanation on page \number\pageno.
```

结果：

```
Don't look for this explanation on page 30.
```

\advancenumber

如果页码 $n \geq 0$ ，此命令给 n 加上 1；如果 $n < 0$ 则给 n 减去 1。

◊ \nopagenumbers

默认情况下，plain TeX 会在页面的页脚生成居中的页码。此命令让 TeX 生成空的页脚。

³ 更准确来说，它从 \count0 到 \count9 依次显示各寄存器的值，但忽略末尾的零值。比如，若 \count0 到 \count3 为 (17, 0, 0, 7) 而其他值都为 0，则 TeX 显示页码为 [17.0.0.7]。

\folio

此命令生成当前页码，它的值等于记录在 \pageno 中的 n 。若 $n \geq 0$ ，TeX 用十进制数形式生成 n ；若 $n < 0$ ，TeX 用小写罗马数字形式生成 $-n$ 。

例子：

```
This explanation appears on page \folio\ of the book.
```

结果：

```
This explanation appears on page 146 of the book.
```

■ 页眉和页脚

```
\headline [⟨token list⟩ parameter]
\footline [⟨token list⟩ parameter]
```

这两个参数分别指定当前的页眉和页脚。Plain TeX 输出例行程序将把页眉放在各页顶部，页脚放在各页底部。默认的页眉为空，而默认的页脚为居中的页码。

页眉和页脚的宽度应该和 \hsize 相等（必要时可以用 \hfil，第 161 页）。在页眉和页脚中你永远应该加上字体设定命令，因为你无法预知 TeX 调用输出例行程序时的当前字体。如果你不明确指明字体，TeX 将使用分页时所用的字体。

不要试图用 \headline 或 \footline 生成多行的页眉或页脚。即使 TeX 不会报错，它也只会出很难看的结果。在第 292 页中有生成多行页眉或页脚的方法。

例子：

```
\headline = {\tenrm My First Reader\hfil Page \folio}
```

结果：

My First Reader (at the top of page 10)	Page 10
--	---------

例子：

```
\footline = {\tenit\ifodd\pageno\hfil\folio
            \else\folio\hfil\fi}
% Produce the page number in ten-point italic at
% the outside bottom corner of each page.
```

■ 标记

```
\mark { <text> }
```

此命令让 TeX 添加内容为 $\langle mark text \rangle$ 的标记到当前正构造的列表中。一般地，此标记命令不应该在“内部”构造中使用，例如数学公式或者用 `\hbox`、`\vbox` 或 `\vtop` 构造的盒子。这是因为 TeX 构造页面主盒子时将看不到这样的标记。但是，若你在普通水平模式中用 `\mark`，或者在主竖直列的 `hbox` 中直接使用 `\mark`，该标记将会被转移到主竖直列。在 *The TeXbook* 第 259–260 页中有如何使用 `\mark` 的例子。

```
\firstmark  
\botmark  
\topmark
```

这些命令展开为之前用 `\mark` 生成的标记项的标记文本。标记文本以记号列形式表示。执行分页工作时，TeX 需要将页面内容放入 `\box255`，然后调用输出例行程序；在这两个步骤之间 TeX 设定这些命令的值如下：

- `\firstmark` 包含页面第一个标记的记号列。
- `\botmark` 包含页面最后一个标记的记号列。
- `\topmark` 包含紧跟在页面上边的那个标记的记号列。这个标记就是上一个页面的最后一个标记，即上一页的 `\botmark`。如果上一页没有标记此命令为空。

如果此页面没有任何标记，TeX 设定 `\firstmark` 和 `\botmark` 都等于 `\topmark`，即前面最近的那个标记。*The TeXbook* 第 258 页底部的表格说明了 `\firstmark`、`\botmark` 和 `\topmark` 这三者的关系。

```
\splitfirstmark  
\splitbotmark
```

这两个命令展开为之前用 `\mark` 在竖直盒子 V 中生成的标记项的标记文本。标记文本以记号列形式表示。当你用 `\vsplit` 命令（第 152 页）分割 V 时，TeX 按如下方式设定这两个命令的值：

- `\splitfirstmark` 包含 V 中第一个标记的记号列。
- `\splitbotmark` 包含 V 中最后一个标记的记号列。

如果没有 `\vsplit` 命令，或者前面最近的 `\vsplit` 不包含任何标记，这两个命令不生成记号列。

插入项

■ 脚注

◊ `\footnote {argument1} {argument2}`
`\vfootnote {argument1} {argument2}`

这两个命令用于生成脚注。其中 `{argument1}` 是脚注的“参考符号”，而 `{argument2}` 是脚注文本。脚注文本必要时可以包含多个段落，也可以包含陈列公式等其他构造，但不应该包含任何插入项（比如其他脚注）。

在数学公式的子公式中，或者组成页面的盒子的子盒子中，或者任何形式的插入项中，你都不应该使用这些命令。如果你不确定这些限制出现在哪里，为安全起见，你可以只在段落内或者段落间直接使用 `\footnote` 和 `\vfootnote`。

这些限制没看起来那么严重，因为你可以用 `\vfootnote` 命令添加几乎任何内容的脚注。`\footnote` 和 `\vfootnote` 都会在脚注前面插入参考符号，区别在于 `\vfootnote` 不会在正文中插入参考符号。因此，你可以不用考虑上下文的限制，显式地将参考符号放到它应该在的地方，然后在下一个段落使用 `\vfootnote` 命令。这可能会导致脚注跑到下一页去，此时你可以将 `\vfootnote` 命令移动到上一个段落中。在极少情况下，你需要修改文档正文，以让脚注和参考符号出现在同一页中。

例子：

```
To quote the mathematician P\'olya is a ploy.\footnote
{*{This is an example of an anagram, but not a strict one.}}
```

结果：

```
To quote the mathematician P\'olya is a ploy.*
```

```
:
```

* This is an example of an anagram, but not a strict one.

例子：

```
$$f(t)=\sigma\sigma t\raise 1ex \hbox{\dag}$$
\vfootnote {\dag{The $\sigma\sigma$ notation was explained
in
the previous section.}}
```

结果：

$$f(t) = \sigma\sigma t^\dagger$$

⋮

† The $\sigma\sigma$ notation was explained in the previous section.

■ 一般插入项

```
\topinsert <vertical mode material> \endinsert
\midinsert <vertical mode material> \endinsert
\pageinsert <vertical mode material> \endinsert
```

这些命令生成不同形式的插入项，它们指示（或者允许） \TeX 重新放置 $\langle\text{vertical mode material}\rangle$ ：

- \topinsert 试图将内容放在当前页的顶部。如果那里放不下， \topinsert 将把该内容移动到顶部可放下的下一个页面。
- \midinsert 试图将内容放在当前位置，如果当前位置放不下， \midinsert 将把该内容移动到顶部可放下的下一个页面。
- \pageinsert 将把内容单独放在下一页。为免得到松散页面，不要忘记在内容最后用 \vfil 命令或者其他方式填满页面。

由于 \TeX 可以移动 $\langle\text{vertical mode material}\rangle$ 的位置，我们称它为“浮动体”。插入项对于图形和表格非常有用，因为你可以将它们放在所需的地方，而不用管哪里会分页。

这些命令每个都会结束当前段落，因此你应该仅在段落之间使用它们。你也不要在盒子内部或者另一个插入项内部使用它们。如果有多个插入项要放在同一个位置， \TeX 将保持它们的先后顺序。

例子：

```
\pageinsert
% This text will appear on the following page, by itself.
This page is reserved for a picture of the Queen of Hearts
sharing a plate of oysters with the Walrus and
the Carpenter.
\endinsert

\endinsert
```

此命令结束由 `\topinsert`、`\midinsert` 或 `\pageinsert` 命令开始的插入项。

`\insert <number> {<vertical mode material>}`

这个原始命令提供了构造插入项的内部机制，但除了在定义宏时会用到，其他时候很少用到它。`\footnote`、`\vfootnote`、`\topinsert`、`\midinsert` 和 `\pageinsert` 命令的定义都建立在 `\insert` 命令之上。

设计文档的插入项时，不同的插入项需要分配不同的整数代码⁴ n ，这个代码可以用 `\newinsert` 命令（第 260 页）得到。`\insert` 命令本身仅添加 `<vertical mode material>` 到当前水平列或竖直列。而输出例行程序负责移动 `\box n` 中的插入内容到输出页面。

TEX 将所有插入项按照它们的代码分组。每个插入代码关联四个寄存器：

- `\box n` 给 TEX 用于累积第 n 种插入内容。TEX 分页时会尽可能多地往 `\box n` 中放入第 n 种插入项。然后输出例行程序将把这些内容移动了实际页面上。你可以用 `\ifvoid` 命令（第 254 页）检查 `\box n` 中是否有任何内容。如果不能放下全部内容，TEX 将把剩下内容留到下一页。
- `\count n` 为放大因子 f 。TEX 计算第 n 种插入项所占用的页面纵向空间时，将把插入内容的纵向长度乘以 $f/1000$ 。因此对双栏插入项通常设 f 为 500，而对边注项设为 0。
- `\dimen n` 设定每页放置的第 n 种插入项的最大数目。
- `\skip n` 设定页面包含第 n 种插入项时 TEX 添加的额外间隔。这个间隔不包含该插入项本身占用的空间。例如，当页面包含脚注时，就会用到此间隔。

⁴ The TEXbook 用“类”这个术语表示此代码。为避免和“类”（第 57 页）的其他意思混淆，我们用不同的术语。

TeX 只设定 `\box n`, 你需要设定其他三个寄存器, 以让 TeX 正确计算插入项所需的竖直空间。The *TeXbook* 第 122–125 页中详细介绍了 TeX 如何处理此命令, 以及插入项和分页方式之间的关系。

参阅: `\floatingpenalty` (第 142 页).

修改输出例行程序

`\output [⟨token list⟩ parameter]`

这个参数包含了当前的 输出例行程序, 也就是 TeX 当发现一个断页点的时候会展开的一个符号列表。TeX 把当前页面放入 `\box255`, 所以 `\output` 负责对 `\box255` 进行处理——不管是把它输出到 dvi 还是把它放到别的什么地方去。输出例行程序也负责把页眉和页脚附加到页面上。

`\plainoutput`

这个命令调用 plain TeX 的输出例行程序。Plain TeX 把 `\output` 定义为一个仅仅包含 `\plainoutput` 的符号列表。

`\shipout ⟨box⟩`

这个命令让 TeX 把 `⟨box⟩` 输出到 .dvi 文件。TeX 在 `\shipout` 时, 把 `⟨box⟩` 中的任何 `\write` 命令进行展开。`\shipout` 的主要用途是在输出例行程序中, 不过你可以在任何其他地方使用这个命令。

`\deadcycles [⟨number⟩ parameter]`

这个参数包含这 TeX 在上次 `\shipout` 后已经初始化的输出例行程序的个数。⁵ 如果 `\deadcycles` 变得太大, 那你很有可能把 TeX 弄死循环了, 也就是说, 循环往复地对同一个页面进行分页操作。

⁵ 更加准确地说, TeX 在执行 `\shipout` 的时候, 把 `\deadcycles` 归 0, 然后每次执行 `\output` 的时候把它增加 1。

`\maxdeadcycles [⟨number⟩ parameter]`

如果 `\deadcycles` 超过了 `\maxdeadcycles`, 那 TeX 就会假定输出例行程序被用户死循环了, TeX 这个时候就会报错, 并且执行它自己默认的简单输出例行程序, 也就是 `\shipout\box255`, 以期望从循环中跳出。Plain TeX 把 `\maxdeadcycles` 设置为 25。

`\outputpenalty [⟨number⟩ parameter]`

TeX 在断页的时候设置这个参数。如果断点是在一个 惩罚的项目上, TeX 就会移除这个惩罚, 然后把 `\outputpenalty` 设置成断点处的值, 否则 `\outputpenalty` 就被设成 0。

如果你希望在 TeX 选择的断点之外的其他地方断页, 而不在该断点处断页, 那么在重构这个页面时, 你就需要在 TeX 选择的断点处重新设置这个惩罚。你可以通过 `\penalty\outputpenalty` 来达到这个目的。

`\holdinginserts [⟨number⟩ parameter]`

当 TeX 在处理一个断页时, 这个参数如果大于 0, 那么 TeX 就会阻止处理插入。当你写一个输出例行程序的时候, 如果你希望重新处理该页的内容, (例如输出例行程序本身用的 `\vsize` (第 143 页) 和断页程序用的并非相同值), 那么把这个参数设为 1 就会很有用。

分割竖直列表

`\vsplit ⟨number⟩ to ⟨dimen⟩`

这个命令使得 TeX 把寄存器中编号为 `⟨number⟩` 的盒子 B_2 切割成两个部分。如果 B_2 是一个页面, 那这个命令会使用相同的算法来把这个页面分为两部分; 这种情况下, 分割处就是断页的地方。 B_2 这个盒子必须是个竖直盒子, 不是水平盒子。TeX 把从开头到分割处的部分放到另一个盒子 B_1 中, 而把分割处到结尾的部分放到 B_2 中。然后 `\vsplit` 就输出了 B_1 这个盒子。一般情况下, 你会用类似下面例子的方式, 把 B_1 盒子指定到另一个盒子寄存器中。如果分割处恰好是 B_2 盒子的结尾部分, 那 B_2 盒子就会在 `\vsplit` 后变成一个空盒子。

TeX 使用它一贯的断页算法来分割这个竖直盒子。它使用 `⟨dimen⟩` 的长度来指定 B_1 盒子需要的高度 `\pagegoal`。而 B_1 的竖直长度未必就

一定是 $\langle dimem \rangle$ ，因为 TeX 不一定能完美地达到其分页的目标。TeX 在计算这个竖直分割的时候，不考虑插入，因此 B_2 插入的原本的竖直列表依然会被保留，但并不会受到分割的影响。

例子：

```
\setbox 20 = \vsplit 30 to 7in  
% 把 30 号盒子截去 7 英寸，并放到 20 号盒子中。
```

\splitmaxdepth [$\langle dimen \rangle$ parameter]

此参数指定 \vsplit 产生的盒子的最大允许深度。 \splitmaxdepth 在盒子中和 \maxdepth (第 144 页) 在页面中扮演的角色相同。

\splittopskip [$\langle glue \rangle$ parameter]

这个参数指定 TeX 在 \vsplit 产生的盒子的头部插入的粘连。 \splittopskip 在盒子中和 \topskip (第 144 页) 在页面中扮演的角色相同。

参阅： \splitbotmark , \splitfirstmark (第 147 页).



7

水平和竖直模式命令

这一章介绍水平和竖直模式下具有对应或一致形式的命令。这些命令提供盒子、间隔、标线、指引线以及阵列。有关本章中使用的惯例的说明，参见“命令的描述”（第 3 页）。

产生间隔

■ 固定宽度水平间隔

\thinspace

该命令产生一个正的紧排，它的宽度是 1/6 em（第 61 页），也就是说，它使得 TeX 向右移动一个相应的距离。例如，当你需要嵌套引用，并希望将不同层次的引号分开时，它便非常有用。TeX 不会在 \thinspace 处断行。

例子：

```
“‘\thinspace‘一个引用。’\thinspace’’\par
24,\thinspace 29--31,\thinspace 45,\thinspace 102
```

结果：

“‘一个引用。’”
24, 29–31, 45, 102

\negthinspace

该命令产生一个负的紧排，它的宽度是 1/6 em（第 61 页），也就是说，它使得 TeX 向左移动一个相应的距离。当需要将两个间隔稍远的字符靠近一些时便非常有用。TeX 不会在 \negthinspace 处断行。

例子：

The horror, the horror\negthinspace, the horror of it all!

结果：

The horror, the horror, the horror of it all!

\enspace

该命令产生一个宽度为 1 en 的紧排（1 em 的一半，参见 第 61 页）。TeX 不会在 \enspace 处断行，除非紧接其后的是粘连。在符号列表中，符号与其后的文字通常由 \enspace 来分隔。

例子：

引理 1.\enspace 这里有一只白兔。

结果：

引理 1. 这里有一只白兔。

◊ \enskip

\quad

\quad

\quad

这里的每个命令都产生一个既不能伸展也不能收缩的水平粘连。TeX 不能在这些粘连处断行。对于 plain TeX 的默认字体 cmr10 来说，这些粘连的宽度（与当前字体有关）如下：

命 令	间 距	图示
\enskip	1/2 em	→ ←
\quad	1 em	→ ←

\qquad	2 em	→ ←	
--------	------	------	--

例子：

```
en\enskip skip; quad\quad skip; qquad\qquad skip
```

结果：

```
en skip; quad skip; qquad skip
```

■ 固定长度竖直间隔

- ◊ \smallskip
\medskip
\bigskip

这些命令可以产生连续较大的竖直间隔：

<u>smallskip</u>	<u>medskip</u>	<u>bigskip</u>
------------------	----------------	----------------

\smallskip 产生 3 点的竖直间隔，并可以伸展或收缩 1 点。\\medskip 相当于两个 \\smallskip，而 \\bigskip 相当于两个 \\medskip。

由于这些命令具有固有的竖直特性，因此它们的出现宣告一个段落的结束。由此产生的间距需要再加上正常的段落间距。

例子：

```
Hop \smallskip skip \medskip and \bigskip jump.
```

结果：

```
Hop
```

```
skip
```

```
and
```

```
jump.
```

```
\smallskipamount [⟨glue⟩ parameter]  
\medskipamount [⟨glue⟩ parameter]  
\bigskipamount [⟨glue⟩ parameter]
```

这些参数用于确定由 \\smallskip、\\medskip 和 \\bigskip 命令产生的粘连的量。通过更改这些参数，你可以改变命令的效果。（对于 plain TeX）它们相应的默认值是 1/4、1/2 以及 1 行距。我们建议你保持

这个比例，无论何时，你可以通过修改 `\baselineskip` (第 136 页) 来修改参数值。

■ 可变尺寸间隔

◊ `\hskip <dimen1> plus <dimen2> minus <dimen3>`
`\vskip <dimen1> plus <dimen2> minus <dimen3>`

这些命令可以分别产生水平的和竖直的粘连。在最简单以及最常见的情况下，即当只有 $\langle dimen_1 \rangle$ 存在时，`\hskip` 向右移动 $\langle dimen_1 \rangle$ ，而 `\vskip` 向页面下方移动 $\langle dimen_1 \rangle$ 。在一般情况下，这些命令产生一个粘连，其自然尺寸为 $\langle dimen_1 \rangle$ ，可伸长量为 $\langle dimen_2 \rangle$ ，可收缩量为 $\langle dimen_3 \rangle$ 。命令中的 `plus <dimen2>` 或 `minus <dimen3>`，或两者都可以省略。如果两者都需要，则 `plus` 必须出现在 `minus` 之前。被省略的变量取值为零。任何 $\langle dimen \rangle$ 都可以取负值。

在数学模式下，你可以使用 `\hskip`，但不能使用 `mu` (见“数学单位”，第 81 页) 作为任何尺寸的单位。如果你希望使用 `mu` 作为单位，可以使用 `\mskip` (第 227 页) 代替之。

例子：

```
\hbox to 2in{one\hskip 0pt plus .5in two}
```

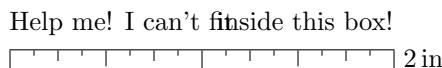
结果：



例子：

```
\hbox to 2in{Help me! I can't fit
{\hskip 0pt minus 2in} inside this box!}
```

结果：



例子：

```
\vbox to 4pc{\offinterlineskip%
  \hbox{一}\vskip 0pc plus 1pc \hbox{二}
  \vskip .5pc \hbox{三}}
```

结果：

一

二

三

```
\hglue <glue>
\vglue <glue>
```

`\hglue` 命令生成水平的粘连，它不会在断行处消失；`\vglue` 命令生成竖直的粘连，它不会在分页处消失。除此以外，这些命令与 `\hskip` 和 `\vskip` 相似。你可以使用 `\vglue` 在文档第一页的标题上部等页面顶部产生空白间隔，但就此目的而言，随后的 `\topglue` 通常是最好的选择。

```
\topglue <glue>
```

该命令¹ 可以精确地产生从页面顶部至该页上第一个盒子顶部的间隔。可以认为页面顶部位于页面第一行之上的一行无形文字的基线处。更加准确地，它在 `\hoffset` 和 `\voffset` 确定的原点之上，且间距等于 `\topskip`。

由于 TeX 通常以非常复杂的方式调整由 `\topskip` 产生的粘连，所以这个命令是非常有用的。通过使用 `\topglue`，你能够控制页面上第一个盒子的位置，而不必担心那些复杂的调整。

```
\kern <dimen>
```

该命令产生的效果取决于 TeX 遇到它时所处的模式：

- 在水平模式下，TeX 向右（正的紧排）或向左（负的紧排）移动。
- 在竖直模式下，TeX 向页面下方（正的紧排）或页面上方（负的紧排）移动。

因此，正的紧排产生空白的间隔，而负的紧排使得 TeX 倒退到已经存在的东西上。在 TeX 中，正的紧排将两个字母分开，而不是使它们更近一些。紧排的这一设计与一些计算化排版系统是不同的。

¹ `\topglue` 是在 TeX 3.0 中引入的，比其它由新 TeX (第 17 页) 引入的增强特性稍晚。在 *The TeXbook* 的 18th 版中第一次得到描述。

紧排与粘连相似，不同的是 (a) 紧排不能被伸展或收缩；(b) 如果紧排后紧接粘连，并且不是数学公式的一部分时， \TeX 不能在紧排处断行或分页。如果 \TeX 发现紧排位于一行或一页的结尾，该紧排将会被忽略。如果你希望得到不会消失的紧排的效果，使用 $\backslash hglue$ 或 $\backslash vglue$ 。

你可以在数学模式中使用 $\backslash kern$ ，但你不能使用 μ 作为 $\langle dimen \rangle$ 的单位（见“数学单位”，第 81 页）。如果你希望使用 μ 作为单位，可以使用 $\backslash mkern$ （第 227 页）代替之。

例子：

```
\centerline{$\Downarrow$}\kern 3pt % 竖直的紧排
\centerline{$\Longrightarrow$}\kern 6pt % 水平的紧排
{\bf 留意我的警示!! }\kern 6pt % 另一水平的紧排
$\Longleftarrow$%
\kern 3pt % 另一竖直的紧排
\centerline{$\Uparrow$}
```

结果：

↓
 \Longrightarrow 留意我的警示！ \Longleftarrow
 ↑

$\backslash hfil$	$\backslash vfil$
$\backslash hfill$	$\backslash vfill$

这些命令产生可无限伸展的水平或竖直的粘连，它们将取代任何存在的有限伸展。 $\backslash hfil$ 和 $\backslash hfill$ 产生水平的粘连，而 $\backslash vfil$ 和 $\backslash vfill$ 产生竖直的粘连。

$\backslash vfil$ 和 $\backslash vfill$ 有着类似的行为。

例子：

```
\hbox to 2in{左 \hfil 中 \hfil 右}
```

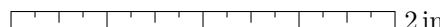
结果：

左 中 右


例子：

```
\hbox to 2in{左 \hfil 中 \hfill 右}
```

结果：

左中 右


例子：

```
\leftline{%
\vbox to 4pc{%
\hbox{上}\vfil\hbox{中}\vfil \hbox{下}}\quad
\vbox to 4pc{%
\hbox{上}\vfil\hbox{中}\vfill\hbox{下}}}
```

结果：

上	上
中	中
下	下

```
\hss
\vss
```

这些命令产生水平和竖直的可无限伸展和收缩的粘连。粘连可以收缩至负的距离，产生的效果是（对于 `\hss` 而言）沿一行向左移动或（对于 `\vss` 而言）向页面顶部移动。

例子：

```
\line{正文文本 \hfil\hbox to 0pt{页边空白 \hss}}
% ‘页边空白 \hss’ 收缩到宽度为零的水平盒子中。
```

结果：

正文文本	页边空白
------	------

例子：

```
\vbox to 1pc{\hrule width 6pc % Top of box.
\hbox{1} \vskip 1pc\hbox to 2pc{\hfil 2}
% \vss 消除了由 \vskip 产生的额外间距。
\vss \hbox to 3pc{\hfil 3}
\hrule width 6pc}% 盒子的底部
```

结果：

A horizontal line with three boxes labeled 1, 2, and 3. Box 1 is at the top, box 2 is in the middle, and box 3 is at the bottom. The boxes are aligned horizontally.

2

```
\hfilneg
\vfilneg
```

这些命令可以消除其前接的 `\hfil` 或 `\vfil` 产生的效果。当 `\hfil` 和 `\vfil` 产生正的无限可伸展的粘连时，`\hfilneg` 和 `\vfilneg` 产生负的无限可伸展粘连。（因此，`n \hfilneg` 消除 `n \hfil` 产生的效果，对于 `\vfilneg` 也是相似的。）`\hfilneg` 和 `\vfilneg` 主要的用途是抵消由宏引入的 `\hfil` 或 `\vfil` 产生的效果。

如果它们是一个盒子中仅有的可无限伸展的粘连，`\hfilneg` 和 `\vfilneg` 则具有奇异的特性，即它们产生与 `\hfil` 和 `\vfil` 完全一致的效果。

例子：

```
\leftline{\hfil 位于右端 \hfilneg}  
% 消除了 \leftline 在其参数右侧产生的 \hfil。
```

结果：

位于右端

例子：

```
\def\aa{\hbox to 1pc{\hfil 2}\vfil}  
\vbox to 4pc{\hbox{1} \vfil \aa  
 \vfilneg \hbox to 2pc{\hfil 3}}
```

结果：

1

2

3

参阅：`\hbadness` 和 `\vbadness`（第 175 页），`\hfuzz` 和 `\vfuzz`（第 176 页），■ 及“指引线”（第 72 页）。

操作盒子

■ 构造 hbox 和 vbox

```
\hbox {<horizontal mode material>}
\hbox to <dimen> {<horizontal mode material>}
\hbox spread <dimen> {<horizontal mode material>}
```

该命令产生一个 hbox (水平盒子) 以容纳 *<horizontal mode material>*。*<horizontal mode material>* 两侧的大括号定义一个编组。TeX 不会对 *<horizontal mode material>* 断行，因为当其组装盒子时它处于受限水平模式下。一旦盒子生成后，TeX 不会改变其大小。

当你希望将一些文字全部放在一行上时，`\hbox` 是有用的。如果你所用的 `\hbox` 使得 TeX 不能以一种可以接受的方式断行时，TeX 会给出 hbox 过满的警告。

`hbox` 的宽度取决于 `\hbox` 的参数：

- 如果你只设定了 *<horizontal mode material>*，`hbox` 将具有参数本身的宽度。
- 如果你设定了 `to <dimen>`，则 `hbox` 的宽度将是 *<dimen>*。
- 如果你设定了 `spread <dimen>`，则 `hbox` 的宽度将是参数本身的宽度再加上 *<dimen>*，也就是说，`hbox` 将伸展 *<dimen>*。

当盒子中内容的宽度没有盒子宽，需要使用空白间隔来填充 `hbox` 时，`\hfil` 命令（第 161 页）是非常有用的。

例子：

```
\hbox{ugly suburban sprawl}
\hbox to 2in{ugly \hfil suburban \hfil sprawl}
\hbox spread 1in {ugly \hfil suburban \hfil sprawl}
% 在上面两行中，如果去掉 \hfil，将会得到 ‘underfull hbox’
警告。
```

结果：

ugly suburban sprawl
ugly suburban sprawl
ugly suburban sprawl



3 in

```
\vtop <vertical mode material>
\vtop to <dimen> {<vertical mode material>}
\vtop spread <dimen> {<vertical mode material>}
\vbox {<vertical mode material>}
\vbox to <dimen> {<vertical mode material>}
\vbox spread <dimen> {<vertical mode material>}
```

这些命令产生一个 `vbox` (竖直盒子) 以容纳 `<vertical mode material>`。`<vertical mode material>` 两侧的大括号定义一个编组。当 TeX 组装盒子时它处于内部竖直模式下。一旦盒子生成后, TeX 不会改变其大小。

`\vtop` 和 `\vbox` 之间的不同点在于 TeX 放置已构成的 `vbox` 的基准点的位置。通常情况下, 由 `\vtop` 获得的基准点位于或接近于已构成的 `vbox` 的顶部; 而 `\vbox` 获得的基准点位于或接近于已构成的 `vbox` 的底部。因此, 一行全部由 `\vtop` 构成的 `vbox`, 其顶部将会近似在一条线上; 相应地, 一行全部由 `\vbox` 构成的 `vbox`, 其底部将会近似在一条线上。

当你希望将一些文字放在单独的一页上时, `\vtop` 和 `\vbox` 是有用的。(为了这个目的, 它通常不在乎你使用的命令。) 如果你使用这些命令使得 TeX 不能以一种可以接受的方式分页时, TeX 会在 `\output` 处于活动的情况下, 给出 `vbox` 过满或不足的警告。

`vbox` 的高度取决于 `\vtop` 或 `\vbox` 的参数。对于 `\vbox`, TeX 采用如下方法确定其高度:

- 如果你只设定 `<vertical mode material>`, `vbox` 将具有参数本身的高度。
- 如果你设定 `to <dimen>`, `vbox` 的高度将是 `<dimen>`。
- 如果你设定了 `spread <dimen>`, 则 `vbox` 的高度将是参数本身的高度再加上 `<dimen>`, 也就是说, `vbox` 将在竖直方向上伸展 `<dimen>`。

对于 `\vtop`, TeX 采用 `\vbox` 一样的规则来构造盒子, 然后以下面描述的方法在高度和深度之间分配竖直方向上的扩展。

通常, 构造的 `vbox` 的宽度是其内部最宽项目的宽度。² 在高度与深度间分配竖直扩展的规则更加复杂:

- 对于 `\vtop`, 如果第一个项目是一个盒子或标线, 则其高度是其第一个项目的高度。否则, 其高度是零。深度是减去高度后的竖直扩展。

² 更准确地说, 是从基准点至构造的 `vbox` 最右侧之间的距离。因此, 如果你使用 `\moveright` 或 (负数距离的) `\moveleft` 向右移动任何项目, 那么构造的 `vbox` 则可能更宽。

- 对于 `\vbox`, 如果最后一个项目是一个盒子或标线, 则其深度是其最后一个项目的深度。否则浓度是零。高度是减去深度后的竖直扩展。³

当盒子中内容的高度没有盒子竖直伸展高, 需要使用空白间隔来填充 `vbox` 时, `\vfil` 命令 (第 161 页) 是非常有用的。

例子:

```
\hbox{\hsize = 10pc \raggedright\parindent = 1em  
\vtop{在这个例子中, 我们将看到如何使用 \vbox 产生双栏的效果。  
每个 \vbox 都包含两个段落, 除了设置为左对齐以外均依照 \TeX  
通常的规则进行排版。 \par  
这实际上并不是获得真正双栏的最好的方法, 因为两栏}  
\hskip 2pc  
\vtop{\noindent  
并不对称, 而且我们无法自动选择第一栏在何处断开,  
甚至不能确定第一栏的最后一行。 \par  
然而, 将行文放在 \vbox 中是一个将文本放在页面上  
你希望的位置上的有用的技巧。  
}}
```

结果:

在这个例子中, 我们将看到如何使用 `vbox` 产生双栏的效果。每个 `vbox` 都包含两个段落, 除了设置为左对齐以外均依照 `\TeX` 通常的规则进行排版。

这实际上并不是获得真正双栏的最好的方法, 因为两栏

并不对称, 而且我们无法自动选择第一栏在何处断开,甚至不能确定第一栏的最后一行。

然而, 将行文放在 `vbox` 中是一个将文本放在页面上你希望的位置上的有用的技巧。

³ 实际上, 规则可以更加复杂。假定在深度确定后, 使用给出的两条规则, 则深度将会大于 `\boxmaxdepth`。随后, 深度将被缩小至 `\boxmaxdepth`, 高度也相应地被调整。

例子：

```
\hbox{\hsize = 1in \raggedright\parindent = 0pt
\vtop to .75in{\hrule 这个盒子的深度是 .75in。 \vfil\hrule}
\qquad
\vtop{\hrule 这个盒子具体其本身的深度。 \vfil\hrule}
\qquad
\vtop spread .2in{\hrule 这个盒子的深度比其自身的深度多
.2in。 \vfil\hrule}}
```

结果：

这个盒子的深度 是.75in。	<u>这个盒子具体其 本身的深度。</u>	这个盒子的深度 比其自身的深度 多.2in。
--------------------	---------------------------	------------------------------

例子：

```
% 显示 \vbox 是如何底部对齐，而非顶部对齐的。
\vbox{\hsize = 1in \raggedright
\vbox to .5in{\hrule 这个盒子的深度是 .5in。 \vfil\hrule}
\qquad
\vbox to .75in{\hrule 这个盒子的深度是 .75in。 \vfil\hrule}}
```

结果：

这个盒子的深度 是.75in。	<u>这个盒子的深度</u> 是.5in。
--------------------	--------------------------

`\boxmaxdepth [⟨dimen⟩ parameter]`

这个参数包含的是一个尺寸 D 。如果盒子的深度超过 D , TeX 将不会构造这个盒子。如果你生成一个盒子，其深度 d 大于 D , TeX 将超出的深度传递给盒子的高度，即高效地将盒子的基准点向下移动 $d - D$ 。如果你将 `\boxmaxdepth` 设为零, TeX 会将盒子向上排一列，以便它们的底线都位于同一个水平线上。Plain TeX 将 `\boxmaxdepth` 设为 `\maxdimen` (第 259 页)，因此，`\boxmaxdepth` 不会影响你的盒子，除非你有意改动它。

```
\underbar {argument}
```

这个命令将 *argument* 放入到一个 hbox，并在其下划线，而不考虑任何突出到盒子的基线以下的东西。

例子：

```
\underbar{为什么不学习 \TeX?}
```

结果：

Why not learn TeX?

```
\everyhbox [ token list] parameter  
\everyvbox [ token list] parameter
```

这些参数包含的是记号列表。TeX 会在开始构造每个 hbox 或 vbox 时对其进行扩展。来自于扩展的任何项目随后成为盒子的项目列表的开始。这些记号列表默认是空的。

■ 设置和获取盒子的内容

```
\setbox register = box  
\box register
```

这些命令分别设置和获取编号为 *register* 的盒子寄存器的内容。需要注意，设置盒子寄存器与设置其它寄存器有细微的差别：你应该使用 `\setbox n =`，而不是 `\box n =`。

使用这些命令获取盒子寄存器中的内容时，有清空其内容的副作用，所以盒子寄存器将失效。如果你不希望这些发生，你可以使用 `\copy`（参见后面的内容）来获取内容。如果你不在乎你使用之后的盒子寄存器中的内容，你应该优先使用 `\box` 而不是 `\copy`，这样可以不致于用废弃的盒子来消耗 TeX 的内存。

例子：

```
\setbox0 = \hbox{蘑菇}  
\setbox1 = \vbox{\copy0\box0\box0}  
\box1
```

结果：

蘑菇
蘑菇

```
\copy register
```

这个命令可以产生盒子寄存器 $\langle register \rangle$ 的一个拷贝。当你希望获取一个盒子的内容，但又不希望破坏其中的内容时是有用的。(使用 $\backslash\text{box}$ 获取寄存器内容会使寄存器失效。)

例子：

```
\setbox0 = \hbox{美好}
祝你拥有 \copy0 \box0 \box0 的一天！
```

结果：

祝你拥有 美好美好的一天！

```
\unhbox \langle register \rangle
\unvbox \langle register \rangle
```

这些命令可以给出盒子寄存器 $\langle register \rangle$ 容纳的列表，并且使盒子寄存器失效。 $\backslash\text{unhbox}$ 适用于包含 hbox 的盒子寄存器，而 $\backslash\text{unvbox}$ 适用于包含 vbox 的盒子寄存器。当你不在乎你使用之后的盒子寄存器中的内容，你应该优先使用这些命令而不是 $\backslash\text{unhcopy}$ 和 $\backslash\text{unvcopy}$ ，这样可以不至于用废弃的盒子来消耗 T_EX 的内存。

例子：

```
\setbox0=\hbox{素甲鱼深深地叹息着，用一只手背抹着眼泪。}
\setbox1=\hbox{他想说话，可是有好一阵子泣不成声。}
\unhbox0 \unhbox1
% \box0 \box1 将并排设置两个 \hbox
% (并且输出难看的过满行)。
\box1 % 什么也不输出。
```

结果：

素甲鱼深深地叹息着，用一只手背抹着眼泪。他想说话，可是有好一阵子泣不成声。

```
\unhcopy \langle register \rangle
\unvcopy \langle register \rangle
```

这些命令可以给出盒子寄存器 $\langle register \rangle$ 容纳的列表，并且保持寄存器的内容不被改变。 $\backslash\text{unhcopy}$ 适用于包含 hbox 的盒子寄存器，而 $\backslash\text{unvcopy}$ 适用于包含 vbox 的盒子寄存器。

例子：

```
\setbox0=\hbox{素甲鱼深深地叹息着，用一只手背抹着眼泪。}
\setbox1=\hbox{他想说话，可是有好一阵子泣不成声。}
\unhcopy0 \unhcopy1\par\noindent
% \box0 \box1 将并排设置两个 hbox
% (并且输出难看的过满行)。
\box1 % 输出一个(不能被断行) hbox。
```

结果：

素甲鱼深深地叹息着，用一只手背抹着眼泪。他想说话，可是有好一阵子泣不成声。
他想说话，可是有好一阵子泣不成声。

参阅：\wd, \dp, \ht（第 172 页）。

■ 移动盒子

```
\moveleft <dimen> <box>
\moveright <dimen> <box>
```

这些命令将 *<box>* 向左或右移动 *<dimen>*（这个数值可以是负的）。你只能对存在于竖直列表中的盒子使用 \moveleft 和 \moveright。

例子：

```
\vbox{\vbox{Phoebe}\vbox{walked}%
\moveleft 20pt\vbox{a}\moveright 20pt\vbox{crooked}%
\vbox{mile.}}
```

结果：

Phoebe
walked
a
 crooked
 mile.

```
\lower <dimen> <box>
\raise <dimen> <box>
```

这些命令将 *<box>* 向上或下移动 *<dimen>*（这个数值可以是负的）。你只能对存在于水平列表中的盒子使用 \raise 和 \lower。

例子：

关于你鼻子上的 \lower 6pt \hbox{肿块}, 你觉得
\raise 6pt \hbox{沮丧} 吗?

结果：

关于你鼻子上的 肿块, 你觉得 沮丧吗?

■ 盒子寄存器的尺寸

```
\ht <register> [ <dimen> parameter ]
\dp <register> [ <dimen> parameter ]
\wd <register> [ <dimen> parameter ]
```

这些参数分别表示盒子寄存器 *<register>* 的高度、深度和宽度。你可以使用这些命令获得一个盒子的尺寸。你也可以改变一个盒子的尺寸，但这是一个需要技巧的事情；如果你希望成为冒险者，你可以从 *The TeXbook* pages 388–389 中获得全部的技巧。

例子：

\setbox0 = \vtop{\hbox{一个}\hbox{米黄色的}\hbox{兔子}}%
那个盒子的宽度是 \the\wd0, 高度是 \the\ht0,
深度是 \the\dp0。

结果：

那个盒子的宽度是 40.0pt, 高度是 7.72pt, 深度是 29.60999pt。

■ 支架、幻影和空盒子

\strut

这个命令产生一个宽度为零的盒子，盒子的高度 (8.5pt) 和深度 (3.5pt) 与用 plain TeX 的默认字体 cmr10 排出的典型的一行文字是一致的。当你使用 \offinterlineskip 或类似的命令关闭 TeX 的行间粘连，如你正在构建一个阵列时，该命令主要用于强迫一行具有相同的高度。如果一行的自然高度过矮，你可以在行中插入一个 \strut 使它回到标准的高度。支架将使得一行的高度和深度更大一些，但它不会打印出任何东西，或者消耗任何水平间隔。

如果你正在使用一个大于或小于 cmr10 的字体排版，你应该根据所处环境重新定义 `\strut`。

例子：

```
\noindent % 使我们处于水平模式下。  
\offinterlineskip % 我们得到固有的间隔。  
% 下面这些 vbox 中的句号在竖直方向不是等间距的。  
\vtop{\hbox{.}\hbox{.}\hbox{.}}  
 \hbox{.\vrule height 4pt depth Opt}}\qqquad  
% 因为支架的缘故，  
% 下面这些 vbox 中的句号在竖直方向上则是等间距的。  
\vtop{\hbox{.\strut}\hbox{.(\strut)\hbox{.x}\strut}}  
 \hbox{.\vrule height 4pt depth Opt\strut}}
```

結果：

:(
x
.)
.x
.)

\mathstrut

这个命令产生一个宽度为零的幻影公式，它的高度与深度与左圆括号是一致的。`\mathstrut` 实际上的定义是`\vphantom{`}`。它主要用于使得根号、下横线和上横线能与公式中其它的根号、下横线和上横线成行排列。除能根据出现在数学公式中的不同样式自动调节以外，它与`\strut`（第 172 页）非常相似。

例子：

結果：

$$\overline{a_1 a_2} \wedge \overline{b_1 b_2} \quad \text{versus} \quad \overline{a_1 a_2} \wedge \overline{b_1 b_2}$$

\phantom{<argument>}

此命令产生一个空盒子，它的大小和位置与假定 $\langle argument \rangle$ 被排版出来时的相同。因某种原因需要手工给一个符号留出空间时，`\phantom` 就可派上用场。

例子：

```
$1\phantom{9}2$
```

结果：

```
1 2
```

```
\hphantom {argument}
\vphantom {argument}
```

这些命令产生幻影盒子，并不打印任何东西：

- `\hphantom` 产生一个宽度与 $\langle argument \rangle$ 一致的盒子，但高度与深度均为零。
- `\vphantom` 产生一个高度与深度与 $\langle argument \rangle$ 一致的盒子，但宽度为零。

它们的主要目的是迫使一个子公式具有特定的最小水平或竖直尺寸。

例子：

```
$$\left.\left[\vphantom{\frac{u}{v}}\right.\right] \star \left.\left[\vphantom{\frac{u}{v}}\right.\right] \quad \{ \quad \}
```

结果：

$$\left[t \right] \star \left[\frac{u}{v} \right] \quad \{ \quad \}$$

```
\smash {argument}
```

该命令排出 $\langle argument \rangle$ ，但迫使其包含的盒子的高度和深度设为零。你可以联合使用 `\smash` 和 `\vphantom` 来获得任何你期望高度与深度的子公式。

例子：

```
$$\{\smash{r_m}\brace r_n}\vphantom{r} \Longrightarrow r$$
```

结果：

$$\left\{ \begin{matrix} r_m \\ r_n \end{matrix} \right\} \Longrightarrow r$$

\null

这个命令产生一个空的 hbox。

例子：

\setbox0 = \null

空盒子 \null 的宽度是 \the\wd0, 高度是 \the\ht0,
深度是 \the\dp0。

结果：

空盒子 的宽度是 0.0pt, 高度是 0.0pt, 深度是 0.0pt。

■ 有关畸形盒子的参数

\overfullrule [〈dimen〉 parameter]

这个参数设定 TeX 附加在过满水平盒子后的标线的宽度。Plain TeX 将其设为 5pt。

\hbadness [〈number〉 parameter]

\vbadness [〈number〉 parameter]

这些参数设定了报告盒子未满或过满警告时的水平和竖直的恶劣状态的阈值。`\hbadness` 适用于 hbox, 而 `\vbadness` 适用于 vbox。如果一个盒子的恶劣状态超过了阈值, TeX 将报告一个错误。如果你提高阈值 (plain TeX 的默认值是 1000), TeX 将不再经常给出警告。需要注意的是, 设定 `\hbadness` 和 `\vbadness` 对你的文档排版结果没有影响, 它们只影响你获得的错误信息。参见 *The TeXbook* page 302, 以获得 TeX 如何确定在什么时候对过满或未满盒子进行警告的准确描述。

例子：

```
\hbadness = 10000 % 取消任何关于 hbadness 的警告。
\hbox to 2in{a b}\par
\hbadness = 500 % 当 hbadness 超过 500 就给出警告。
\hbox to 2in{a\hskip .5pt plus .5in b}
```

结果（在日志中）：

```
Underfull \hbox (badness 5091) detected at line 4
\tenrm a b
```

```
\hbox(6.94444+0.0)x144.54, glue set 3.70787
.\tenrm a
.\glue 0.0 plus 36.135
.\tenrm b
```

\badness

该命令给出 TeX 刚刚产生的（水平的或竖直的）盒子的恶劣状态的数值。如果盒子是过满的，\badness 将是 1000000；在所有其它情况下，它将在 0 至 10000 之间。

```
\hfuzz [ <dimen> parameter]
\vfuzz [ <dimen> parameter]
```

这些参数设置在 TeX 认为一个盒子过满之前，一个盒子能够超出其本身大小的量。\\hfuzz 适用于水平盒子，而 \\vfuzz 适用于竖直盒子。Plain TeX 将两个参数均设为 0.1pt。

例子：

```
\hfuzz = .5in
\hbox to 2in{This box is longer than two inches.}
% 没有错误的结果。
```

结果：

This box is longer than two inches.



参阅：\\tolerance（第 123 页）。

从列表中获取最后的项目

```
\lastkern
\lastskip
\lastpenalty
\lastbox
```

这些控制序列产生位于当前列表最后的项目值。它们不是真正的命令，因为它们只能以参数的一部分出现。如果列表中的最后的项目不是所指类型，它们将给出零值（或者空盒子，在 \lastbox 的情况下）。例如，如果在当前列表中最后一项是一个紧排，\lastkern 给出那个紧排的尺寸；如果它不是紧排，它给出一个尺寸 0。

使用 \lastbox 具有从列表中移除最后一个盒子的附加效应。如果你希望原来的 \lastbox 仍保留在列表中，你不得不向列表中增加它的一个拷贝。 \lastbox 不允许用于数学列表或主竖直列表。

在可能插入所示各种项目的宏调用后，这些控制序列是非常有用的。

例子：

```
\def\aa{\kern 15pt}
\aa\aa\hskip 2\lastkern \par
% 在 ‘三’ 之前产生三倍的紧排。
\def\aa{\hbox{二}}
\aa
\setbox0 = \lastbox % 将 ‘二’ 去掉。
\box0。
```

结果：

```
一二 二      三
一三二。
```

```
\unkern
\unskip
\unpenalty
```

如果当前列表的最后一项分别是紧排、粘连或惩罚的类型，这些命令可以将它们从列表中去除。如果项目不是正确的类型，这些命令将不起作用。与 \lastbox 类似，你不能将它们应用于处于数学模式或主竖直列表中。在已知道会插入你不希望存在的特定项目的宏调用后，这些

命令是非常有用的。 \TeX 不提供 \unbox 命令，因为 \lastbox 产生了近乎一致的效果。

标线与指引线

```
\hrule
\hrule height <dimen> width <dimen> depth <dimen>
\vrule
\vrule width <dimen> height <dimen> depth <dimen>
```

\hrule 命令用于产生水平的标线；相应的 \vrule 命令用于产生竖直的标线。你可以指定标线的宽度、高度或深度的任意或全部。 \TeX 会为你省略的那些提供默认值。你可以以任何顺序给定标线的尺寸；上面给出的形式只是可能的组合中的两个。你甚至可以一个给定类型的尺寸给出多个值---如果你这么做，起作用的是最后一个值。

如果你没有设定水平标线的宽度，则标线将水平地扩展到包含标线的盒子或阵列的最内侧的边界。如果你没有设定水平标线的高度，其默认值是 $0.4pt$ ；如果你没有设定水平标线的深度，其默认值是 $0pt$ 。

如果你没有设定竖直标线的宽度，其默认值是 $0.4pt$ ；如果你没有设定竖直标线的高度或深度，则标线将地扩展到包含标线的盒子或阵列的最内侧的边界。

\TeX 以内的竖直项目处理水平标线，而以内的水平项目处理竖直标线。因此，水平标线只有竖直模式下是有效的，而竖直标线只在水平模式下是有效的。如果觉得意外，可以直观化地观测。水平标线从左向右移动，并且将竖直的项目有序地分开，而竖直标线从上到下，将水平项目有序地分开。

例子：

```
\hrule\smallskip
\hrule width 2in \smallskip
\hrule width 3in height 2pt \smallskip
\hrule width 3in depth 2pt
```

结果：



例子：

```
% 在这里，你能看到基线与 \hrule 的高度与深度之间的关系。
\leftline{
    \vbox{\hrule width .6in height 5pt depth 0pt}
    \vbox{\hrule width .6in height 0pt depth 8pt}
    \vbox{\hrule width .6in height 5pt depth 8pt}
    \vbox{\hbox{基线}\kern 3pt \hrule width .6in}
}
```

}

结果：



例子：

```
\hbox{(\ {\vrule} {\vrule width 8pt} )}
\hbox {(\ {\vrule height 13pt depth 0pt}
    {\vrule height 13pt depth 7pt} x)}
% 圆括号定义了两个连续的盒子的高度与深度；
% 而 ‘x’ 位于基线之上。
```

结果：



- ◊ **\leaders** *<box or rule>* *<skip command>*
- \cleaders** *<box or rule>* *<skip command>*
- \xleaders** *<box or rule>* *<skip command>*

这些命令用于产生指引线，也就是用某一个模式（见“指引线”，第 72 页）的复本填充水平或竖直的间隔。*<box>* 或 *<rule>* 确定一个指引体，即模式的单个复本，而 *<skip command>* 确定了用一行或一列的指引体填充的窗体。模式将重复多次，直到填充了整个窗体。如果 *<skip command>* 是一个水平的间距，窗体中包括的是行向的指引线，TeX 必须处于水平模式下；如果 *<skip command>* 是一个竖直的间距，窗体中包括的是列向的指引线，TeX 必须处于竖直模式下。

这些命令的差异是它们在空间内如何组织重复的模式以及在什么地方放置剩余的间距：

- 对于 **\leaders**，TeX 将一行指引线与包含 **\leaders** 这一命令结果的最内侧的盒子 *B* 的左侧对齐。它将一列指引线与 *B* 的顶端对齐。

整个落在窗体中的指引线将被保留。任何在窗体顶部或底部剩余的间隔将空着。

- 对于 `\cleaders`, 指引线在窗体内是居中的。
 - 对于 `\xleaders`, 模式是均匀分布在窗体中的。如果剩余空间是 l , 指引体被重复 n 次, \TeX 将在相邻的指引线以及指引线的两个端点(左和右或者上和下)之间填充宽度或高度为 $l/(n+1)$ 的间隔。

例子：

```
\def\pattern{\hbox to 15pt{\hfil.\hfil}}
\line{Down the Rabbit-Hole {\leaders\pattern\hfil} 1}
\line{The Pool of Tears {\leaders\pattern\hfil} 9}
\line{A Caucus-Race and a Long Tale {\cleaders\pattern
\hfil} 19}
\line{Pig and Pepper {\xleaders\pattern\hfil} 27}
```

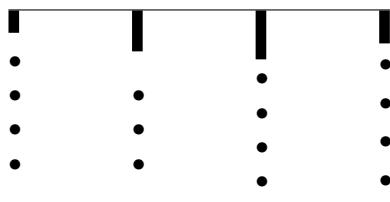
结果：

Down the Rabbit-Hole	1
The Pool of Tears	9
A Caucus-Race and a Long Tale	19
Pig and Pepper	27

例子：

```
\def\bulletfill{\vbox to 3ex{\vfil\hbox{$\bullet$}\vfil}}%
\def\mybox{\vbox to 1in}
\def\myrule{\hrule width 4pt}\hspace{2in}
\hrule \line{%
    \mybox{\myrule depth 8pt \leaders\bulletfill\vfill}
    \hfil
    \mybox{\myrule depth 15pt \leaders\bulletfill\vfill}
    \hfil
    \mybox{\myrule depth 18pt \cleaders\bulletfill\vfill}
    \hfil
    \mybox{\myrule depth 12pt \xleaders\bulletfill\vfill}%
}\hrule
```

结果：



```
\dotfill
```

```
\hrulefill
```

这些命令分别采用一行位于基线上的点和水平线填充封闭的水平空间。通常，在`\dotfill`或`\hrulefill`与任何其之前或之后的文字之间留一个空格是一个好的主意（参见下面的例子）。

例子：

```
\hbox to 3in{开始 {\dotfill} 结束}  
\hbox to 3in{瑞典语 {\hrulefill} 芬兰语}
```

结果：

开始..... 结束
瑞典语—— 芬兰语

```
\leftarrowfill  
\rightarrowfill
```

这些命令用向左或向右的箭头来填充封闭的水平空间。

例子：

```
\hbox to 3in{\vrule \rightarrowfill \ 3 in  
\leftarrowfill\vrule}
```

结果：

|————→ 3 in ←————|

阵列

■ 制表阵列

```
\+ <text> & <text> & ... \cr
\tabalign
```

在制表阵列中，这些命令开始单独的一行。`\+` 与 `\tabalign` 之间唯一的不同在于 `\+` 是一个外部宏，当 TeX 高速读取记号时，你不能使用它（见“外部的”，第 82 页）。

在制表阵列中，如你将 ‘`&`’ 放在所有已经存在的制表符的右侧，那么 ‘`&`’ 将在那个位置上建立一个新的制表符。

例子：

```
\cleartabs % Nullify any previous \settabs.
\+ {\bf if }$a[i] < a[i+1]$ &{\bf then}&\cr
\+&&$a[i] := a[i+1]$;\cr
\+&&{\it found }$:=\$ {\bf true};\cr
\+&&{\bf else}\cr
\+&&{\it found }$:=\$ {\bf false};\cr
\+&&{\bf end if};\cr
```

结果：

```
if a[i] < a[i + 1] then
    a[i] := a[i + 1];
    found := true;
else
    found := false;
end if;
```

```
\settabs <number> \columns
\settabs \+ <sample line> \cr
```

这个命令的第一种形式为制表阵列定义一系列制表符的停止位置。它告诉 TeX 设置制表符停止位置以便将每一行分成 `<number>` 等

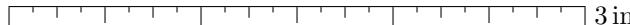
份。TeX 仍将行的长度设为 `\hsize`。你可以通过减小 `\hsize` 来使得阵列更加紧密。

例子：

```
{\hsize = 3in \settabs 3 \columns
\+$1$& 一 & 第一 \cr
\+$2$& 二 & 第二 \cr
\+$3$& 三 & 第三 \cr}
```

结果：

1	一	第一
2	二	第二
3	三	第三



这个命令的第二利形式是利用样例行中的 ‘&’ 指示的位置来定义制表符停止位置。样例行本身不会出现在最终输出中。当你使用这种形式，你通常希望将一些宽度大于需要排列的最宽的内容放在样例行中，以便在列之间产生间隔。正象我们在下面的例子中所做的那样。最后一个制表符后的内容是无关紧要的，因为 TeX 在 `\cr` 出现的地方不需要任何位置指示。

由 `\settabs` 建立起来的制表符设置一直有效，直到你使用一个新的 `\settabs` 命令或结束包含 `\settabs` 命令的编组。该命令的两种形式都是这样的。

例子：

```
% The first line establishes the template.
\settabs \+$1$\quad & three\quad & seventh\cr
\+$1$& 一 & 第一 \cr
\+$2$& 二 & 第二 \cr
\+$3$& 三 & 第三 \cr
```

结果：

1	一	第一
2	二	第二
3	三	第三

\cleartabs

这个命令消除位于当前列右侧的所有制表符。它主要用于排版计算机程序等一些应用，在这类应用中，从一行至另一行，制表符的位置是变化的。

参阅: `\cr`、`\endline`、`\crrcr` (第 186 页)。

■ 常规阵列

```
\halign {<preamble> \cr <row> \cr ... <row> \cr}
\halign to <dimen>{<preamble> \cr <row> \cr ... <row> \cr}
\halign spread <dimen>{<preamble> \cr <row> \cr ... <row> \cr}
```

这一命令产生由一系列行组成的水平的阵列，每一行又包括了一系列的列。 \TeX 调节列的宽度以容纳每一列中最宽的那个条目。

只有当 \TeX 位于竖直模式下水平阵列才能出现。我们建议你在使用该命令之前首先学习一下常见的阵列 (第 47 页)。

阵列由导言和随后的需要排列的文字构成。导言描述随后行的版式，由一系列列的模板构成，列之间由 ‘&’ 分隔，用 `\cr` 作为结束标志。每一行由一系列列的条目构成，也由 ‘&’ 分隔，用 `\cr` 作为结束标志。在模板中，‘#’ 表示 \TeX 应该在一个列条目中插入相应文字的位置。与之相反，`\settabs` 使用 ‘#’ 的固定隐式模板，即它只插入文字本身。

\TeX 在受限水平模式中对每一列条目进行排版，即像 `hbox` 的内容一样，并且隐式地将它封闭在一个编组中。

这个命令的 `to` 形式指导 \TeX 构造一个宽度为 `<dimen>` 的阵列，在需要的时候调节列的间距。这个命令的 `spread` 形式指导 \TeX 构造一个比其本身宽度再宽 `<dimen>` 的阵列。这些形式与 `\hbox` (第 164 页) 相应的形式是类似的。

参见 `\tabskip` (第 190 页) 中使用 `to` 形式的例子。

例子：

```
\tabskip = 1em \halign{%
  \hfil\it#\hfil&\hfil#\hfil##\hfil$#\cr
  美国 & 华盛顿 & 美元 &1.00\cr
  法国 & 巴黎 & 法郎 &0.174\cr
  以色列 & 耶路撒冷 & 新谢克尔 &0.507\cr
  日本 & 东京 & 日元 &0.0829\cr}
```

结果：

美国	华盛顿	美	元	\$1.00
法国	巴黎	法	朗	\$0.174
以色列	耶路撒冷	新谢克尔		\$0.507
日本	东京	日	元	\$0.0829

```
\valign { <preamble>\cr <column>\cr ... <column>\cr }
\valign to <dimen>{ <preamble>\cr <column>\cr ... <column>\cr }
\valign spread <dimen>{ <preamble>\cr <column>\cr ... <column>\cr }
```

这一命令产生由一系列列组成的竖直的阵列，每一列又包括了一系列的行。 \TeX 调节行的高度以容纳每一列中最高的那一个条目。

只有当 \TeX 位于水平模式下竖直阵列才能出现。因为竖直阵列 (a) 在概念上有一些困难，并且 (b) 不经常使，因此，我们建议在你试图使用 `\valign` 命令之前首先学习一下常见的阵列（第 47 页）以及 `\halign` 命令（参见上面的描述）。

阵列由导言和随后的需要排列的文字构成。导言描述随后列的版式，由一系列行的模板构成，列之间由 ‘&’ 分隔，用 `\cr` 作为结束标志。每一列由一系列行的条目构成，也由 ‘&’ 分隔，用 `\cr` 作为结束标志。在模板中，‘#’ 表示 \TeX 应该在一个行条目中插入相应文字的位置。

\TeX 在内部竖直模式中对每一行条目进行排版，即像竖直盒子的内容一样，并且隐式地将它封闭在一个编组中。它总是将 `vbox` 的深度设为零。随后，列中的任何文字或其它水平模式内容将 \TeX 带回通常的水平模式。（这只是在内部竖直模式下 \TeX 行为常见准则的一个应用。）常见的段落参数适用于这种情形：行条目具有 `\parindent`（第 112 页）的起始缩进，而且它的每一行都附加了 `\leftskip` 和 `\rightskip`（第 115 页）粘连。

尤其需要注意的是，列条目包含的文字具有 `\hsize`（第 113 页）的宽度。除非你重新设置 `\hsize` 到你希望得到的行宽，否则你总能遇到过满水平盒子的情况，或者发现第一列占据了整页的宽度，或者两种情况都会出现。

正常情况下，你需要在每个模板中包括支架，以便行不会由于阵列中条目高度的变化而变得弯曲。你可以使用 `\strut` 获得支架。

这个命令的 `to` 形式指导 \TeX 构造一个竖直伸展为 `<dimen>` 的阵列，在需要的时候调节行的间距。这个命令的 `spread` 形式指导 \TeX 构造一个比其本身高度再高 `<dimen>` 的阵列。这些形式与 `\vbox`（第 166 页）相应的形式是类似的。

例子：

```
{\hsize=1in \parindent=0pt
\valign{\strut\strut\strut\strut\cr
bernaise&curry&hoisin&hollandaise\cr
ketchup&marinara&mayonnaise&mustard\cr
rarebit&tartar\cr}}
```

结果：

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

例子：

```
% 与上例相同的内容，但没有支架
% (这个例子显示了支架存在的必要性)。
```

```
{\hsize=1in \parindent=0pt
\valign{\#\#\#\#\cr
bernaise&curry&hoisin&hollandaise\cr
ketchup&marinara&mayonnaise&mustard\cr
rarebit&tartar\cr}}
```

结果：

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

\ialign

除了首先将 `\tabskip` 粘连设为零，并将 `\everycr` 清空以外，这个命令的行为与 `\halign` 相似。

\cr

这个命令用于结束水平或竖直阵列的导言、水平的或制表阵列的行，或竖直阵列的列。你可以通过设置 `\everycr` (第 191 页) 参数的值，引导 TeX 采取特定的行动，无论什么时候它遇到 `\cr`。

\endline

这个命令是 `\cr` 的另一种形式。在你重新定义了 `\cr`，但还需要利用其原始的定义时，这个命令是有用的。

\crcr

如果该命令紧随 \cr 或 \noalign 之后出现, TeX 会忽略它。除此之外, 它的行为与 \cr 类似。它主要的应用是作为一种安全措施以避免由期望以 \cr 作为结束符的宏包而引起的错误信息。如果你将 \crcr 放在用以表示宏定义中此类参数的后面, 宏将正常工作, 无论其参数是否以 \cr 作为结束标记。

\omit

这个命令告诉 TeX 在处理特定的列或行条目时, 忽略水平的或竖直的阵列中的一个模板。 \omit 必须以第一个项目出现在列或行条目中, 它有效地使用简单的模板 '#' 覆盖来自导言中的模板。

例子:

```
\tabskip = 2em\halign{%
    \hfil\it#\hfil&\hfil#\hfil&#\hfil\$#\cr
    美国 & 华盛顿 & 美元 &1.00\cr
    \omit \dotfill 法国 \dotfill& 巴黎 & 法朗 &0.174\cr
    以色列 & 耶路撒冷 & 新谢克尔 &0.507\cr
    日本 & 东京 & 日元 &0.0829\cr}
```

结果:

美国	华盛顿	美 元	\$1.00
.法国.	巴黎	法 朗	\$0.174
以色列	耶路撒冷	新谢克尔	\$0.507
日本	东京	日 元	\$0.0829

例子:

```
{\hsize=1.2in \parindent=0pt
\valign{(#)\strut&(#)\strut&(#)\strut&(#)\strut\cr
bernaise&curry&hoisin&hollandaise\cr
ketchup\omit\strut{\bf MARINARA!}&mayonnaise&mustard\cr
rarebit&tartar\cr}}
```

结果:

(bernaise)	(ketchup)	(rarebit)
(curry)	MARINARA!	(tartar)
(hoisin)	(mayonnaise)	
(hollandaise)	(mustard)	

\span

这个命令的意义取决于它出现在导言中，还是在阵列条目中。

- 通常，T_EX 并不扩展它读入的导言中的记号。将 `\span` 放在导言中记号的前面，根据宏扩展的 T_EX 常用规则，会导致这些记号立即得到扩展。
- 用 `\span` 取代 ‘&’，放在两列或两行条目之前，会导致这些列或行合并到一起。对于水平阵列，合并的列的宽度是列成员宽度之和。对于竖直阵列，合并的行的高度是行成员高度之和。合并的行或合并的列的模板形成一个编组，因此，在 `\span` 前的字体设置命令影响下一个‘&’前的一切。

`\span` 很少单独用在模板之外，但它为定义 `\multispan` 提供了基本的机制。

\multispan <number>

该命令告诉 T_EX，随后水平阵列的行中的 `<number>` 列或竖直阵列的列中的 `<number>` 行应该合并成一个单独的列或行（就像 `\span` 那样），并且它们的模板应该被忽略（就像 `\omit` 那样）。

例子：

```
\tabskip = 13pt\halign{%
  \hfil\it#\hfil&\hfil#\hfil&#\hfil\$#\cr
  美国 & 华盛顿 & 美元 &1.00\cr
  法国 & 巴黎 & 法朗 &0.174\cr
  以色列 & 耶路撒冷 &
  \multispan 2 \hfil\it(无信息)\hfil \cr
  日本 & 东京 & 日元 &0.0829\cr}
```

结果：

美国	华盛顿	美元	\$1.00
法国	巴黎	法郎	\$0.174
以色列	耶路撒冷	(无信息)	
日本	东京	日元	\$0.0829

例子：

```
{\hsize=1.2in \parindent=0pt  
\valign{(#)\strut&(#)\strut&(#)\strut&(#)\strut\cr  
    bernaise&curry&hoisin&hollandaise\cr  
    \multispan 3$$\left\{\{\rm ketchup\atop\rm marinara\right\}\right.\cr  
        \right\}$$&mustard\cr  
    rarebit&tartar\cr}}
```

結果：

(bernaise)		(rarebit)
(curry)		(tartar)
(hoisin)	{ ketchup marinara }	
(hollandaise)		(mustard)

```
\noalign { \langle vertical mode material\rangle }  
\noalign { \langle horizontal mode material\rangle }
```

该命令在水平阵列的当前行之后插入 *<vertical mode material>*，或者在竖直阵列的当前列之后插入 *<horizontal mode material>*。插入的内容可以是文本、粘连、标线，或者其它任何东西。

`\noalign` 最常见的用途是在一行或列后插入额外的间隔。如果你希望在水平阵列的每行后都插入额外的间隔，可以使用 `\openup`（第 138 页）。

例子：

```
\halign{%
  \hfil\it#\hfil\tabskip=2em&\hfil#\hfil##%
  \hfil\$#\tabskip=0em\cr
% 改变 \tabskip 将阻止下面的标线突出来。
美国 & 华盛顿 & 美元 & 1.00\cr
法国 & 巴黎 & 法朗 & 0.174\cr
\noalign{\smallskip\hrule\smallskip}
以色列 & 耶路撒冷 & 新谢克尔 & 0.507\cr
日本 & 东京 & 日元 & 0.0829\cr}
```

結果：

美国	华盛顿	美	元	\$1.00
法国	巴黎	法	朗	\$0.174
以色列	耶路撒冷	新谢克尔		\$0.507
日本	东京	日	元	\$0.0829

例子：

```
{\hsize=1in \parindent=0pt
\valign{\strut\strut\strut\strut\cr
\noalign{\vrule width 2pt\quad}
bernaise&curry&hoisin&hollandaise\cr
\noalign{\vrule width 2pt\quad}
ketchup&marinara&mayonnaise&mustard\cr
\noalign{\vrule width 2pt\quad}
rarebit&tartar\cr
\noalign{\vrule width 2pt\quad}}}
```

结果：

bernaise	ketchup	rarebit	
curry	marinara	tartar	
hoisin	mayonnaise		
hollandaise	mustard		

\tabskip [⟨glue⟩ parameter]

该参数指定 TeX 在水平阵列的列之间或竖直阵列的行之间放置水平或竖直粘连的量。TeX 也在水平阵列的第一列的左侧和最后一行的右侧，以及竖直阵列的第一行的上端和最后一行的下端放置 \tabskip 粘连。你可以在模板中改变 \tabskip---这一改变将影响所有紧接其后的 & 的粘连以及最后一行或列之后的粘连。

例子：

```
\halign to 3.5in{%
\hfil\it#\tabskip = 2em plus 8pt
\hfil\&\hfil#\hfil\#&\tabskip = 1em
\&\hfil$#\tabskip = 0em\cr
美国 & 华盛顿 & 美元 &1.00\cr
法国 & 巴黎 & 法朗 &0.174\cr
以色列 & 耶路撒冷 & 新谢克尔 &0.507\cr
日本 & 东京 & 日元 &0.0829\cr}
```

结果：

美国	华盛顿	美 元	\$1.00
法国	巴黎	法 朗	\$0.174
以色列	耶路撒冷	新谢克尔	\$0.507
日本	东京	日 元	\$0.0829

例子：

```
{\hsize = 1in \parindent=0pt \tabskip=5pt
\valign{\#&strut&#&strut\tabskip = 3pt
&#&strut&#&strut\cr
bernaise&curry&hoisin&hollandaise\cr
ketchup&marinara&mayonnaise&mustard\cr
rarebit&tartar\cr}}
```

结果：

bernaise	ketchup	rarebit
curry	marinara	tartar
hoisin	mayonnaise	
hollandaise	mustard	

\hidewidth

该命令告诉 TeX 忽略水平阵列中下一个列条目的宽度。如果你有一个条目长于同一列中其它大多数，并且，你希望条目伸到列外，而不是使得该条目所在列变得更宽时，该命令是有用的。如果 `\hidewidth` 在条目的左侧，则条目向左伸展；如果 `\hidewidth` 在条目的右侧，则条目向右伸展。

例子：

```
\tabskip = 25pt\halign{%
\hfil\it#\hfil&\hfil\#&\hfil\$#\cr
美国 &\hidewidth 华盛顿 & 美元 &1.00\cr
法国 & 巴黎 & 法郎 &0.174\cr
以色列 & 耶路撒冷 & 新谢克尔 &0.507\cr
日本 & 东京 & 日元 &0.0829\cr}
```

结果：

美国	华盛顿	美 元	\$1.00
法国	巴黎	法 朗	\$0.174
以色列	耶路撒冷	新谢克尔	\$0.507
日本	东京	日 元	\$0.0829

\everycr [*<token list>* parameter]

无论什么时候，当 TeX 遇到处于每个导言结尾处、水平阵列的每行结尾处以及竖直阵列的每列结尾处的 `\cr` 时，它扩展 *<token list>*。

\everycr 只在 \cr 后扩展。因此，你可以通过向 \everycr 赋与导言、行、列相关列表，引导 TeX 在导言、行或列的结尾处执行特定的命令。

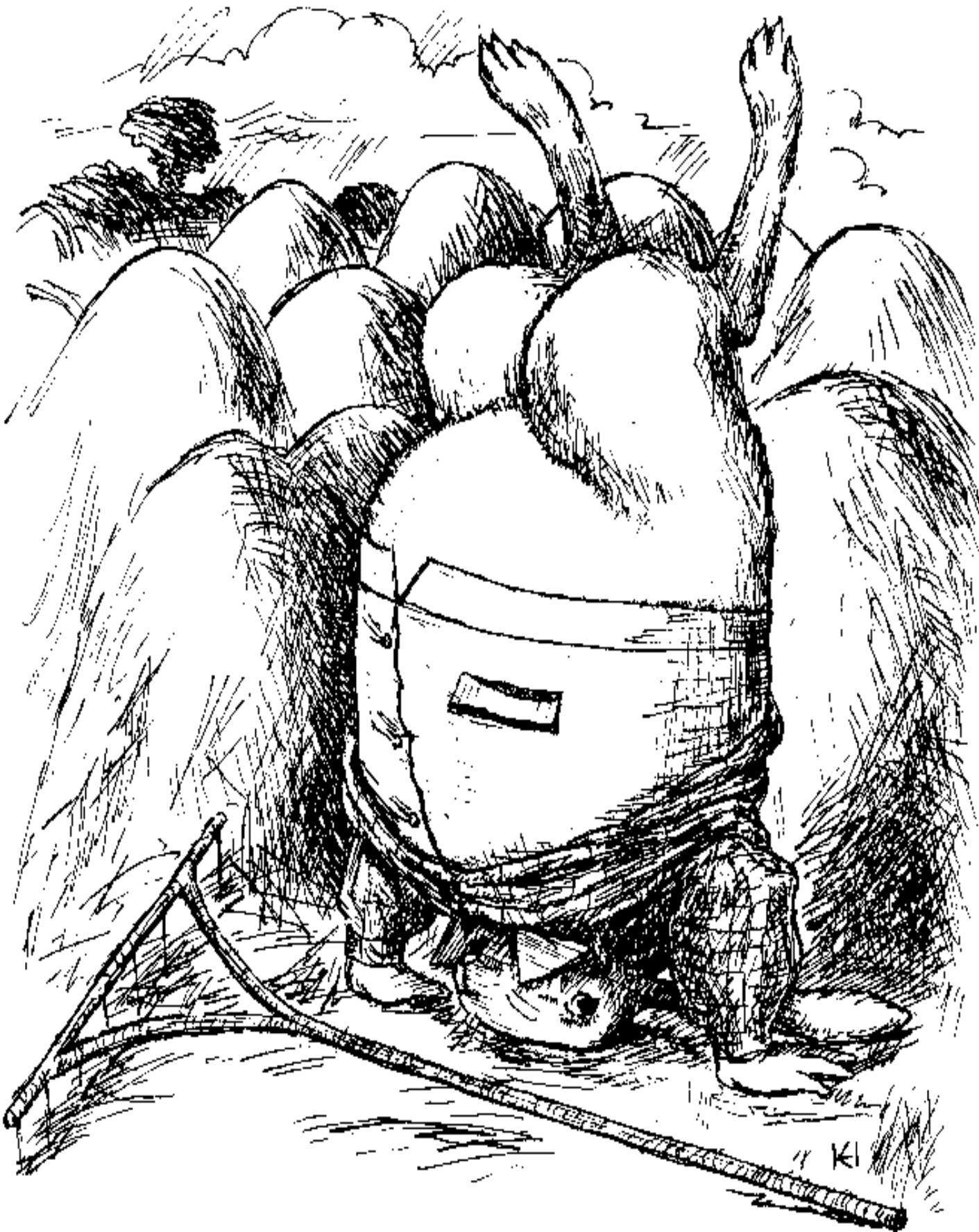
\everycr 的记号不应该包括 \noalign 以外的任何命令。这是因为，\everycr 记号会出现在阵列的最后一一个 \cr。任何一个非 \noalign 命令会导致 TeX 认为它新开了一行或一列，因此，TeX 会抱怨丢失了 \cr，插入 \cr，再次插入 \everycr 记号，并且无限地重复这些动作。

例子：

```
\everycr={\noalign{\smallskip\hrule\smallskip}}
\halign{#\tabskip = 11pt&\hfil#\hfil&\hfil#\hfil
       \tabskip = 0pt\cr
$1\$& 一 & 第一 \cr
$2\$& 二 & 第二 \cr
$3\$& 三 & 第三 \cr}
```

结果：

1	一	第一
2	二	第二
3	三	第三



8

数学公式命令

这一章包括排印数学公式所需要的命令。在“命令描述”（第 3 页）这一节中给出了这章的惯例。

简单公式排版

■ 希腊字母

α	<code>\alpha</code>	μ	<code>\mu</code>	σ	<code>\sigma</code>
β	<code>\beta</code>	ν	<code>\nu</code>	ς	<code>\varsigma</code>
χ	<code>\chi</code>	ω	<code>\omega</code>	Σ	<code>\Sigma</code>
δ	<code>\delta</code>	Ω	<code>\Omega</code>	τ	<code>\tau</code>
Δ	<code>\Delta</code>	ϕ	<code>\phi</code>	θ	<code>\theta</code>
ϵ	<code>\epsilon</code>	φ	<code>\varphi</code>	ϑ	<code>\vartheta</code>
ε	<code>\varepsilon</code>	Φ	<code>\Phi</code>	Θ	<code>\Theta</code>
η	<code>\eta</code>	π	<code>\pi</code>	υ	<code>\upsilon</code>
γ	<code>\gamma</code>	ϖ	<code>\varpi</code>	Υ	<code>\Upsilon</code>
Γ	<code>\Gamma</code>	Π	<code>\Pi</code>	ξ	<code>\xi</code>
ι	<code>\iota</code>	ψ	<code>\psi</code>	Ξ	<code>\Xi</code>
κ	<code>\kappa</code>	Ψ	<code>\Psi</code>	ζ	<code>\zeta</code>
λ	<code>\lambda</code>	ρ	<code>\rho</code>		
Λ	<code>\Lambda</code>	ϱ	<code>\varrho</code>		

输入这些命令可以排印出数学公式中的相应的希腊字母符号。你只能在数学模式中使用它们，所以如在普通的文本中使用它们时，你必须

把它们括在美元符号 (\$) 内。TeX 并不包含这些数学中使用的希腊字母所对应的正体字符的命令，不过你可以很方便地得到这些字符。比如说，你可以在公式中使用 ‘{\rm o}’ 来得到一个小写的 omicron ‘o’，又比如，你可以使用 ‘{\rm B}’ 得到大写的 beta (‘B’).

注意不要混淆下面的符号：

- \upsilon (‘v’), {\rm v} (‘v’), 和 \nu (‘\nu’).
- \varsigma (‘\varsigma’) 和 \zeta (‘\zeta’).

使用数学的意大利字体 (\mit) 可以得到斜体的大写希腊字母。

在计算在希腊字母周围插入多少的空白时，TeX 把它们当作正常的符号。

例子：

如果 \$\\rho\$ 和 \$\\theta\$ 都是正数，那么 \$f(\\theta) - \\Gamma_{\\mit \\Gamma} \\theta \\Gamma_{\\mit \\Gamma} \\rho < f(\\rho) - \\Gamma_{\\mit \\Gamma} \\rho\$.

结果：

如果 ρ 和 θ 都是正数，那么 $f(\theta) - \Gamma_\theta < f(\rho) - \Gamma_\rho$.

■ 各种普通数学符号

\diamond	∞ \infty	\exists \exists	∂ \partial
\Re	\Re	\forall \forall	\surd \surd
\Im	\Im	\hbar \hbar	\wp \wp
\angle	\angle	ℓ \ell	\flat \flat
\triangle	\triangle	\aleph \aleph	\sharp \sharp
\backslash	\backslash	\imath \imath	\natural \natural
\mid	\mid	\jmath \jmath	\clubsuit \clubsuit
\parallel	\parallel	∇ \nabla	\diamondsuit \diamondsuit
\Vert	\Vert	\neg \neg	\heartsuit \heartsuit
\emptyset	\emptyset	\lnot \lnot	\spadesuit \spadesuit
\bot	\bot	/ (上标点)	
\top	\top	/ \prime	

这些命令可以排印各种符号。为了把它们和其它的符号，比如关系符号等，区分开来，它们被称为普通数学符号。你只能在数学模式中使用这些符号，所以如果在普通的文本中使用，你必须使用美元符号 (\$) 把它们括起来。

当你想在 ‘ i ’ 或 ‘ j ’ 上加上重音符号，则需要使用 \imath 和 \jmath 命令来表示它们本身。

上标点符号 (') 是一个 `\prime` 的上标的简写。`\prime` 本身可以排印一个很大的丑陋的撇号。)

`\|` 和 `\Vert` 命令是等价的，就像 `\neg` 和 `\lnot` 命令一样。`\vert` 符号可以排印出和 ‘|’ 相同的效果。

由 `\backslash`, `\vert`, 和 `\Vert` 排印的命令叫做 分界符。使用 `\bigm` 等 (第 222 页) 命令可以排印大号的这些字符。

例子：

```
The Knave of $\heartsuit$, he stole some tarts.
```

结果：

```
The Knave of ♠, he stole some tarts.
```

例子：

如 $\hat{\imath} < \hat{\jmath}$ 则 $i' \leq j'$.

结果：

如 $\hat{i} < \hat{j}$ 则 $i' \leq j'$.

例子：

```
$$\{{x-a}\over{x+a}\}\biggm|\backslash{y-b}\over{y+b}$$
```

结果：

$$\frac{x-a}{x+a} \Bigg| \frac{y-b}{y+b}$$

■ 二元运算符

◊	<code>\vee</code>	<code>\wedge</code>	<code>\amalg</code>	<code>\cap</code>	<code>\cup</code>	<code>\uplus</code>	<code>\sqcap</code>	<code>\sqcup</code>	<code>\dagger</code>	<code>\ddagger</code>	<code>\land</code>	<code>\lor</code>	<code>\cdotp</code>	<code>\diamond</code>	<code>\bullet</code>	<code>\circ</code>	<code>\bigcirc</code>	<code>\odot</code>	<code>\ominus</code>	<code>\oplus</code>	<code>\oslash</code>	<code>\otimes</code>	<code>\pm</code>	<code>\mp</code>	<code>\triangleleft</code>	<code>\triangleright</code>	<code>\bigtriangledown</code>	<code>\bigtriangleup</code>	<code>\ast</code>	<code>\star</code>	<code>\times</code>	<code>\div</code>	<code>\setminus</code>	<code>\wr</code>
---	-------------------	---------------------	---------------------	-------------------	-------------------	---------------------	---------------------	---------------------	----------------------	-----------------------	--------------------	-------------------	---------------------	-----------------------	----------------------	--------------------	-----------------------	--------------------	----------------------	---------------------	----------------------	----------------------	------------------	------------------	----------------------------	-----------------------------	-------------------------------	-----------------------------	-------------------	--------------------	---------------------	-------------------	------------------------	------------------

这些命令可以排印各种二元运算符。二元运算符是 TeX 的一种符号集。TeX 在不同的符号集周围会插入不同的空白。当 TeX 需要在同一个数

学公式中间断行时，它会考虑在二元运算符后面进行断行---不过仅在它出现在公式的最外层时，而不是在一个组中。

除了这些命令以外，TEX 也把 ‘+’ and ‘-’ 作为二元运算符。它把 ‘/’ 当作一个普通符号，因为虽然事实上在数学中它是一个二元运算，但是它在周围加入的空白更少时看上去更漂亮。

例子：

```
 $$z = x \div y \quad \hbox{当且仅当} \quad z \times y = x \text{ 且 } y \neq 0$$  
z \times y = x \; ; \hbox{且} ; y \neq 0 $$
```

结果：

$$z = x \div y \quad \text{当且仅当} \quad z \times y = x \text{ 且 } y \neq 0$$

*

命令 * 表示乘法符号 (\times)，也是一个二元符号。乘法符号在文本中的数学公式中出现时表现得和一个分词符类似。这就是说，TEX 仅会在公式该点需要断行时排版 \times 符号。因为 TEX 永远不会在陈列公式中断行，所以 * 在陈列公式 中是没有任何作用的。

例子：

```
Let $c = a*b$. In the case that $c=0$ or $c=1$, let
$\Delta$ be $(\hbox{the smallest } q)\ast(\hbox{the
largest } q)$ in the set of approximate $\tau$-values.
```

结果：

```
Let  $c = ab$ . In the case that  $c = 0$  or  $c = 1$ , let  $\Delta$  be (the smallest  $q$ )  $\times$ 
(the largest  $q$ ) in the set of approximate  $\tau$ -values.
```

■ 关系符号

\diamond	\asymp	\gg	\bowtie
\cong	\cong	\ll	\propto
\dashv	\dashv	\models	\approx
\vdash	\vdash	\neq	\sim
\perp	\perp	\neq	\simeq
\mid	\mid	\notin	\frown
\parallel	\parallel	\in	\smile
\doteq	\doteq	\ni	\subset
\equiv	\equiv	\owns	\subseteq
\geq	\geq	\prec	\supset
\geq	\geq	\preceq	\supseteq
\leq	\leq	\succ	\sqsubset
\leq	\leq	\succeq	\sqsupset

这些命令可以排印各种关系符号。关系符号是 TeX 的数学符号中的类之一。TeX 在不同的类之间插入不同的空白长度。当 TeX 需要在在一个数学公式处断行，它会考虑在一个关系符后进行断行---不过仅在它出现在公式的最外层时，而不是在一个组中。

除了这里列出的命令以外，TeX 也把 ‘=’ 和 “arrow” 命令 (第 202 页) 作为关系运算符。

一些关系符有多种命令表达方式，你可以使用任何一个来排印它们：

- ‘ \geq ’ (\geq 和 \geq).
- ‘ \leq ’ (\leq 和 \leq).
- ‘ \neq ’ (\neq , \neq , 和 $\not=$).
- ‘ \exists ’ (\ni 和 \owns).

在这些符号前加上 `\not`, 可以排印它们的非运算:

$\not\asymp$	$\not\leq$	$\not\simeq$
$\not\cong$	$\not\prec$	$\not\subset$
$\not\equiv$	$\not\preceq$	$\not\subseteq$
$\not=$	$\not\succ$	$\not\supset$
$\not\geq$	$\not\sucessq$	$\not\supseteq$
$\not\geq$	$\not\approx$	$\not\sqsubseteq$
$\not\leq$	$\not\sim$	$\not\sqsupseteq$

例子:

我们可以得到 $AB \perp AC$, 且
 $\triangle ABF \not\sim \triangle ACF$.

结果:

我们可以得到 $AB \perp AC$, 且 $\triangle ABF \not\sim \triangle ACF$.

■ 左右定界符

<code>\lbrace</code>	<code>\lbrack</code>	<code>\lceil</code>
<code>\{</code>	<code>\rbrack</code>	<code>\rceil</code>
<code>\rbrace</code>	<code>\langle</code>	<code>\lfloor</code>
<code>\}</code>	<code>\rangle</code>	<code>\rfloor</code>

这些命令排印各种左右定界符。数学家用定界符指明公式各部分的边界。左定界符又称为“开符号”，右定界符又称为“闭符号”。开符号和闭符号是 TeX 数学公式中的两种字符类。TeX 在不同类的数学符号之间留下不同大小的间隔。你也许认为在开符号和闭符号旁边的间隔是对称的，但实际上并非如此。

有些左定界符和右定界符可以用不止一个命令排印:

- ‘{’ (`\lbrace` 和 `\{`)
- ‘}’ (`\rbrace` 和 `\}`)
- ‘[’ (`\lbrack` 和 ‘[’)
- ‘]’ (`\rbrack` 和 ‘]’)

左右方括号（两种形式皆可）在数学模式之外也可以使用。

除这些命令之外，TeX 还将 ‘(’ 视为左定界符，将 ‘)’ 视为右定界符。

利用 `\left` 和 `\right` (第 214 页) 命令，你可以让 TeX 选择定界符的尺寸。或者利用某个 `\bigx` 命令（见 `\big` 等，第 222 页），你可以选择特定尺寸的定界符。

例子：

集合 $\{\mid x \mid x > 0\}$ 是空集。

结果：

集合 $\{x \mid x > 0\}$ 是空集。

箭头

\leftarrow	<code>\leftarrow</code>	\rightarrow	<code>\rightarrow</code>
\leftarrow	<code>\gets</code>	\rightarrow	<code>\rightharpoondown</code>
\Leftarrow	<code>\Leftarrow</code>	\rightarrow	<code>\leftharpoonup</code>
\rightarrow	<code>\rightarrow</code>	\rightarrow	<code>\rightharpoonup</code>
\rightarrow	<code>\to</code>	\rightleftharpoons	<code>\rightleftharpoons</code>
\Rightarrow	<code>\Rightarrow</code>	\mapsto	<code>\mapsto</code>
\Leftrightarrow	<code>\Leftrightarrow</code>	\longmapsto	<code>\longmapsto</code>
\Downarrow	<code>\Downarrow</code>	\downarrow	<code>\downarrow</code>
\Downarrow	<code>\Downarrow</code>	\Downarrow	<code>\Downarrow</code>
\Longleftarrow	<code>\Longleftarrow</code>	\uparrow	<code>\uparrow</code>
\Longrightarrow	<code>\Longrightarrow</code>	\Uparrow	<code>\Uparrow</code>
\Longrightarrow	<code>\Longrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\Longleftrightarrow	<code>\Longleftrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\Longleftrightarrow	<code>\Longleftrightarrow</code>	\nearrow	<code>\nearrow</code>
\Longleftrightarrow	<code>\Longleftrightarrow</code>	\searrow	<code>\searrow</code>
\iff	<code>\iff</code>	\nwarrow	<code>\nwarrow</code>
\hookleftarrow	<code>\hookleftarrow</code>	\swarrow	<code>\swarrow</code>
\hookrightarrow	<code>\hookrightarrow</code>		

这些命令提供各种箭头。它们被划分为关系符号（第 200 页）。上面的竖直箭头同时也是定界符，因此你可以用 `\big` 等命令让它们变大（第 222 页）。

命令 \iff 和 \Longleftarrow 的差别之处在于，它在箭头两边生成额外间隔。

你可以用 `\buildrel` (第 212 页) 命令将符号或者其他文字放在箭头上方。

例子：

$\$ \$ f(x) \mapsto f(y) \iff x \mapsto y \$ \$$

结果：

$$f(x) \mapsto f(y) \iff x \mapsto y$$

■ 已命名的数学函数

\diamond	cos	\cos	sinh	\sinh	hom	\hom
	sin	\sin	tanh	\tanh	ker	\ker
	tan	\tan	det	\det	inf	\inf
	cot	\cot	dim	\dim	sup	\sup
	csc	\csc	exp	\exp	lim	\lim
	sec	\sec	ln	\ln	lim inf	\liminf
	arccos	\arccos	log	\log	lim sup	\limsup
	arcsin	\arcsin	lg	\lg	max	\max
	arctan	\arctan	arg	\arg	min	\min
	cosh	\cosh	deg	\deg	Pr	\Pr
	coth	\coth	gcd	\gcd		

这些命令以惯用的罗马字体排印各种数学函数的名称。如果你给这些命令中的任何一个加上上标或下标, TeX 将在通常的位置排版它。在陈列样式中, 对于 \det、\gcd、\inf、\lim、\liminf、\limsup、\max、\min、\Pr 和 \sup, TeX 将上标和下标当成极限那样排版, 即将它们直接放在函数名的上边或下边。

例子:

```
$\cos^2 x + \sin^2 x = 1 \quad \max_{a \in A} g(a) = 1$
```

结果:

$$\cos^2 x + \sin^2 x = 1 \quad \max_{a \in A} g(a) = 1$$

\bmod

此命令排印一个标明公式内的模运算的二元运算符。

例子:

```
$$x = (y+1) \bmod 2$$
```

结果:

$$x = (y + 1) \bmod 2$$

\pmod

此命令在公式末尾排印放在圆括号中的模运算。

例子:

```
$$x \equiv y+1 \pmod 2$$
```

结果:

$$x \equiv y + 1 \pmod{2}$$

■ 巨算符

\cap	\bigcap	$\backslash\bigcap$	\cup	\bigcup	$\backslash\bigcup$	\int	\smallint	$\backslash\smallint$
\cup	\bigcup	$\backslash\bigcup$	\uplus	\biguplus	$\backslash\biguplus$	\int	\int	$\backslash\int$
\odot	\bigodot	$\backslash\bigodot$	\vee	\bigvee	$\backslash\bigvee$	\oint	\oint	$\backslash\oint$
\oplus	\bigoplus	$\backslash\bigoplus$	\wedge	\bigwedge	$\backslash\bigwedge$	\prod	\bigprod	$\backslash\prod$
\otimes	\bigotimes	$\backslash\bigotimes$	\coprod	\bigcoprod	$\backslash\bigcoprod$	\sum	\bigsum	$\backslash\sum$

这些命令排印各种巨算符。TeX 在文内样式中排印小号字符，而在陈列样式中排印大号字符。巨算符是 TeX 数学符号的其中一类。TeX 在不同类数学符号间留下不同大小的间隔。

名称中带有 ‘big’ 的巨算符和对应的二元运算符（比如 \cap (\cap)，见第 198 页）不同，因为它们通常出现公式的开头。TeX 给巨算符留下间隔与二元运算符的不同。

不要混淆 ‘ \sum ’ (\sum) 和 ‘ Σ ’ (Σ)，或者 ‘ \prod ’ (\prod) 和 ‘ Π ’ (Π)。 Σ 和 Π 排印大写希腊字母，它们尺寸更小，外观也不同。

巨算符可以带有极限。下极限用下标指定，而上极限用上标指定。

例子：

```
$$\bigcap_{k=1}^r (a_k \cup b_k)$$
```

结果：

$$\bigcap_{k=1}^r (a_k \cup b_k)$$

例子：

```
$$\int_0^\pi \sin^2 ax dx = \frac{\pi}{2}$$
```

结果：

$$\int_0^\pi \sin^2 ax dx = \frac{\pi}{2}$$

\limits

在文内样式中，TeX 通常将极限放在巨算符后边。此命令让 TeX 将极限放在巨算符的上边和下边，而不是在后边。

如果你多次使用 `\limits`、`\nolimits` 或 `\displaylimits`, 仅最后一个命令生效。

例子:

```
Suppose that $\bigcap\limits_{i=1}^N q_i$ contains at least
two elements.
```

结果:

Suppose that $\bigcap_{i=1}^N q_i$ contains at least two elements.

`\nolimits`

在陈列样式中, TeX 通常将极限放在巨算符的上边和下边。此命令让 TeX 将极限放在巨算符后边, 而不是上边和下边。

积分算符 `\int` 和 `\oint` 是两个例外——除非像 `\int\limits` 这样指定, TeX 总是将极限放在算符的后边。(plain TeX 将 `\int` 和 `\oint` 定义为算符后跟着 `\nolimits` 的宏——这就是其默认表现有所不同的原因。)

如果你多次使用 `\limits`、`\nolimits` 或 `\displaylimits`, 仅最后一个命令生效。

例子:

```
$$\bigcap\nolimits_{i=1}^N q_i$$
```

结果:

$$\bigcap_{i=1}^N q_i$$

`\displaylimits`

此命令让 TeX 在陈列样式中将极限放在算符(包括积分符)的上边和下边, 在文内样式中将极限放在算符的后边。

如果你多次使用 `\limits`、`\nolimits` 或 `\displaylimits`, 仅最后一个命令生效。

例子:

```
$a(\lambda) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x)e^{-i\lambda x} dx$$
```

结果:

$$a(\lambda) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x)e^{-i\lambda x} dx$$

■ 标点

\cdotp
\ldotp

这两个命令分别排印居中的圆点和在基线上的圆点。它们仅可用于数学模式中。TeX 将它们视为标点，在前面不留间隔而在后面留下一点间隔。与此相反，对于用 \cdot 命令（第 198 页）生成的居中圆点，TeX 在其两侧留下相同大小的间隔。

例子：

\$x \cdotp y \quad x \ldotp y \quad x \cdot y\$

结果：

$x \cdot y \quad x \ldotp y \quad x \cdot y$

\colon

此命令排印一个冒号标点，它只能用在数学模式中。冒号标点 \colon 和冒号字符 (:) 的区别在于，‘:’ 是一个运算符，因此 TeX 在其左侧留下额外间隔，然而在 \colon 左侧却不留额外间隔。

例子：

\$f \colon t \quad f : t\$

结果：

$f : t \quad f : t$

上标和下标

◆ $\underline{}$ $\langle argument \rangle$ $\overline{}$ $\langle argument \rangle$
 $\backslash sb$ $\langle argument \rangle$ $\backslash sp$ $\langle argument \rangle$

各栏的两个命令都是等价的。第一栏的命令将 $\langle argument \rangle$ 排版为下标，而第二栏的命令将 $\langle argument \rangle$ 排版为上标。 $\backslash sb$ 和 $\backslash sp$ 命令主要用于无法使用下划线和插入符的终端中，或者用在重新定义了 ‘_’ or ‘^’ 但需要其原始定义的情况下。这些命令也用于设定求和号和积分号的下极限和上极限。

如果下标或上标不是单个记号，你需要将它放在编组中。TeX 并不处理下标和上标的优先级，因此它将拒绝类似 a_i_j 、 a^i_j 或 a^{i_j} 的公式。

下标和上标排版时通常用标号样式，或者小标号样式，如果它们是二阶标号，比如下标中的下标或下标中的上标。利用 `\scriptstyle` 和 `\scriptscriptstyle` 命令（第 207 页），你可以将数学公式的任何文本设为标号或小标号样式。

对任何以罗马字体排印命名数学函数（见第 203 页）的命令，你都可以给它添加下标和上标。在某些情形中（同样见第 203 页），下标和上标分别出现在函数名的下边和上边，如下面例子中的 `\lim` 和 `\det` 所示。

例子：

```
$x_3 \quad t_{\max} \quad a_{i_k} \quad \sum_{i=1}^n q_i \quad x^3 \quad e^{t \cos \theta} \quad r^{x^2} \quad \int_0^\infty f(x) dx
\lim_{x \leftarrow 0} f(x) \quad \det^{z \in A} \quad \sin^2 t
$$\lim_{x \leftarrow 0} f(x) \quad \det^{z \in A} \quad \sin^2 t$$
```

结果：

$$x_3 \quad t_{\max} \quad a_{i_k} \quad \sum_{i=1}^n q_i \quad x^3 \quad e^{t \cos \theta} \quad r^{x^2} \quad \int_0^\infty f(x) dx$$

$$\lim_{x \leftarrow 0} f(x) \quad \det^{z \in A} \quad \sin^2 t$$

■ 选用样式

```
\textstyle
\scriptstyle
\scriptscriptstyle
\displaystyle
```

这些命令覆盖 TeX 排版公式时通常使用的样式及其字体。如同类似 `\it` 的字体设置命令，它们在其所在编组结束前一直有效。当 TeX 给你要排版的公式选用了不合适的样式时，你可以使用这些命令。

例子：

```
$t + {\scriptstyle t} + {\scriptscriptstyle t} + {t}
```

结果：

$$t + t + t$$

```
\mathchoice{math_1}{math_2}{math_3}{math_4}
```

此命令让 TeX 根据当前样式选择并排版其中一个子公式 $math_1$ 、 $math_2$ 、 $math_3$ 或 $math_4$ 。也就是说，如果在陈列样式中，TeX 将

`\mathchoice` 排版为 $\langle math_1 \rangle$ ；在文本样式中排版为 $\langle math_2 \rangle$ ，在标号样式中排版为 $\langle math_3 \rangle$ ；而在小标号样式中排版为 $\langle math_4 \rangle$ 。

例子：

```
\def\mc{{\mathchoice{D}{T}{S}{SS}}}
The strange formula $ \mc_{} $ illustrates a
mathchoice.
```

结果：

The strange formula T_{SS} illustrates a mathchoice.

`\mathpalette <argument1> <argument2>`

此命令提供一种生成适用于四种样式的数学结构的简便方法。¹ 要使用它，通常你需要定义一个额外的宏，假设我们称它为 `\build`。调用 `\mathpalette` 就应该用 `\mathpalette\build<argument>` 这种形式。

`\build` 测试 `TEX` 位于何种样式，并相应地排版 `<argument>`。它应该定义为有两个参数。当你调用 `\mathpalette` 时，它以 #1 为选择样式的命令，#2 为 `<argument>` 转而调用 `\build`。因此，在 `\build` 的定义中，通过将某些东西放在 ‘#1’ 前面，就可以用当前样式排版它。在 *The T_EXbook* 第 360 页中有如何使用 `\mathpalette` 的例子，而在 *The T_EXbook* 第 151 页中有它如何运作的进一步解释。

¹ 译注：该宏定义为 `\def\mathpalette{\#1\#2{\mathchoice{\#1\displaystyle{\#2}}{\#1\textstyle{\#2}}{\#1\scriptstyle{\#2}}{\#1\scriptscriptstyle{\#2}}}}`。

复合符号

■ 数学重音

- ◊ \acute{} 锐音符, 如同 \acute{x}
- \bar{} 下线符, 如同 \bar{x}
- \barwedge{} 上线符, 如同 \bar{x}
- \breve{} 短音符, 如同 \breve{x}
- \check{} 抑扬符, 如同 \check{x}
- \ddot{} 双点符, 如同 \ddot{x}
- \dot{} 上点符, 如同 \dot{x}
- \grave{} 钝音符, 如同 \grave{x}
- \hat{} 尖角符, 如同 \hat{x}
- \widehat{} 宽尖角符, 如同 $\widehat{x+y}$
- \tilde{} 波浪符, 如同 \tilde{x}
- \widetilde{} 宽波浪符, 如同 $\widetilde{z+a}$
- \vec{} 向量符, 如同 \vec{x}

这些命令在数学公式上排印重音标记。你通常需要在它们后面留下空格。宽重音可以应用到多字符子公式中; TeX 将把重音放在子公式的中间。其他重音仅在应用到单个字符时才有用。

例子:

```
$\dot{t}^n \quad \widetilde{v_1 + v_2}$
```

结果:

$$\dot{t}^n \quad \widetilde{v_1 + v_2}$$

`\mathaccent <mathcode>`

此命令让 TeX 排版字体族和字符编码由 `<mathcode>` 给出的数学重音。(TeX 忽略数学码中的类。) 请参阅 *The TeXbook* 附录 G 对 TeX 如

何放置该重音的详细介绍。经常将 `\mathaccent` 放在宏定义中，以给数学重音一个名称。

例子：

```
\def\acute{\mathaccent "7013}
```

参阅：“Accents”（第 100 页）。

■ 分式和其他堆叠运算

☆ `\over`
`\atop`
`\above <dimen>`
`\choose`
`\brace`
`\brack`

这些命令将一个子公式堆放在另一个子公式之上。我们将解释 `\over` 如何作用，然后说明其他命令与它的关系。

`\over` 命令通常用于排印分式。如果你按下面几种形式之一撰写：

```
$$<formula_1>\over<formula_2>$$
$<formula_1>\over<formula_2>$
\left<delim><formula_1>\over<formula_2>\right><delim>
{<formula_1>\over<formula_2>}
```

你将得到分子为 $<formula_1>$ 分母为 $<formula_2>$ 的分式，即 $<formula_1>$ 除以 $<formula_2>$ 。在前面三种形式中，`\over` 非显式地包含在一个编组中；它吸收左边和右边的内容直到遇到边界，即编组的开头和结尾。

你不可以在一个公式中多次使用 `\over` 或这批命令的其他命令。因此下面的公式：

```
$$a \over n \choose k$$
```

是不合法的。这不是什么严重的限制，因为你总可以将其中一个命令放在花括号中。作此限制的原因是，如果你把这些命令的其中两个放在同一个公式中，TeX 将不知道如何划分它们。

其他命令与 `\over` 类似，但有所不同：

- `\atop` 去掉分式的横线。
- `\above` 给出厚度为 $<dimen>$ 的分式横线。

- `\choose` 去掉分式横线，并将结构放在圆括号中。(称它为“选择”，是因为 $\binom{n}{k}$ 表示从 n 个东西中任取 k 个的所有选取方式的数目。)
- `\brace` 去掉分式横线，并将结构放在花括号中。
- `\brack` 去掉分式横线，并将结构放在方括号中。

例子：

```
$$\{n+1 \over n-1} \qquad \{n+1 \atop n-1} \qquad \{n+1 \brace n-1} \qquad \{n+1 \brack n-1}$$
```

结果：

$$\frac{n+1}{n-1} \quad n+1 \atop n-1 \quad \frac{n+1}{n-1} \quad \binom{n+1}{n-1} \quad \left\{ \begin{matrix} n+1 \\ n-1 \end{matrix} \right\} \quad \left[\begin{matrix} n+1 \\ n-1 \end{matrix} \right]$$

```
\overwithdelims <delim1> <delim2>
\atopwithdelims <delim1> <delim2>
\abovewithdelims <delim1> <delim2> <dimen>
```

这里的每个命令都将一个子公式堆放在另一个子公式之上，并将整个结构的左边用 $\langle \text{delim}_1 \rangle$ ，右边用 $\langle \text{delim}_2 \rangle$ 包围。这些命令遵循与 `\over`、`\atop` 和 `\above` 相同的规则。`\abovewithdelims` 后面的 $\langle \text{dimen} \rangle$ 指定分式横线的厚度。

例子：

```
$$m \overwithdelims () n \qquad
m \atopwithdelims || n \qquad
m \abovewithdelims \{\} 2pt n$$
```

结果：

$$\left(\frac{m}{n} \right) \quad \left| \begin{matrix} m \\ n \end{matrix} \right| \quad \left\{ \begin{matrix} m \\ n \end{matrix} \right\}$$

\cases

此命令排印一个表示从多个情形中选择的数学形式。每种情形由两部分组成，两者以‘&’分隔。TeX 将第一部分视为数学公式，第二部分视为普通文本。每个情形之后必须加上 `\cr`。

例子：

```
 $$g(x,y) = \cases{f(x,y), & if $x < y$\cr f(y,x), & if $x > y$\cr 0, & otherwise.\cr}$$
```

结果：

$$g(x,y) = \begin{cases} f(x,y), & \text{if } x < y \\ f(y,x), & \text{if } x > y \\ 0, & \text{otherwise.} \end{cases}$$

```
\underbrace {<argument>}
\overbrace {<argument>}
\underline {<argument>}
\overline {<argument>}
\overleftarrow {<argument>}
\overrightarrow {<argument>}
```

这些命令将可伸长的花括号、横线或箭头放在由 *<argument>* 给出的子公式的上边或下边。TeX 将让这些结构足够宽以适应内容。当 TeX 排印可伸长的花括号、横线或箭头时，它只考虑包含 *<argument>* 的盒子的尺寸。如果你在一个公式中使用这些命令中的两个以上，其中排印的花括号、横线或箭头之间可能无法恰当地对齐。你可以使用 `\mathstrut` 命令（第 173 页）克服此困难。

例子：

```
 $$\displaylines{ \underbrace{x \circ y}\quad \overbrace{x \circ y}\quad \underline{x \circ y}\quad \overline{x \circ y}\quad \overleftarrow{x \circ y}\quad \overrightarrow{x \circ y} \\ \overline{r + t} \quad \overline{r + t} }$$
```

结果：

$$\underbrace{x \circ y}_{\bar{r} + \bar{t}} \quad \overbrace{x \circ y}^{\overline{r + t}} \quad \underline{x \circ y} \quad \overline{x \circ y} \quad \overleftarrow{x \circ y} \quad \overrightarrow{x \circ y}$$

```
\buildrel {formula} \over {relation}
```

此命令将 $\langle formula \rangle$ 所在的盒子放在 $\langle relation \rangle$ 上边。TeX 处理间隔时将结果视为一个关系符（见“类”，第 57 页）。

例子：

```
$\backslash buildrel \rm def \over \equiv$
```

结果：

$$\stackrel{\text{def}}{=}$$

■ 圆点

☆ `\ldots`
`\cdots`

这两个命令都排印三个一排的圆点。对于 `\ldots`，圆点放在基线上；对于 `\cdots`，圆点放在中轴线上（见第 225 页对 `\vcenter` 的解释）。

例子：

```
$t_1 + t_2 + \cdots + t_n \qquad x_1, x_2, \ldots, x_r$
```

结果：

$$t_1 + t_2 + \cdots + t_n \qquad x_1, x_2, \ldots, x_r$$

☆ `\vdots`

此命令排印三个竖直的圆点。

例子：

```
$$\eqalign{f(\alpha_1) &= f(\beta_1) \cr
&\noalign{\kern -4pt} \% \cr
&\phantom{f(\alpha_1)} \vdots \cr
f(\alpha_k) &= f(\beta_k) \cr}$$
```

结果：

$$f(\alpha_1) = f(\beta_1)$$

⋮

$$f(\alpha_k) = f(\beta_k)$$

\ddots

此命令排印斜线上的三个圆点。它常用于表示沿矩阵对角线的重复。

例子：

```
 $$\pmatrix{0&\ldots&0\cr\noalign{\vdots}\vdots&\ddots&\vdots\cr 0&\ldots&0}$$
```

结果：

$$\begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}$$

参阅：\dots（第 99 页）。

■ 定界符

```
\lgroup  
\rgroup
```

这两个命令排印大号的左和右圆括号，它们分别作为开定界符和闭定界符。这两个定界符的最小可用尺寸为 \Big。如果使用更小的尺寸，你将得到奇怪的字符。

例子：

```
 $$\lgroup\dots\rgroup\qquad\bigg\lgroup\dots\bigg\rgroup$$
```

结果：

$$(\dots) \quad \left(\dots \right)$$

☆ \left
\right

这两个命令必须按照下面模式一起使用：

```
\left \langle delim_1 \rangle \langle subformula \rangle \right \langle delim_2 \rangle
```

这个构造将让 TeX 排印 $\langle subformula \rangle$ ，并用定界符 $\langle delim_1 \rangle$ 和 $\langle delim_2 \rangle$ 包围它。TeX 调整定界符的竖直尺寸以适应 $\langle subformula \rangle$ 的竖直尺寸（高度加深度）。 $\langle delim_1 \rangle$ 和 $\langle delim_2 \rangle$ 不需要相对应。举个例子，在使用 \left 和 \right 时，你可以将 ‘]’ 作为左定界符，而将 ‘(’ 作为右定界符。

\left 和 \right 有个重要性质是它们定义了一个编组，即它们能够充当左和右花括号。当你在 \left 和 \right 之间放上 \over (第 210 页) 或其他相关命令时，此编组性质就很有用，因为你无需在 \over 构造的分式两边加上花括号。

如果你需要左定界符但不需要右定界符，你可以用 ‘.’ 代替你不需要的定界符，这样它就变成一个空白（宽度为 \nulldelimiterspace）。

例子：

```
$\$\\left\\Vert\\matrix{a&b\\cr c&d\\cr}\\right\\Vert
\\qquad \\left\\uparrow q_1\\atop q_2\\right..$$
```

结果：

$$\begin{array}{|c c|} \hline a & b \\ \hline c & d \\ \hline \end{array} \quad \begin{array}{l} \uparrow q_1 \\ q_2 \end{array}$$

\delimiter <number>

此命令排印用 <number> 刻画其特性的定界符。<number> 通常用十六进制表示。在 TeX 需要定界符的任何地方你都可以用 \delimiter 命令代替一个字符（尽管此命令很少在宏定义之外的地方使用）。假设 <number> 为十六进制数 $cs_1s_2s_3l_1l_2l_3$ 。则 TeX 知道该定界符属于第 c 类，小号变体为 $s_1s_2s_3$ ，而大号变体为 $l_1l_2l_3$ 。这里 $s_1s_2s_3$ 表示第 s_1 族位置 s_2s_3 的数学字符， $l_1l_2l_3$ 类似。这里使用与 \mathcode (第 267 页) 一样的约定。

例子：

```
\def\\vert{\\delimiter "026A30C} % As in plain TeX.
```

```
\delimiterfactor [ <number> parameter ]
\delimitershortfall [ <number> parameter ]
```

这两个参数共同确定了定界符高度与其中子公式的竖直尺寸的关系。\\delimiterfactor 给出定界符高度相对子公式竖直尺寸的最小比例，而 \\delimitershortfall 给出定界符高度相对子公式竖直尺寸的最大差距。

假设包含子公式的盒子的高度为 h 深度为 d ，且令 $y = 2 \max(h, d)$ 。设 \\delimiterfactor 的值为 f ，\\delimitershortfall 的值为 δ 。则 TeX 选取的定界符高度至少为 $y \cdot f / 1000$ ，且至少为 $y - \delta$ 。特别地，如果 \\delimiterfactor 恰好为 1000，则 TeX 将试着生成一个至少和其中的子公式一样高的定界符。见 *The TeXbook* 第 152 页和第 446 页（规则 19）中 TeX 如何使用这些参数的细节。Plain TeX 设定 \\delimiterfactor 为 901，\\delimitershortfall 为 5pt。

参阅: `\delcode` (第 268 页)、`\vert`、`\Vert` 和 `\backslash` (第 197 页)。■

■ 矩阵

```
\matrix {<line> \cr ... <line> \cr}
\pmatrix {<line> \cr ... <line> \cr}
\bmatrix {<line> \cr ... <line> \cr}
```

这三个命令每个都排印一个矩阵, 输入矩阵时各行的元素之间用 ‘&’ 分隔, 而各行用 `\cr` 结尾。(这里使用与阵列一样的形式。) 这些命令之间的区别如下:

- `\matrix` 排印一个四周空白不带定界符的矩阵。
- `\pmatrix` 排印一个两边带圆括号的矩阵。
- `\bmatrix` 排印一个将第一行和第一列视为标号的矩阵。(第一行的第一个元素通常为空白。) 矩阵的其他元素被圆括号包含。

通过增加竖直延伸, TeX 能够为 `\pmatrix` 和 `\bmatrix` 制作足够大的圆括号。如果你需要用不同于圆括号的定界符包围矩阵, 你应当将 `\matrix` 与 `\left` 和 `\right` (第 214 页) 合起来使用。

例子:

```
$$\displaylines{
\matrix{t_{11}&t_{12}&t_{13}\cr
t_{21}&t_{22}&t_{23}\cr
t_{31}&t_{32}&t_{33}}\quad\quad\quad
\left(\begin{array}{ccc} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{array}\right)
\pmatrix{t_{11}&t_{12}&t_{13}\cr
t_{21}&t_{22}&t_{23}\cr
t_{31}&t_{32}&t_{33}}\quad\quad\quad
\bmatrix{t_{11}&t_{12}&t_{13}\cr
t_{21}&t_{22}&t_{23}\cr
t_{31}&t_{32}&t_{33}}\quad\quad\quad
\left|\begin{array}{ccc} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{array}\right|}$$
```

结果:

$$\begin{array}{ccc} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{array} \quad \left\{ \begin{array}{ccc} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{array} \right\}$$

$$\begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} \quad \begin{matrix} c_1 & c_2 & c_3 \\ r_1 & r_2 & r_3 \end{matrix} \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix}$$

■ 根号与根数

◊ `\sqrt {argument}`

此命令排印 `argument` 的平方根。

例子：

```
$\$x = {-b\pm\sqrt{b^2-4ac}} \over 2a}$$
```

结果：

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

◊ `\root {argument}_1 \of {argument}_2`

此命令排印 `argument` 的 `argument` 次根号。

例子：

```
$\root \alpha \of {r \cos \theta} $
```

结果：

$$\sqrt[r \cos \theta]{}$$

`\radical {number}`

此命令排印用 `number` 刻画其特性的根数符号。它使用的定界码表示法与 `\delcode` 命令（第 268 页）的相同。

例子：

```
\def\sqrt{\radical "270370} % as in plain TeX
```

方程编号

◊ `\eqno`

`\leqno`

这两个命令给陈列公式加上方程编号。`\eqno` 将编号放在右侧，而 `\leqno` 将编号放在左侧。这两个命令必须放在公式末尾。如果你有个

多行陈列公式，而你希望给不止一行编号，你可以用 `\eqalignno` 或 `\leqalignno` 命令（第 218 页）。

这两个命令只能在陈列数学模式中使用。

例子：

```
$$e^{i\theta} = \cos \theta + i \sin \theta \eqno{(11)}$$
```

结果：

$$e^{i\theta} = \cos \theta + i \sin \theta \quad (11)$$

例子：

```
$$\cos^2 \theta + \sin^2 \theta = 1 \leqno{(12)}$$
```

结果：

$$(12) \quad \cos^2 \theta + \sin^2 \theta = 1$$

多行陈列公式

```
\displaylines { <line> \cr ... <line> \cr }
```

此命令排印一个多行陈列公式，其中的各行独立地居中放置。你可以使用 `\noalign` 命令（第 189 页）改变多行陈列公式中两行的间隔。

如果你希望给多行陈列公式的某个或某些方程添加编号，你应当使用 `\eqalignno` 或 `\leqalignno`。

例子：

```
$$\displaylines{ (x+a)^2 = x^2+2ax+a^2 \cr (x+a)(x-a) = x^2-a^2 \cr }$$
```

结果：

$$\begin{aligned} (x+a)^2 &= x^2 + 2ax + a^2 \\ (x+a)(x-a) &= x^2 - a^2 \end{aligned}$$

```
\eqalign { <line> \cr ... <line> \cr }
```

```
\eqalignno { <line> \cr ... <line> \cr }
```

```
\leqalignno { <line> \cr ... <line> \cr }
```

这些命令排印一个多行陈列公式，其中某些行的对应部分竖直对齐。`\eqalignno` 和 `\leqalignno` 命令还允许你给某个或某些行添加方程编号。`\eqalignno` 将方程编号放在右侧，而 `\leqalignno` 将编号放在左侧。

陈列公式的每行用 `\cr` 结尾。各行需要对齐的各部分（多半是等号）前面加上 ‘&’。方程编号放在公式末尾，它的前面也要加上 ‘&’。你可以在单个陈列公式中多次使用这些命令以排印多组方程。在这种情形中，只有最右边或最左边的那组方程可以用 `\eqalignno` 或 `\leqalignno` 编号。

你可以使用 `\noalign` 命令（第 189 页）改变多行陈列公式中两行的间隔。

例子：

```
$$\left.\begin{aligned}f_1(t) &= 2t \\ f_2(t) &= t^3 \\ f_3(t) &= t^2 - 1\end{aligned}\right\} \quad \left.\begin{aligned}g_1(t) &= t \\ g_2(t) &= 1\end{aligned}\right\}$$
```

结果：

$$\left\{ \begin{array}{l} f_1(t) = 2t \\ f_2(t) = t^3 \\ f_3(t) = t^2 - 1 \end{array} \right\} \quad \left\{ \begin{array}{l} g_1(t) = t \\ g_2(t) = 1 \end{array} \right\}$$

例子：

```
$$\begin{aligned}\sigma^2 &= E(x - \mu)^2 \\ &= \frac{1}{n} \sum_{i=0}^n (x_i - \mu)^2 \\ &= E(x^2) - \mu^2\end{aligned} \quad (12)
```

结果：

$$\begin{aligned}\sigma^2 &= E(x - \mu)^2 \\ &= \frac{1}{n} \sum_{i=0}^n (x_i - \mu)^2 \\ &= E(x^2) - \mu^2\end{aligned} \quad (12)$$

例子：

```
$$\leqalignno{  
 \sigma^2 &= E(x-\mu)^2 \\  
 &= E(x^2) - \mu^2}
```

结果：

$$(6) \quad \sigma^2 = E(x - \mu)^2$$

$$(7) \quad = E(x^2) - \mu^2$$

例子：

```
$$\eqalignno{  
 (x+a)^2 &= x^2 + 2ax + a^2 \\  
 (x+a)(x-a) &= x^2 - a^2}
```

% same effect as \displaylines but with an equation number

结果：

$$(x + a)^2 = x^2 + 2ax + a^2 \tag{19}$$

$$(x + a)(x - a) = x^2 - a^2$$

数学公式字体

- ◊ \cal use calligraphic uppercase font
- \mit use math italic font
- \oldstyle use old style digit font

这些命令让 TeX 用指定的字体排版之后的文本。你只能在数学模式中使用它们。\\mit 命令可用于排印斜体大写希腊字母。你还可以用“选择字体”（第 102 页）中的那些命令改变数学模式中的字体。

例子：

```
 ${\cal XYZ} \quad  
 {\mit AaBb\Gamma\Delta\Sigma} \quad  
 {\oldstyle 0123456789}$
```

结果：

\mathcal{XYZ} $AaBb\Gamma\Delta\Sigma$ 0123456789

```
\itfam   family for italic type  
\bfffam family for boldface type  
\slfam   family for slanted type  
\ttfam   family for typewriter type
```

这些命令定义几种用于数学模式的字体族。它们主要用在 `\it`、`\bf`、`\sl` 和 `\tt` 命令的定义中，使这些命令能在数学模式中使用。

```
\fam [⟨number⟩ parameter]
```

在数学模式时，TeX 通常用字符的数学码指定的字体族排版该字符。但是，如果 TeX 在数学模式中遇到第 7 类（变量）字符，它将用由 `\fam` 的值给出的字体族排版该字符，只要 `\fam` 的值在 0 和 15 之间。如果 `\fam` 的值不在该范围内，TeX 就像通常情形那样使用字符的数学码指定的字体族。TeX 在进入数学模式时设定 `\fam` 为 -1。在数学模式之外，`\fam` 无任何效果。

通过赋予 `\fam` 不同的值，你能让 TeX 用不同方式排版普通字符，比如变量。举个例子，设定了 `\fam` 为 `\ttfam`，你可以让 TeX 用打字机字体排版变量。Plain TeX 在定义 `\tt` 宏时，除了其他设定之外，还设定 `\fam` 等于 `\ttfam`。

例子：

```
\def\bf{\fam\bffam\tenbf} % As in plain TeX.
```

```
\textfont ⟨family⟩ [⟨fontname⟩ parameter]  
\scriptfont ⟨family⟩ [⟨fontname⟩ parameter]  
\scriptscriptfont ⟨family⟩ [⟨fontname⟩ parameter]
```

这三个参数分别选择 TeX 排版指定族的指定样式时所用的字体。这些选择在数学模式之外无任何效果。

例子：

```
\scriptfont2 = \sevensy % As in plain TeX.
```

参阅：“字体风格”（第 103 页）。

构造数学符号

■ 增大定界符

$\hat{\diamond}$	$\backslash\big$	$\backslash\Big$	$\backslash\bigg$	$\backslash\Bigg$
	$\backslash\bigl$	$\backslash\Bigl$	$\backslash\biggl$	$\backslash\Biggl$
	$\backslash\bigm$	$\backslash\Bigr$	$\backslash\biggm$	$\backslash\Biggm$
	$\backslash\bigr$		$\backslash\biggr$	$\backslash\Biggr$

这些命令让定界符比它们的正常尺寸还大。这四栏中的命令生成依次增大的尺寸。 $\backslash\big$ 、 $\backslash\bigl$ 、 $\backslash\bigr$ 和 $\backslash\bigm$ 的区别在于增大的定界符所属的类：

- $\backslash\big$ 生成一个普通符号。
- $\backslash\bigl$ 生成一个开符号。
- $\backslash\bigr$ 生成一个闭符号。
- $\backslash\bigm$ 生成一个关系符号。

TEX 从字符所属的类确定要在该字符两边留下多大的空格。

例子：

```
$(x) \quad \backslash\bigl(x\backslash\bigr) \quad \backslash\Bigl(x\backslash\Bigr) \quad \backslash\biggl(x\backslash\biggr)
\quad \backslash\biggl(x\backslash\biggr) \quad \backslash\Biggl(x\backslash\Biggr)\quad
[x] \quad \backslash\bigl[x\backslash\bigr] \quad \backslash\Bigl[x\backslash\Bigr] \quad \backslash\biggl[x\backslash\biggr]
\quad \backslash\biggl[x\backslash\biggr] \quad \backslash\Biggl[x\backslash\Biggr]$$
```

结果：

$$(x) \quad (x) \quad (x) \quad \left(x \right) \quad [x] \quad [x] \quad [x] \quad \left[x \right] \quad \left[x \right]$$

■ 大符号的一部分

```
\downbracefill
\upbracefill
```

这两个命令分别排印朝上和朝下的可伸展水平花括号。TeX 将让花括号足够宽。这两个命令用于定义 `\overbrace` 和 `\underbrace` (第 212 页)。

例子：

```
$$\hbox{ to } 1in{\downbracefill} \quad
\hbox{ to } 1in{\upbracefill}$$
```

结果：



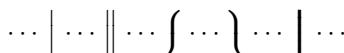
```
\arrowvert
\Arrowvert
\l moustache
\r moustache
\bracevert
```

这些命令排印某些大定界符的一部分，把它们也用作定界符。它们取自 cmex10 数学字体中的字符。

例子：

```
$$\cdots \Big| \cdots \Big\| \cdots \Big\} \cdots \Big( \cdots \Big| \cdots
```

结果：



对齐部分公式

■ 对齐数学重音

`\skew <number> <argument1> <argument2>`

此命令将重音 $\langle argument_1 \rangle$ 相对 $\langle argument_2 \rangle$ 从它的正常位置往右移动 $\langle number \rangle$ 个数学单位。此命令常用于调整在其他重音之上的重音的位置。

例子：

```
$$\skew 2\bar{\bar{z}}\quad\skew 3\tilde{\tilde{y}}\quad
\skew 4\tilde{\hat{x}}$$
```

结果：

$\bar{\bar{z}}$ $\tilde{\tilde{y}}$ $\tilde{\hat{x}}$

`\skewchar [<number> parameter]`

字体的 `\skewchar` 是字体中的某个字符，它在字体度量文件中定义的紧排确定了数学重音的位置。也就是说，假设 TeX 要给字符 ‘x’ 加上数学重音，则 TeX 检查字符对 ‘x\skewchar’ 是否有个紧排；如果有，它就以该紧排的值移动该重音。TeX 放置数学重音的完整算法（这涉及到很多事情）在 *The TeXbook* 附录 G 中描述。

如果 `\skewchar` 的值不在 0–255 的范围内，TeX 将紧排的值当作零。

注意 $\langle font \rangle$ 是一个控制序列，它是字体的名称，而不是字体文件的名称 $\langle fontname \rangle$ 。小心：对 `\skewchar` 的赋值在编组结束时并不会还原。如果你想局部改变 `\skewchar`，你需要显式地保存和还原它的原始值。

`\defaultskewchar [<number> parameter]`

在执行 `\font` 命令读取字体的度量文件时，TeX 设定该字体的 `\skewchar` 等于 `\defaultskewchar`。如果 `\defaultskewchar` 的值不在 0–255 的范围内，TeX 默认就不设定 `\skewchar` 的值。Plain TeX 设定 `\defaultskewchar` 等于 -1，一般不需要改动它。

■ 坚直对齐素材

```
\vcenter { <vertical mode material> }
\vcenter to <dimen> { <vertical mode material> }
\vcenter spread <dimen> { <vertical mode material> }
```

每个数学公式都有一个不可见的“轴线”，TEX 将它作为该公式的水平中心线。举个例子，由分式组成的公式的轴线就在分数线的中心。`\vcenter` 命令让 TEX 将 *<vertical mode material>* 放入竖直盒子中，并将该竖直盒子与当前公式的轴线居中对齐。

此命令的第一种形式如上所述居中放置素材。后两种形式竖直扩展或收缩素材，如同 `\vbox` 命令（第 166 页）。

例子：

```
$$\{n \choose k\} \def \rm buildrel \over \equiv \gt
\vcenter{\hsize 1.5 in \noindent the number of
combinations of $n$ things taken $k$ at a time}$$
```

结果：

$$\binom{n}{k} \stackrel{\text{def}}{=} \begin{array}{l} \text{the number of combinations of } n \text{ things taken } k \\ \text{at a time} \end{array}$$

生成间隔

■ 固定宽度数学间隔

```
\!
\,
\gt
\;
```

这些命令在公式中生成各种大小的额外间隔。它们使用数学单位来定义，因此 TEX 会根据当前样式调整间隔的大小。

- `\!` 生成负的细小间隔，即它让相邻子公式的间隔减去该细小间隔的大小。

- \, 生成细小间隔。
 - \> 生成中等间隔。
 - \; 生成较大间隔。

例子：

```
$$00\quad0!\quad0\quad0\,,\quad0\quad0\quad>\quad0\quad0\quad;\quad0\quad0\quad  
{\scriptstyle 00\quad0!\quad0\quad0\,,\quad0\quad0\quad>\quad0\quad0\quad;\quad0}$$
```

结果：

00 00 00 00 00 00 00 00 00 00 00

```
\thinmuskip [ <muglue> parameter ]
\medmuskip [ <muglue> parameter ]
\thickmuskip [ <muglue> parameter ]
```

这些参数定义了数学模式中细小、中等和较大间隔的大小。

例子：

\$00\quad 0\mskip\thinmuskip0\quad 0\mskip\medmuskip0
 \quad 0\mskip\thickmuskip0\$

结果：

00 00 00 00

\jot [*dimen*] parameter

此参数定义为三个点的距离（除非你改变了它）。在用 `\openup` 命令
分开陈列公式各行时，`\jot` 是一个实用的度量单位。²

例子：

\$\$\vbox{\halign{\$\hfil#\hfil\$\cr x\cr y\cr}}\$\$

\$\$\backslash openup{2}\backslash jot\backslash vbox{\backslash halign{\backslash hfil\#1\backslash hfil\backslash cr x\backslash cr y\backslash cr}}\backslash closeup{2}\$\$

结果：

x

y

x

y

² 译注：下面的例子为译者所加。请参阅 \openup 命令（第 138 页）。

■ 可变宽度数学间隔

`\mkern <mudimen>`

此命令生成一个宽度为 $\langle mudimen \rangle$ 的紧排，即空白间隔。该紧排用数学单位表示，因此在不同样式中有不同的尺寸。除了使用数学单位外，此命令与水平模式的 `\kern` (第 160 页) 的表现类似。

例子：

```
$0\mkern13mu 0 \qquad {\scriptscriptstyle 0 \mkern13mu 0} $
```

结果：

```
0 0     o o
```

`\mskip <mudimen1> plus <mudimen2> minus <mudimen3>`

此命令生成一个水平粘连，它的自然宽度为 $\langle mudimen_1 \rangle$ ，伸长量为 $\langle mudimen_2 \rangle$ ，收缩量为 $\langle mudimen_3 \rangle$ 。该粘连用数学单位表示，因此将随着样式的变化而变化。除了使用数学单位外，此命令与 `\hskip` (第 159 页) 的表现类似。

例子：

```
$0\mskip 13mu 0 \quad {\scriptscriptstyle 0 \mskip 13mu 0} $ ■
```

结果：

```
0 0     o o
```

`\nonscript`

在排版标号或小标号样式时，如果 TeX 在粘连或紧排跟前遇到此命令，它就丢弃该粘连或紧排。`\nonscript` 在其他样式中无任何效果。

此命令提供一种“收紧”标号和小标号样式中的间隔的方法；通常用小号字体排版这两个样式。在宏定义之外的地方，此命令很少用到。

例子：

```
\def\ab{a\nonscript\; b}
$ \ab^{\ab} $
```

结果：

```
a bab
```

参阅：`\kern` (第 160 页) 和 `\hskip` (第 159 页)。

■ 陈列公式的间隔参数

`\displaywidth [〈dimen〉 parameter]`

此参数指定 TeX 对陈列公式所允许的最大宽度。如果 TeX 无法将陈列公式放入这样宽的空间中，它将生成一个过满的水平盒子并给出警告。TeX 在遇到 ‘\$\$’ 开始陈列公式时就设定 `\displaywidth` 的值。它的初始值为 `\hsize` (第 113 页)，除非段落形状改变了。见 *The TeXbook* 第 188–189 页中对此参数的更仔细的说明。

`\displayindent [〈dimen〉 parameter]`

此参数指定 TeX 对陈列公式的缩进量。TeX 在遇到 ‘\$\$’ 开始陈列公式时就设定 `\displayindent` 的值。通常它的初始值为零，但如果段落形状表明该陈列公式需要移动距离 s ，TeX 就设定 `\displayindent` 等于 s 。见 *The TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

`\predisplaysize [〈dimen〉 parameter]`

TeX 设定此参数等于陈列公式之前的文本行的宽度。TeX 利用 `\predisplaysize` 确定是否让陈列公式的起始点位于前一行结尾处的左边，即它在外观上是否可能与前一行重叠。如果有重叠，TeX 在排版陈列公式时使用 `\abovedisplayskip` 和 `\belowdisplayskip` 粘连；否则 TeX 使用 `\abovedisplayshortskip` 和 `\belowdisplayshortskip` 粘连。见 *The TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

`\abovedisplayskip [〈glue〉 parameter]`

此命令指定当陈列公式的起始点位于前一行结尾处的左边时，即它在外观上可能与前一行有重叠时，TeX 在陈列公式之前插入的竖直粘连的大小。Plain TeX 设定 `\abovedisplayskip` 等于 `12pt plus3pt minus9pt`。见 *The TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

`\belowdisplayskip [〈glue〉 parameter]`

此命令指定当陈列公式的起始点位于前一行结尾处的左边时，即它在外观上可能与前一行有重叠时，TeX 在陈列公式之后插入的竖直粘连的大小。Plain TeX 设定 `\belowdisplayskip` 等于 `12pt plus3pt minus9pt`。见 *The TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

`\abovedisplayshortskip [⟨glue⟩ parameter]`

此命令指定当陈列公式的起始点位于前一行结尾处的右边时，即它在外观上不会与前一行有重叠时， \TeX 在陈列公式之前插入的竖直粘连的大小。Plain \TeX 设定 `\abovedisplayshortskip` 等于 `0pt plus3pt`。见 *The \TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

`\belowdisplayshortskip [⟨glue⟩ parameter]`

此命令指定当陈列公式的起始点位于前一行结尾处的右边时，即它在外观上不会与前一行有重叠时， \TeX 在陈列公式之后插入的竖直粘连的大小。Plain \TeX 设定 `\belowdisplayshortskip` 等于 `7pt plus3pt minus4pt`。见 *The \TeXbook* 第 188–189 页中对此参数的更仔细的介绍。

■ 其他的数学间隔参数

`\mathsurround [⟨dimen⟩ parameter]`

此参数指定 \TeX 在文内数学公式（即放在两个 \$ 之间的公式）两边插入的间隔的大小。见 *The \TeXbook* 第 162 页对此行为的进一步解释。Plain \TeX 设定 `\mathsurround` 为 `0pt`。

`\nulldelimiterspace [⟨dimen⟩ parameter]`

此参数指定空定界符生成的间隔的大小。Plain \TeX 设定 `\nulldelimiterspace` 等于 `1.2pt`。

`\scriptspace [⟨dimen⟩ parameter]`

此参数指定 \TeX 在上标或下标前后插入的间隔的大小。上标或下标之后的 `\nonscript` 命令（第 227 页）可以取消此间隔。Plain \TeX 设定 `\scriptspace` 等于 `0.5pt`。

分类数学结构

\mathord	\mathopen
\mathop	\mathclose
\mathbin	\mathpunct
\mathrel	

这些命令让 TeX 把随后的结构归入指定的类（见 *The TeXbook* 第 154 页对类的定义）。它们按照类编号的大小顺序排列，从 0 到 6。它们主要用于按照指定的类调整该结构两边的间隔大小。

例子：

```
$\mathop{\rm minmax}\limits_{t \in A \cup B} t$  
% By treating minmax as a math operator, we can get TeX to  
% put something underneath it.
```

结果：

$$\minmax_{t \in A \cup B} t$$

\mathinner

此命令让 TeX 将随后的结构视为“内部公式”，比如分式，并据此调整间隔。它与上面刚提到的类命令类似。

特殊处理数学公式

```
\everymath [ <token list> parameter ]  
\everydisplay [ <token list> parameter ]
```

这两个命令分别指定 TeX 在每个文内公式或陈列公式开头插入的记号列。你可以利用 \everymath 或 \everydisplay 在每个数学公式开头作特殊处理。你务必清楚，若你需要同时处理两种公式，你必须同时设定这两个参数。

例子：

```
\everydisplay={\heartsuit\quad}
\everymath = {\clubsuit}
$3$ is greater than $2$ for large values of $3$.
$$4>3$$
```

结果：

$\heartsuit 3$ is greater than $\heartsuit 2$ for large values of $\clubsuit 3$.

$\heartsuit \quad 4 > 3$



9

一般操作命令

这章介绍 T_EX 的编程功能和不适合放入前几章的所有内容。在“命令描述”(第 3 页)一节中，给出了这章的惯例。

命名及修改字体

```
\font  
\font <control sequence> = <fontname>  
\font <control sequence> = <fontname> scaled <number>  
\font <control sequence> = <fontname> at <dimen>
```

单独使用时，\font 控制序列指代当前的字体。 \font 在这个时候并不是一个真正的命令，它仅仅作为其它命令的一个参数。

其它的三个 \font 形式，字体名 <fontname> 用来指代定义一个字体所需要的一系列文件。在这些形式下，\font 是命令。每种形式都有两个效果：

- 1) 它定义了一个名为 <control sequence> 的控制序列，用来选择名为 <fontname> 的字体，而且该字体可能被缩放(见后)。
- 2) 它使 T_EX 载入 <fontname> 的字体信息文件 (.tfm 文件)。

字体的名字往往表示它的设计大小，比如 cmr10 表示设计大写为 10 点的计算机现代字体。字体的设计大小保存在字体信息文件中。

如果用户既没有指定 `scaled <number>` 也没有指定 `at <dimen>`, 那么这个字体载入时, 使用设计大小---设计大小的含意是, 字体在这个大小时, 表示最佳。否则将会载入一个缩放版的字体:

- 如果指定了 `scaled <number>`, 则字体将被放大 $<number>/1000$ 倍。
- 如果指定了 `at <dimen>`, 则字体通过缩放大 $<dimen>/ds$ 倍, 变为 $<dimen>$ 大小, 其中, ds 是 `<fontname>` 的设计大小。`<dimen>` 和 ds 的单位往往使用点。

放大率可以小于 1, 这样做就会使字体缩小尺寸。

你往往需要为载入的字体的每一个使用的放大率提供一个字体轮廓文件 (第 65 页). 当然, 一些设备驱动可以使用打印机内置的字体。这些字体不需要字体轮廓文件。

更多信息请参见“字体”(第 64 页)和“放大率”(第 77 页).

例子:

```
\font\tenttt = cmtt10
\font\bigttfont = cmtt10 scaled \magstep2
\font\eleventtfont = cmtt10 at 11pt
First we use {\tenttt regular CM typewriter}.
Then we use {\eleventtfont eleven-point CM typewriter}.
Finally we use {\bigttfont big CM typewriter}.
```

结果:

First we use regular CM typewriter. Then we use eleven-point CM typewriter. Finally we use big CM typewriter.

`\fontdimen <number> [<dimen> parameter]`

这些参数可以用来定义控制序列 `` (注意, `` 和 `<fontname>` 不同, `<font-name>` 是用来表示字体文件的文件名的) 所定义的各种尺寸大小。这些参数的值, 都在 `` 所使用的字体信息文件中定义了, 但是你可以在执行 TeX 时得到或改变这些值。这些参数的数值及其意义为:

数值	意义
1	字体每点的倾斜程度
2	词间距
3	词间伸长长度
4	词间收缩长度
5	x 字符的高度 (<code>1ex</code> 的长度)
6	m 字符的宽度 (<code>1em</code> 的长度)
7	额外空白长度

\TeX 需要使用字体每点的倾斜程度来计算重音符号的位置。使用词间距来控制单词间的距离（见 $\backslash\text{spaceskip}$, 第 107 页). 使用额外空白长度来控制句号以后所留的空白（见 $\backslash\text{spaceskip}$, 第 107 页). 在 *The \TeXbook* 433 页中列出了 plain \TeX 中字体所使用的这些参数的数值。数学符号有另外 15 个参数，不过我们在此不展开讨论。

注意：这些参数被设定后，在当前组结束时，并不返回初始值。如果你只是希望在局部的文本中改变这些值，你需要将它的原值保存，以便稍后恢复。

例子：

```
Here's a line printed normally.\par
\fontdimen2\font = 3\fontdimen2\font
% Triple the interword spacing.
\noindent Here's a really spaced-out line.
```

结果：

```
Here's a line printed normally.
Here's a really spaced-out line.
```

```
\magnification = <number>
\mag [ <number> parameter ]
```

使用 `\magnification` 可以定义“缩放因子” f ，缩放因子决定了你的文档的放大率（见“放大率”，第 77 页）。`\magnification` 必须在文档的第一页被编译出前定义。

这个命令把 f 定义成 $\langle number \rangle$ 的同时，也会定义 `\hsize` 为 6.5true in, `\vsize` 为 8.9true in, 即适合 $8\frac{1}{2}$ 英尺乘 11 英尺的页面的大小。缩放因子 f 的值必需介于 0 到 32768 之间，它确定了该文档的放大率为 $f/1000$ 。因此缩放因子为 1000 就给出了单位放大率，也就是说，它不改变文档的尺寸。我们习惯使用 1.2 的幂次值来定义缩放因子，因为很多的字体就是基于这种因子的。用 `\magstep` 和 `\magstephalf` 命令你可以更方便地定义这种因子。

`\magnification` 不是一个参数，你不能使用它来得到缩放因子。如果你使用类似 `\dimen0 = \magnification` 之类的语句，TeX 将会报错。

而 `\mag` 参数则包含缩放因子。改变 `\mag` 的值也缩放了页面尺寸，而这一般不是你所想要的。因此，一般情况下，最好是使用 `\magnification` 而不是 `\mag` 来改变页面的放大率。¹

例子：

```
\magnification = \magstep2
% 把字体放大 1.44 倍 (=1.2x1.2)
```

`\magstep <number>`

这个命令展开为一个放大率因子，这个因子可以把你文档中所有（除了给定 `true` 尺寸的）东西放大 1.2^r 倍，其中 r 是 $\langle number \rangle$ 的值。 $\langle number \rangle$ 必须在 0 到 5 之间。²

例子：

```
\magnification = \magstep1 % 放大 1.2 倍。
```

¹ 译注：`\mag` 是 TeX 的原始命令，而 `\magnification` 是 plain TeX 中定义的命令。`\magnification` 依赖于 `\mag`，它用 `\mag` 设置放大率后恢复了默认的页面尺寸。在 LaTeX 中没有 `\magnification` 命令，一般也不建议使用 `\mag` 命令，因为这与 LaTeX 的设计原则相悖。

² 译注：即在 plain TeX 中定义 `\magstep1` 为 1200, `\magstep2` 为 1440, 等等。

\magstephalf

这个命令展开为一个放大率因子，这个因子可以把你文档中所有（除了给定 `true` 尺寸的）东西放大 $\sqrt{1.2}$ 倍，也就是 1 和 1.2 的等比中项。³

例子：

```
\magnification = \magstephalf
```

把信息转为记号

■ 数值

\number <number>

这个命令可以把一个数表示成字符记号序列。这里的数可以是一个确定的整数，或是一个 `<number>` 参数，也可以是一个 `<number>` 寄存器。

例子：

```
\number 24 \quad \count13 = -10000 \number\count13
```

结果：

```
24 -10000
```

☆ \romannumeral <number>

这个命令可以把一个数表示成罗马字符记号列。这里的数可以是一个确定的整数，或是一个 `<number>` 参数，也可以是一个 `<number>` 寄存器。如果这个数是负的，则 `\romannumeral` 不产生任何记号。

例子：

```
\romannumeral 24 \quad (\romannumeral -16)\quad
```

```
\count13 = 6000 \romannumeral\count13
```

结果：

```
xxiv () mmmmmmm
```

³ 译注：即在 plain TeX 中定义 `\magstephalf` 为 1095。

■ 环境信息

`\time [⟨number⟩ parameter]`

T_EX 把这个参数设置为从当日午夜到现在所经过的分钟数。比如在中午, `\time` 就是 720. 这个命令和下面的三个命令使用的是你计算机中所记录的时间。T_EX 只在开始运行的时候向系统获取一次时间, 所以如果你不改变这个值的话, 先前运行的 `\time` 和最后运行的 `\time` 的值是完全相同的。

`\day [⟨number⟩ parameter]`

T_EX 把这个参数设置为今天的日期数。这是一个介于 1 和 31 之间的数。`\day` 只在程序开始运行的时候被设定(见上面 `\time` 的说明)。

`\month [⟨number⟩ parameter]`

T_EX 把这个参数设置为当前的月份。这是一个介于 1 和 12 之间的数。`\time` 只在程序开始运行的时候被设定(见上面 `\time` 的说明)。

`\year [⟨number⟩ parameter]`

T_EX 把这个参数设置为当前的(公元)年份。这是一个类似 1991 之类的数。`\year` 只在程序开始运行的时候被设定(见上面 `\time` 的说明)。

`\fmtname`

`\fmtversion`

这个命令会产生当前使用的 T_EX 格式(比如 plain T_EX 或 L^AT_EX)的名字和版本号。

例子:

本书使用 `\fmtname` 格式, 版本 `~\fmtversion`。

结果:

本书使用 eplain 格式, 版本 1.9: 26 April 1991 (and plain 3.141592653). ■

\jobname

这个命令产生调用 TeX 的文件的文件本名。比如你的输入文件为 `hatter.tex`, `\jobname` 会被展开成 `hatter.` `\jobname` 在你生成文档的辅助文件时会很管用。

例子：

```
\newwrite\indexfile \openout\indexfile = \jobname.idx
% 打开 ‘hatter.tex’ 的索引文件 ‘hatter.idx’.
```

■ 变量的值

\meaning <token>

这个命令会产生 `<token>` 的定义。它对于诊断输出很有用。你可以用类似的方法使用 `the` 命令 (第 249 页) 来得到 寄存器和其它 TeX 中的东西的值信息。

例子：

```
[{\tt \meaning\eject}] [\meaning\tensrm] [\meaning Y]
```

结果：

```
[macro:->\par \break ] [select font cmr10] [the letter Y]
```

\string <control sequence>

这个命令会把 `<control sequence>` 表示成其名字的字符串，包括转义符。转义符会被表示成当前 `\escapechar` 的值。TeX 把在此中的所有字符的类码设为 12 (其它)。

你可以使用 `\csname` 命令执行这个命令的反操作 (第 248 页)。它会把一个字符串转为一个控制序列。

例子：

```
控制序列 {\tt \string\bigbreak}
```

结果：

```
控制序列 \bigbreak
```

\escapechar [<number> parameter]

这个参数在把一个控制序列名转化为一系列字符记号时，指定 TeX 用来表示转义符的字符的 ASCII 码。这个转化发生在你使用 `\string` 命令时或者 TeX 在产生诊断信息时。转义符的默认值是 92，即反斜

杠的 ASCII 码。如果 `\escapechar` 不在 0–255 之间, TeX 则在转换时不包括转义符。

例子:

```
\escapechar = ‘!
控制序列 {\tt \string\bigbreak}
```

结果:

```
控制序列!bigbreak
```

`\fontname `

这个命令产生 `` 的字体文件名。该字体名就是定义 `` 时所的 `<fontname>`。

例子:

```
\font\myfive=cmr5 [\fontname\myfive]
```

结果:

```
[cmr5]
```

编组

```
\begingroup
\endgroup
```

这两个命令开始或结束一个编组。`\begingroup` 不会和右花括号匹配, `\endgroup` 也不会和左花括号匹配。

在扫描输入时, TeX 如同其它控制序列一样对待 `\begingroup` 和 `\endgroup`。尤其体现在, 你可以定义一个包括 `\begingroup` 但不包括 `\endgroup` 的宏, 或者与之相反。这个技巧往往在你定义一对宏时会非常有用, 一个用来开始一个环境, 另一个用来结束这个环境。但是你不可以使用 `\begingroup` 和 `\endgroup` 来替换除了用来括起一个组以外功能的花括号。

例子：

```
\def\aa{一 \begingroup \it 二 }
\def\enda{\endgroup 四}
\aa 三 \enda
```

结果：

```
一 二 三 四
```

```
◊ { }                                \bgroup \egroup
```

左花括号和右花括号命令的作用是用来开始或结束一个编组。除了在扫描输入时 TeX 会把 `\bgroup` 和 `\egroup` 像其它控制序列一样对待以外，`\bgroup` 和 `\egroup` 控制序列与 ‘{’ 和 ‘}’ 是等价的。

当你定义两个成对的宏时，`\bgroup` 和 `\egroup` 会非常有用。这两个宏中其中一个开始一个由花括号定界的结构（不一定是一个组），而另一个结束该结构。你不能使用常规的花括号来定义这样的宏，否则你的宏定义会包含没有匹配的花括号，而这样的定义是不被 TeX 所接受的。一般情况下你需要在你没有办法使用花括号时使用这些命令。

例子：

一个编组的 `{\it` 边界 `\}/` 可由花括号定义。

结果：

一个编组的 边界 可由花括号定义。

例子：

```
\def\aa{一 \vbox\bgroup
% 在这里你不能使用一个 { 来代替 \bgroup,
% 否则 TeX 不会识别这个宏
\def\enda{\{\#1\egroup} 二
% 这有点技巧性，因为 \egroup 事实上和左花括号匹配，
% 而它右边的右边的花括号和 \bgroup 匹配。
% 不过这个歪门邪道行得通！
\aa \enda{\hrule width 1in}
```

结果：

```
一 _____ 二
```

\global

这个命令使得它随后的定义或赋值成为全局性的（见“全局的”，第 66 页），而不是仅局限在所处的组之中生效。你可以把 \global 放在包括宏定义和寄存器赋值在内的任何定义或赋值前。

例子：

```
{\global\let\la = \leftarrow}  
$a \la b$
```

结果：

$a \leftarrow b$

\globaldefs [⟨number⟩ parameter]

这个参数控制 TeX 是否把定义和其它的赋值作为全局的：

- 如 \globaldefs 是（默认值）零，当且仅当一个定义是由直接或者间接的 \global 命令指定的。（\gdef 和 \xdef 命令（第 246 页）间接地在定义前面加了 \global 命令）。
- 如果 \globaldefs 大于零，所有的定义和赋值都会间接地在前面加上 global。
- 如果 \globaldefs 小于零，所有的 \global 都被忽略。

\aftergroup ⟨token⟩

当 TeX 在读取输入时遇到了这个命令，它就把 ⟨token⟩ 保存下来。并在当前组后插入展开后的 ⟨token⟩。如果一个编组有好几个 \aftergroup，则每个记号全都会依次插入这个编组的最后。

下面的例子向你展示了如何用 \aftergroup 来延迟处理一个在条件测试中产生的记号。

例子：

```
\def\neg{负} \def\pos{正}
% 因为一个 \aftergroup 只能作用到一个记号上,
% 而不是到一系列的记号上, 甚至不能是一个用括号定界的文本,
% 所以下面的代码是必需的.
\def\arith#1{是否 $#1>0$? \begingroup
    \ifnum #1>-1 是 \aftergroup\pos
    \else 否 \aftergroup\neg\fi
    , 它是 \endgroup. }
\arith 2
\arith {-1}
```

结果：

是否 $2 > 0$? 是, 它是正. 是否 $-1 > 0$? 否, 它是负.

\afterassignment *<token>*

当 TeX 遇到这个命令, 它会把 *<token>* 保存在一个特殊的地方。在它下次执行赋值时, 它会把展开后的 *<token>* 放到其后。如果你调用了几次 \afterassignment, 那仅有最后一次是生效的。\\afterassignment 命令的一个用处是可以写定义形式的宏命令, 例子如下。

精确的 \afterassignment 行为定义请参见 *The TeXbook* page 279.

例子：

```
\def\setme{\afterassignment\setmeA\count255}
\def\setmeA{$\number\count255\advance\count255 by 10
            +10=\number\count255$}
算术式: \setme = 27
% 在展开 \setme 以后, TeX 把 \count255
% 设为 27, 然后调用 \setmeA.
```

结果：

算术式: $27 + 10 = 37$

宏

■ 定义宏

```
\def <control sequence> <parameter text> {<replacement text> }
```

这个命令根据 *<parameter text>* 和 *<replacement text>* 把 *<control sequence>* 定义为一个宏。请参阅第 75 页获取关于如何定义宏的完整信息。

例子：

```
\def\add#1+#2=?{#1+#2&=
 \count255=#1 \advance\count255 by #2 \number\count255\cr}■
 $$\eqalign{
 \add 27+9=?
 \add -5+-8=?}$$
```

结果：

$$\begin{aligned} 27 + 9 &= 36 \\ -5 + -8 &= -13 \end{aligned}$$

```
\edef <control sequence> <parameter text> {<replacement text> }
```

这个命令和 `\def` 一样可以定义宏。唯一的区别在于，`TeX` 会把 `\edef` 的 *<replacement text>* 立即展开（但是不会执行它）。因此，任何在 *<replacement text>* 中的内容会被展开，但那些产生盒子或粘连的排列或命令依然保持原来的样子。比如 `TeX` 在处理这个定义时，在由 `\edef` 的定义中的 *<replacement text>* 中的 `\hbox` 命令依然会保持命令竹的格式，而不会变成一个盒子。一般情况下，并不能明显看出什么会被展开，什么不会，但你可以在 *The TeXbook* 页 212-215 找到一张可被展开的控制序列列表。

你可以使用 `\noexpand` 命令（第 249 页）来阻止一个控制序列的展开。你可以使用 `\expandafter`（第 249 页）命令来延迟一个控制序列的展开。

\write, \message, \errmessage, \wlog 和 \csname 命令会把它们的记号列使用和 \edef 替换文本相同的法则进行展开。

例子：

```
\def\aa{xy} \count255 = 1
\edef\bb{\ifnum \count255 > 0\aa\fi z}
% 和 \def\bb{wxyz} 等价
\def\aa{} \count255 = 0 % 不影响 \bb
\bb
```

结果：

wxyz

\gdef <control sequence> <parameter text> {<replacement text>}

这个命令等价于 \global\def.

\xdef <control sequence> <parameter text> {<replacement text>}

这个命令等价于 \global\edef.

\long

这个命令加在宏定义前面。它告诉 TeX 该宏的参数可以包括 \par 记号 (第 110 页), 该记号一般表示一个段的结束。如果 TeX 尝试把一个没有 \long 的宏定义展开, 而该宏的任意一部分的参数包括了一个 \par 记号, TeX 就会报错。这是一个失控的参数。这个行为的目的, 是为了确保宏参数的完整结束。而 \long 就给你提供了一个回避这个行为的方法。

例子：

```
\long\def\aa#1{\par\hrule\smallskip#1\par\smallskip\hrule}
\aa{这是第一行。 \par
这是第二行。 }
% 没有 \long, TeX 会报错
```

结果：

这是第一行。

这是第二行。

\outer

这个命令加在宏定义前。它告诉 TeX 这个宏是外部的（第 82 页），所以不能在某些情况下使用。如果这个宏出现在那些被禁止的情况下，TeX 会报错。

例子：

```
\outer\def\chapterhead#1{%
  \eject\topglue 2in \centerline{\bf #1}\bigskip}
% 在一个被禁止的情况下使用 \chapterhead
% 会出现错误信息.
```

`\chardef <control sequence>=<charcode>`

上面的命令把 `<charcode>` 定义为 `<control sequence>`。虽然 `\chardef` 常在定义字符时用到，你也可以使用它来给 0–255 之间的任何一个数定义一个名字，即使你没用以字符码的形式使用该数。

例子：

```
\chardef\percent = '\% 21\percent, {\it 19\percent}
% 排印出正体和意大利体的百分号
```

结果：

21%, 19%

`\mathchardef <control sequence>=<mathcode>`

这个命令把 `<control sequence>` 定义为一个给定数学字符码的数学字符。这个控制序列仅在数学模式下有效

例子：

```
\mathchardef\alphachar = "010B % 类似 plain TeX 的定义
$\alphachar$
```

结果：

α

■ 其它定义方法

`\let <control sequence> = <token>`

这个命令会使得 `<control sequence>` 去获取当前 `<token>` 的含义。即使你对 `<token>` 重新定义，`<control sequence>` 的含义仍然不变。虽然 `<token>` 一般情况下都是一个控制序列，它其实也可以是一个字符记号。

```
\futurelet <control sequence> <token1> <token2>
```

这个命令告诉 TeX 把 <token₂> 的设为 <control sequence> 的定义 (可以由 \let 完成), 然后按正常的方式来处理 <token₁> 和 <token₂>. 在宏定义的最后 \futurelet 很有用, 因为它可以给你提供一个在 TeX 处理未处理的记号前, 查看它后面的记号的方法。

例子:

```
\def\predict#1{\toks0={#1}\futurelet\next\printer}
% \next 会获取 \predict 后的标点符号.
\def\printer#1{一个 \punc\ 放在 \the\toks0 前。}
\def\punc{%
  \ifx\next; 分号 \else
    \ifx\next, 逗号 \else
      ``\next''\fi\fi}
\predict{三月}; \predict{四月}, \predict{七月}/
```

结果:

一个 分号 放在 三月前。一个 逗号 放在 四月前。一个 “/” 放在 七月前。

```
\csname <token list> \endcsname
```

这个命令可以由 <token list> 产生一个控制序列。它提供了一种把记号列合并成控制序列的方式, 包括一些你一般情况下不能直接写的形式。<token list> 可以包含控制序列。 \csname 命令会把这个记号列展开, 使用和 \edef 替换文本相同的法则 (第 245 页)。如果最后的展开会产生不是字符的东西, TeX 会报错。 \csname 把一个记号列转为一个控制序列; 你可以用 \string (第 240 页) 的方法做相反的事情。

例子:

```
\def\capTe{Te}
本书关于 \csname\capTe X\endcsname.
```

结果：

本书关于 TeX.

■ 控制展开

`\expandafter <token1> <token2>`

这个命令会让 TeX 在展开一层 $\langle token_2 \rangle$ 后把 $\langle token_1 \rangle$ 根据宏展开法则进行展开。有时你想展开 $\langle token_2 \rangle$, 而它前面有类似 ‘\’ 或 `\string` 的东西阻止它的展开，这时这个命令会非常有用。

例子：

```
\def\aa{xyz}
\tt % Use this font so ‘\’ prints that way.
[\string\aa] [\expandafter\string\aa]
[\expandafter\string\csname TeX\endcsname]
```

结果：

`[\aa] [xyz] [\TeX]`

`\noexpand <token>`

这个命令让 TeX 跳过一个可以展开的记号 $\langle token \rangle$ 的展开。如果 $\langle token \rangle$ 不能被展开，比如它是一个字符，TeX 则不会理会 `\noexpand`, 而把 $\langle token \rangle$ 按一般的方法处理。

例子：

```
\def\bunny{兔子}
\edef\magic{把 \noexpand\bunny\ 从帽子中取出! }
% 如果没有 \noexpand, \bunny 会永远替换为‘兔子’
\let\oldbunny=\bunny \def\bunny{兔} \magic
\let\bunny=\oldbunny \magic
```

结果：

把 兔 从帽子中取出!! 把 兔子 从帽子中取出!!

`\the <token>`

这个命令一般会把 $\langle token \rangle$ 展开，表示成一个字符串。其中 $\langle token \rangle$ 可以是以下的任何形式：

- TeX 参数，比如，`\parindent` 或者 `\deadcycles`

- 寄存器, 比如, `\count0`
- 输入字符的字符码, 比如, `\catcode`(``
- 字符参数, 比如, `\fontdimen3\sevenbf`
- 字体的 `\hyphenchar` 或 `\skewchar`, 比如, `\skewchar\teni`
- `\lastpenalty`, `\lastskip`, 或者 `\lastkern` (当前的水平或竖直列表的最后一项的数值).
- `\chardef` 或 `\mathchardef` 定义的控制序列

此外, `\the` 可以在下面两种情况下, 展开成非字符记号:

- `\the `, 会展开成当前定义的控制序列, 这个控制序列和控制序列 `` 所选择的字体相同。
- `\the <token variable>`, 会复制变量的值, 并且对副本进行展开, 比如 `\the\everypar`

请参阅 *The TeXbook* 页 214–215 来获取 `\the` 在各种情况下行为的更详细描述。

例子:

目前页面的竖直长度为 `\the\vsiz`。

‘(’ 字符的类码是 `\the\catcode`(``。

结果:

目前页面的竖直长度为 548.4975pt。‘(’ 字符的类码是 12。

参阅: “把信息转为记号” (第 238 页), `\showthe` (第 270 页).

■ 条件测试

`\if <token1> <token2>`

此命令测试 `<token1>` 和 `<token2>` 的字符编码是否相同, 不区分它们的类别码。在测试之前, TeX 展开 `\if` 之后的记号, 直到得到两个无法继续展开的记号。这两个记号就是 `<token1>` 和 `<token2>`。展开过程也包括将控制序列替换为用 `\let` 设定的某个字符记号。TeX 将无法继续展开的控制序列视为字符编码 256 的记号。

例子：

```
\def\first{abc}
\if\first true\else false\fi;
% ``c'' is left over from the expansion of \first.
% It lands in the unexecuted ``true'' part.
\if a\first\ true\else false\fi;
% Here ``bc'' is left over from the expansion of \first
\if \hbox\relax true\else false\fi
% Unexpandable control sequences test equal with ``if''
```

结果：

```
false; bc true; true
```

\ifcat $\langle token_1 \rangle$ $\langle token_2 \rangle$

此命令测试 $\langle token_1 \rangle$ 和 $\langle token_2 \rangle$ 的类别码是否相同。在测试之前, TeX 展开 \ifcat 之后的记号, 直到得到两个无法继续展开的记号。这两个记号就是 $\langle token_1 \rangle$ 和 $\langle token_2 \rangle$ 。展开过程也包括将控制序列替换为用 \let 设定的某个字符记号。TeX 将无法继续展开的控制序列视为类别码 16 的记号。

例子：

```
\ifcat axtrue\else false\fi;
\ifcat ]}true\else false\fi;
\ifcat \hbox\day true\else false\fi;
\def\first{12345}
\ifcat (\first true\else false\fi
% ``2345'' lands in the true branch of the test
```

结果：

```
true; false; true; 2345true
```

\ifx $\langle token_1 \rangle$ $\langle token_2 \rangle$

此命令测试 $\langle token_1 \rangle$ 和 $\langle token_2 \rangle$ 是否相同。与 \if 和 \ifcat 不同, \ifx 命令不会展开 \ifx 之后的记号, 因此 $\langle token_1 \rangle$ 和 $\langle token_2 \rangle$ 就是紧随 \ifx 之后的两个记号。TeX 按照如下三种情形处理:

- 1) 如果一个记号是宏而另一个记号不是, 则这两个记号不相同。
- 2) 如果两个记号都不是宏, 它们满足下面条件之一时相同:
 - a) 两个记号都是字符 (或表示字符的控制序列), 且它们的字符编码和类别码相同;

- b) 两个记号指代相同的 TeX 命令或字体等。
- 3) 如果两个记号都是宏，它们满足下面两个条件时相同：
- 它们的“第一层”展开即它们的替换文本相同；
 - 它们有相同的 `\long` (第 246 页) 和 `\outer` (第 246 页) 状态。

注意特别地任何两个未定义的控制序列是相同的。

此测试通常比 `\if` 更为有用。

例子：

```
\ifx\alice\rabbit true\else false\fi;
% true since neither \rabbit nor \alice is defined
\def\aa{a}%
\ifx a\aa true\else false\fi;
% false since one token is a macro and the other isn't
\def\first{\aa}\def\second{\aa}\def\aa{a}%
\ifx \first\second true\else false\fi;
% false since top level expansions aren't the same
\def\third{\aa}\def\fourth{\aa}%
\ifx\third\fourth true\else false\fi
% false since parameter texts differ
```

结果：

true; false; false; false

`\ifnum <number1> <relation> <number2>`

此命令测试 `<number1>` 和 `<number2>` 是否满足 `<relation>`，即 ‘<’、‘=’ 或者 ‘>’ 关系。这两个数可以是类似 127 的常数、类似 `\pageno` 或 `\count22` 的计数寄存器或者类似 `\hbadness` 的数值参数。在执行测试之前，TeX 展开 `\ifnum` 之后的记号，直到它得到一串形如 `<number1> <relation> <number2>` 的记号序列，而且随后的记号不能成为 `<number2>` 的一部分。

例子：

```
\count255 = 19 \ifnum \count255 > 12 true\else false\fi
```

结果：

true

`\ifodd <number>`

此命令测试 $\langle number \rangle$ 是否为奇数。在执行测试之前, TeX 展开 `\ifodd` 之后的记号, 直到它得到一串形如 $\langle number \rangle$ 的记号序列, 而且随后的记号不能成为 $\langle number \rangle$ 的一部分。

例子:

```
\count255 = 19  
\ifodd 5 true\else false\fi
```

结果:

```
true
```

`\ifdim` $\langle dimen_1 \rangle$ $\langle relation \rangle$ $\langle dimen_2 \rangle$

此命令测试 $\langle dimen_1 \rangle$ 和 $\langle dimen_2 \rangle$ 是否满足 $\langle relation \rangle$, 即 ‘ $<$ ’、‘ $=$ ’ 或 ‘ $>$ ’ 关系。这两个尺寸可以是类似 `1in` 的常数、类似 `\dimen6` 的尺寸寄存器或者类似 `\parindent` 的尺寸参数。在执行测试之前, TeX 展开 `\ifdim` 之后的记号, 直到它得到一串形如 $\langle dimen_1 \rangle$ $\langle relation \rangle$ $\langle dimen_2 \rangle$ 的记号序列, 而且随后的记号不能成为 $\langle dimen_2 \rangle$ 的一部分。

例子:

```
\dimen0 = 1000pt \ifdim \dimen0 > 3in true\else false\fi
```

结果:

```
true
```

`\ifhmode`
`\ifvmode`
`\ifmmode`
`\ifinner`

这些命令测试 TeX 处于何种模式中:

- `\ifhmode` 为真, 如果 TeX 处于普通或受限水平模式中。
- `\ifvmode` 为真, 如果 TeX 处于普通或内部竖直模式中。
- `\ifmmode` 为真, 如果 TeX 处于文内数学或陈列数学模式中。
- `\ifinner` 为真, 如果 TeX 处于“内部”模式中: 即位于受限水平模式、内部竖直模式或者文内数学模式中。

例子：

```
\def\modes{{\bf
  \ifhmode
    \ifinner {\tt H}\else {\tt H}\fi
  \else\ifvmode
    \ifinner {\tt V}\else {\tt V}\fi
  \else\ifmmode {\tt M}\else
    error\fi\fi\fi}}
Formula \$\modes\$; then \modes,
\hbox{next \modes\ and \vbox{\modes}}.
\par\modes
```

结果：

Formula **M**; then **H**, next **IH** and **IV**.
V

```
\ifhbox {register}
\ifvbox {register}
\ifvoid {register}
```

这些命令测试编号为 *register* 的盒子寄存器的内容。设编号 *register* 为 *n*。则有：

- `\ifhbox` 为真，如果 `\boxn` 是一个水平盒子。
- `\ifvbox` 为真，如果 `\boxn` 是一个竖直盒子。
- `\ifvoid` 为真，如果 `\boxn` 是空的，即它不包含盒子。

例子：

```
\setbox0 = \vbox{} % empty but not void
\setbox1 = \hbox{a}
\setbox2 = \box1 % makes box1 void
\ifvbox0 true\else false\fi;
\ifhbox2 true\else false\fi;
\ifvoid1 true\else false\fi
```

结果：

true; true; true

```
\ifeof {number}
```

此命令测试输入流文件是否结束。它为真当且仅当第 *number* 个输入流还没打开，或者已经打开但对应的文件已经全部读完（或文件不存在）。

```
\ifcase <number>\<case0 text> \or <case1 text> \or ... \or <casen text>
    \else <otherwise text> \fi
```

此命令引入一个带编号的多重分支测试。假设 *<number>* 的值为 *k*, 在 *<case_k text>* 存在时 TeX 将展开它, 否则就展开 *<otherwise text>*。你可以省略 *\else* ——此时, 在各个分支都不满足时 TeX 将不展开任何东西。

例子:

```
\def\whichday#1{\ifcase #1<day 0>\or Sunday\or Monday%
    \or Tuesday\or Wednesday\or Thursday\or Friday%
    \or Saturday\else Nonday\fi
    \is day \##1. }
\whichday2 \whichday3 \whichday9
```

结果:

Monday is day #2. Tuesday is day #3. Nonday is day #9.

```
\iftrue
\iffalse
```

这两个命令提供始终为真或始终为假的测试。它们主要用于在宏定义中记录测试结果。

例子:

```
\def\isbigger{\let\bigger=\iftrue}
\def\isnotbigger{\let\bigger=\iffalse}
% These \let's MUST be buried in macros! If they aren't,
% TeX erroneously tries to match them with \fi.
\def\test#1#2{\ifnum #1>#2 \isbigger\else\isnotbigger\fi}
\test{3}{6}
\bigger$3>6$\else$3\leq6$\fi
```

结果:

$3 \leq 6$

```
\else
```

此命令提供条件测试为“假”时的另一选择。

```
\fi
```

此命令结束条件测试文本。

```
\newif \if<test name>
```

假设 *<test name>* 为 *alpha*, 此命令命名了三个控制序列 *\alphatrue*、*\alphafalse* 和 *\ifalpha*。通过创建记录真/假信息的逻辑变量, 你可以定义自己的测试:

- *\alphatrue* 设定逻辑变量 *alpha* 为真;
- *\alphafalse* 设定逻辑变量 *alpha* 为假;
- *\ifalpha* 是一个条件测试, 它为真当且仅当逻辑变量 *alpha* 为真。

该逻辑变量 *alpha* 实际上不存在, 但 TeX 假装它是存在的。在执行 *\newif\ifalpha* 后, 该逻辑变量的初始值为假。

\newif 是一个外部命令, 因此你不能在宏定义中使用它。

例子:

```
\newif\iflong \longtrue
\iflong Rabbits have long ears.
\else Rabbits don't have long ears.\fi
```

结果:

Rabbits have long ears.

■ 重复操作

```
\loop \if\Omega \beta \repeat
\repeat
```

这些命令提供了 TeX 的循环结构。这里 α 和 β 是任意的命令序列, 而 *\if\Omega* 是在“条件测试”(第 250 页)中描述的任何一种条件测试。*\repeat* 已经替代了对应于该测试的 *\fi*, 因此你不可以显式写出 *\fi* 以结束测试。这也遗憾地导致你不可以给测试带上 *\else* 部分。如果你想在相反意义上使用测试, 你需要重新安排该测试, 或者用 *\newif* (见上面) 定义一个辅助测试并按照你的需要使用这个测试(见下面第二个例子)。

TeX 按照下列步骤展开 *\loop* 循环:

- 1) 展开 α 。
- 2) 执行 *\if\Omega*。如果结果为假, 循环就此中止。
- 3) 展开 β 。
- 4) 重复此循环。

例子：

```
\count255 = 6
\loop
  \number\count255\
  \ifnum\count255 > 0
    \advance\count255 by -1
  \repeat
```

结果：

```
6 5 4 3 2 1 0
```

例子：

```
\newif\ifnotdone % \newif uses \count255 in its definition
\count255=6
\loop
  \number\count255\
  \ifnum\count255 < 1 \notdonefalse\else\notdonetrue\fi
  \ifnotdone
    \advance\count255 by -1
  \repeat
```

结果：

```
6 5 4 3 2 1 0
```

■ 什么也不做

\relax

此命令让 TeX 不做任何事情。此命令常用在需要提供一个命令但却无事可做的环境中。

例子：

```
\def\medspace{\hskip 12pt\relax}
% The \relax guards against the possibility that
% The next tokens are ‘plus’ or ‘minus’.
```

\empty

此命令展开后得不到任何记号。它与 \relax 的区别在于，在宏展开后它消失了。

寄存器

■ 使用寄存器

```
\count <register> = <number>      \count <register>
\dimen <register> = <dimen>        \dimen <register>
\skip <register> = <glue>          \skip <register>
\muskip <register> = <muglue>       \muskip <register>
\toks <register> = <token variable>\toks <register>
\toks <register> = {<token list> }
```

左边列出的六个命令用于给寄存器赋值，其中的‘=’是可选的。右边列出的五个控制序列并不是真正的命令，因为它们只能作为参量的一部分。它们给出了特定寄存器的内容。虽然在文本中你不能将这些控制序列单独用作命令，但你可以用 `\the` 命令将它们转换为文本再排版它们的值。

你可以用 `\newcount` 及相关命令命名和预留寄存器（第 260 页）。用这些命令获取寄存器更加安全，因为它们保证不会造成冲突。

`\count` 寄存器存储一个整数，可正可负。该整数可以很大，比你可能需要的都大。⁴ TeX 将计数寄存器 0–9 用于记录页码（见 *The TeXbook* 第 119 页）。`\count255` 是无需预留就能使用的唯一计数寄存器。

例子：

```
\count255 = 17 \number\count255
```

结果：

17

`\dimen` 寄存器存储一个尺寸。寄存器 `\dimen0` 到 `\dimen9` 及 `\dimen255` 都可用于临时用途。

例子：

```
\dimen0 = 2.5in
\hbox to \dimen0{\$Leftarrow\$hfil\$Rightarrow\$}
```

结果：

⇐ ⇒



⁴ 这里是本书仅有的一个练习：找出 TeX 可以接受的最大整数。

\skip 寄存器存储粘连的尺寸。与 \dimen 寄存器不同，它除了记录粘连的自然尺寸外，还记录其伸长量和收缩量。寄存器 \skip0 到 \skip9 及 \skip255 都可用于临时用途。

例子：

```
\skip2 = 2in
$ \Rightarrow \$\hskip \skip2 \$\Leftarrow $
```

结果：

⇒ ←


\muskip 寄存器与 \skip 寄存器相似，但其中的粘连始终以 mu 为单位（见“数学单位”，第 81 页）。一个 mu 的大小与当前字体有关，比如在下标中通常就比在普通文本中小一点。寄存器 \muskip0 到 \muskip9 及 \muskip255 都可用于临时用途。

例子：

```
\muskip0 = 24mu % An em and a half, no stretch or shrink.
\$ \mathop{a \mskip \muskip0 b} \limits^{\smash{a \mskip \muskip0 b}} \$%
% Note the difference in spacing.
```

结果：

$\begin{matrix} a & b \\ a & b \end{matrix}$

你可以指定一个记号变量（一个寄存器或一个参数）或一个记号列给 \toks 寄存器。当你指定一个记号列给记号寄存器时，记号列中的记号不会被展开。

一旦用 \the 命令将记号列中的记号插入文本中，TeX 就像直接读入那样展开它们。它们的类别码是 TeX 在输入中首次看到时给出的。

例子：

```
\toks0 = {the \oystereaters\ were at the seashore}
% This assignment doesn't expand \oystereaters.
\def\oystereaters{Walrus and Carpenter}
\toks1 = \toks0
% the same tokens are now in \toks0 and \toks1
Alice inquired as to whether \the\toks1.
```

结果：

Alice inquired as to whether the Walrus and Carpenter were at the seashore.

这个控制序列生成 TeX 所能接受的最大尺寸 $\langle dimen \rangle$ (差不多为 18 英尺)。它不是真正的命令，因为它只能作为其他命令的参数的一部分。

例子：

```
\maxdepth = \maxdimen % Remove restrictions on \maxdepth.
```

参阅：\advance (第 261 页), \multiply, \divide (第 262 页), \setbox, \box (第 169 页).

■ 命名和预留寄存器等

\newcount	\newread
\newdimen	\newwrite
\newskip	\newfam
\newmuskip	\newinsert
\newtoks	\newlanguage
\newbox	

这些命令预留并命名一个指定类型的实体：

- \newcount、\newdimen、\newskip、\newmuskip、\newtoks 以及 \newbox 各自预留一个指定类型的寄存器。
- \newread 和 \newwrite 分别预留一个输入流和输出流。
- \newfam 预留一个数学字体族。
- \newinsert 预留一个插入项类型。(预留一个插入项类型包括预留几个不同的寄存器。)
- \newlanguage 预留一套连字模式。

为避免编号冲突，在需要这些实体之一时你应当使用这些命令，除非在很局部的范围内。

这些命令之间还有一个很重要的区别：

- 用 \newcount、\newdimen、\newskip、\newmuskip 以及 \newtoks 定义的控制序列各自指明一个相应类型的实体。例如在执行命令

```
\newdimen\listdimen
```

之后，控制序列 \listdimen 可以用作一个尺寸。

- 用 \newbox、\newread、\newwrite、\newfam、\newinsert 以及 \newlanguage 定义的控制序列各自表示相应类型的实体的编号。例如在执行命令

```
\newbox\figbox
```

之后，控制序列 `\figbox` 必须和类似 `\box` 的命令一起使用，比如：

```
\setbox\figbox = \vbox{...}
```

```
\countdef <control sequence> = <register>
\dimendef <control sequence> = <register>
\skipdef <control sequence> = <register>
\muskipdef <control sequence> = <register>
\toksdef <control sequence> = <register>
```

这些命令将 `<control sequence>` 定义为指定类型的编号为 `<register>` 的寄存器。为避免编号冲突，通常你应当使用前面那些命令 (`\newcount` 等)，而不是使用这些命令。实际上，前面那些命令就是用这些命令来定义的。

例子：

```
\countdef\hatters = 19 % \hatters now refers to \count19
\toksdef\hares = 200 % \hares now refers to \toks200
```

参阅：`\newif` (第 256 页), `\newhelp` (第 279 页).

■ 寄存器中的算术运算

```
\advance <count register> by <number>
\advance <dimen register> by <dimen>
\advance <skip register> by <glue>
\advance <muskip register> by <muglue>
```

此命令给寄存器加上一个兼容的数量。对于 `<glue>` 或 `<muglue>`，三个部分（自然尺寸、伸长量和收缩量）都被加上。任何数量都可以是负的。为方便这些计算（以及其他赋值），`<glue>` 可以通过丢掉伸长量和收缩量转换为 `<dimen>`，而 `<dimen>` 可以通过取以 scaled point 为单位的值转换为 `<number>`（见“尺寸”，第 61 页）。在这些命令中，你可以忽略单词 `by`——这两种写法 TeX 都能理解。

例子：

```
\count0 = 18 \advance\count0 by -1 \number\count0\par
\skip0 = .5in \advance\skip0 by 0in plus 1in % add stretch
\hbox to 2in{a\hskip\skip0 b}
```

结果：

A horizontal ruler scale from 0 to 2 inches with major tick marks every 1/8 inch. The scale is labeled with '17' at the 0-inch mark, 'a' at the 1-inch mark, and 'b' at the 2-inch mark.

`\multiply <register> by <number>`
`\divide <register> by <number>`

这两个命令将 $\langle register \rangle$ 中的值乘以或除以 $\langle number \rangle$ (它可以为负数)。其中的寄存器可以是 $\backslash count$ 、 $\backslash dimen$ 、 $\backslash skip$ 或 $\backslash muskip$ 寄存器。对于 $\backslash skip$ 或 $\backslash muskip$ 寄存器 (第 258 页)，在寄存器中粘连的三个部分都被修改。在这些命令中，你可以忽略单词 `by` —— 这两种写法 TeX 都能理解。

在 $\langle dimen \rangle$ 前面加上 $\langle number \rangle$ 或小数，你可以得到它的倍数，比如 $-2.5\dimen2$ 。在 $\langle glue \rangle$ 前面也可以使用此写法，但小心注意——结果是一个 $\langle dimen \rangle$ 而不是 $\langle glue \rangle$ 。比如用 $2\baselineskip$ 将得到一个 $\langle dimen \rangle$ ，它等于 \baselineskip 的自然尺寸的两倍，但没有伸长量和收缩量。

例子：

```
\count0 = 9\multiply \count0 by 8 \number\count0 ;
\divide \count0 by 12 \number\count0 \par
\skip0 = 20pt plus 2pt minus 3pt \multiply \skip0 by 3
Multiplied value of skip0 is \the\skip0.\par
\dimen0 = .5in \multiply\dimen0 by 6
\hbox to \dimen0{a\hfil b}
```

結果：

72: 6

Multiplied value of skip0 is 60.0pt plus 6.0pt minus 9.0pt.

3

b

 3 in

结束任务

◊ \bye

此命令让 TeX 填满并生成最后的页面，打印遗留的插入项，并结束任务。这是结束输入文件的常用方式。

\end

此命令让 TeX 生成最后的页面并结束任务。但由于它不会填满页面，一般情形用 \bye 比用 \end 更好些。

输入和输出

■ 操作输入文件

◊ \input <filename>

此命令让 TeX 从文件 <filename> 读取输入。当该文件用完后，TeX 回到之前的输入来源继续读取。你可以嵌套输入文件到任何层级（在合理范围内）。

在排版一个大文档时，通常在主文件写上一系列 \input 命令，以引入文档的各个子文件。用这种方式，在打草稿阶段你可以很方便地处理文档的单个部分。还有个好习惯就是，将所有宏定义放在一个独立文件中，并在主文件一开始用 \input 命令引入它。

TeX 扫描文件名的规则一般和扫描记号的不同（见第 64 页）。如果 TeX 系统要求文件名包含扩展名（通常在前面加上点号表示），则 TeX 以 .tex 作为默认扩展名。

例子：

```
\input macros.tex  
\input chap1 % equivalent to chap1.tex
```

\endinput

此命令让 TeX 在这一行读完后停止从当前文件读取输入。

\inputlineno

此命令生成一个给出当前行编号的数（不是字符串），当此处有错误时，它就是写在错误信息中的行数。

\openin *number* = *filename*

此命令让 TeX 打开名为 *filename* 的文件，让我们可以通过编号为 *number* 的输入流读取它。*number* 必须在 0 和 15 之间。一旦打开了一个文件并将它连接到一个输入流，你就可以用 \read 命令以输入流编号读取其内容。

你可以把同一个文件和多个输入流连接。这样你就可以从文件的多个不同位置读取内容，每个输入流对应一个位置。

你应当用 \newread（第 260 页）命令分配 \openin 的流编号。

例子：

```
\newread\auxfile \openin\auxfile = addenda.aux
% \auxfile now denotes the number of this opening
% of addenda.aux.
```

\closein *number*

此命令让 TeX 关闭编号为 *number* 的输入流，结束输入流与文件之间的关联。此编号的输入流以后就可以用于其他文件。在文件使用完成后你应当关闭输入流。

例子：

```
\closein\auxfile
```

\read *number* to *control sequence*

此命令让 TeX 从编号 *number* 的输入流关联的文件中读取一行，并将该行的记号赋予 *control sequence*。该控制序列就成为一个不带参数的宏。在读取操作中 TeX 不会执行宏展开。如果该行包含未配对的左花括号，TeX 将读取额外的行直到花括号全部配对。如果直到文件结束还无法配对所有花括号，TeX 将报错。

如果 *number* 大于 15 或者还未用 \openin 关联文件，TeX 将在终端上显示提示符 ‘*control sequence* =’，等待你键入一行输入，然后将该

行输入赋予 $\langle control\ sequence \rangle$ 。如果 $\langle number \rangle$ 小于零，它同样从终端读取输入，但不显示提示符。

例子：

```
\read\auxfile to \holder  
% Expanding \holder will produce the line just read.
```

■ 操作输出文件

\openout $\langle number \rangle = \langle filename \rangle$

此命令让 TeX 打开名为 $\langle filename \rangle$ 的文件，让我们可以通过编号为 $\langle number \rangle$ 的输出流写入它。 $\langle number \rangle$ 必须在 0 和 15 之间。一旦打开了一个文件并将它连接到一个输出流，你就可以用 **\write** 命令以输出流编号读取其内容。

\openout 命令生成一个作为盒子一部分的小玩意。直到 TeX 将该盒子送出到 .dvi 文件时 **\openout** 才会起作用，除非你在 **\openout** 之前加上 **\immediate** 命令。

如果你把同一个文件和多个输出流关联，TeX 不会报错，但这样做将得到一些垃圾！

你应当用 **\newwrite** (第 260 页) 为 **\openout** 分配编号。

例子：

```
\newwrite\auxfile \openout\auxfile = addenda.aux  
% \auxfile now denotes the number of this opening  
% of addenda.aux.
```

\closeout $\langle number \rangle$

此命令让 TeX 关闭编号为 $\langle number \rangle$ 的输出流，结束输出流与文件之间的关联。此编号的输出流以后就可以用于其他文件。在文件使用完成后你应当关闭输出流。

\closeout 命令生成一个作为盒子一部分的小玩意。直到 TeX 将该盒子送出到 .dvi 文件时 **\closeout** 才会起作用，除非你在 **\closeout** 之前加上 **\immediate** 命令。

例子：

```
\closeout\auxfile
```

\write $\langle number \rangle \{ \langle token\ list \rangle \}$

此命令让 TeX 将 $\langle token\ list\rangle$ 写入编号为 $\langle number\rangle$ 的输出流的关联文件中。它生成一个作为盒子一部分的小玩意。直到 TeX 将该盒子送出到 .dvi 文件时才会实际写入文件，除非你在 \write 之前加上 \immediate 命令。

对于非立即 (immediate) 的 \write, TeX 直到实际写入文件时才展开 $\langle token\ list\rangle$ 中的宏。宏展开遵循与 \edef (第 245 页) 一样的规则。任何不是宏名称的控制序列被写成 \escapechar 加上控制序列名，后面再加一个空格。 $\langle token\ list\rangle$ 中的任何 ‘#’ 记号被重复两遍，即写成 ‘##’。

如果 $\langle number\rangle$ 不在 0 到 15 的范围内，TeX 将 $\langle token\ list\rangle$ 写入到日志文件。如果 $\langle number\rangle$ 大于 15 或该输出流未关联文件，TeX 还同时将 $\langle token\ list\rangle$ 写到终端中。

例子：

```
\def\aa{a a}
\write\auxfile{\hbox{$x#y$} \aa}
% Writes the string '\hbox {$x##y$} a a' to \auxfile.
```

\immediate

此命令应当放在 \openout、\closeout 或 \write 之前。它让 TeX 立即执行指定的文件操作。

例子：

```
\immediate\write 16{I'm stuck!}
% has the same effect as \message
```

\special { $\langle token\ list\rangle$ }

此命令让 TeX 下次送出页面时将 $\langle token\ list\rangle$ 直接写入 .dvi 文件中。\\special 的一个典型用法是，让设备驱动器将指定名称的图片文件整合到输出页面中。\\special 命令生成一个包含 $\langle token\ list\rangle$ 和一个特定位置的小玩意，该位置是将 \\special 命令换成零尺寸盒子时该盒子应该出现的位置。你可以怎样使用 \\special 命令，完全取决于你可用的设备驱动器。

例子：

```
\special{graphic expic}
% Display the graphics file 'expic' here.
```

`\newlinechar [⟨number⟩ parameter]`

此参数表示输出时所用的换行符。在读取 `\write`、`\message` 或 `\errmessage` 命令的参量时，如果 TeX 遇到这个字符，它就新起一行。如果 `\newlinechar` 不在 0–255 的范围内，输出中就不会任何换行符。Plain TeX 设定 `\newlinechar` 等于 -1。

例子：

```
\newlinechar = '\^J
\message{This message appears^J on two lines.}
```

结果（在日志中）：

```
This message appears
on two lines.
```

参阅：`\newread`, `\newwrite` (第 260 页).

■ 解释输入字符

`\catcode ⟨charcode⟩ [⟨number⟩ table entry]`

此表格项表示 ASCII 码为 `⟨charcode⟩` 的字符的类别码。第 55 页列出了所有的类别码。通过修改一个字符的类别码，你可以让 TeX 用不同方式处理该字符。

例子：

```
\catcode '[' = 1 \catcode ']' = 2
% Make [ and ] act like left and right braces.
```

`\active`

此命令表示活动字符的类别码，即数字 13。

例子：

```
\catcode '@' = \active % Make @ an active character.
```

`\mathcode ⟨charcode⟩ [⟨number⟩ table entry]`

此表格项表示 ASCII 码为 `⟨charcode⟩` 的字符的数学码（见“数学码”，第 80 页）。数学码指明在数学模式中如何解释该字符。

例子：

```
\mathcode\> = "313E % as in plain TeX
% The > character has class 3 (relation), family 1 (math
% italic), and character code "3E
```

\delcode *<charcode>* [*<number>* table entry]

此表格项表示 ASCII 码为 *<charcode>* 的输入字符的定界码。定界码指引 TeX 找到最佳输出字符以将该输入字符排版为定界符。

通常以十六进制表示 *<number>*。假设 *<number>* 等于十六进制数 $s_1s_2s_3l_1l_2l_3$ 。则当该字符用作定界符时，该字符的小型变体为 $s_1s_2s_3$ ，大型变体为 $l_1l_2l_3$ 。这里 $s_1s_2s_3$ 表示在第 s_1 族的 s_2s_3 位置的数学字符， $l_1l_2l_3$ 类似。这与 \mathcode (第 267 页) 中的用法一样，只是 \mathcode 还另外指定了字符类。

例子：

```
\delcode ' ( = "028300 % As in plain TeX.
```

\endlinechar [*<number>* parameter]

此参数表示 TeX 在每个输入行后添加的行尾符。取不在 0–255 范围内的值则表示不添加行尾符。Plain TeX 设定 \endlinechar 为 ‘\^M’ (即 *<return>* 符的 ASCII 码)。

\ignorespaces

此命令让 TeX 读取和展开记号，直到找到一个非空格记号，并忽略它找到的任何空格记号。**\ignorespaces** 常用在宏结尾处，使得调用宏时不受后面的空格符或换行符的影响。(但是 \ignorespaces 之后的空行依然生成 \par 记号)。

例子：

```
\def\aa#1{yes #1\ignorespaces}
\aa{may}
be
```

结果：

yes maybe

控制与 *T_EX* 的交互

\errorstopmode

此命令让 *T_EX* 遇到错误时暂停以和用户交互。这是 *T_EX* 运行的正常模式。

\scrollmode

此命令让 *T_EX* 遇到大多数错误时不暂停，而仅在终端上显示错误信息。在回应错误信息时键入 ‘s’ 或 ‘S’ 就进入卷动模式。

\nonstopmode

此命令让 *T_EX* 遇到错误时不暂停，即使那些无法找到文件的错误，而仅仅在终端上显示错误信息。在回应错误信息时键入 ‘R’ 或 ‘r’ 就进入直达模式。

\batchmode

此命令让 *T_EX* 遇到错误时不暂停，且取消在终端上显示此后的输出。在回应错误信息时键入 ‘Q’ 或 ‘q’ 就进入批处理模式。

\pausing [*number*] parameter

如果此参数大于零，*T_EX* 将在每个输入行后暂停，让你有机会将该行替换为另外一行。如果你键入替换行，*T_EX* 就将原来的行替换为此行；如果你键入 *return* 符，*T_EX* 将使用原来的行。

将 \pausing 设定为 1，是在 *T_EX* 处理文档时修补它的常用方法。例如，你可以用这种方式插入 \show 命令（见下面）。

帮助调试

■ 显示内部数据

```
\show <token>
\showthe <argument>
\showbox <number>
\showlists
```

这些命令在 TeX 运行时记录信息到日志文件中：

- `\show` 记录 `<token>` 的含义。
- `\showthe` 记录用 `\the <argument>` 生成的所有记号（见第 249 页）。
- `\showbox` 记录编号为 `<number>` 的盒子寄存器的内容。日志文件中的前导点号的个数表示内部盒子的嵌套层级。
- `\showlists` 记录 TeX 正在构造的各个列表的内容。（这些列表将会互相嵌套。）见 *The TeXbook* 第 88–89 页中对 `\showlists` 的输出的进一步介绍。

对于 `\show` 和 `\showthe`, TeX 还在终端中显示这些信息。对于 `\showbox` 和 `\showlists`, TeX 仅在 `\tracingonline` (第 273 页) 大于零时才在终端显示信息；如果 `\tracingonline` 小于或等于零（默认情形），这些信息就不在终端显示。

在遇到类似 `\show` 的命令时，TeX 暂停以和用户交互。请求交互并不表示有错误发生，但这确实给你机会让 TeX 显示其他东西。如果你不想看到其他东西，键入 `<return>` 符即可。

要控制 `\showbox` 生成的输出的数量，你可以设定 `\showboxbreadth` 和 `\showboxdepth` (第 278 页)。这两个参数的默认值分别为 5 和 3，因而在下面的日志输出中每个盒子只显示五个项目。（‘`..etc.`’ 表示盒子内的其他未显示的项目。）

例子：

```
\show a
\show \hbox
\show \medskip
\show &
```

结果 (在日志中) :

```
> the letter a.  
> \hbox=\hbox.  
> \medskip=macro:  
->\vskip \medskipamount .  
> alignment tab character &.
```

例子:

```
\showthe\medskipamount  
\toks27={\hbox{Joe's\quad\ Diner}}  
\showthe\toks27
```

结果 (在日志中) :

```
> 6.0pt plus 2.0pt minus 2.0pt.  
> \hbox {Joe's\quad \ Diner}.
```

例子:

```
\setbox 3=\vbox{\hbox{A red dog.}\hrule A black cat.}  
\showbox 3
```

结果 (在日志中) :

```
> \box3=  
\vbox(16.23332+0.0)x53.05565  
. \hbox(6.94444+1.94444)x46.41675  
. \tenrm A  
. \glue 3.33333 plus 1.66498 minus 1.11221  
. \tenrm r  
. \tenrm e  
. \tenrm d  
. etc.  
. \rule(0.4+0.0)x*  
. \hbox(6.94444+0.0)x53.05565  
. \tenrm A  
. \glue 3.33333 plus 1.66498 minus 1.11221  
. \tenrm b  
. \tenrm l  
. \tenrm a  
. etc.
```

例子：

```
\vbox{A \hbox
  {formula
    $x \over y\showlists$}}
```

结果（在日志中）：

```
### math mode entered at line 3
\mathord
.\fam1 y
this will be denominator of:
\fraction, thickness = default
\\mathord
.\fam1 x
### restricted horizontal mode entered at line 2
\tenrm f
\tenrm o
\tenrm r
\tenrm m
\kern-0.27779
\tenrm u
\tenrm l
\tenrm a
\glue 3.33333 plus 1.66666 minus 1.11111
spacefactor 1000
### horizontal mode entered at line 1
\hbox(0.0+0.0)x20.0
\tenrm A
\glue 3.33333 plus 1.66498 minus 1.11221
spacefactor 999
### internal vertical mode entered at line 1
prevdepth ignored
### vertical mode entered at line 0
prevdepth ignored
```

参阅：`\showboxbreadth`、`\showboxdepth`（第 278 页）。

■ 指定追踪的内容

`\tracingonline [⟨number⟩ parameter]`

如果此参数大于零，除了记录到日志文件外，`TEX` 还在终端上显示追踪的结果（包括`\showbox`和`\showlists`）。

`\tracingcommands [⟨number⟩ parameter]`

如果此参数大于或等于 1，`TEX` 将在日志文件中记录它执行的大多数命令。如果`\tracingonline` 大于或等于零，这些信息还将出现在终端上。排版一个单词的首个字符算作一个命令，但（仅对追踪的目的而言）排版之后的字符和标点却不算作命令。如果`\tracingcommands` 大于或等于 2，`TEX` 还将记录展开但不执行的命令，比如条件测试及它们的输出。

例子：

```
\tracingcommands = 1 If $x+y>0$ we quit.\par
On the other hand, \tracingcommands = 0
```

结果（在日志中）：

```
{vertical mode: the letter I}
{horizontal mode: the letter I}
{blank space }
{math shift character $}
{math mode: the letter x}
{the character +}
{the letter y}
{the character >}
{the character 0}
{math shift character $}
{horizontal mode: blank space }
{the letter w}
{blank space }
{the letter q}
{blank space }
{\par}
{vertical mode: the letter 0}
{horizontal mode: the letter 0}
{blank space }
{the letter t}
{blank space }
{the letter o}
{blank space }
{the letter h}
{blank space }
{\tracingcommands}
```

\tracinglostchars [⟨number⟩ parameter]

如果此参数大于零，每次因某字符不在当前字体中而丢弃它时，
 TeX 将在日志文件中记录一个声明。如果 **\tracingonline** 大于零，此
 信息还将显示在终端上。Plain TeX 设定该参数的默认值为 1（与其他参数不一样）。

例子：

```
\tracinglostchars = 1
A {\nullfont few} characters.
```

结果（在日志中）：

```
Missing character: There is no f in font nullfont!
Missing character: There is no e in font nullfont!
Missing character: There is no w in font nullfont!
```

\tracingmacros [⟨number⟩ parameter]

如果此参数大于或等于 1, TeX 将在日志文件中记录它执行的每个宏的展开和参量。如果 \tracingmacros 大于或等于 2, TeX 还将记录每个记号列的展开，比如 \output 或 \everycr。如果 \tracingonline 大于零，这些信息还将显示在终端上。

例子：

```
\def\aa{first \b, then \c}
\def\b{b} \def\c{c}
\tracingmacros = 2
Call \aa once.
```

结果（在日志中）：

```
\aa ->first \b , then \c
```

```
\b ->b
```

```
\c ->c
```

\tracingoutput [⟨number⟩ parameter]

如果此参数大于零，TeX 将在日志文件中记录它送出到 .dvi 文件的每个盒子的内容。如果 \tracingonline 大于零，这些信息还将显示在终端上。追踪输出中每行开头的点号个数表示该行盒子的嵌套层级。你可以用 \showboxbreadth 和 \showboxdepth (第 278 页) 控制追踪的数量。

若你想确定页面为何出现多余空格，设定 \tracingoutput 为 1 就特别有用。

例子：

```
% This is the entire file.  
\tracingoutput = 1 \nopagenumbers  
One-line page. \bye
```

结果（在日志中）：

```
Completed box being shipped out [1]  
\vbox(667.20255+0.0)x469.75499  
. \vbox(0.0+0.0)x469.75499, glue set 13.99998fil  
. \glue -22.5  
. \hbox(8.5+0.0)x469.75499, glue set 469.75499fil  
. \vbox(8.5+0.0)x0.0  
. \glue 0.0 plus 1.0fil  
. \glue 0.0 plus 1.0fil minus 1.0fil  
. \vbox(643.20255+0.0)x469.75499, glue set 631.2581fill  
. \glue(\topskip) 3.05556  
. \hbox(6.94444+1.94444)x469.75499, glue set 386.9771fil  
. \hbox(0.0+0.0)x20.0  
. \tenrm 0  
. \tenrm n  
. \tenrm e  
. \tenrm -  
. etc.  
. \glue 0.0 plus 1.0fil  
. \glue 0.0 plus 1.0fill  
. \glue(\baselineskip) 24.0  
. \hbox(0.0+0.0)x469.75499, glue set 469.75499fil  
. \glue 0.0 plus 1.0fil
```

\tracingpages [⟨number⟩ parameter]

如果此参数大于零，*T_EX* 将在日志文件中记录它尝试各种分页所计算的代价。如果 *\tracingonline* 大于零，这些信息还将显示在终端上。在放置一个盒子或插入项到当前的页面列表，或者在处理页面的某个可能分页点时，*T_EX* 都在日志输出中写上一行。检查此输出将有助于你确定糟糕分页的起因。见 *The T_EXbook* 第 112–114 页中此类输出的示例和解释。

某些 *T_EX* 实现形式忽略 *\tracingpages* 的取值以更快地运行。如果你需要此参数，确保你用的 *T_EX* 形式能用此参数。

`\tracingparagraphs [⟨number⟩ parameter]`

如果此参数大于零, TeX 将在日志文件中记录它尝试各种断行所计算的代价。如果 `\tracingonline` 大于零, 这些信息还将显示在终端上。TeX 在段落的结尾处生成这些输出。见 *The TeXbook* 第 98–99 页中此类输出的示例和解释。

某些 TeX 实现形式忽略 `\tracingparagraphs` 的取值以更快地运行。如果你需要此参数, 确保你用的 TeX 形式能用此参数。

`\tracingrestores [⟨number⟩ parameter]`

如果此参数大于零, TeX 将在日志文件中记录那些在编组结尾处被还原的值。如果 `\tracingonline` 大于零, 这些信息还将显示在终端上。

某些 TeX 实现形式忽略 `\tracingrestores` 的取值以更快地运行。如果你需要此参数, 确保你用的 TeX 形式能用此参数。

`\tracingstats [⟨number⟩ parameter]`

如果此参数大于或等于 1, TeX 将报告它运行任务时所用的各种资源 (见 *The TeXbook* 第 300 页中这些资源的列表和解释)。另外, 如果 `\tracingstats` 大于或等于 2, TeX 将在 `\shipout` (第 151 页) 页面时报告所用的内存占用。这些报告写在日志文件的末尾。如果 `\tracingonline` 大于零, 这些信息还将显示在终端上。如果出现 TeX 超出其中某个容量的问题, `\tracingstats` 提供的这些信息将帮助你准确定位问题的起因。

某些 TeX 实现形式忽略 `\tracingstats` 的取值以更快地运行。如果你需要此参数, 确保你用的 TeX 形式能用此参数。

下面的例子显示了在某个 TeX 实现中实际的追踪输出。在其他 TeX 实现中也许会有所不同。

例子：

```
\tracingstats=1
```

结果（在日志中）：

```
Here is how much of TeX's memory you used:
4 strings out of 5540
60 string characters out of 72328
5956 words of memory out of 262141
921 multiletter control sequences out of 9500
14794 words of font info for 50 fonts, out of 72000 for 255
14 hyphenation exceptions out of 607
7i,4n,1p,68b,22s stack positions out of 300i,40n,60p,3000b,4000s
```

`\tracingall`

此命令让 TeX 打开所有可用的追踪形式，并设定 `\tracingonline` 为 1 以让追踪输出显示在终端上。

`\showboxbreadth [⟨number⟩ parameter]`

此参数指定 TeX 为 `\showbox` 或 `\tracingoutput` 生成输出时，对各级盒子的列表项显示的最大数目。Plain TeX 设定 `\showboxbreadth` 等于 5。

`\showboxdepth [⟨number⟩ parameter]`

此参数指定 TeX 为 `\showbox` 或 `\tracingoutput` 生成输出时，所显示的列表的最深层级。Plain TeX 设定 `\showboxdepth` 等于 3。

■ 传递信息

`\message {⟨token list⟩}`

`\errmessage {⟨token list⟩}`

这两个命令在终端上显示由 `⟨token list⟩` 给出的信息，并将它们记录到日志中。信息中的任何宏都将被展开，但是命令不会被执行。这与 TeX 用于 `\edef` (第 245 页) 的规则一样。

对于 `\errmessage`, TeX 如同显示自己的错误信息一样暂停运行，并在请求帮助时显示 `\errhelp` 记号列。

要生成多行的信息，你可以用 `\newlinechar` 字符（第 267 页）。

例子：

```
\message{Starting a new section.}
```

```
\wlog {\iotaoken list}
```

此命令在日志文件写入 $\langle token list \rangle$ 。TeX 按照与 `\edef`（第 245 页）一样的规则展开 $\langle token list \rangle$ 。

例子：

```
\wlog{Take two aspirins and call me in the morning.}
```

结果（在日志中）：

```
Take two aspirins and call me in the morning.
```

```
\errhelp [ \langle token list \rangle parameter ]
```

此参数包含 TeX 用户在 `\errmessage` 命令中请求帮助时显示的记号列表。我们建议在用 `\errmessage` 生成错误信息时，将 `\errhelp` 设定为描述错误形式的字符串，并用 `\newhelp` 生成该字符串。你可以用 `\newlinechar` 字符生成多行的信息。

```
\newhelp \langle control sequence \rangle { \langle help text \rangle }
```

此命令将 $\langle help text \rangle$ 给出的错误信息赋予 $\langle control sequence \rangle$ 。它提供一种定义帮助文本的快捷方式，帮助文本进一步解释了错误信息。在用 `\errmessage` 命令报出错误之前，你应当将 $\langle control sequence \rangle$ 赋予 `\errhelp`。这样在显示错误信息时，如果用户键入 ‘H’ 或 ‘h’，TeX 将显示出帮助文本。

例子：

```
\newhelp\pain{Your input includes a token that I find^^J
    to be offensive. Don't bother me again with this^^J
    document until you've removed it.}
\errhelp = \pain \newlinechar = '\^^J
% ^^J will start a new line
\errmessage{I do not appreciate receiving this token}
```

结果（在日志中）：

```
! I do not appreciate receiving this token.
1.8 ...t appreciate receiving this token.}
```

```
? H
\Your input includes a token that I find
    to be offensive. Don't bother me again with this
    document until you've removed it.
```

\errorcontextlines [⟨number⟩ parameter]

此命令确定 TeX 遇到错误时打印的成对信息行的数目，不包括最上面和最下面这两行。将它设为 0 就可以去掉冗长的错误信息。你仍然可以强制显示完整的信息，只要在回应错误时这样输入：

```
I\errorcontextlines=100\oops
```

这是因为未定义的控制序列将引起另一个错误。Plain TeX 设定 \errorcontextlines 等于 5。

参阅：\write (第 265 页)、\escapechar (第 240 页)。

初始化 TeX

\dump

这个命令不能在编组中使用，它可以把 TeX 记录的内容导出成一个格式文件（第 65 页）。如果使用的是 `virtex` 这种特殊的 TeX 形式，⁵ 你能够以很快的速度载入这个格式文件，载入后的 TeX 状态和你当初导

⁵ 译注：`virtex` 和 `initex` 在现代的 TeX 系统中不再被使用。在远古时代，人们必须使用 `initex` 命令来导出格式文件，用 `virtex` 命令载入格式文件，而 `tex` 程序不能导出或载入格式文件。而在现代的发行版中，`tex` 命令可以处理这一切。

出格式文件时的状态并无二致。`\dump` 命令会同时结束本次运行。由于 `\dump` 只能在 `initex` 中运行，而不是生产版本的 TeX，所以它仅仅在你安装 TeX 时会用到。

`\everyjob` [*<token list>* parameter]

此参数包含一个记号列，TeX 在开始每个任务时展开它们。由于对 `\everyjob` 的赋值无法影响当前任务（在你赋值时已经太迟了），它仅对制作格式文件的人有用。



10 | 建议和技巧

TeX 是一个复杂的程序，偶尔地会以你看不懂的方式变得不听话。这一章，我们提供一些解决问题的小贴士并阐释一些有用的技巧。

纠正不良分页

有时，TeX 在你想要放在一起的素材中间分页——例如，章节标题和随后的文本，或者一个互有联系的项目列表。有两种校正方法：

- 强制素材被放在一起。
- 在别的地方要求分页。

要强制 TeX 把素材放在同一页上，最简单的办法是使用 `\vbox` 命令（第 166 页）来使素材被封装在一个竖直盒子里。这里使用竖直盒子比水平盒子要好，因为大多数情况下，需要放在一起的是竖直模式素材，比如连续的段落。竖直盒子的前后，你都要放一个隐含或明显的分段命令（或者空行，或者 `\par`）；否则的话，TeX 会试图使竖直盒子成为相邻段落的一部分。竖直盒子的方法有一个重要的局限：不能用于不足一个段落的小部分文本。

有时候，要把一个自然段的各行放置在一起，你可以把它封装在一个编组中，并在编组开始的地方（或者段落结束前的其它位置）令 `\interlinepenalty`（第 141 页）的值为 10000。这样就告诉了 TeX，“绝对不能”在段内分页。但是，如果 TeX 发现所有要分页的地方都是“绝对不能”的，它可能不管三七二十一，就在自然段里分页。

在段落之后使用 `\nobreak` 命令（第 139 页）会阻止 TeX 在下一项内容之前分页（除非下一项恰好是一个小于 10000 的惩罚项）。这也是防止在章节标题之后分页的最佳方案，因为章节标题通常类似于一个自然段落。`\nobreak` 需要放在段落结束的空行或者 `\par` 之后，从而避免 TeX 将 `\nobreak` 作为段落的一部分处理。`\nobreak` 还需要出现在段落末尾所有适合分页的断点之前才有效。TeX 插入在下一个段落之前的粘连就是这样的一个断点，其他的还有你明确插入在段落后面的任何竖直粘连。因此，`\nobreak` 通常出现在段落或者标题结束之后的第一个位置。

你可以使用 `\eject` 命令（第 140 页），强制 TeX 在特定的地方分页。在段落内部，你可以使用一个组合：`'\vadjust{\vfill\eject}'`（第 120 页）在下一行之后强制断开。`\eject` 之前 `\vfill`（第 161 页）的作用是让 TeX 使用空白来填充页面。但是，使用 `\eject` 来纠正分页问题的主要缺点在于：如果文档的页边距改变，分页的地方可能未必就如你所愿那样。

如果你没有在 `\eject` 之前使用 `\vfill` 命令，TeX 将尽其所能地重新分配额外的空白间距，然后发牢骚：“an underfull `\vbox` (badness 10000) has occurred while `\output` is active.” 上面讲的封装素材在一起的任一方法中，类似的问题也会出现。

`\filbreak` 命令（第 140 页）提供了一种方式来把一个自然段或几个段落（或者其他竖直模式素材）放在同一页面上。当你用 `\filbreak` 来封闭一个自然段的时候，如果段落内容可以放置在一页，TeX 会有效地忽略那些 `\filbreak`；如果不是，TeX 将在第一个 `\filbreak` 之前分页。如果在连续段落中，每个自然段前后都有 `\filbreak`，比如这样：

```
\filbreak
<paragraph>
\filbreak
<paragraph>
\filbreak
\filbreak
⋮
<paragraph>
\filbreak
```

TEX 将使每一段都是同页。如果在 `\filbreak` 处分页，TEX 将用空白填充页面底部。

有时，你可以改变一个或几个段落的 `\looseness` 参数（第 124 页）以修改页面的长度。设置 `\looseness` 为负会让TEX 试图缩减段落的行数；为正则增加其行数。改变 `\looseness` 的缺点在于：受影响区域的单词间距不是最佳的。要了解更多的断行信息，可以设置 `\tracingpages`（第 276 页）为 1。

保留页尾

有时你需要在单个页面上修改某些地方但是又不想重排全部文档。如果页面长度没有因为修改而改变很多，那就可以实现。你要做的是固定页面的尾部不变，所需的办法跟纠正不良分页那里一样。

如果原来的页尾就是正好是两个段落中间，你可以用前面描述的方法在同样位置强制分页。其他情况下，你必须要在指定位置同时强制断行和分页。如果新页面比旧页面要短，命令序列：

```
\vadjust{\vfill\eject}\break
```

是一个捷径。但是，如果新的页面要长一些的话，问题就非常难办，因为TEX 可能已经尽量使页面非常紧凑了。这时唯一的办法就是设定 `\looseness`（第 124 页）为负值，来缩短页面上的一些竖直间距，如果 `\parskip`（第 144 页）非零，压缩一下，或者祭出压箱底的绝招：略微减小 `\baselineskip`（第 136 页）。

保留页首空白

通常你可以使用 `\vskip` 命令（第 159 页）在页面上添加空白竖直间距，但这一招对页首无效，因为在紧接着分页后的所有粘连，紧排和惩罚，都会被TEX 会丢弃。一个替代：`\topglue` 命令（第 160 页），能产生永不消失的粘连。

纠正不良断行

如果 TeX 在需要放在一行的内容中间断行了，有好几种方法能纠正这种情况：

- 你可以使用 `\break` 命令（第 121 页），在旁边强制断行。
- 你可以在两个单词之间插入一个带子（~）（见第 105 页），来防止在它们之间断开。
- 你可以在一些单词里插入一个或者多个自定连字符，来告诉 TeX 新的断词规则（参考 `\-`，第 127 页）。
- 你可以使用 `\hbox` 命令（第 164 页），把几个单词封装在一个水平盒子里。

除了插入临时连字符以外，其他几种方法都有这样的缺点：TeX 不能产生最佳的断行设置。一旦如此，TeX 将产生一个或多个未满或溢出盒子，并且对此抱怨。水平盒子的方法还有另外一个缺点：因为 TeX 处理一个水平盒子的方式是不考虑其中的内容而置其为一个基本单位，结果盒子内部的单词间距可能会与一行内其它地方不一致。

纠正溢出或未满盒子

如果 TeX 抱怨一个溢出盒子，那表示你在一个盒子里放置了超出盒子空间的内容。类似的，如果 TeX 抱怨的是一个未满盒子，那是说你没有往盒子里加入足够多的内容。这些抱怨可以在许多情形下发生，所以，让我们先看看一般的情况：

- 一个水平溢出盒子，如果是段落中的一行，表示那一行太长而且 TeX 不能重新安排段落来缩短它。如果你设置 `\emergencystretch`（第 124 页）的值非零，TeX 将在单词间插入更多的空白，可能有助于解决这个问题。另一个办法是设置 `\tolerance`（第 123 页）10000，但它往往会导致行内太多的空白。还有一个办法是对在 TeX 里未定义断词规则的关键单词插入自定连字符。如果这些都不管用，那你可能只好去重写那个段落了。一个不是很好的方法是增加 `\hfuzz`（第 176 页），来允许 TeX 排版超出右边界的一行。

- 一个水平未满盒子，如果是段落里的一行，表示那一行太短了， \TeX 不能重新安排段落来使之变长。 \TeX 将加大单词间的空白，甚至超过正常限度。上面提到的对付溢出行的两种纠正办法：插入临时的连字符和重写段落，都可以用于未满的行。对于使用 `\raggedright` 命令（第 116 页）得到的左对齐格式，就没有未满行的问题。
- 抱怨信息：

```
Underfull \vbox (badness 10000) has occurred
while \output is active
```

表明 \TeX 因为没有足够的素材而没有填满一页。可能的原因是：你使用了竖直盒子来把素材放在一起，但是在接近页面的底部时， \TeX 遇到了一个不适合放在那里的竖直盒子。结果那个盒子被移至下一页，而在当前页上留下了太多空白。在这种情况下，或者你要在别的地方插入更多的空白，或者将那个竖直盒子打破，变得小一点。

出现这一抱怨信息的另一个可能原因是：一个长段落覆盖了整页而没有分段。通常 \TeX 不会自动调整基行距离，因此，在一页的末尾，就没法填满不足一个行间距的空白。如果页面长度 `\vsize`（第 143 页）恰好不是基线距离 `\baselineskip`（第 136 页）的整数倍，这种情况就会出现。

关于这一抱怨信息还有一个原因：与上面相仿，段落间的粘连 `\parskip`（第 144 页），被固定为不能拉伸或挤压的值。后面的这两种情况，你可以增加 `\vfuzz`（第 176 页）予以解决。

- 抱怨信息：

```
Overfull \vbox (296.30745pt too high) has occurred
while \output is active
```

说的是你构造了一个长于一页的竖直盒子。弄短一点吧。

- 对于 `\hbox` 或 `\vbox` 命令（第 164, 166 页）所产生的溢出盒子问题，矫正的诀窍是：从盒子里拿一些东西出去；用 `\hss` 或 `\vss`（第 162 页）插入负值的粘连；或者增加盒子的尺寸。
- 对于 `\hbox` 或 `\vbox` 命令（第 164, 166 页）所产生的未满盒子问题，只需用 `\hfil` 或 `\vfil`（第 161 页）来填满即可。

恢复丢失的单词间距

如果你发现 TeX 把两个单词连在一起，可能的原因是有一个控制序列吃掉了其后所有的空格。在那个控制序列后面放一个控制空格 (`\u`) 即可。

避免多余的单词间空白

如果在文档中有不想要或者未预料到的空白，以我们的经验，最可能的原因是一行的结尾或大括号后的空格。（当你用类别码来做华丽的效果时，还会有其他许多类似的原因。）TeX 通常将行尾处理为空格，并且会排出左右大括号后面的空格。

如果多余的空白是在一行内，由大括号后的空格引起的，那就移除它。如果多余的空白是在行尾，由大括号后的空格引起的，那就紧接着在大括号后放一个 ‘%’。‘%’ 用于注释开始，但不一定真的要有注释内容。

宏定义如果写的不仔细，也可能导致多余的空白。当你应用一个宏，但是有多余的空白，那就要检查其定义，确保大括号后没有额外的空格以及紧接着括号之后，宏定义没有换行。人们经常在大括号后对宏定义换行，从而容易阅读。安全起见，应该在所有大括号之后要换行的地方放一个 ‘%’。可能有些地方不需要，但是这么做起码没有坏处。¹

当你不能准确在源文档中定位多余的空间的时候，可以尝试设置 `\tracingcommands` (第 273 页) 为 2。在日志文件里，每一处空格都会输出一个 {blank space} 命令。

知道 TeX 关于空格的规则会有帮助。

- 1) 输入行开始之前的空格将被忽略。
- 2) 输入行结束之后的空格在任何情况下都被忽略。虽然行尾本身就被当作空格。（一个空白行产生的是 `\par` 标记。）
- 3) 多于一个的空格，如果在一起，将被处理为一个空格。因此，宏调用的参量之后的空格将不会和宏定义末尾的空格合并。你得到的是两个空格。
- 4) 控制字符之后的空格将被忽略。
- 5) 数字，尺寸以及粘连设定中的 ‘plus’ 和 ‘minus’ 之后的空格在执行中也被忽略。²

¹当然，极为偶然的情况下，你可能真的需要在大括号后直接换行。

²实际上，TeX 在这些地方只会忽略单个的空格。因为多个空格会被处理为一个空格，所以结果就是忽略所有空格。

如果你改变了空格符或行尾符的类别码，这些规则就不算数了。

避免陈列公式周围的额外空白

如果在陈列公式之前有太多空白，有可能是因为你输入的时候在陈列公式之前留了一个空行。这个空行将开始一个新的段落，并且把 TeX 转入竖直模式。当 TeX 在竖直模式里遇到一个'\$'，就会切换到水平模式并插入段间粘连（\parskip），接下来又插入一个行间粘连（\baselineskip）。然后当显示开始的时候，它插入更多粘连（或者是 \abovedisplayskip 或者是 \abovedisplayshortskip，取决于前一行的长度）。只有最后一个粘连是你想要的。为了避免也得到段间粘连，在陈列公式之前不要留空行或者结束段落（比如用 \par）。

类似的，如果在陈列公式之后有太多的空白，可能是因为在输入时留了一个空行。去掉它即可。

避免段后的额外空白

如果使用宏得到的段落之后有太多竖直空白，可能里头包含了宏产生的段间粘连，一个空段，以及另外一个段间粘连。你可以通过在那个宏之后插入

```
\vskip -\parskip
\vskip -\baselineskip
```

来消除第二个段落间距。

改变段落形状

有几个 TeX 的参数——\hangindent, \leftskip, 等等——影响 TeX 塑段和折行的方式。Plain TeX 的命令，如 \narrower 和 \hang，会间接的用到这些参数；你也可以直接对它们赋值。如果你已经使用了这

些命令中的一个（或者改变了这些参数中的一个），但是看上去对段落没有任何效果，问题可能在于，段落结束之前你先结束了一个组。例如：

```
{\narrower She very soon came to an open field, with  
a wood on the other side of it: it looked much darker  
than the last wood, and Alice felt a little timid  
about going into it.}
```

这个段落不会变窄，因为最后那个右边大括号在 TeX 拆段组行之前就结束了 `\narrower` 的组。作为补救，放一个 `\par` 在右边大括号之前；这样你才能看到想要的效果。

把段落放入盒子

假设你有一些文字段落想放到页面的一个特定位置。显而易见的方式是用一个合适尺寸的水平盒子来封装段落，然后放到想要的位置。哎呀，这个办法不管用，因为 TeX 不能在受限水平模式下断行。这么做，你将得到一个有些误导的错误消息，声称 you're missing the end of a group。摆脱这个限制的方法是使用：

```
\vbox{\hsize = <dimen> ... <paragraphs> ...}
```

其中，`<dimen>` 是你希望的一行的长度。这么做就可以了，特别是当你想用盒子（带线框的盒子，不是 TeX 盒子）封装一些段落的时候。

画线

你可以使用 `\hrule` 和 `\vrule` 命令（第 178 页）来画线。所需知道的是 (a) 在什么地方可以用哪个命令，以及 (b) TeX 如何在你未明确指出的情况下决定线的长度。

- TeX 处于竖直模式时，只能用 `\hrule`；TeX 处于水平模式时，只能用 `\vrule`。所以你不能在水平盒子里使用横线或者在竖直盒子里使用竖线。然而，通过设定所有三个尺寸 (dimension)，你可以把横线画的又高又细，使它看起来像竖线。类似的，你也可以把竖线画的又矮又宽，使它看起来像横线。

- 如果你没有明确设定宽度，竖直盒子里的横线就会和盒子一样宽。水平盒子里的竖线也类似。如果你得到的线太长或太短，请检查外面盒子的尺寸。

一个例子，假设你想要

Help! Let
me out of
here!

The following input will do it:

```
\hbox{\vrule
  \vbox{\hrule \vskip 3pt
    \hbox{\hskip 3pt
      \vbox{\hsize = .7in \raggedright
        \noindent Help! Let me out of here!}%
      \hskip 3pt}%
    \vskip 3pt \hrule}%
\vrule}
```

我们需要把文本放到竖直盒子里来让 TeX 把它当作段落处理。四层盒子是必要的---如果你有所怀疑，可以尝试在更少的盒子情况下运行这个例子。

创建多行的页眉页脚

你可以使用 `\headline` 和 `\footline` 命令（第 146 页）来创建页眉和页脚，但是处理多于一行的页眉和页脚，就不大灵光。然而，通过重新定义 TeX 的输出例行程序的一些宏，就可以实现。

为了得到多行页眉，你需要做三件事：

- 1) 重新定义 `\makeheadline` 宏，TeX 的输出例行程序将会用到它。
- 2) 把额外的行所需的竖直距离加给 `\voffset`。
- 3) 对 `\vsize` 减少同样的大小。

下面的例子演示了你可以怎样做：

```
\advance\voffset by 2\baselineskip
\advance\vsize by -2\baselineskip
```

```
\def\makeheadline{\vbox to 0pt{\vss\noindent
  Header line 1\hfil Page \folio\break
  Header line 2\hfil\break
  Header line 3\hfil}%
  \vskip\baselineskip}
```

一般你照着这个定义的样子做就可以了，只需要替换你自己的页眉内容并选一个合适的倍数给 `\baselineskip`（页眉行数减一）。

对多行页脚的处理也类似：

- 1) 重新定义 `\makefootline` 宏，`TEX` 的输出例行程序将会用到它。
- 2) 把额外的行所需的竖直距离加给 `\voffset`。
- 3) 从 `\vsize` 减去额外的行所需的竖直距离。

下面的例子演示了你可以怎样做：

```
\advance\vsize by -2\baselineskip
\def\makefootline{%
  \lineskip = 24pt
  \vbox{\raggedright\noindent
  Footer line 1\hfil\break
  Footer line 2\hfil\break
  Footer line 3\hfil}}
```

同样，一般你照着这个定义的样子做就可以了。`\lineskip` 的值决定了页面上正文最后一行与页脚第一行的基行距离。

找出匹配错误的花括号

大多数情况下，如果有花括号匹配错误，`TEX` 会在你的输入文档里非常接近错误发生的地方给出诊断信息。但是，`TEX` 停止之前最令人沮丧的错误之一是

```
(\end occurred inside a group at level 1)
```

这表明在你的文档中有一个额外的左花括号或者缺少一个右花括号，但是并没有提示可能出现问题的地方。那么，怎么找到它呢？

一条有用的调试技巧是，在文档内部等距离地插入五到六处类似下面的内容（不要放在已知的编组里）：

```
}% a fake ending
```

假设问题是多了一个左花括号。如果它在第三和第四个伪装结尾 (fake ending) 之间，你将从前三个伪装结尾的地方得到出错消息，第四个则不会。原因是 TeX 将抱怨并忽略前三个伪装结尾，但是第四个正好匹配了那个额外的左花括号。这样你就知道了那个额外的左花括号在第三和第四个伪装结尾之间某个地方。如果嫌疑区域仍然太大，就去除已有的伪装结尾，并在嫌疑区域内重复上述步骤。如果问题是缺少一个右花括号，一旦你发现左边那个，就能确定右边的了。

这个方法不是必杀技。特别是，如果文档中有几个很大的组，这个方法就不那么灵了。然而大多数情况下，你可以适当地变化一下这个方法来找出暗地捣乱的花括号。

如果其他方法都不管用了，那就试着删掉文档后半部分（注意备份！）或者在中间插入一个 `\bye` 命令。如果问题仍然存在，那你就可知它在前一半文档中；如果没有问题了，你也可以可知问题在后一般文档里。重复这个过程，最终你将会发现错误所在。

设置尺寸

设置尺寸的最简单办法是直接赋值，比如：

```
\hsize = 6in
```

你也可以用其他尺寸来设定或者混用多种长度单位，这要多费一点事。有两种方法来创建这样的尺寸组合：

- 1) 你可以增加一个尺寸到另一个尺寸参数或寄存器。例如：

```
\hsize = 6in \advance\hsize by 3pc % 6in + 3pc
```

- 2) 你可以将一个尺寸取为另一个尺寸（或粘连）参数或寄存器的倍数。这时候，TeX 将丢掉粘连的伸长量和收缩量把它转换为尺寸。例如：

```
\parindent = .15\hsize  
\advance\vsiz by -2\parskip
```

创建混合字体

有时候新建一种“混合字体”会很有用处。混合字体命名为一个控制序列 \mathcal{F} ，它的字符基于一种字体 f_1 ，但是有一些来自另一种字体 f_2 。然后你可以像调用其他字体一样使用 \mathcal{F} 来设定文本使用混合字体。

要创建混合字体，你可以定义 \mathcal{F} 为一个宏。宏定义中，首先选择字体 f_1 ，然后定义控制序列来产生从 f_2 借来的字符。举个例子，假如你要创建一种混合字体 $\backslash\text{britrm}$ ：基于 cmr10 但是其中的美元符号借用 cmti10 的英镑符号。英镑符号在 cmti10 的位置恰好跟美元符号在 cmr10 里的位置一样。你可以这样做：

```
\def\text{britrm}{%
    \tenrm % \tenrm names the cmr10 font
    \def\${\{\text{tenit}\char`\$}\} % \tenit names the cmti10 font.■
}
```

现在，当你使用 $\text{\text{britrm}}$ 字体的时候， $\$$ 将生成英镑符号。

另一种方法会有同样的效果：首先改动类别码（category codes）来激活所考虑的字符，然后定义它们。例如：

```
\catcode`*=active
\def*{\tentt\char`\*}
```

这个例子里，星号将从 \tentt 字体借用。如果接下来输入：

```
Debbie was the * of the show.
```

效果将会是

```
Debbie was the * of the show.
```

原文呈现

原文是“输入什么样，就是什么样”的文本。比较常见的是排版计算机输入，包括计算机程序和 TeX 输入。计算机输入不易以原文呈现的原因是：

- 1) 有些字符（控制字符，转义符，花括号等）在 TeX 里有特殊的含义。
- 2) 行尾和多个空格会被转换为一个空格。

为了呈现原文，你必须取消这些特殊含义和转换。最好的办法是使用宏。

要取消特殊含义，你可以改变那些字符的类别码。接下来的宏演示你如何做：

```
\chardef \other = 12
\def\deactivate{%
  \catcode`\\" = \other  \catcode`\' = \other
  \catcode`\} = \other  \catcode`\$ = \other
  \catcode`\& = \other  \catcode`\# = \other
  \catcode`\% = \other  \catcode`\~ = \other
  \catcode`\^ = \other  \catcode`\_ = \other
}
```

小心！一旦这样改变了类别码，就不能再使用任何控制序列了，因为现在不再有转义符了。你需要想法回到正常的操作模式。接下来，介绍完另一个问题：取消多个空格和行尾的转换以后，我们将予以解释。

Plain T_EX 有两个命令：`\obeyspaces`（第 107 页）和 `\obeylines`（第 122 页），几乎已经解决了上述问题，除了两件事：保留行首的空格以及保留多个空白行。对此，你需要更强大的方法——使用我们将要定义的 `\obeywhitespace` 宏。

T_EX 总是收集多行文本为段落。为了原文呈现各行，一个办法是把每一行转换为独立的段落。³ 你可以重新定义行尾字符来产生一个控制序列 `\par`。下面三个宏定义演示了如何做：

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}
{%
  The group delimits the text over which ^^M is active.
  \makeactive\^^M %
  \gdef\obeywhitespace{%
    % Use \gdef so the definition survives the group.
    \makeactive\^^M %
    \let\^^M = \newline %
    \aftergroup\removebox % Kill extra paragraph at end.
    \obeyspaces %
  }%
}
\def\newline{\par\indent}
\def\removebox{\setbox0=\lastbox}
```

一个要点是：`^^M` 必须被激活，无论是定义 `\obeywhitespace` 还是使用它的时候。

³ 另一个办法是把行尾字符变为 `\break` 命令，并且在每一行尾施加无穷大的粘连。

为了在原文文本之后回到正常的操作模式，你需要选定一个原文文本中几乎不用的字符来作为临时的转义符。竖线 (|) 经常是一个好的选择。为此，需要宏定义：

```
\def\verbatim{\par\begingroup\deactivate\obeywhitespace  
  \catcode '\| = 0 % Make | the new escape character.  
}  
  
\def\endverbatim{\endgroup\endpar}  
  
\def\|{|}
```

在原文文本中，你可以用双竖线 (||) 表示单竖线，并用 `\endverbatim` 来结束原文部分。

这个技巧可以有很多变化：

- 如果一种编程语言有关键字，你可以把关键字变为一个命令，来产生粗体的关键字。这时候每一个关键字前面都要跟一个临时转义符。
- 如果有一个字符（假设是竖线）在原文文本中从来没有用到，你可以激活它并且用它结束原文部分。宏定义将类似于：

```
{\catcode '\| = \active  
 \gdef\verbatim{  
   \par\begingroup\deactivate\obeywhitespace  
   \catcode '| = \active  
   \def |{\endgroup\par}%  
 }
```

我们在这里只是提供了简单的办法来排版计算机程序。在反映程序的语法和语义环境方面，原文呈现经常不是一个清楚、易读的印刷习惯。如果你想在这个主题上走的更远，我们推荐阅读：

Baecker, Ronald M., and Marcus, Aaron, *Human Factors and Typography for More Readable Programs.* Reading, Mass.: Addison-Wesley, 1990.

使用外部宏

如果 TeX 警告 “*forbidden control sequence*”，很有可能你在非外部环境中使用了外部宏（见“*外部的*”，第 82 页）。一个外部宏的定义总是以 `\outer` 开始。外部宏不能被用于宏参量 (argument)、宏定义、阵列的导言或者条件文本：只有当某个条件检验有特定的结果是才展开的文本。有些宏被以外部的方式定义是因为我们不想在这些环境里使用它们，一旦使用，必定出错。解决的办法就是重新定义宏或者移动它们到合适的环境里。

在不恰当的环境使用外部宏，还会引起 TeX 警告：不受控制 (runaway) 的情形或者不完备的条件。因为错误消息提供不了线索，这些问题通常难于诊断。如果你得到了这样的错误消息，找找看哪里用了外部宏。可能你并不知道那些宏属于外部的，‘`\show\alpha`’（第 270 页）将显示 `\alpha` 的定义，并告之 `\alpha` 是否外部宏。

改变类别码

有时候需要在文档的部分内容里局部的改变类别码。例如，排版一个计算机程序 或者将激活的字符用于专门目的。你需要反激活那些字符来使 TeX 可以像其他字符一样使用它们。

如果局部地改变了类别码，你可能会悲哀地发现 TeX 没有理会你的努力。通常，原因有两个：

- 1) TeX 决定输入字符的类别码，读到字符的时候就贴上相应的类别码。假设读到波浪符 (~)，但是在 TeX 的胃还没有实际消化那个波浪符之前改变了它的类别码（见“TeX 剖析”，第 49 页）。TeX 将会继续认定原来贴上的类别码。这样的问题通常是因为波浪符是一个宏定义的一部分，而那个宏里改变了波浪符的类别码。
- 2) 当 TeX 匹配一个宏引用和它的定义时，匹配的不仅仅是参数的字符而且包括他们的类别码。如果定义里字符的类别码与宏引用里的不同，TeX 不会认为字符匹配正确。这样的效果匪夷所思，因为看起来是匹配的。例如，如果你已经定义了宏：

```
\def\eurodate#1/#2/#3{#2.#1.#3}
```

然后斜线字符在引用和定义 \eurodate 的时候必须有相同的类别码。

如果问题是因为嫌疑字符作为宏参量 (argument)，诊治方法是重新定义宏为一对宏 \mstart 和 \mfinish，分别在参量文本的前后应用。 \mstart 设定类别码而 \mfinish 结束这一编组来取消变动。

使宏文档易读

为了使得宏文档易读，你可以在文档开头设定空格的类别码为 9 (忽略字符)，\endlinechar (第 268 页) 为 -1。然后，你就可以自由地使用空格和断行而不必担心产生多余的空白。忽略字符不会产生空白，但仍然决定控制序列定义。如果你真的需要一个空格，可以用 \space 命令 (第 104 页)。

当然在文档最后，你需要恢复空格和行尾的类别码 (分别是 10 和 5)：可以把整个文档封装在一个编组里，也可以明确地恢复其正常值。

如果使用编组封装，你还需要设定 `\globaldefs` 为 1 以使得整个宏定义是全局的，在编组之外可见。

下面是这种格式宏文档的一个小例子：

```
\catcode '\ = 9 \endlinechar = -1

\def \makeblankbox #1 #2 {
    \hbox{\lower \dp0 \vbox{\hiderule {#1} {#2}
    \kern -#1 % overlap rules
    \hbox to \wd0{\hiderule {#1} {#2}%
        \raise \ht0 \vbox to #1{} % vrule height
        \lower \dp0 \vtop to #1{} % vrule depth
        \hfil \hiderule {#2} {#1} }
    \kern -#1 \hiderule {#2} {#1} } }

\def\hiderule #1 #2 {
    \kern -#1 \hrule height#1 depth#2 \kern -#2 }
\def\hiderule #1 #2 {
    \kern -#1 {\dimen0 = #1 \advance \dimen0 by #2
    \vrule width \dimen0 } \kern -#2 }

\catcode '\ = 10 \endlinechar = '\^M
```

不改变类别码的话，这些宏需要写的更加紧凑，使用很少的空格和行尾更多的 ‘%’。



11

理解错误信息

解释 TeX 的错误信息有时候就像，你跟医生抱怨你感觉疲乏无力，医生却递给你一份血生化细目表。对你的痛苦的解释也许就在那里，但却不容易弄清楚它是什么。掌握一些简单规则对你理解 TeX 错误信息并从中受益将大有帮助。

你的第一个目标应当是理解什么东西导致 TeX 报错。第二个目标（如果是交互式使用 TeX）应当是在一次运行中捕获尽可能多的错误。¹

第一个例子

我们来看一个例子。假设你的输入中包含下面这行：

```
We skip \quid a little bit.
```

你本想输入 ‘\quad’，却输入成 ‘\quid’。此时 TeX 将给出下面的信息：

```
! Undefined control sequence.  
1.291 We skip \quid  
a little bit.  
?
```

此信息将同时出现在终端上和日志文件中。第一行始终以感叹号 (!) 开头，告诉你出现了什么问题。在 ‘?’ 提示符之前的最后两行（在此例子中同样也是接下来的两行）告诉你 TeX 在何处发现此错误。它在当前输入

¹ 译注：本章的各个节标题为译者所加。

文件的第 291 行发现了错误，两个信息行之间的断行表示 TeX 在第 291 行的精确位置，即在 \quid 后面。当前输入文件是指，在运行时的终端输出上，最接近的非闭合左圆括号后显示的文件（见第 10 页）。

这种错误，即控制序列未定义，是你最常出现的错误之一。如果你在提示符后键入另一个‘?’，TeX 将显示下列信息：

```
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 the next 1 to 9 tokens of input,  
H for help, X to quit.
```

这些选择各自的含义如下：

- 如果你键入〈return〉，TeX 将继续处理你的文档。在此例子中它就忽略了 \quid。
- 如果你键入‘S’（或者‘s’——这时候大小写是等价的），TeX 将继续处理你的文档且不再暂停，除非找不到某文件。但错误信息还是会出现在终端上和日志文件中。
- 如果你键入‘R’或‘r’，你将得到与‘S’相同的结果，但此时 TeX 即使找不到文件也不会暂停。
- 如果你键入‘Q’或‘q’，TeX 将继续处理你的文档，且遇到错误后将不会再暂停，也不会在终端上显示。错误信息还是会记录到日志文件中。
- 如果你键入‘X’或‘x’，TeX 将尽力清理干净，丢弃正在构造的页面，并退出。你仍然可以打印或查看 TeX 已经生成的页面。
- 如果你键入‘E’或‘e’，TeX 将如同‘X’或‘x’那样清理并中止，然后打开文本编辑器，跳转到错误所在的行。（并非所有系统都支持此选项。）
- 如果你键入‘H’或‘h’，你将在终端上看到对此错误的进一步解释，可能还有对此错误的一些建议。这些信息同样会记录到你的日志文件中。对于上述例子的未定义控制序列，你将得到下列信息：

```
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.
```

- 如果你键入‘?’，你将得到和上面相同的信息。

另外两个选择，即键入 ‘I’ 或一个小整数，提供一种让 TeX 回到正常轨道上的方法，使得该错误不会在文档后面导致更多的错误：

- 如果你键入 ‘I’ 或 ‘i’ 并加上一些文本，TeX 将把文本插入错误位置之后，放在 TeX 处理的最内层级。在上面这个例子中，这表示在 TeX 的原始输入的位置，即在 ‘\quid’ 之后。稍后我们将给出一个例子，说明把某些东西插入到最内层级与插入到原始输入的区别。在上面这个控制序列未定义的例子中，如果你键入：

```
I\quad
```

TeX 将执行 \quad 命令并在你需要的地方生成一个全方间隔。

- 如果你键入一个小于 100（错误信息让人误解为小于 10）的正整数，TeX 将从它处理的最内层级删除该数目的记号。（如果你键入大于或等于 100 的整数，TeX 将只删除 10 个记号！）

第二个例子

这里是另一种常见错误的例子：

```
Skip across \hskip 3cn by 3 centimeters.
```

这个例子的错误信息为：

```
! Illegal unit of measure (pt inserted).
<to be read again>
          c
<to be read again>
          n
1.340 Skip across \hskip 3cn
                                by 3 centimeters.
```

这里 TeX 注意到 ‘3’ 后面不是一个正确的度量单位，因此它假设该度量单位为点。TeX 将重新读取 ‘cn’ 的记号并将它们插入你的输入中，而这并不是你想要的。在这种情形下若想得到更好的结果，你可以先键入 ‘2’ 以跳过 ‘cn’。这样你将得到如下信息：

```
<recently read> n

1.340 Skip across \hskip 3cn
                                by 3 centimeters.
```

现在你可以键入 ‘`I\hskip 3cm`’ 以得到你所要的间距（加上你已经得到的 3pt 间距）。²

第三个例子

如果你输入某些仅可用于数学模式的内容，TeX 将帮你切换到数学模式，不管这是否是你真正想要的。例如：

```
So \spadesuit s are trumps.
```

下面是 TeX 的错误信息：

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                \spadesuit
1.330 So \spadesuit
                s are trumps.
```

由于 `\spadesuit` 符号仅可在数学模式中使用，TeX 已经在它前面插入一个 ‘\$’。在插入某个记号之后，TeX 位于该记号的前面，在此情形中就是位于 ‘\$’ 前面，并准备读取输入。键入 ‘2’ 将让 TeX 跳过 ‘\$’ 和 ‘`\spadesuit`’ 记号，让它准备处理 ‘`s are trumps.`’ 中的 ‘`s`’。（若你只是让 TeX 继续，它将在数学模式中排版 ‘`s are trumps`’。）

第四个例子

在下面这个例子中，TeX 对错误的诊断是完全不正确的：

```
\hbox{One \vskip 1in two.}
```

它给出的错误信息如下：

```
! Missing } inserted.
<inserted text>
                }
```

² 若改为键入 ‘`I\unskip\hskip 3cm`’，你就可以去掉这个 3pt 的间距。

```
<to be read again>
    \vskip
1.29 \hbox{One \vskip
               1in two.}
```

问题在于，当 TeX 位于受限水平模式中，即正在构建水平盒子时，不能使用 \vskip。但 TeX 不是丢掉 \vskip，而是在它前面插入一个右花括号，以结束该水平盒子。如果你接受 TeX 的修改，TeX 遇到稍后的正确右花括号时将再次报错。它还将对该右花括号之前的任何不能出现在竖直模式的东西报错。这些额外的错误将是特别让人迷惑的，因为这些错误所说的都是假的，它们是不恰当插入右花括号后产生的结果。你的最佳做法是键入 ‘5’ 以跳过在 ‘}\vskip 1in’ 中的所有记号。

第五个例子

这里还有个相似的例子，它的错误信息比你之前看到的都要长：

```
\leftline{Skip \smallskip a little further.} But no more.
```

错误在于 \smallskip 只能在竖直模式中使用。给出的错误信息如下：

```
! Missing } inserted.
<inserted text>
}
<to be read again>
    \vskip
\smallskip ->\vskip
               \smallskipamount
<argument> Skip \smallskip
               a little further.
\leftline #1->\line {#
                           \hss }
1.93 ...Skip \smallskip a little further.}
                                         But no more.
```

这些错误信息带你巡视了在 plain TeX 中用于实现 \leftline 的各个宏——这些宏也许是你并不在乎的。第一行告诉你 TeX 试图通过插入右花括号解决此问题。TeX 还未读到右花括号，因此若你想的话你可以删除

它。第一行（带 ‘!’ 的那行）之后的信息每部分都由两行组成。下面是各部分的两行信息的含义：

- 1) 第一对双行表示 TeX 已经插入，但尚未读到的右花括号。
- 2) 第二对双行表示在读取右花括号后，TeX 将重新读取 ‘\vskip’ 命令（它来自 \smallskip 宏的定义中）。
- 3) 第三对双行表示 TeX 是在展开 \smallskip 时发现的错误。此部分也显示 \smallskip 的定义，并表示在展开和执行该定义时已经读到哪里。具体地说，它是在执行 \vskip 命令时失败的。一般地，以控制序列加 ‘->’ 开始的信息行表示 TeX 已展开执行的宏的名称。
- 4) 第四对双行表示 TeX 在处理宏参量时发现了 \smallskip，同时也表示 TeX 在该参量中的位置，即它正在处理 \smallskip（失败了）。通过往前查找接下来两行，你可以看到此参量是传递给 \leftline 的。
- 5) 第五对双行表示 TeX 是在展开 \leftline 宏时发现的错误。（在这个例子中，错误发生在 TeX 解释多个不同层级的宏定义的中间。）在 #1 后面的位置表示它最后读取到的是 \leftline 的第一个（在这里也是唯一的）参量。
- 6) 最后对双行表示 TeX 在你的输入文件的位置。注意此位置远远超出它插入右花括号并重新读取 ‘\vskip’ 的位置。这是因为 TeX 已经从输入文件读取了 \leftline 的完整参量，即使它仅仅处理该参量的一部分。这两行信息开头的圆点表示输入行前面一部分内容没显示出来。前面那部分内容实际上就包括让 \vskip 变得不合法的 \leftline 控制序列。

在类似这样的长篇信息中，你通常发现只有第一行和最后两行是有用的；但知道其他行是怎么回事有时候也有益处。任何你插入或删除的文本将在最内层级中插入或删除。在这里例子中，插入或删除将出现在 TeX 插入的右花括号前。特别要注意，在此情形中 TeX 将不会把你可能插入的文本放在你的输入文本中，而是放在多层宏定义的最里面。（原始的宏定义当然并不会被改动。）

你可以用 \errorcontextlines（第 280 页）命令限制 TeX 生成的错误信息双行的数目。如果你并非对 TeX 给你的所有信息都感兴趣，你可以设定 \errorcontextlines 为 0。这样错误信息将只有最前面两行和最后面两行。

最后，我们提及可能出现在错误信息行对开头的另外两个指示词：

- <output> 表示 TeX 在它的输出例行程序中间遇到此错误。

- <write> 表示 TeX 在执行 `\write` 命令时遇到此错误。TeX 将在实际执行 `\write` 时才检测这样的错误（在 `\shipout` 时），而不是在刚遇到 `\write` 命令时。



12 | 实用宏集概述

这一章描述 `eplain.tex`, 一组用于扩展 plain TeX 的宏和其他定义。对各种宏的描述解释了它们的目的，但通常不解释它们的工作原理或给出如何使用它们的详情。那些信息包含在 `eplain.tex` 源码文件以及它的文档中。在“资源”（第 18 页）中介绍了获取 `eplain.tex` 的方法。

预备定义

我们先介绍几个修改类别码及定义两种常用类别码的宏。

```
\def\makeactive#1{\catcode`#1 = \active \ignorespaces}%
\chardef\letter = 11 \chardef\other = 12
\def\uncatcodespecials{%
  \def\do##1{\catcode`##1 = \other}%
}\dospecials}%
Defined in plain.
```

为了将 ‘`^M`’ 定义为活动字符，你需要将定义放在编组中，并借助某些额外机制。`\letreturn` 宏让你无需额外机制就可以定义 ‘`^M`’（它的定义在下面可以看到）。

```
{\makeactive`^M \long\gdef\letreturn#1{\let`^M = #1}}%
```

这些宏用于吸收一个、两个或三个参量。

```
\def\gobble#1{} \def\gobbletwo#1#2{}%
\def\gobblethree#1#2#3{}%
```

现在我们建立一些惯例，用于读取这个文件的其他部分。在这个文件中我们允许名称带有 ‘@’ 的“私有”控制序列。这些控制序列在这个文件之外无法使用（除非你再次修改 ‘@’ 的类别码）。

```
\catcode`@ = \letter      % Allow control sequences with @.
\let\@plainwlog = \wlog % Don't log register allocations.
\let\wlog = \gobble
\newlinechar = '^^J
```

接下来两个宏提供了诊断输出的便捷方式。`\loggingall` 打开所有追踪，但让追踪输出仅写在日志文件中，而不在终端上显示。`\tracingboxes` 使得盒子被追踪时其内容完全显示。（TEX 通常只显示嵌套到三层的盒子，每个盒子显示五个项目。）

```
\def\loggingall{\tracingcommands\tw@\tracingstats\tw@
  \tracingpages@ne\tracingoutput@ne
  \tracinglostchars@ne\tracingmacros\tw@
  \tracingparagraphs@ne\tracingrestores@ne
  \showboxbreadth\maxdimen\showboxdepth\maxdimen}%
\def\tracingboxes{\showboxbreadth = \maxdimen
  \showboxdepth = \maxdimen}%
```

标线的默认厚度为 0.4 点。要生成默认厚度为任何值的标线，你可以重新定义 `\vruledefaultwidth`、`\hruledefaultheight` 及 `\hruledefaultdepth`，并将 `\hrule` 和 `\vrule` 换为 `\ehrule` 和 `\evrule`。（名称中的 ‘e’ 代表 “explain”。）如果你显式给出尺寸（比如 `\ehrule height 16pt`），TEX 将使用它。

```
\newdimen\hruledefaultheight \hruledefaultheight = 0.4pt
\newdimen\hruledefaultdepth \hruledefaultdepth = 0.0pt
\newdimen\vruledefaultwidth \vruledefaultwidth = 0.4pt
\def\ehrule{\hrule height\hruledefaultheight
  depth\hruledefaultdepth}%
\def\evrule{\vrule width\vruledefaultwidth}%
```

`\%` 通常用于写出 ‘%’ 字符，但它无法包含在 `\write` 的记号列中。你可以用 `\percentchar` 达到此目的。我们还将 `^^L` 重新定义为非外部宏，因此你可以在宏定义或者参量中使用它。

```
{\catcode`\% = \other \gdef\percentchar{ \% }%
 \def^^L{\par
 }%
```

\tokstostring 将它的参量转换为一列字符记号。它仅用到在 TeX 的食道中处理的展开。此特性是和 \edef 一起使用时所必需的。它用于宏的交叉引用（第 326 页）。

为了将参量在空格处分开，我们要用到两个子宏。@\ttsA 用于寻找空格，而@\ttsB 用于处理不带空格的控制序列。每个空格都被替换为\spacesub 的展开。

```
\def\tokstostring#1{\@ttsA#1 \ttsmarkA}%
\def@\ttsA#1 #2\ttsmarkA{\ifempty{#1}\else
    \@ttsB #1\@ttsmarkB
    \ifempty{#2}\else
        \spacesub@\ttsA#2\ttsmarkA\fi\fi}%
\def@\ttsB#1{\ifx #1\@ttsmarkB\else
    \string #1%
    \expandafter@\ttsB\fi}%
\def@\ttsmarkB{\@ttsmarkB}%
\def\@ttsmarkB{\@ttsmarkB}%
\def\spacesub{+}%
```

\ifempty 检测它的参量是否为空。

```
\def\ifempty#1{\@ifempty #1\@emptymarkA\@emptymarkB}%
\def@\ifempty#1#2\@emptymarkB{\ifx #1\@emptymarkA}%
\def@\emptymarkA{\@emptymarkA}%
```

\for 宏实现了其他传统编程语言中的 for 循环的 TeX 版本。这些宏直接取自 L^ATeX。

```
\def\for#1:=#2\do#3{\edef\@fortmp{#2}%
    \ifx\@fortmp\empty \else
        \expandafter\@forloop#2,\@nil,\@nil\@#1{#3}\fi}%
\def@\nnil{@nil}%
\def@\fornoop#1\@#2#3{%
\def\@forloop#1,#2,#3\@#4#5{\def#4{#1}\ifx #4\@nil
    \else #5\def#4{#2} \ifx #4\@nil \else
        #5\@forloop #3\@#4{#5}\fi\fi}%
\def\@iforloop#1,#2\@#3#4{\def#3{#1}\ifx #3\@nil
    \let\@nextwhile=\@fornoop \else #4\relax
    \let\@nextwhile=\@iforloop\fi
    \@nextwhile#2\@#3{#4}}%
```

`\obeywhitespace` 用于重新生成你的输入中的换行、空行和空格。它结合了 `\obeylines` 和 `\obeyspaces` 的效果，并把行首的空格也打印出来。制表符不受它影响；它们仍然生成正常的粘连。

```
\def\alwayspace{\hglue\fontdimen2\the\font \relax}%
{\makeactive\^M \makeactive\ %
\gdef\obeywhitespace{%
\makeactive\^M\def\par\indent{%
\aftergroup\@removebox% Kill extra paragraph at end.
\makeactive\ \let =\alwayspace}}%
\def\@removebox{\setbox0=\lastbox}
```

当你不想用 `\over`, 而“ $1/2$ ”看来又不够好时，你最好用 `\frac` 在正文中打印分式。这个宏是 *The T_EXbook* 练习 11.6 的答案。

```
\def\frac#1/#2{\leavevmode
\kern.1em \raise .5ex \hbox{\the\scriptfont0 #1}%
\kern-.1em $/$%
\kern-.15em \lower .25ex \hbox{\the\scriptfont0 #2}}%
```

下面这些宏生成 T_EX 圈中常用的标识。*AMS-T_EX* 标识取自 *The T_EXbook* 第 420 页。*L^AT_EX* 标识取自 `latex.tex` 并稍作修改（我们修改了‘A’的字体）；类似地，*BIBT_EX* 标识用 `\sevenrm` 字体代替真正的小型大写字体。*METAFONT* 标识的 `.mf` 源文件出现在 *METAFONT* 手册中：

Knuth, Donald E., *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

```
\def\LaTeX{L\kern-.26em \raise.6ex\hbox{\fiverm A}%
\kern-.15em TeX}%
\def\AMSTeX{$\cal A\kern-.1667em \lower.5ex\hbox{$\cal M$}}\kern-.125em\$-\kern-.1667em T\kern-.05em\lower.7ex\hbox{E}%
\def\BibTeX{{\rm B\kern-.05em \kern-.1667em I\kern-.025em B}%
\kern-.08em T\kern-.1667em\lower.7ex\hbox{E}}%
\font\mflogo = logo10
\def\MF{\mflogo META}\tenrm \{-\}\mflogo FONT}}
```

接下来的两个宏用于生成盒子。`\blackbox` 生成一个方块，它用在列表宏中（第 322 页）。`\makeblankbox`（取自 *The T_EXbook* 第 311 页）生成一个空心矩形，它的边框厚度由参量指定。

```
\def\blackbox{\vrule height .8ex width .6ex depth -.2ex}%
```

```
\def\makeblankbox#1#2{%
  \hbox{\lower\dp0\vbox{\hiderule{#1}{#2}%
    \kern -#1% overlap rules
    \hbox to \wd0{\hiderule{#1}{#2}%
      \raise\ht0\vbox to #1{}% vrule height
      \lower\dp0\vtop to #1{}% vrule depth
      \hfil\hiderule{#2}{#1}}%
    \kern-#1\hiderule{#2}{#1}}}}%
\def\hiderule#1#2{\kern-#1\hrule height#1 depth#2
  \kern-#2}%
\def\hiderule#1#2{\kern-#1{\dimen0 = #1
  \advance\dimen0 by #2 \vrule width\dimen0}\kern-#2}%

\numbername 从数字生成其书面名称（若数字大于十，这个宏仅重新生成参量的数值。）

\def\numbername#1{\ifcase#1%
  zero\or one\or two\or three\or four\or five%
  \or six\or seven\or eight\or nine\or ten\or #1\fi}%

```

\testfileexistence 判定文件 \jobname.#1 是否非空，并相应地设定 \iffileexists 的值。参量中的文件名无需以空格记号结尾，因为这个宏已经提供了空格记号。

```
\newif\iffileexists
\def\testfileexistence#1{\begingroup
  \immediate\openin0 = \jobname.#1\space
  \ifeof 0\global\fileexistsfalse
  \else \global\fileexiststrue\fi
  \immediate\closein0
\endgroup}%

```

陈列公式

TeX 默认将陈列公式（放在 \$\$ 之间的公式）居中显示。`\leftdisplay` 让陈列公式默认为左对齐显示。你也可以用 `\centereddisplay` 命令恢复居中显示。

这些宏提供让陈列公式左对齐显示之外的更多功能。对每个陈列公式，若 `\eqno` 存在则 `\ifeqno` 为真，若 `\leqno` 存在则 `\ifleqno` 为真。

如果其中一种公式编号存在，`\eqn` 将排印公式编号的内容。而 `\eq` 始终排印公式本身的内容。

这些宏基于 *The TeXbook* 第 376 页中的代码。

```
\newif\ifeqno \newif\ifleqno
\newtoks\@eqtoks \newtoks\@eqnotoks
\def\eq{\the\@eqtoks}\def\eqn{\the\@eqnotoks}%
\def\displaysetup#1$${%
  \@displaytest#1\eqno\eqno\@displaytest}%
\def\@displaytest#1\eqno#2\eqno#3\@displaytest{%
  \if #3% No \eqno, check for \leqno:
    \olddisplaytest#1\leqno\leqno\olddisplaytest
  \else
    \eqnotrue \leqnofalse % Have \eqno, not \leqno.
    \eqnotoks = {#2}\@eqtoks = {#1}%
  \fi
  \generaldisplay$$}%
\def\olddisplaytest#1\leqno#2\leqno#3\olddisplaytest{%
  \@eqtoks = {#1}%
  \if #3%
    \eqnofalse % No \leqno; we're done.
  \else
    \eqnotrue \leqnotrue % Have \leqno.
    \eqnotoks = {#2}%
  \fi}%

```

通过自己定义类似 `\leftdisplay` 的宏，你可以给陈列公式设定不同的形式。这个宏定义中必须在 `\everydisplay` 中调用 `\displaysetup`，以确保 `\displaysetup` 在每个陈列公式开始都被调用。这个宏定义必须也包含 `\generaldisplay` 的定义。

```
\newtoks\previouseverydisplay
\def\leftdisplay{%
  \previouseverydisplay = \everydisplay
  \everydisplay =
  {\the\previouseverydisplay \displaysetup}%
\def\generaldisplay{%
  \leftline{%
    \strut \indent \hspace{\leftskip}
    \dimen0 = \parindent
  }}
```

```

\advance\dimen0 by \leftskip
\advance\displaywidth by -\dimen0
\@redefinealignmentdisplays
\ifeqno \ifleqno
  \kern-\dimen0
  \rlap{$\displaystyle{\eqn}$}%
  \kern\dimen0
\fi\fi
$\displaystyle{\eqn}%
\ifeqno \ifleqno\else
  \hfill $\displaystyle{\eqn}%
\fi\fi}}}%
\def\centereddisplays{\let\displaysetup = \relax}%

```

\leftdisplays 必须历经辛苦地确保 \displaylines、\eqalignno 和 \leqalignno 还能正常使用。\\eq 在数学模式中排版，而 \\halign 在数学模式中是不合法的。我们用 \\vcenter 改变环境，以让 \\halign 重新变成合法的。我们还去掉模板左边的 \\hfil 命令以得到左对齐格式。除这些改变之外，这些宏与 plain.tex 中的相同。

```

\def\@redefinealignmentdisplays{%
\def\displaylines##1{\displ@y
  \vcenter{\halign{\hbox to\displaywidth{$\@align
    \displaystyle####\hfil$\cr\cr##1\cr\cr}}}%
\def\eqalignno##1{\displ@y
  \vcenter{\halign to\displaywidth{%
    $\@align\displaystyle####$\tabskip\z@skip
    &$\@align\displaystyle{}####$%
      \hfil\tabskip\centering
      &\llap{$\@align####$\tabskip\z@skip\cr\cr
      ##1\cr\cr}}}%
\def\leqalignno##1{\displ@y
  \vcenter{\halign to\displaywidth{%
    $\@align\displaystyle####$\tabskip\z@skip
    &$\@align\displaystyle{}####$%
      \hfil\tabskip\centering
      &\kern-\displaywidth
      \rlap{\kern-\parindent\kern-\leftskip$%
        \@align####$}}%
```

```
\tabskip\displaywidth\crcr
##1\crcr}}}}}%
```

日期时间

TeX 开始运行时设定了 `\time`、`\day`、`\month` 和 `\year` 参数的值。`\monthname` 排印月份名称的三字母缩写。`\timestring` 排印类似当前时间，类似于“1:14 p.m.”。`\timestamp` 排印完整日期文本，类似于“23 Apr 1964 1:14 p.m.”。

```
\def\monthname{%
\ifcase\month
  \or Jan\or Feb\or Mar\or Apr\or May\or Jun%
  \or Jul\or Aug\or Sep\or Oct\or Nov\or Dec%
\fi}%
\def\timestring{\begingroup
\count0 = \time \divide\count0 by 60
\count2 = \count0    % The hour.
\count4 = \time \multiply\count0 by 60
\advance\count4 by -\count0    % The minute.
\ifnum\count4<10 \toks1 = {0}%
\else \toks1 = {}%
\fi
\ifnum\count2<12 \toks0 = {a.m.}%
\else \toks0 = {p.m.}%
\advance\count2 by -12
\fi
\ifnum\count2=0 \count2 = 12 \fi % Make midnight '12'.
\number\count2:\the\toks1 \number\count4
\thinspace \the\toks0
\endgroup}%
\def\timestamp{\number\day\space\monthname\space
\number\year\quad\timestring}%
%
```

列表

\numberedlist 排印编号列表; \endnumberedlist 结束该列表。类似地, \unorderedlist 排印无序列表。这两种列表中的项目以 \li (“list item”) 开头。如果不需要列表项目间的额外间隔, 你可以在列表开始处加上 \listcompact。列表可以任意嵌套。

更一般地, 你可以通过改变下列寄存器的值控制列表项目间的间隔。如果列表项目经常比较长, 你也许希望设定非零的 \interitemskip。各列表项目的左缩进量等于 \parindent 加 \listleftindent; 各列表项目的右缩进量等于 \listrightindent。

```
\newskip\abovelistskip  \abovelistskip = .5\baselineskip
\newskip\interitemskip  \interitemskip = 0pt
\newskip\belowlistskip  \belowlistskip = .5\baselineskip
\newdimen\listleftindent \listleftindent = \parindent
\newdimen\listrightindent \listrightindent = 0pt
\def\listcompact{\interitemskip = 0pt \relax}%
```

编号列表和无序列表都使用下面这些宏。我们并不改变 \parindent 的值, 因为很多现有的宏比如 \footnote 依赖于 \parindent。我们必须考虑到项目包含不止一个段落的可能性。在这种情形中, 除第一个之外的所有段落都将被缩进。我们用 \leftskip 和 \rightskip 缩进列表项目。我们通过改变 \everydisplay 设定陈列公式的缩进。

```
\newdimen\@listindent
\def\beginlist{%
  \@listindent = \parindent
  \advance\@listindent by \listleftindent
  \everydisplay = \expandafter{\the\everydisplay
    % Don't lose user's \everydisplay:
    \advance\displayindent by \@listindent
    \advance\displaywidth by -\@listindent
    \advance\displaywidth by -\listrightindent}%
  \nobreak\vskip\abovelistskip
  \parskip = 0pt
  % \leftskip shifts nested lists to the right on the page.■
  \advance\leftskip by \@listindent
```

```
\advance\rightskip by \listrightindent}%
\def\printitem{\par\noindent
  \llap{\hspace{-\listleftindent} \marker \enspace}}%
\def\endlist{\vskip\belowlistskip}%
```

通常重新定义 `\numberedmarker` 宏，你可以改变项目标签的样式。

```
\newcount\numberedlistdepth \newcount\itemnumber
\newcount\itemletter
\def\numberedmarker{%
  \ifcase\numberedlistdepth
    (impossible)%
  \or \itemnumberout\%
  \or \itemletterout\%
  \else *\%
  \fi}%
```

这里是 `\numberedlist` 和 `\unorderedlist` 的定义。这两个定义的结构是相同的。

```
\def\numberedlist{\environment{@numbered-list}%
  \advance\numberedlistdepth by 1
  \itemnumber = 1 \itemletter = 'a
  \beginlist \let\marker = \numberedmarker
  \def\li{%
    \ifnum\itemnumber=1\else \vskip\interitemskip \fi
    \printitem
    \advance\itemnumber by 1 \advance\itemletter by 1
  }%
  \def\itemnumberout{\number\itemnumber}%
  \def\itemletterout{\char\itemletter}%
  \def\endnumberedlist{\par
    \endenvironment{@numbered-list}\endlist}%
}
```

```
\newcount\unorderedlistdepth
\def\numberedmarker{%
  \ifcase\numberedlistdepth
    (impossible)%
  \or \blackbox
  \or ---%
```

```
\else *%
\fi}%
\def\unorderedlist{\environment{@unordered-list}%
\advance\unorderedlistdepth by 1
\beginlist \itemnumber = 1
\let\marker = \unorderedmarker
\def\li{%
\ifnum\itemnumber=1\else \vskip\interitemskip \fi
\printitem \advance\itemnumber by 1
}}%
\def\endunorderedlist{\par
\endenvironment{@unordered-list}\endlist}%

```

原文呈现

`\listing` 宏用 `\tt` 字体排印指定文件的原文呈现。它基于 *The TeXbook* 第 380 页中的代码。制表符生成固定大小的间隔，而换页符生成一个分页。其他控制字符生成该字体所在位置的字符，这通常不太有用。通过重新定义 `\setuplistinghook`，你可以在读入文件前针对特别字体和/或使用环境作额外定制。

```
\def\listing#1{%
\par \begingroup \@setuplisting \setuplistinghook
\input #1 \endgroup}%
\let\setuplistinghook = \empty
\def\@setuplisting{%
\uncatcodespecials
\obeywhitespace \makeactive` \makeactive^^I
\def^^L{\vfill\eject}\tt}%
{\makeactive` \gdef`{\relax\lq}}% Defeat ligatures.
{\makeactive^^I\gdef^^I{\hskip8\fontdimen2\tt \relax}}%
```

目录

\writetocentry 宏在 \jobname.toc 文件中写入一个宏调用。 \writetocentry 的第一个参数，比如 “chapter”，用于构成所调用的宏的名称。第二个参数是显示在目录项中的文本。 \writetocentry 添加页码到宏调用中。例如：

```
\writetocentry{chapter}{Introduction}
```

将生成下列这行：

```
\tocchapterentry{Introduction}{2}
```

到 .toc 文件中，表示 ‘Introduction’ 从第 2 页开始。

利用 \writenumberedtocentry 你还可以提供第三个参数，比如章编号。例如：

```
\writenumberedtocentry{chapter}{The second chapter}{2}
```

将写入这一行：

```
\tocchapterentry{The second chapter}{2}{14}
```

你也可以自己用 \write 命令写入 \tocfile。¹

```
\newwrite\tocfile \newif\iftocfileopened
\def\opentocfile{\iftocfileopened\else
    \tocfileopenedtrue
    \immediate\openout\tocfile = \jobname.toc
\fi}%
\def\writetocentry#1#2{\ifrewritetocfile
    \opentocfile
    \write\tocfile{%
        \expandafter\noexpand \csname toc#1entry\endcsname
        {#2}{\folio}}%
\fi\ignorespaces}%
\def\writenumberedtocentry#1#2#3{\ifrewritetocfile
    \opentocfile
    \write\tocfile{%
        \expandafter\noexpand \csname toc#1entry\endcsname
        {#2}{#3}{\folio}}%
\fi\ignorespaces}%

```

¹ 译注：实际上，在本书原本所用的旧 eplain 1.9 里面，\writetocentry 调用 \writenumberedtocentry，并在后者中根据第三个参数是否非空写入不同的目录项。但在 \writenumberedtocentry 的定义中有问题，导致写入 tocfile 文件的目录项始终有三个参数。在翻译时译者也同时修正了旧 eplain 中的这个错误。

要排印出目录，只需用 `\readtocfile` 读取 `.toc` 文件。你应该在首次使用 `\writetocentry` 之前调用 `\readtocfile`。在处理目录且不想重新生成它时，务必不要改写 `.toc` 文件——如果你这样做，文件内容将会丢失。命令 `\rewritetocfilefalse` 将禁止这种改写。

```
\newif\ifrewritetocfile \rewritetocfiletrue
\def\readtocfile{\testfileexistence{toc}%
  \iffileexists
    \input \jobname.toc
    \ifrewritetocfile \opentocfile \fi
  \fi}%
```

这里给出可能出现的 `\toc...entry` 宏的一些定义。这些定义只是作为例子而已——在目录中使用指引线通常不是最好的做法。

```
\def\tocchapterentry#1#2{\line{\bf #1 \dotfill \ #2}}%
\def\tocsectionentry#1#2{%
  \line{\quad\sl #1 \dotfill \ \rm #2}}%
\def\tocsubsectionentry#1#2{%
  \line{\quad\quad\rm #1 \dotfill \ #2}}%
```

交叉引用

接下来的宏提供了符号式的交叉引用，让你可以在文档中通过名称而非实际页码提及其它部分的东西。`\xrdef{foo}` 定义标签 `foo` 为当前页码，而 `\xrefn{foo}` 生成该页码，比如 77。你更常用的是类似“see p. 77”的写法，因此 `\xref{foo}` 生成“p. 77”。如果 `foo` 未定义，就会得到一个警告信息。`\xrefwarningfalse` 取消这种警告。

这些宏没有提供对重复定义的保护。要检查重复定义，你可以将交叉引用文件排序，并用工具或肉眼检查同一个符号的相邻定义。

```
\newif\ifxrefwarning \xrefwarningtrue
\def\xrdef#1{\begingroup
  \xrlabel{#1}%
  \edef\@wr{\@writexrdef{\the\xrlabeltoks}}%
  \@wr
\endgroup \ignorespaces}%
```

```
\def\@writexrdef#1{\write\reffile{%
  \string\gdef
    \expandafter\string\csname#1\endcsname
    {\noexpand\folio}\percentchar}}%
\def\xrefnumber#1{%
  \xrlabel{#1}%
  % \@xrlabeltoks now has the control sequence name.
  \toks0 =
    \expandafter{\csname\the\@xrlabeltoks\endcsname}%
  \expandafter \ifx\the\toks0\relax
    \ifxrefwarning \message{Undefined label
      '\tokstostring{#1}'.}\fi
    \let\spacesub = \space
    \expandafter\xdef\the\toks0
      {'\tt \tokstostring{#1}}}\fi
  \the\toks0}%
\def\xref#1{p.\thinspace\xrefnumber{#1}}%
\def\xrefn#1{\xrefnumber{#1}}%
```

这个宏将一个标签转换为一列字符记号，并放在寄存器 `\labeltoks` 中。除了普通字符之外，标签还可以包含空格和控制序列，但不能包含花括号。

```
\newtoks\@xrlabeltoks
\def\xrlabel#1{\begingroup
  \escapechar = '\_ \edef\tts{\tokstostring{#1\_}}%
  \global\@xrlabeltoks = \expandafter{\tts}%
\endgroup}%

```

需要运行两遍才能得到正确的交叉引用，因为这些定义是写出到辅助文件 `\jobname.aux` 里的。`\readreffile` 命令用于将它们读取回来。如果你不在第一个定义之前执行此命令，你将丢失上一遍运行得到的定义。

```
\newwrite\reffile \newif\ifreffileopened
\def\openreffile{\ifreffileopened\else
  \reffileopenedtrue
  \immediate\openout\reffile = \jobname.aux
\fi}%
\def\readreffile{%
  \testfileexistence{aux}}%
```

```

\iffileexists
  \begingroup
    \@setletters
    \input \jobname.aux
  \endgroup
\else
  \message{No cross-reference file; I won't give you
  warnings about undefined labels.}%
  \xrefwarningfalse
\fi
\openreffile}%
\def\@setletters{%
  \catcode`_ = \letter \catcode`+ = \letter
  \catcode`- = \letter \catcode`@ = \letter
  \catcode`0 = \letter \catcode`1 = \letter
  \catcode`2 = \letter \catcode`3 = \letter
  \catcode`4 = \letter \catcode`5 = \letter
  \catcode`6 = \letter \catcode`7 = \letter
  \catcode`8 = \letter \catcode`9 = \letter
  \catcode`(` = \letter \catcode`)` = \letter}%

```

按同样的方式，你可以用 `\eqdef` 和 `\eqref` 给出公式的符号式名称。

`\eqdef` 插入自己的 `\eqno` 命令，因此它必须用在能用 `\eqno` 的地方。

```

\newcount\eqnumber
\def\eqdef#1{\global\advance\eqnumber by 1
  \expandafter\xdef
    \csname#1eqref\endcsname{\the\eqnumber}%
  \immediate\write\reffile{\string\def
    \expandafter\string\csname#1eqref\endcsname
    {\the\eqnumber}}%
\eqno
\eqprint{\the\eqnumber}%

```

`\eqref` 生成“(公式编号)”。通过重新定义 `\eqprint`，你可以实现更加复杂的格式。举个例子，你可以重新定义它，使得公式编号中包含章编号。

```

\def\eqref#1{%
  \expandafter \ifx \csname#1eqref\endcsname \relax
    \ifxrefwarning \message{Undefined equation label

```

```
'#1'.}\fi  
\expandafter\def\csname#1eqref\endcsname{00}%">  
\else \eqprint{\csname#1eqref\endcsname}%">  
\fi}%">  
\def\eqprint#1{(#1)}%
```

环境

这些宏让你可以将手稿的一部分定义为命名编组（环境）。类似 TeX 的编组，这些编组也可以嵌套，而且实际上它们的嵌套可以与 TeX 编组的嵌套相互交织。如果开始的名称和结束的名称不匹配，你将得到一个错误信息。这些宏如此设计，使得当你得到这种错误信息时，你有机会方便地定位错误的来源。

用 `\environment {foo}` 开始一个环境，用 `\endenvironment{foo}` 结束该环境，其中 `foo` 是该环境的名称。我们的宏稍微改进了 *The TeXbook* 练习 5.7，添加对 `\begingroup` 和 `\endgroup` 是否配对的检查，并确保 `\environment` 和 `\endenvironment` 能够匹配。

```
\def\environment#1{\ifx\@groupname\undefined\else  
    \errhelp = \@unnamedendgrouphelp  
    \errmessage{'\@groupname' was not closed by  
        \string\endenvironment}\fi  
\def\@groupname{#1}%">  
\begingroup  
    \let\@groupname = \undefined \ignorespaces}%">  
\def\endenvironment#1{\endgroup  
    \def\@thearg{#1}%">  
    \ifx\@groupname\@thearg  
    \else  
        \ifx\@groupname\undefined  
            \errhelp = \@isolatedendenvironmenthelp  
            \errmessage{Isolated  
                \string\endenvironment\space for '#1'}%  
        \else  
            \errhelp = \@mismatchedenvironmenthelp
```

```

\errmessage{Environment '#1' ended,
but '@groupname' started}%
\endgroup % Probably a typo in the names.

\fi
\fi
\let\@groupname = \undefined \ignorespaces}%

```

你也可以给上述这些错误定义它们各自的帮助信息。

```

\newhelp\@unnamedendgroup{%
  Most likely, you just forgot an^^J%
  \string\endenvironment.

  Maybe you should try inserting another^^J%
  \string\endgroup to recover.}%

\newhelp\@isolatedendenvironment{%
  You ended an environment X, but^^J%
  no \string\environment\space to start it
  is anywhere in sight.^^J%
  You might also be at an
  \string\endenvironment\space that would match^^J%
  a \string\begin{group}, i.e., you forgot an
  \string\endgroup.}%

\newhelp\@mismatchedenvironment{%
  You started an environment X, but^^J%
  you ended it with Y. Maybe you made a typo
  in one or the other^^J%
  of the names.}%

```

某些环境应当不允许出现在其他环境内部。我们称这些环境为“外部环境”。\checkenv 检测是否不存在当前有效的外部环境，若存在的话给出警告信息。要使用 \checkenv，你必须在每个外部环境开头处执行 \environmenttrue 命令。

```

\newif\ifenvironment
\def\checkenv{%
  \ifenvironment
    \errhelp = \@interwovenenvhelp
    \errmessage{Interwoven environments}%
    \endgroup
  \fi}%
\newhelp\@interwovenenvhelp{%

```

Perhaps you forgot to end the previous ^J%
 environment? I'm finishing off the current group, ^J%
 hoping that will fix it. }%

对齐

这三个宏 `\flushleft`、`\flushright` 和 `\center` 将后面各行文本以指定方式对齐。这种命令必须单独写在一行中。命令和文本应该包含在一个编组中——编组的结束就表示文本的结束。整个编组作为一个段落排版，各行视情况在一边或两边用空白填充。空行依原样显示。

```
\begingroup
\catcode `'^M = \active
\globaldefs = 1 %
\def\flushleft{\beforejustify %
  \aftergroup\endflushleft %
  \def^M{\null\hfil\break}%
  \def\@eateol^M{}@\eateol}%
\def\flushright{\beforejustify %
  \aftergroup\endflushright %
  \def^M{\break\null\hfil}%
  \def\@eateol^M{\hfil\null}@\eateol}%
\def\center {\beforejustify %
  \aftergroup\endcenter %
  \def^M{\hfil\break\null\hfil}%
  \def\@eateol^M{\hfil\null}@\eateol}%
\endgroup
```

在 `\flushleft`、`\flushright` 和 `\center` 的定义中，下述这些命令出现在 `\aftergroup` 的后面；它们在编组结束后被调用，以执行一些必需的清理工作。

```
\def\endflushleft{\unpenalty
  {\parfillskip = 0pt plus 1 fil\par}%
  \ignorespaces}%
\def\endflushright{%
  % Remove the \hfil\null\break we just put on.
  \unskip \setbox0=\lastbox \unpenalty
```

```
% We have fil glue at the left of the line;
% \parfillskip shouldn't affect that.
{\parfillskip = 0pt \par}\ignorespaces}%
\def\@endcenter{%
% Remove the \hfil\null\break we just put on.
\unskip \setbox0=\lastbox \unpenalty
% We have fil glue at the left of the line;
% \parfillskip must balance it.
{\parfillskip = 0pt plus 1fil \par}\ignorespaces}%
\def\beforejustify{%
\par\noindent
\catcode`\^M = \active
\checkenv \environmenttrue}%
```

表格

\makecolumns 宏允许你给出表格的所有元素，而无需担心如何分栏。例如，在输入一个冗长的按字母排序的名单，并分多栏显示时，你通常不知道在何处结束一栏开始下一栏。另外，如果添加了另一项，各栏的划分将会改变。

\makecolumns 有两个（定界）参量：表格元素的总数以及表格的栏数。例如，‘\makecolumns 37/3：’指定一个三栏表格，其元素为接下来的 37 行。要调整表格在页面上的位置，你可以修改 \parindent 以确定表格左侧的空白，以及 \hsize 以确定从页面左边缘到表格右侧的距离。要允许在 \valign 之前分页，你可以修改 \abovecolumns penalty。

```
\newcount\abovecolumns penalty
\abovecolumns penalty = 10000
\newcount\@linestogo      % Lines remaining to process.
\newcount\@linestogoincolumn % Lines remaining in column.
\newcount\@columndepth     % Number of lines in a column.
\newdimen\@columnwidth      % Width of each column.
\newtoks\crtok  \crtok = {\cr}%
\def\makecolumns#1/#2: {\par \begingroup
  \columndepth = #1 \advance\columndepth by #2
  \advance\columndepth by -1
```

```

\divide \columndepth by #2
\@linestogoincolumn = \columndepth \@linestogo = #1
\def\@endcolumnactions{%
  \ifnum \@linestogo<2
    \the\crtok \egroup \endgroup \par
    % End \valign and \makecolumns.
  \else
    \global\advance\@linestogo by -1
    \ifnum\@linestogoincolumn<2
      \global\@linestogoincolumn = \columndepth
      \the\crtok
    \else &\global\advance\@linestogoincolumn by -1
    \fi
  \fi}%
\makeactive\^\^M\let\return\@endcolumnactions
\@columnwidth = \hsize
\advance\@columnwidth by -\parindent
\divide\@columnwidth by #2
\penalty\abovecolumns penalty
\noindent % It's not a paragraph (usually).
\valign\bgroup
  &\hbox to \@columnwidth{\strut ##\hfil}\cr
}%
} The next end-of-line starts everything going.

```

脚注

脚注通常用升高的数字作为参考符号。我们定义的 `\numberedfootnote` 宏就是这样的。它还重新定义了 `\vfootnote`, 以支持比 plain TeX 更一般的脚注格式。尺寸寄存器 `\footnotemarkseparation` 控制脚注符号（比如数字）与脚注文本之间的间隔。`\everyfootnote` 记号在每个脚注之前插入。

Plain TeX 中的 `\footnote` 和 `\vfootnote` 定义保留在 `\@plainfootnote` 和 `\@plainvfootnote` 中, 以备不时之需。

```
\newcount\footnotenumber \newtoks\everyfootnote
```

```

\newdimen\footnotemarkseparation
\footnotemarkseparation = .5em
\let\@plainfootnote = \footnote
\let\@plainvfootnote = \vfootnote
\def\vfootnote#1{\insert\footins\bgroup
  \interlinepenalty\interfootnotelinepenalty
  \splittopskip\ht\strutbox \splitmaxdepth\dp\strutbox
  \floatingpenalty\@MM
  \leftskip\z@skip \rightskip\z@skip \spaceskip\z@skip
  \xspaceskip\z@skip
  \everypar = {}%
  \the\everyfootnote
  \indent\llap{#1\kern\footnotemarkseparation}\footstrut
  \futurelet\next\fo@t}%
\def\numberedfootnote{\global\advance\footnotenum by 1
  \@plainfootnote{$^{\{\number\footnotenum\}}$}}%

```

双栏

\doublecolumns 命令开始双栏输出，而 \singlecolumn 命令恢复单栏输出。两者可以在同个页面上相互切换。用 \abovedoublecolumnskip 和 \belowdoublecolumnskip 指定的粘连插入到双栏素材的前面和后面。

它们的实现方式来源于 *The TeXbook* 第 417 页。

```

\newskip\abovedoublecolumnskip
\newskip\belowdoublecolumnskip
\abovedoublecolumnskip = \bigskipamount
\belowdoublecolumnskip = \bigskipamount
\newdimen\gutter           \gutter = 2pc
\newdimen\doublecolumnhsize \doublecolumnhsize = \hsize
\newbox\@partialpage       \newdimen\singlecolumnhsize
\newdimen\singlecolumnvsize \newtoks\previousoutput
\def\doublecolumns{\par % Don't start in horizontal mode.
  \previousoutput = \expandafter{\the\output}

```

```
\advance\doublecolumnhsize by -\gutter
\divide\doublecolumnhsize by 2
\output = {\global\setbox\@partialpage =
    \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
\pagegoal = \pagetotal \break % Expands \output above.
\output = {\doublecolumnoutput}%
\singlecolumnhsize = \hsize
\singlecolumnvsize = \vsize
\hsize = \doublecolumnhsize \vsize = 2\vsize}%

\@doublecolumnspli 宏执行实际的分栏。插入项被当作单栏素材；  
如果这不是你想要的，你需要修改输出例行程序。在 \@doublecolumnspli 完成之后，\box255 将包含双栏素材。双栏素材前面是调用 \doublecolumns 前的单栏素材。 \box4 将包含无法放入该页面的素材。
```

```
\def\@doublecolumnspli{%
    \splittopskip = \topskip \splitmaxdepth = \maxdepth
    \dimen0 = \singlecolumnvsize
        \advance\dimen0 by -\ht\@partialpage
        \advance\dimen0 by -\ht\footins
        \advance\dimen0 by -\skip\footins
        \advance\dimen0 by -\ht\topins
    \begingroup
        \vbadness = 10000
        \global\setbox1=\vsplit255 to \dimen0 \wd1=\hsize
        \global\setbox3=\vsplit255 to \dimen0 \wd3=\hsize
    \endgroup
    \global\setbox4=\vbox{\unvbox255
        \penalty\outputpenalty}%
    \global\setbox255=\vbox{\unvbox\@partialpage
        \hbox to \singlecolumnhsize{\box1\hfil\box3}%
        \vfill}%
}
```

\doublecolumnoutput 是真正的输出例行程序。我们调用 \output 执行实际的盒子送出工作。

```
\def\doublecolumnoutput{\@doublecolumnsplit
  \hsize = \singlecolumnhsize \vsize = \singlecolumnvsize
  \previousoutput \unvbox4}%
\singlecolumn 恢复单栏排版。它假定已调用了 \doublecolumns.
\def\singlecolumn{\par % Don't start in horizontal mode.
  \output = {\global\setbox1 =
    \vbox{\unvbox255\vskip\abovedoublecolumnskip}}%
  \pagegoal = \pagetotal \break \setbox255 = \box1
  {\singlecolumnvsize = \ht255
    \divide\singlecolumnvsize by 2
    \advance\singlecolumnvsize by +\ht\@partialpage
    \advance\singlecolumnvsize by +\ht\footins
    \advance\singlecolumnvsize by +\skip\footins
    \advance\singlecolumnvsize by +\ht\topins
  \@doublecolumnsplit}%
  \hsize = \singlecolumnhsize
  \vsize = \singlecolumnvsize
  \output = \expandafter{\the\previousoutput}%
  \unvbox255}%
```

收尾

现在我们必须撤销开始的修改（见第 315 页）。我们还给出版本标识，它可以在 \fmtname 和 \fmtversion 中得到。

```
\let\wlog = \@plainwlog \catcode`@ = \other
\def\fmtname{eplain}%
{\edef\plainversion{\fmtversion}%
 \xdef\fmtversion{1.0: 15 May 1990
 (and plain \plainversion)}%
}%
```




13 | 命令速查表

这一章将简单地描述原始的 TeX 与 plain TeX 中定义的命令，包括控制序列与特殊字符。我们忽略了那些只在 plain TeX 定义内部使用的命令。需要注意的是，普通字符，如 ‘a’ 或 ‘6’ 也是命令，并且是最常见的命令（见“字符”，第 56 页）。

为了保持描述的简洁，我们采用如下惯例。

- 命令前面的星号表明该命令是原始的，即内建于 TeX 计算程序中的命令（见“原始的”，第 87 页）。
- 在命令描述中出现“音乐”、“标点”、“函数”、“符号”、“关系”、“定界符”、“运算符”，则暗示该命令只在数学模式中有效。
- 除非有其它说明，“显示”一词应用于 TeX 发送至日志文件中的信息。如果 `\tracingon` 是正值，TeX 也会将其发送至终端。我们使用名词“陈列”表示陈列公式（参见第 62 页），即在 `$$` 之间的内容。
- 短语“输出 x ”表明指定命令将对 x 进行排版，并将结果放在盒子中。在不会引起歧义的情况下，我们有时会省略“输出”一词。比如，我们对 `\alpha` 的描述是“数学希腊字母 α ”，而不是“输出数学希腊字母 α ”。

*`\hskip` 字间距（第 104 页）

`\!` 数学中的负的细小间隔（第 225 页）

`\textnormal` 文本中的元音变音符，比如 ö 中那样（第 100 页）

`#` 引入宏参数，或在对齐导言中指示文本条目的位置（第 75 页，第 48 页）

\# 从当前字体中输出 # 字符 (第 98 页)
\$ 开始或结束数学公式 (第 16 页)
\\$ 从当前字体中输出 \$ 字符 (第 98 页)
* % 开始注释 (第 13 页)
\% 从当前字体中输出 % 字符 (第 98 页)
& 在对齐中分隔模板与条目 (第 184 页)
\& 从当前字体中输出 & 字符 (第 98 页)
,, 数学中的素数符号, 比如 p' 中那样 (第 197 页)
\, 文本中的尖音符, 比如 é 中那样 (第 100 页)
* 允许断行的乘号 (第 199 页)
\+ 开始制表符行 (第 182 页)
\, 数学中的细小间隔 (第 225 页)
\,- 指定合法的断字点 (第 127 页)
\. 文本中的点重音符, 比如 ñ 中那样 (第 100 页)
\,/ 对之前字符的斜体校正 (第 106 页)
\; 数学中的较大间隔 (第 225 页)
\= 文本中的长音符号, 比如 ë 中那样 (第 100 页)
* \ 开始控制序列 (第 10 页)
\> 数学中的中等间隔 (第 225 页)
\^ 以上标形式输出指定的子公式 (第 206 页)
\~ 文本中的抑扬符号, 比如 ô 中那样 (第 100 页)
\^L 等同于原始的 \par (第 110 页)
* \^M 行结束标志 (第 105 页)
_ 以下标形式输出指定的子公式 (第 206 页)
_ 底线: _ (第 98 页)
\` 文本中的抑音符, 比如 è 中那样 (第 100 页)
\{ 开始编组 (第 242 页)
\{ 数学中的左大括号定界符: { (第 201 页)
\| 数学中的平行线: || (第 197 页)
\} 结束编组 (第 242 页)
\} 数学中的右大括号定界符: } (第 201 页)
\~ 字间间隔, 该处不能断行 (第 105 页)
\~ 文本中的颤化符, 比如 å 中那样 (第 100 页)
\aa 斯堪的纳维亚字母: å (第 97 页)
\AA 斯堪的纳维亚字母: Å (第 97 页)

- *\above 输出带有指定厚度横线的分式 (第 210 页)
- *\abovedisplayskip 当前一行可以放在陈列公式的缩进部分时, TeX 在陈列公式之前插入的粘连, 其默认值是 0 点 plus 3 点 (第 229 页)
- *\abovedisplayskip 当前一行不能放在陈列公式的缩进部分时, TeX 在陈列公式之前插入的粘连, 其默认值是 12 点 plus 3 点 minus 9 点 (第 228 页)
- *\abovewithdelims 输出带有指定厚度横线且被指定定界符围绕的分式 (第 211 页)
- *\accent 将指定重间符加在后接字符之上 (第 101 页)
- \active 活动字符的类别码, 即数字 13 (第 267 页)
- \acute 数学中的尖音符, 比如 \acute{x} 中那样 (第 209 页)
- *\adjdemerits 对造成相邻行单词间距不相容的断行附加的额外缺陷, 其默认值为 10000 (第 125 页)
- *\advance 向 \count 寄存器增加一个数 (第 261 页)
 - \advancenopage 如果 \pageno 为正值, 加一; 如果为负值, 减一 (第 145 页)
 - \ae æ 连字 (第 97 页)
 - \AE Ä 连字 (第 97 页)
- *\afterassignment 直到下一赋值完成后才扩展随后的记号 (第 244 页)
- *\aftergroup 直到当前编组结束后才扩展随后的记号 (第 243 页)
- \aleph 数学中唯一的希伯来字母: \aleph (第 197 页)
- \allowbreak 执行 \penalty0, 即在通常不能断行或分页的地方允许断行或分页 (第 121 页, 第 139 页)
- \alpha 数学希腊字母 α (第 196 页)
- \amalg 合并运算符: \amalg (第 198 页)
- \angle 角度符号: \angle (第 197 页)
- \approx 近似关系: \approx (第 200 页)
- \arccos 反余弦函数: \arccos (第 203 页)
- \arcsin 反正弦函数: \arcsin (第 203 页)
- \arctan 反正切函数: \arctan (第 203 页)
- \arg 辐角 (相位) 函数: \arg (第 203 页)
- \arrowvert 可延长的双箭头的竖直部分 (第 223 页)
- \Arrowvert 可延长的单箭头的竖直部分 (第 223 页)
- \ast 星号运算符: * (第 198 页)
- \asymp 渐近关系: \asymp (第 200 页)

- *\atop 输出没有横线的分式 (第 210 页)
- *\atopwithdelims 输出没有横线且被指定定界符围绕的分式 (第 211 页)
- \b 数学中下横线重音符, 比如 \bar{x} 中那样 (第 209 页)
- \backslash 反斜线符号: \ (第 197 页)
- *\badness 在上一个生成的盒子中设定的粘连的劣度 (第 176 页)
- \bar 数学中横线重音符, 比如 \bar{x} 中那样 (第 209 页)
- *\baselineskip 从一基线到另一基线的正常竖直距离的粘连, 其默认值是 12 点 (第 136 页)
- *\batchmode 不在错误处停下, 也不向终端输出 (第 269 页)
- *\begingroup 开始由 \endgroup 结束的编组 (第 241 页)
- \begin{section} 开始文档的一个主要部分 (第 130 页)
- *\belowdisplayshortskip 当前一行可以放在陈列公式的缩进部分时, TeX 在陈列公式之后插入的粘连, 其默认值是 7 点 plus 0.3 点 minus 4 点 (第 229 页)
- *\belowdisplayskip 当前一行不能放在陈列公式的缩进部分时, TeX 在陈列公式之后插入的粘连, 其默认值是 12 点 plus 3 点 minus 9 点 (第 228 页)
- \beta 数学希腊字母 β (第 196 页)
- \bf 使用粗体, 即执行 \tenbf\fam=\bffam (第 103 页)
- \bffam 数学中的粗体字族 (第 221 页)
- \bgroup 隐式编组开始字符 (第 242 页)
- \big 使得指定定界符大于正常大小, 但对于文本依旧足够小 (第 222 页)
- \Big 使得指定定界符约 11.5 点高 (第 222 页)
- \bigbreak 用 \penalty-200 给出合适分页点, 并生成大小为 \bigskipamount 的粘连 (第 140 页)
- \bigcap 大的求交运算符 (它不是用来输出大写字母的!): \bigcap (第 204 页)
- \bigcirc 大的圆圈运算符: \bigcirc (第 198 页)
- \bigcup 大的求并运算符: \bigcup (第 204 页)
- \bigg 使得指定定界符高度约 14.5 点 (第 222 页)
- \Bigg 使得指定定界符高度约 17.5 点 (第 222 页)
- \biggl 大小与 \bigg一样, 但间隔与开符号一样 (第 222 页)
- \Biggl 大小与 \Bigg一样, 但间隔与开符号一样 (第 222 页)
- \biggm 大小与 \bigg一样, 但间隔与关系符号一样 (第 222 页)
- \Biggm 大小与 \Bigg一样, 但间隔与关系符号一样 (第 222 页)

\biggr 大小与 \bigg一样，但间隔与闭符号一样 (第 222 页)
 \Biggr 大小与 \Bigg一样，但间隔与闭符号一样 (第 222 页)
 \bigl 大小与 \big一样，但间隔与开符号一样 (第 222 页)
 \Bigl 大小与 \Big一样，但间隔与开符号一样 (第 222 页)
 \bigm 大小与 \big一样，但间隔与关系符号一样 (第 222 页)
 \Bigr 大小与 \Big一样，但间隔与关系符号一样 (第 222 页)
 \bigodot 大的圆圈点运算符: \odot (第 204 页)
 \bigoplus 大的圆圈加号运算符: \oplus (第 204 页)
 \bigotimes 大的圆圈乘号运算符: \otimes (第 204 页)
 \bigr 大小与 \big一样，但间隔与闭符号一样 (第 222 页)
 \Bigr 大小与 \Big一样，但间隔与闭符号一样 (第 222 页)
 \bigskip 输出 \bigskipamount 粘连 (第 158 页)
 \bigskipamount 用于大的竖直间距的粘连，其默认值为 12 点 plus
 4 点 minus 4 点 (第 158 页)
 \bigsqcup 大的方形求并运算符: \sqcup (第 204 页)
 \bigtriangledown 向下的三角运算符: \bigtriangledown (第 198 页)
 \bigtriangleup 向上的三角运算符: \bigtriangleup (第 198 页)
 \biguplus 大的求并的加法运算符: \biguplus (第 204 页)
 \bigvee 大的逻辑“或”运算符: \bigvee (第 204 页)
 \bigwedge 大的逻辑“和”运算符: \bigwedge (第 204 页)
 *\binoppenalty 一个二元数学运算符后断行或分页的额外惩罚，默认
 值是 700 (第 126 页)
 \bmod 模运算符，比如 $n \bmod 2$ 中那样 (第 203 页)
 \bordermatrix 带有行和列标志的矩阵 (第 216 页)
 \bot 格底符号: \bot (第 197 页)
 *\botmark 在刚放入盒子的页面内的最后一个标记项 (第 147 页)
 \bowtie 领结关系: \bowtie (第 200 页)
 *\box 将一指定的盒子寄存器中的盒子附加到当前列表中，并使寄存器
 失效 (第 169 页)
 *\boxmaxdepth vbox 的最大深度，其默认值是 \maxdimen (第 168 页)
 \brace \$n\brace k\$ 输出带有一对大括号的标记: $\{^n_k\}$ (第 210 页)
 \bracevert 可延长的大括号的竖直部分 (第 223 页)
 \brack \$n\brack k\$ 输出带有一对方括号的标记: $[^n_k]$ (第 210 页)
 \break 执行 \penalty-10000，即强制断行或分页 (第 121 页，
 第 139 页)
 \breve 数学中的短音重音符，象 \check{x} 中一样 (第 209 页)

- *`\brokenpenalty` 在自定项目处断行的惩罚，默认值是 100 (第 141 页)
- `\buildrel` 在一指定关系上方输出指定公式 (第 212 页)
- `\bullet` 圆点运算符: • (第 198 页)
- `\bye` 用空白间隔 `\vfill` 最后一页, `\supereject` 它, 并且 `\end` 作业 (第 263 页)
- `\c` 文本中的变音符号重音符, 比如 \c{c} 中那样 (第 100 页)
- `\cal` 数学中, 对于大写字母使用书法字体, 比如 $\mathcal{X}\mathcal{Y}\mathcal{Z}$ 中那样 (第 220 页)
- `\cap` 交集运算符: \cap (第 198 页)
- `\cases` 输出数学中的条件式, 比如 $\{\dots\}$ 中那样 (第 211 页)
- *`\catcode` 指定字符的类别码 (第 267 页)
- `\cdot` 居中点操作符: · (第 198 页)
- `\cdotp` 居中句号: · (第 206 页)
- `\cdots` 数学中的居中多点: ··· (第 213 页)
- `\centerline` 输出文字居中的行 (第 108 页)
- *`\char` 从当前字体中输出指定编码的字符 (第 99 页)
- *`\chardef` 定义指定控制序列, 使之成为 0 至 255 之间的字符编码 (第 247 页)
- `\check` 数学中的抑制重音符, 比如 \check{x} 中那样 (第 209 页)
- `\chi` 数学希腊字母 χ (第 196 页)
- `\choose` $\$n \choose k$ 输出组合标记: $\binom{n}{k}$ (第 210 页)
- `\circ` 圆圈操作符: ○ (第 198 页)
- *`\cleaders` 输出在第一盒子之前和最后一个盒子之后保留一半剩余空间的指引线 (第 179 页)
- `\cleartabs` 为制表阵列清除所有的制表符 (第 183 页)
- *`\closein` 关闭指定的输入流 (第 264 页)
- *`\closeout` 关闭指定的输出流 (第 265 页)
- *`\clubpenalty` 对分页前剩余单行的额外惩罚, 默认值为 150 (第 141 页)
- `\clubsuit` 梅花花色符号: ♣ (第 197 页)
- `\colon` 数学中的分号: : (第 206 页)
- `\cong` 全等关系: \cong (第 200 页)
- `\coprod` 上积运算符: \coprod (第 204 页)
- *`\copy` 与 `\box` 一样, 但不会使寄存器失效 (第 169 页)
- `\copyright` 版权符号: © (第 98 页)
- `\cos` 余弦函数: \cos (第 203 页)

- \cosh 双曲余弦函数: \cosh (第 203 页)
- \cot 余切函数: \cot (第 203 页)
- \coth 双曲余切函数: \coth (第 203 页)
- *\count 指定的整数寄存器 (第 258 页)
- *\countdef 定义指定的控制序列, 使之成为与 \count 寄存器相对应的数值 (第 261 页)
- *\cr 在对齐中结束一行 (或一列) (第 186 页)
- *\crcr 如果最后一个命令是 \cr 或 \noalign, 则什么也不做; 否则等同于 \cr (第 187 页)
- \csc 余割函数: \csc (第 203 页)
- *\csname 开始一个以 \endcsname 结束的控制序列命名 (第 248 页)
- \cup 求并运算符: \cup (第 198 页)
- \d 文本中底点重音符, 比如 r 中那样 (第 100 页)
- \dag 文本中的剑形符号: \dag (第 98 页)
- \dagger 数学中的剑形运算符: \dagger (第 198 页)
- \dashv 右十字转门关系: \dashv (第 200 页)
- *\day 月份中的当天, 作为一个数字 (第 239 页)
- \ddag 文本中的双剑形符号: \ddag (第 98 页)
- \ddagger 数学中的双剑形运算符: \ddagger (第 198 页)
- \ddot 数学中的双点重音符: \ddot{x} (第 209 页)
- \ddots 数学中的斜多点: \ddots (第 214 页)
- *\deadcycles 自从最后 \shipout 后的 \output 初始化数值 (第 151 页)
- *\def 定义一个控制序列, 使之成为宏 (第 245 页)
- *\defaulthyphenchar 默认断字字符代码 (第 130 页)
- *\defaultskewchar 默认重音符偏移字符代码 (第 224 页)
- \deg 角度函数: \deg (第 203 页)
- *\delcode 指定字符的定界符代码 (第 268 页)
- *\delimiter 输出指定的定界符 (第 215 页)
- *\delimiterfactor 1000 乘以定界符的最小尺寸与完成覆盖公式的尺寸的比例 1000, 其默认值为 901 (第 215 页)
- *\delimitershortfall 公式高度与分界符高度之间的最小差异, 默认值为 5 点 (第 215 页)
- \delta 数学希腊字母 δ (第 196 页)
- \Delta 数学希腊字母 Δ (第 196 页)
- \det 行列式函数: \det (第 203 页)
- \diamond 菱形运算符: \diamond (第 198 页)

- \diamondsuit 方块花色符号: \diamond (第 197 页)
- \dim 维度函数: dim (第 203 页)
- *\dimen 指定尺寸寄存器 (第 258 页)
- *\dimendef 定义指定控制序列, 使之成为对应于 \dimen 寄存器的一个数值 (第 261 页)
- *\discretionary 指定三个文本, 前两个分别用在分行前后, 第三个为不分行时用 (第 127 页)
- *\displayindent TeX 设定该值为陈列公式的缩进量 (第 228 页)
- *\displaylimits 只在陈列样式中, 将极限放在运算符的上或下 (第 205 页)
- \displaylines 输出指定的每行居中的多行显示陈列公式 (第 218 页)
- *\displaystyle 在公式中使用陈列样式尺寸 (第 207 页)
- *\displaywidowpenalty 在一页开始处单行位于陈列公式前的惩罚, 默认值为 50 (第 141 页)
- *\displaywidth TeX 设定该值为陈列公式的宽度 (第 228 页)
- \div 除号运算符: \div (第 198 页)
- *\divide 用指定的 \count 寄存器除以指定的整数 (第 262 页)
- \dot 数学中的点重音符, 比如 \dot{x} 中那样 (第 209 页)
- \doteq 点等于关系: \doteq (第 200 页)
- \dotfill 用点填充闭合的水平间隔 (第 181 页)
- \dots 序列的省略号: x_1, \dots, x_n (第 99 页)
- *\doublehyphendemerits 连续以连字号结束行的处罚, 默认为 10000 (第 126 页)
- \downarrow 关系: \downarrow (第 202 页)
- \Downarrow 关系: \Downarrow (第 202 页)
- \downbracefill 用开口向下的大括号填充闭合的 hbox:  (第 223 页)
- *\dp 指定盒子寄存器中的盒子的深度 (第 172 页)
- *\dump 结束作业, 并产生格式文件 (第 280 页)
- *\edef 定义一个控制序列, 使之成为宏, 立即展开替换文本 (第 245 页)
- \egroup 隐式编组结束字符 (第 242 页)
- \eject 结束当前段落, 并强制分页, 伸展当前页 (第 140 页)
- \ell 数学中的草书字母: ℓ (第 197 页)
- *\else 条件式的错误或默认情况下的替换 (第 255 页)
- *\emergencystretch 如果 \tolerance 没有得到满足时, 加到每行上的额外的伸展 (第 124 页)
- \empty 不作任何扩展的宏 (第 257 页)

\emptyset 空集符号: \emptyset (第 197 页)

*\end \output 最后一页, 并且结束作业 (第 263 页)

*\endcsname 结束以 \csname 开始的控制序列名称 (第 248 页)

\endgraf 等同于原始的 \par (第 110 页)

*\endgroup 结束以 \begingroup 开始的编组 (第 241 页)

*\endinput 结束从当前文件中的输入 (第 263 页)

\endinsert 结束插入 (第 150 页)

\endline 等同于原始的 \cr (第 186 页)

*\endlinechar T_EX 在每个输入行结尾插入的字符, 默认为 \^M (第 268 页)

\enskip 宽度为 $1/2\text{em}$ 的水平粘连 (第 157 页)

\enspace $1/2\text{em}$ 紧排 (第 157 页)

\epsilon 起始希腊字母 ϵ (第 196 页)

\eqalign 输出指定的多行显示行列公式, 其指定部分是竖直对齐的 (第 218 页)

\eqalignno 输出带有公式编号的多行行列公式, 其指定部分是竖直对齐的 (第 218 页)

*\eqno 放指定的公式编号放在行列公式的右侧 (第 217 页)

\equiv 等价关系: \equiv (第 200 页)

*\errhelp 当用户在面对 \errmessage 寻求帮助时, T_EX 展开并显示的记号列 (第 279 页)

*\errmessage 给出指定的错误信息 (第 278 页)

*\errorcontextlines T_EX 遇到错误时显示的双行信息的数目, 默认值为 5 (第 280 页)

*\errorstopmode 在错误信息处停下来等待交互过程 (第 269 页)

*\escapechar T_EX 显示控制序列名称时的前导字符 (第 240 页)

\eta 数学希腊字母 η (第 196 页)

*\everycr 在 \cr 后, 或者在非 \cr 或 \noalign 后的 \crrcr 后 T_EX 展开的记号列表 (第 191 页)

*\everydisplay 任一行列公式开始时, T_EX 展开的记号列表 (第 230 页)

*\everyhbox 任一水平盒子开始时, T_EX 展开的记号列表 (第 169 页)

*\everyjob 任一作业开始时, T_EX 展开的记号列表 (第 281 页)

*\everymath 当文本数学模式开始时, T_EX 展开的记号列表 (第 230 页)

*\everypar 任一段落开始时, T_EX 展开的记号列表 (第 113 页)

*\everyvbox 任一垂直盒子开始时, T_EX 展开的记号列表 (第 169 页)

*\exhyphenpenalty 在一显式连字号之后断行的额外惩罚, 默认值为 50 (第 126 页)

\exists “那些存在” 符号: \exists (第 197 页)
\exp 指数函数: \exp (第 203 页)
*\expandafter 只有在展开了下一个记号之后的记号后才展开它
(第 249 页)
*\fam 在数学公式中, TeX 用于第 7 类 (即变量) 字符的字体族
(第 221 页)
*\fi 结束一个条件表达式 (第 255 页)
\filbreak 强制分页, 除非文本达到另一个 \filbreak 时仍适合于本
页 (第 140 页)
*\finalhyphendemerits 对第二到最后一个在连字号处断行的惩罚, 默
认值为 5000 (第 126 页)
*\firstmark 刚处于盒子内的页面的第一个标记项 (第 147 页)
\fivebf 使用 5-pt 粗字体, cmex5 (第 102 页)
\fivei 使用 5-pt 数学斜体, cmmi5 (第 102 页)
\fiverm 使用 5-pt 罗马字体, cmr5 (第 102 页)
\fivesy 使用 5-pt 符号字体, cmsy5 (第 102 页)
\flat 音乐中的降调符号: \flat (第 197 页)
*\floatingpenalty 对跨页分裂插入的惩罚, 默认值为 0 (第 142 页)
\fmtname 当前格式文件的名称 (第 239 页)
\fmtversion 当前格式文件的版本号 (第 239 页)
\folio 以字符输出 \pageno; 如果负值, 则输出罗马数字 (第 146 页)
*\font 定义指定的控制序列用于选择字体 (第 234 页)
*\fontdimen 指定字体的指定参数 (第 235 页)
*\fontname 以字符形式输出指定字体的名称 (第 241 页)
\footline 在每页底部输出线的记号列表 (第 146 页)
\footnote 输出带有指定参考标记的指定脚注 (第 148 页)
\forall “全部” 符号: \forall (第 197 页)
\frenchspacing 使得字间距不依赖于标点符号 (第 106 页)
\frown frown 关系: \sim (第 200 页)
*\futurelet 将其后第三个记号赋给指定的控制序列, 随后展开其后第
二个记号 (第 248 页)
\gamma 数学希腊字母 γ (第 196 页)
\Gamma 数学希腊字母 Γ (第 196 页)
\gcd 最大公分母函数: gcd (第 203 页)
*\gdef 等同于 \global\def, 即定义全局宏 (第 246 页)
\ge 大于或等于关系: \geq (第 200 页)

- \geq 等同于 \ge (第 200 页)
- \gets 获得关系: \leftarrow (第 202 页)
- \gg 远大于关系: \gg (第 200 页)
- *\global 使得随后的定义全局化 (第 243 页)
- *\globaldefs 忽略附加于赋值前面的 \global (第 243 页)
- \goodbreak 表示采用 \penalty-500 的期望的分页 (第 140 页)
- \grave 数学中的抑间符重音符, 比如 \grave{x} 中那样 (第 209 页)
- \H 文本中匈牙利元音变量重音符, 比如 \acute{o} 中那样 (第 100 页)
- *\halign 在列中对齐文本 (第 184 页)
- \hang 当前段落缩进 \parindent (第 117 页)
- *\hangafter 悬挂缩进开始的行数 (第 117 页)
- *\hangindent 悬挂缩进的间隔 (第 117 页)
- \hat 数学中的帽子重音符, 比如 \hat{x} 中那样 (第 209 页)
- *\hbadness 报告未满或过满水平盒子的劣度的阈值, 默认值为 1000
(第 175 页)
- \hbar 数学符号: \hbar (第 197 页)
- *\hbox 产生指定的 hbox (第 164 页)
- \headline 在每页顶部输出线的记号列表 (第 146 页)
- \heartsuit 心形花色符号: \heartsuit (第 197 页)
- *\hfil 输出无限可扩展水平粘连 (第 161 页)
- *\hfill 输出比 \hfil 更加无限扩展的水平粘连 (第 161 页)
- *\hfilneg 输出无限负扩展的水平粘连 (第 162 页)
- *\hfuzz 报告过满 hbox 的间隔阈值, 其默认值为 0.1 点 (第 176 页)
- \hglue 输出在断行时不会消失的水平粘连 (第 160 页)
- \hidewidth 在对齐中忽略一个条目的宽度, 以致于它可以在
\hidewidth 的方向上扩展到盒子外面 (第 191 页)
- *\hoffset 从纸张的左边界相对于一英寸的页面偏移量 (第 143 页)
- *\holdinginserts 如果是正值, 将不从当前页中移去插入 (第 152 页)
- \hom 同调函数: hom (第 203 页)
- \hookleftarrow 关系: \hookleftarrow (第 202 页)
- \hookrightarrow 关系: \hookrightarrow (第 202 页)
- \phantom 产生一个具有自然宽度, 但高度与深度均为零的不可见公式
(第 174 页)
- *\hrule 产生一个水平标线, 只能用于竖直模式中 (第 178 页)
- \rulefill 使用水平标线填充闭合间隔 (第 181 页)
- *\hsize 行的长度, 默认值为 6.5 in (第 113 页)

*\hskip 输出指定的水平粘连 (第 159 页)
*\hss 输出可无限扩展和收缩的水平粘连 (第 162 页)
*\ht 在指定盒子寄存器中盒子的高度 (第 172 页)
*\hyphenation 向断字例外字典中加入指定的词 (第 128 页)
*\hyphenchar 在指定字体中的连字号 (第 129 页)
*\hyphenpenalty 在连字号处断行的额外惩罚, 默认值为 50 (第 126 页)
\i 与重音符一起使用的无点的字母 ‘i’ (第 101 页)
\ialign 开始一个 \tabskip 粘连为零且 \everycr 为空的 \halign
(第 186 页)
*\if 判断两个指定的记号是否具有相同的字符代码 (第 250 页)
*\ifcase 对于指定的数值 n 展开第 n 种情形 (第 255 页)
*\ifcat 判断两个指定的记号是否具有相同的类别码 (第 251 页)
*\ifdim 为了指定的关系在两个指定尺寸之间判断 (第 253 页)
*\ifeof 判断是否处在指定文件的结尾处 (第 254 页)
\iff 当且仅当关系: \iff (第 202 页)
*\iffalse 判断某种情况总是错误的 (第 255 页)
*\ifhbox 判断指定的盒子寄存器是否包含 hbox (第 254 页)
*\ifhmode 判断 TeX 是否处于水平模式中 (第 253 页)
*\ifinner 判断 TeX 是否处于内部模式中 (第 253 页)
*\ifmmode 判断 TeX 是否处于数学模式中 (第 253 页)
*\ifnum 为了指定关系在两个指定数值之间判断 (第 252 页)
*\ifodd 判断指定数值是否是奇数 (第 252 页)
*\iftrue 判断某种情况总是正确的 (第 255 页)
*\ifvbox 判断指定的盒子寄存器是否包含 vbox (第 254 页)
*\ifvmode 判断 TeX 是否处于竖直模式中 (第 253 页)
*\ifvoid 判断指定盒子寄存器是否是无效的 (第 254 页)
*\ifx 判断两个记号是否是相同的, 或两个宏是否具有相同的顶级定义
(第 251 页)
*\ignorespaces 忽略任何接随其后的间隔记号 (第 268 页)
\Im 复数虚部符号: \Im (第 197 页)
\imath 与数学重间符一起使用的无点的字符 ‘i’ (第 197 页)
*\immediate 不作延迟地进行特定的文件操作 (第 266 页)
\in 属于关系: \in (第 200 页)
*\indent 输出一个宽度为 \parindent 的空盒子, 并进入水平模式
(第 111 页)
\inf 下限函数: inf (第 203 页)

\infy 无限符号: ∞ (第 197 页)

*\input 从指定文件开始读入 (第 263 页)

*\inputlineno 当前输入文件的当前行号 (第 264 页)

*\insert 输出指定类别的插入 (第 150 页)

*\insertpenalties 因为插入造成的惩罚之和 (第 142 页)

\int 积分符号: \int (第 204 页)

*\interlinepenalty 段落的行间分页时的额外惩罚, 默认值为 0
(第 141 页)

\iota 数学希腊字母 ι (第 196 页)

\it 使用斜体, 即执行 \tenit\fam=\itfam (第 103 页)

\item 开始一个段落, 采用 \parindent 悬挂缩进, 且其前放置指定的
标签 (第 131 页)

\itemitem 与 \item 一样, 但缩进为 2\parindent (第 131 页)

\itfam 数学中的斜体字族 (第 221 页)

\j 与重音符一起使用的无点字符 ‘j’ (第 101 页)

\jmath 与数学重音符一起使用的无点字符 ‘j’ (第 197 页)

*\jobname 引发 TeX 执行的文件的基本名称 (第 240 页)

\jot 用于分开陈列公式的度量单位 (第 226 页)

\kappa 数学希腊字母 κ (第 196 页)

\ker 核函数: ker (第 203 页)

*\kern 输出指定量的间隔, 在此位置不允许断行 (第 160 页)

\l 波兰字符: l (第 97 页)

\L 波兰字符: L (第 97 页)

\lambda 数学希腊字母 λ (第 196 页)

\Lambda 数学希腊字母 Λ (第 196 页)

\land 逻辑“和”运算符: \wedge (第 198 页)

\langle 左角度分界符: < (第 201 页)

*\language 断字模式的当前设置 (第 129 页)

*\lastbox 如果是一个盒子, 获取并移除当前列表中的最后一项
(第 177 页)

*\lastkern 如果是一个紧排, 获取当前列表中的最后一项 (第 177 页)

*\lastpenalty 如果是一个惩罚, 获取当前列表中的最后一项
(第 177 页)

*\lastskip 如果是一个粘连, 获取当前列表中的最后一项 (第 177 页)

\lbrace 左大括号定界符: { (第 201 页)

\lbrack 左方括号定界符: [(第 201 页)

- *`\lccode` 小写字母的字符代码 (第 103 页)
- `\lceil` 左上限定界符: 「 (第 201 页)
- `\ldotp` 作为标点符号, 位于基线上的点: . (第 206 页)
- `\ldots` 数学中位于基线上的多点: ... (第 213 页)
- `\le` 小于或等于关系: \leq (第 200 页)
- *`\leaders` 使用重复的指定的盒子或标线对指定水平或竖直间隔填充 (第 179 页)
- *`\left.` 输出大小覆盖随后的子公式, 由 `\right.` 结束的指定的分界符 (第 214 页)
- `\leftarrow` 关系: \leftarrow (第 202 页)
- `\Leftarrow` 关系: \Leftarrow (第 202 页)
- `\leftarrowfill` 使用 `\leftarrow` 填充闭合的 hbox: $\leftarrow\cdots\rightarrow$ (第 181 页)
- `\leftharpoondown` 关系: \leftharpoondown (第 202 页)
- `\leftharpoonup` 关系: \leftharpoonup (第 202 页)
- *`\lefthyphenmin` TeX 允许在一个词开始处插入连字号的最小词片段的大小, 默认值为 2 (第 129 页)
- `\leftline` 输出文字靠左边界行 (第 108 页)
- `\leftrightarrow` 关系: \leftrightarrow (第 202 页)
- `\Leftrightarrow` 关系: \Leftrightarrow (第 202 页)
- *`\leftskip` TeX 在每行左侧插入的粘连 (第 115 页)
- `\leq` 等同于 `\le` (第 200 页)
- `\leqalignno` 生成公式编号在左侧, 且指定部分竖直对齐的多行陈列公式 (第 218 页)
- *`\leqno` 在陈列公式的左侧放置指定的公式编号 (第 217 页)
- *`\let` 定义控制序列为随后的记号 (第 247 页)
- `\lfloor` 左下限分界符: \lfloor (第 201 页)
- `\lg` 对数函数: \lg (第 203 页)
- `\lgroup` 左编组定界符 (这里显示的是最小尺寸的): $($ (第 214 页)
- `\lim` 极限函数: \lim (第 203 页)
- `\liminf` 下极限函数: \liminf (第 203 页)
- *`\limits` 在一个大的运算符的上下放置上标和下标 (第 204 页)
- `\limsup` 上极限函数: \limsup (第 203 页)
- `\line` 输出两端对齐的行 (第 109 页)
- *`\linepenalty` 对加到每一行的断行的惩罚, 默认值为 10 (第 125 页)

- *\lineskip 如果行之间的距离小于 \lineskip limit 时, 从一个基线到另一个基线的竖直粘连, 默认值为 1 点 (第 136 页)
- *\lineskip limit 采用 \lineskip 而不是 \baselineskip 的阈值, 默认值为 0 点 (第 136 页)
- \ll 小于关系: << (第 200 页)
- \l l l a p 输出向当前位置左侧伸展的 (没有宽度的) 文本 (第 109 页)
- \l m o u s t a c h e 大的大括号的上半部分: \int (第 223 页)
- \l n 自然对数函数: ln (第 203 页)
- \l n o t 逻辑 “非” 符号: \neg (第 197 页)
- \l o g 对数函数: log (第 203 页)
- *\l o n g 在随后的定义中允许在参数中使用 \par 记号 (第 246 页)
 - \l o n g l e f t a r r o w 关系: \leftarrow (第 202 页)
 - \L o n g l e f t a r r o w 关系: \Leftarrow (第 202 页)
 - \l o n g l e f t r i g h t a r r o w 关系: \longleftrightarrow (第 202 页)
 - \L o n g l e f t r i g h t a r r o w 关系: \Longleftrightarrow (第 202 页)
 - \l o n g m a p s t o 关系: \mapsto (第 202 页)
 - \l o n g r i g h t a r r o w 关系: \rightarrow (第 202 页)
 - \L o n g r i g h t a r r o w 关系: \Rightarrow (第 202 页)
- \l o o p 开始一个由 \repeat 结束的循环 (第 256 页)
- *\l o o s e n e s s 一个段落你希望的行数与最优值之间的差异 (第 124 页)
- \l o r 逻辑 “或” 运算符: \vee (第 198 页)
- *\l o w e r 降低指定盒子指定的数值 (第 171 页)
- *\l o w e r c a s e 将指定文字中的大写字母转化为小写字母 (第 104 页)
- \l q 文本中的左引号字符: ‘ (第 98 页)
- *\l m a g 放大所有尺寸的比例 1000 倍 (第 237 页)
 - \m a g n i f i c a t i o n 与 \mag 一样, 但不放大页面尺寸 (第 237 页)
 - \m a g s t e p 对于指定的 n 得到 $1000 \cdot 1.2^n$ (第 237 页)
 - \m a g s t e p h a l f $1000 \cdot \sqrt{1.2}$ (第 238 页)
- \l m a p s t o 关系: \mapsto (第 202 页)
- *\l m a r k 产生采用指定文的标记项目 (第 147 页)
- *\l m a t h a c c e n t 在接紧的字符上放置指定的数学重音符 (第 209 页)
- *\l m a t h b i n 像二元运算符那样分隔指定的子公式 (第 230 页)
- *\l m a t h c h a r 输出带有指定数学代码的数学字符 (第 99 页)
- *\l m a t h c h a r d e f 定义指定的控制序列, 使之位于 0 至 $2^{15} - 1$ 之间的数学代码 (第 247 页)

*`\mathchoice` 根据当前的样式从四个指定的数学子公式中选择一个
(第 207 页)
`*\mathclose` 像闭合分界符那样分隔指定的子公式 (第 230 页)
`*\mathcode` 指定字符的数学代码 (第 267 页)
`*\mathinner` 像行间分式那样分隔指定的子公式 (第 230 页)
`*\mathop` 像大数学运算符那样分隔指定的子公式 (第 230 页)
`*\mathopen` 像开放分界符那样分隔指定的子公式 (第 230 页)
`*\mathord` 像普通字符那样分隔指定子公式 (第 230 页)
`\mathpalette` 输出根据当前样式展开指定的控制序列的 `\mathchoice`
(第 208 页)
`*\mathpunct` 像标点符号那样分隔指定子公式 (第 230 页)
`*\mathrel` 像关系那样分隔指定子公式 (第 230 页)
`\mathstrut` 输出深度与高度与左圆括号一致, 但宽度为零的不可见盒
子 (第 173 页)
`*\mathsurround` 文本中, 在数学公式前后分隔 TeX 紧排 (第 229 页)
`\matrix` 输出指定矩阵 (第 216 页)
`\max` 最大化函数: `max` (第 203 页)
`*\maxdeadcycles` TeX 给出警告, 并使用其自身的输出程序
 `\deadcycles` 的值默认值为 25 (第 152 页)
`*\maxdepth` 页面上底边盒子的最大深度, 默认值为 4 点 (第 144 页)
`\maxdimen` TeX 可接受的最大尺寸 (第 259 页)
`*\meaning` 以字符形式输出人类能理解的指定记号的意义 (第 240 页)
`\medbreak` 指定使用 `\penalty-100` 进行期望分页, 并输出
 `\medskipamount` 粘连 (第 140 页)
`*\medmuskip` 中等数学间隔的粘连, 默认值为 4 mu plus 2 mu minus
 4 mu (第 226 页)
`\medskip` 输出 `\medskipamount` 粘连 (第 158 页)
`\medskipamount` 中等竖直间隔的粘连, 默认值为 6 点 plus 2 点 minus
 2 点 (第 158 页)
`*\message` 展开并显示指定文本到终端上 (第 278 页)
`\mid` 中项关系: `|` (第 200 页)
`\midinsert` 如果可以有, 在当前位置输出指定文本, 否则在下一页的
 顶部 (第 149 页)
`\min` 最小化函数: `min` (第 203 页)
`\mit` 使用数学斜体, 即执行 `\fam=1` (第 220 页)
`*\mkern` 数学中以 mu 为单位输出指定的紧排 (第 227 页)
`\models` 模关系: `\models` (第 200 页)

- *\month 数字形式的当前月份 (第 239 页)
- *\moveleft 将指定盒子向左移动指定间隔; 只有竖直模式中有效
(第 171 页)
- *\moveright 将指定盒子向右移动指定间隔; 只有竖直模式中有效
(第 171 页)
- \mp 减号与加号运算符: \mp (第 198 页)
- *\mskip 数学中以 mu 为单位输出指定的粘连 (第 227 页)
\mu 数学希腊字母 μ (第 196 页)
- *\multiply 将指定 \count 寄存器乘以指定整数 (第 262 页)
\multispan 使得紧接着的对齐条目跨越指定数目的列 (或行)
(第 188 页)
- *\muskip 指定的数学粘连寄存器 (第 258 页)
- *\muskipdef 定义指定的控制序列为一个与 \muskip 寄存器对应的数值
(第 261 页)
- \nabla 相反的微分符号: ∇ (第 197 页)
- \narrower 使得左右边界都变窄 \parindent (第 114 页)
- \natural 音乐中的本位符号: \natural (第 197 页)
- \nearrow 东北箭头关系: \nearrow (第 202 页)
- \ne 不等于关系: \neq (第 200 页)
- \neg 逻辑“非”符号: \neg (第 197 页)
- \negthinspace $-1/6\text{em}$ 紧排 (第 157 页)
- \neq 不等于关系: \neq (第 200 页)
- \newbox 保留并命名 \box 寄存器 (第 260 页)
- \newcount 保留并命名 \count 寄存器 (第 260 页)
- \newdimen 保留并命名 \dimen 寄存器 (第 260 页)
- \newfam 保留并命名数学字体族 (第 260 页)
- \newhelp 命名指定的帮助信息 (第 279 页)
- \newif 用指定名称定义新的条件式 (第 256 页)
- \newinsert 命名插入类, 并保留相应的 \box、\count、\dimen 和
\skip 寄存器 (第 260 页)
- \newlanguage 保留并命名 \language (第 260 页)
- *\newlinechar 给 \write 等的行结束字符 (第 267 页)
- \newmuskip 保留并命名 \muskip 寄存器 (第 260 页)
- \newread 保留并命名输入流 (第 260 页)
- \newskip 保留并命名 \skip 寄存器 (第 260 页)
- \newtoks 保留并命名 \toks 寄存器 (第 260 页)

\newwrite 保留并命名输出流 (第 260 页)
\ni “反向属于”关系: \ni (第 200 页)
*\noalign 在对齐的行 (或列) 插入其它内容 (第 189 页)
*\noboundary 禁止使用当前字体的边界字符进行连字或紧排 (第 101 页)
\nobreak 执行 \penalty10000, 即禁止断行或分页 (第 121 页,
第 139 页)
*\noexpand 抑制下一个记号的展开 (第 249 页)
*\noindent 进入不缩进段落的水平模式 (第 111 页)
\nointerlineskip 在下一行前禁止行内粘连 (第 138 页)
*\nolimits 将上下标放在大的运算符之后 (第 205 页)
\nonfrenchspacing 使字间间隔依赖于标点符号 (第 106 页)
*\nonscript 在上下标或次上下标样式中禁止任何随后的粘连或紧排
(第 227 页)
*\nonstopmode 不在任何错误处停下, 即使是有关缺失文件的错误
(第 269 页)
\nopagenumbers 禁止打印出页码, 即执行 \footline = \hfil
(第 145 页)
\normalbaselines 将 \baselineskip、\lineskip 和 \lineskip-
limit 设为当前字体大小的正常值 (第 137 页)
\normalbaselineskip 针对当前字体大小的 \baselineskip 值
(第 137 页)
\normalbottom 使得底边距从一页到另一页是相同的 (第 140 页)
\normallineskip 针对当前字体大小的 \lineskip 值 (第 137 页)
\normallineskiplimit 针对当前字体大小的 \lineskiplimit 值
(第 137 页)
\not 宽度为零的斜杠, 用于构建数学关系的否定形式, 比如 \neq 中那样
(第 200 页)
\notin 非包含关系: \notin (第 200 页)
\nu 数学希腊字母 ν (第 196 页)
\null 展开为空的 hbox (第 175 页)
*\nulldelimiterspace 由空的定界符产生的间隔, 默认值为 1.2 点
(第 229 页)
*\nulldfont 没有字符的原始字体 (第 102 页)
*\number 以字符形式输出指定的数值 (第 238 页)
\nwarrow 西北箭头关系: \nwarrow (第 202 页)
\o 丹麦语字母: \o (第 97 页)
\O 丹麦语字母: \O (第 97 页)

\obeylines 使得输入文件中的每个行结束符号等同于 \par (第 122 页)

\obeyspaces 对于输入中的每个间隔字符在输出文件中输出间隔
(第 107 页)

\odot 居中的点运算符: \odot (第 198 页)

\oe \oe 连字 (第 97 页)

\OE \OE 连字 (第 97 页)

\offinterlineskip 从现在开始禁止行间粘连 (第 137 页)

\oint 围线积分运算符: \oint (第 204 页)

\oldstyle 使用古体数字: 1234567890 (第 220 页)

\omega 数学希腊字母 ω (第 196 页)

\Omega 数学希腊字母 Ω (第 196 页)

\ominus 圈减号运算符: \ominus (第 198 页)

*\omit 在对齐中忽略列的(或行的)模板 (第 187 页)

*\openin 准备指定的输入流以从文件中读入 (第 264 页)

*\openout 准备指定的输出流以向文件中写入 (第 265 页)

\openup 使 \baselineskip、\lineskip 和 \lineskiplimit 增加指定的量 (第 138 页)

\oplus 圈加号运算符: \oplus (第 198 页)

*\or 用于分隔 \ifcase 的情形 (第 255 页)

\oslash 圈斜线运算符: \oslash (第 198 页)

\otimes 圈乘号运算符: \otimes (第 198 页)

*\outer 使得随后的宏定义在以高速读入的记号情形下无效 (第 246 页)

*\output 当遇到分页时 TeX 展开的记号列表 (第 151 页)

*\outputpenalty 如果分页发生于惩罚处, 则为惩罚的值, 否则为零
(第 152 页)

*\over 输出一个带有默认厚度横线的分式 (第 210 页)

\overbrace 输出一个覆盖于公式顶部的大括号, 比如 $\overbrace{h+w}$ 中那样
(第 212 页)

*\overfullrule 附加于过满盒子的标线的宽度 (第 175 页)

\overleftarrow 输出一个覆盖于公式顶部的向左的箭头, 比如 $\overleftarrow{r+a}$
中那样 (第 212 页)

*\overline 输出一个覆盖于公式顶部的直线, 比如 $\overline{2b}$ 中那样
(第 212 页)

\overrightarrow 输出一个覆盖于公式顶部的向右的箭头, 比如 $\overrightarrow{i+t}$
中那样 (第 212 页)

- *\overwithdelims 输出一个带有默认厚度横线，且被指定分界符包围的分式 (第 211 页)
- \owns 固有关系: \ni (第 200 页)
- \P 文本中的段落字符: \P (第 98 页)
- *\pagedepth TeX 将其设置为当前页的尖前深度 (第 142 页)
- *\pagefillstretch TeX 将其设置为在当前页的 `filll` 扩展量 (第 143 页)
- *\pagefillstretch TeX 将其设置为在当前页的 `fill` 扩展量 (第 143 页)
- *\pagefilstretch TeX 将其设置为在当前页的 `fil` 扩展量 (第 143 页)
- *\pagegoal TeX 将其设置为当前页的期望高度 (即 \vsize, 当第一个盒被放在该页时) (第 142 页)
- \pageinsert 在随后的页面输出指定的文本，并用完整页 (第 149 页)
- \pageno \count0 寄存器，它包括（可能是负的）页码 (第 145 页)
- *\pageshrink TeX 将其设置为在当前页面总的可收缩量 (第 142 页)
- *\pagestretch TeX 将其设置为在当前页面总的可伸展量 (第 143 页)
- *\pagetotal TeX 将其设置为当前页的自然高度 (第 142 页)
- *\par 结束段落并结束水平模式 (第 110 页)
- \parallel 平行关系: \parallel (第 200 页)
- *\parfillskip TeX 在段落结束处插入的水平粘连 (第 111 页)
- *\parindent TeX 在段落开始处插入的水平间距 (第 112 页)
- *\parshape 指定下一段落中每行的宽度与长度 (第 118 页)
- *\parskip TeX 在段落前插入的竖直粘连 (第 144 页)
- \partial 偏微分符号: ∂ (第 197 页)
- *\pausing 如果是正值，在读入每一行后停下来以便进行可能的替换 (第 269 页)
- *\penalty 输出在此处断行或分页的惩罚（或奖励，如果是负值） (第 122 页, 第 139 页)
- \perp 垂直关系: \perp (第 200 页)
- \phantom 输出指定子公式大小的不可见公式 (第 173 页)
- \phi 数学希腊字母 ϕ (第 196 页)
- \Phi 数学希腊字母 Φ (第 196 页)
- \pi 数学希腊字母 π (第 196 页)
- \Pi 数学希腊字母 Π (第 196 页)
- \plainoutput plain TeX 的 \output 程序 (第 151 页)
- \pm 加减运算符: \pm (第 198 页)

- \pmatrix 输出带圆括号的矩阵 (第 216 页)
- \pmod 放置于公式结尾处的带圆括号的模数标记, 比如 $x \equiv y + 1 \pmod{2}$ 中那样 (第 203 页)
- *\postdisplaypenalty 恰好在显示陈列公式之后的断行的额外惩罚, 其默认值为 0 (第 141 页)
- \Pr 概率函数: \Pr (第 203 页)
- \prec 优先关系: \prec (第 200 页)
- \preceq 优先或等于关系: \preceq (第 200 页)
- *\predisplaypenalty 恰好在陈列公式之前断行的额外惩罚, 其默认值为 0 (第 141 页)
- *\predisplaysize TeX 将其设置为显示陈列公式之前的行的宽度 (第 228 页)
- *\pretolerance 不断字断行时劣度的容许值, 默认值为 100 (第 123 页)
- *\prevdepth 在当前竖直列表中最后一个非标线盒子的深度 (第 137 页)
- *\prevgraf TeX 将其设置为当前段落 (水平模式下) 或前一段落 (竖直模式下) 的行数 (第 120 页)
- \prime 质数数学符号, 比如 r' 中那样 (第 197 页)
- \proclaim 开始定理、引理、假设 … (第 131 页)
- \prod 大的乘积运算符: \prod (第 204 页)
- \propto 比例关系: \propto (第 200 页)
- \psi 数学希腊字母 ψ (第 196 页)
- \Psi 数学希腊字母 Ψ (第 196 页)
- \qquad 输出宽度为 2em 的水平粘连 (第 157 页)
- \quad 输出宽度为 1em 的水平粘连 (第 157 页)
- *\radical 输出指定根号符号 (第 217 页)
- \raggedbottom 允许底边距在页与页间变化 (第 140 页)
- \raggedright 允许右边距在行与行间变化 (第 116 页)
- *\raise 将指定盒子升高指定的量 (第 171 页)
- \rangle 右角度分界符: \rangle (第 201 页)
- \rbrace 右大括号分界符 $\}$ (第 201 页)
- \rbrack 右方括号分界符: $\]$ (第 201 页)
- \rceil 右上限分界符: \lceil (第 201 页)
- \Re 复数实数部分符号: \Re (第 197 页)
- *\read 从指定输入流中读入行 (第 264 页)
- *\relax 什么也不做 (第 257 页)

- *\relpenalty 在关系后的断行或分页的额外惩罚, 默认值为 500
(第 126 页)
- \repeat 结束一个以 \loop 开始的循环 (第 256 页)
- \rfloor 右下限分界符:] (第 201 页)
- \rgroup 右编组定界符 (这里显示的是最小尺寸的): } (第 214 页)
- \rho 数学希腊字母 ρ (第 196 页)
- *\right 在子公式右侧输出由 \left 开始的指定的分界符 (第 214 页)
- \rightarrow 简单关系: \rightarrow (第 202 页)
- \Rightarrow 复杂关系: \Rightarrow (第 202 页)
- \rightarrowfill 使用 \rightarrow 填充所在的水平盒子: \longrightarrow
(第 181 页)
- \rightharpoondown 关系: \rightarrowtail (第 202 页)
- \rightharpoonup 关系: \rightarrowdot (第 202 页)
- \rightleftharpoons 关系: \rightleftharpoons (第 202 页)
- \rightline 输出文字靠右边界的行 (第 108 页)
- *\rightskip TeX 在每行右侧插入的粘连 (第 115 页)
- *\righthyphenmin TeX 允许的单词结尾在连字号之后的最小片断的大
小, 默认值为 3 (第 129 页)
- \rlap 在当前位置的右侧输出 (没有宽度的) 文本 (第 109 页)
- \rm 使用罗马体, 即很执行 \tenrm\fam=0 (第 103 页)
- \rmoustache 大的大括号的下半部分: } (第 223 页)
- \romannumeral 以字符的形式输出指定数值的小写罗马数值表示
(第 238 页)
- \root 输出一指定子公式的指定根, 比如 $\sqrt[3]{2}$ 中那样 (第 217 页)
- \rq 文本中的右引号: ' (第 98 页)
- \S 文本中的小节字符: § (第 98 页)
- \sb 隐式下标字符 (第 206 页)
- *\scriptfont 在指定数学字体族中的上下标样式字体 (第 221 页)
- *\scriptscriptfont 在指定数学字体族中的次上下标样式字体
(第 221 页)
- *\scriptscriptstyle 在公式中使用次上下标样式尺寸 (第 207 页)
- *\scriptspace TeX 在下标或上标之后加入的额外的紧排, 默认值为
0.5 点 (第 229 页)
- *\scriptstyle 在公式中使用上下标样式尺寸 (第 207 页)
- *\scrollmode 不在任何错误处停下, 但在缺失文件错误处停下
(第 269 页)

\searrow 东南箭头关系: \searrow (第 202 页)

\sec 正切函数: sec (第 203 页)

*\setbox 定义指定的盒子寄存器使之成为盒子 (第 169 页)

*\setlanguage 改变到断字规则的指定集合, 但不改变 \language (第 129 页)

\setminus 差集运算符: \setminus (第 198 页)

\settabs 为制表符对齐定义制表符 (第 182 页)

\sevenbf 使用 7-pt 粗字体, cmbx7 (第 102 页)

\seveni 使用 7-pt 数学斜体, cmmi7 (第 102 页)

\sevenrm 使用 7-pt 罗马字体, cmr7 (第 102 页)

\sevensy 使用 7-pt 符号字体, cmsy7 (第 102 页)

*\sfcode 指定字符的间隔系数 (第 107 页)

\sharp 音乐中的升半间符: \sharp (第 197 页)

*\shipout 向 .dvi 文件输出盒子 (第 151 页)

*\show 在日志文件中和终端上显示指定记号的意义 (第 270 页)

*\showbox 显示指定盒子寄存器的内容 (第 270 页)

*\showboxbreadth 在每个嵌套层次上显示的项目的最大数量, 默认值为 5 (第 278 页)

*\showboxdepth 显示的最大嵌套层次, 默认值为 3 (第 278 页)

\showhyphens 在日志文件中和终端上显示指定文本的断字 (第 128 页)

*\showlists 显示所有正被影响的列表 (第 270 页)

*\showthe 在日志文件中和终端上显示 \the 将要输出的内容 (第 270 页)

\sigma 数学希腊字母 σ (第 196 页)

\Sigma 数学希腊字母 Σ (第 196 页)

\sim 相似关系: \sim (第 200 页)

\simeq 相似或等于关系: \simeq (第 200 页)

\sin 正弦函数: sin (第 203 页)

\sinh 双曲正弦函数: sinh (第 203 页)

\skew 将指定字符的重音符偏移指定的量 (第 224 页)

*\skewchar 指定字体中用于定位重音符的字符 (第 224 页)

*\skip 指定的粘连寄存器 (第 258 页)

*\skipdef 定义指定的控制序列使之成为与 \skip 寄存器对应的数值 (第 261 页)

\sl 使用倾斜的字体, 即执行 \tensl\fam=\slfam (第 103 页)

\slash 允许断行的 / 字符 (第 123 页)

\sffam 数学中的倾斜字体族 (第 221 页)

\smallbreak 指示一些期望 \penalty-50 分页，并输出
 \smallskipamount 粘连 (第 140 页)
\smallint 小的积分符号: \int (第 204 页)
\smallskip 输出 \smallskipamount 粘连 (第 158 页)
\smallskipamount 小的竖直间距的粘连，默认值为 3 点 plus 1 点
 minus 1 点 (第 158 页)
\smash 输出高度与浓度均为零的公式 (第 174 页)
\smile 微笑关系: \smile (第 200 页)
\sp 隐式上标字符 (第 206 页)
\space 输出正常的空间粘连 (第 104 页)
*\spacefactor 修改字间粘连的伸展量与收缩量，如果不是 1000
 (第 107 页)
*\spaceskip 如果非零且 \spacefactor < 2000，抑制正常的字间粘连
 (第 107 页)
\spadesuit 黑桃花色符号: \spadesuit (第 197 页)
*\span 在对齐主体中合并条目或在导言中展开记号 (第 188 页)
*\special 将记号写入 .dvi 文件，以便 DVI 浏览程序解释 (第 266 页)
*\splitbotmark 盒子中来自于 \vsplit 的最后一个标记项目
 (第 147 页)
*\splitfirstmark 盒子中来自于 \vsplit 的第一个标记项目
 (第 147 页)
*\splitmaxdepth 来自于 \vsplit 的盒子的最大深度 (第 153 页)
*\splittopskip TeX 在来自于 \vsplit 的盒子的顶部插入的粘连
 (第 153 页)
\sqcap 方形求交运算符: \sqcap (第 198 页)
\sqcup 方形求并运算符: \sqcup (第 198 页)
\sqrt 输出字公式的平方根，比如 $\sqrt{2}$ 中那样 (第 217 页)
\sqsubseteq 方形子集或等于关系: \sqsubseteq (第 200 页)
\sqsupseteq 方形超集或等于关系: \sqsupseteq (第 200 页)
\ss 德语字符: \ss (第 97 页)
\star 星形运算符: \star (第 198 页)
*\string 以字符输出指定的记号，通常为控制序列 (第 240 页)
\strut 在当前字体中，从基线到基线，宽度为零，但高度与深度为标
 准行的盒子 (第 172 页)
\subset 子集关系: \subset (第 200 页)
\subseteq 子集或相等关系: \subseteq (第 200 页)
\succ 后继关系: \succ (第 200 页)

\succeq 后继或相等关系: \succeq (第 200 页)

\sum 大的求和运算符: \sum (第 204 页)

\sup 上限函数: \sup (第 203 页)

\supereject 强制分页, 并输出所有的插入 (第 140 页)

\supset 超集关系: \supset (第 200 页)

\supseteq 超集或相等关系: \supseteq (第 200 页)

\surd 无理数符号: \surd (第 197 页)

\swarrow 西南箭头关系: \swarrow (第 202 页)

\t 文本中的后向连接线重音符 \tilde{u} (第 100 页)

\tabalign 等同于 \+, 除了它不是 \outer (第 182 页)

*\tabskip 对齐中列 (或行) 之间的粘连 (第 190 页)

\tan 正切函数: \tan (第 203 页)

\tanh 双曲正切函数: \tanh (第 203 页)

\tau 数学希腊字母 τ (第 196 页)

\tenbf 使用 10-pt 粗字体, cmbx10 (第 102 页)

\tenex 使用 10-pt 数学扩展字体, cmex10 (第 102 页)

\teni 使用 10-pt 数学斜体, cmmi10 (第 102 页)

\tenit 使用 10-pt 文本斜体, cmti10 (第 102 页)

\tenrm 使用 10-pt 罗马字体, cmr10 (第 102 页)

\tensl 使用 10-pt 倾斜罗马字体, cmsl10 (第 102 页)

\tensy 使用 10-pt 数学符号字体, cmsy10 (第 102 页)

\tentt 使用 10-pt 打字机字体, cmtt10 (第 102 页)

\TeX 输出 TeX 标志 (第 99 页)

*\textfont 在指定数学字体家族中的文本样式 (第 221 页)

\textindent 与 \item 一样, 但不进行悬挂缩进 (第 112 页)

*\textstyle 在公式中使用文本样式尺寸 (第 207 页)

*\the 给出指定记号的值 (第 249 页)

\theta 数学希腊字母 θ (第 196 页)

\Theta 数学希腊字母 Θ (第 196 页)

*\thickmuskip 较大数学间隔所有的粘连, 默认值为 5mu plus 5mu
(第 226 页)

*\thinmuskip 较小数学间隔所有的粘连, 默认值为 3mu (第 226 页)

\thinspace $\frac{1}{6}$ em 紧排 (第 156 页)

\tilde 数学中的否定号重音符, 比如 \tilde{x} 中那样 (第 209 页)

*\time 当天的时间, 从子夜来自的分钟数 (第 239 页)

\times 乘号运算符: \times (第 198 页)
*\toks 指定的记号寄存器 (第 258 页)
*\toksdef 定一指定的控制序列使之成为与 \toks 寄存器相对应的数值
(第 261 页)
*\tolerance 带断字断行时劣度的容许值 (第 123 页)
\to 映射关系: \rightarrow (第 202 页)
\top lattice top symbol: \top (第 197 页)
\topglue 在页面的顶部输出指定的竖直粘连 (第 160 页)
\topinsert 在页面顶部输出指定的文本 (第 149 页)
*\topmark 放入盒子中的在当前页之前的 \botmark (第 147 页)
*\topskip 页眉线与同页上第一行文本之间的粘连, 默认值为 10 点
(第 144 页)
\tracingall 打开最大跟踪 (第 278 页)
*\tracingcommands 显示命令的执行 (第 273 页)
*\tracinglostchars 显示被请求的字符, 但不是被定义的 (第 274 页)
*\tracingmacros 显示宏展开 (第 275 页)
*\tracingonline 在终端及日志文件中显示诊断输出 (第 273 页)
*\tracingoutput 显示输出的盒子的内容 (第 275 页)
*\tracingpages 显示分页计算 (第 276 页)
*\tracingparagraphs 显示断行计算 (第 277 页)
*\tracingrestores 显示在编组结束处保存的值 (第 277 页)
*\tracingstats 显示内存使用统计 (第 277 页)
\triangle 三角符号: \triangle (第 197 页)
\triangleleft 左三角运算符: \triangleleft (第 198 页)
\triangleright 右三角运算符: \triangleright (第 198 页)
\tt 使用打字机字体, 即执行 \tentt\fam=\ttfam (第 103 页)
\ttfam 数学中的打字机字体族 (第 221 页)
\ttraggedright 使用打字机字体, 并允许段落右边界从一行到另一行
不同 (第 116 页)
\u 文本中的短音重音符, 比如 \check{r} 中那样 (第 100 页)
*\uccode 一个字母的大小字母的字符代码 (第 103 页)
*\uchyph 如果是正值, 考虑对大写字母开始的词断词 (第 128 页)
\underbar 不用避免任何下行字母的情况下对指定文本划下划线, 比如
 \underline{fog} 中那样 (第 169 页)
\underbrace 输出一个覆盖公式底部的大括号, 比如 $\underbrace{x+x}$ 中那样
(第 212 页)

- *\underline 在下行字母之下对数学公式划下划线，比如 $\underline{x+y}$ 中那样
(第 212 页)
- *\unhbox 将指定盒子寄存器中盒子的内容随加到当前列表中，并使寄存器失效；只在水平模式中有效。(第 170 页)
- *\unhcopy 与 \unhbox 一样，但不使寄存器失效 (第 170 页)
- *\unkern 如果当前列表的最后一项是紧排，删除它 (第 177 页)
- *\unpenalty 如果当前列表的最后一项是惩罚，删除它 (第 177 页)
- *\unskip 如果当前列表的最后一项是粘连，删除它 (第 177 页)
- *\unvbox 将指定盒子寄存器中盒子的内容随加到当前列表中，并使寄存器失效；只在竖直模式中有效。(第 170 页)
- *\unvcopy 与 \unvbox 一样，但不使寄存器失效 (第 170 页)
- \uparrow 箭头：↑(第 202 页)
- \Uparrow 箭头：↑↑(第 202 页)
- \upbracefill 使用向上开口的大括号填充闭合的 hbox: 
- (第 223 页)
- \updownarrow 箭头：↑↓(第 202 页)
- \Updownarrow 箭头：↑↓↑↓(第 202 页)
- \uplus 并加号运算符：⊕ (第 198 页)
- *\uppercase 将指定文本中的小写字母转换为大写字母 (第 104 页)
- \upsilon 数学希腊字母 ν (第 196 页)
- \Upsilon 数学希腊字母 Υ (第 196 页)
- \v 文本中的抑制重音符，比如 ö 中那样 (第 100 页)
- *\vadjust 在当前行后输出竖直模式内容 (第 120 页)
- *\valign 在行中对齐文本 (第 185 页)
- \varepsilonpsilon 异体的数学希腊字母 ε (第 196 页)
- \varphi 异体的数学希腊字母 φ (第 196 页)
- \varpi 异体的数学希腊字母 ϖ (第 196 页)
- \varrho 异体的数学希腊字母 ϱ (第 196 页)
- \varsigma 异体的数学希腊字母 ς (第 196 页)
- \vartheta 异体的数学希腊字母 ϑ (第 196 页)
- *\vbadness 报告未满或过满竖直盒子使用的劣度阈值，默认值为 1000
(第 175 页)
- *\vbox 输出竖直盒子，其基线是闭合的底下盒子的基线 (第 166 页)
- *\vcenter 在数学轴上居中指定文本 (第 225 页)
- \vdash 左十字转门符号：⊤ (第 200 页)
- \vdots 数学的竖直多点：⋮ (第 213 页)

\vec 数学中的矢量重音符, 比如 \vec{x} 中那样 (第 209 页)
\vee 逻辑“或”运算符: \vee (第 198 页)
\vert 横线关系: $|$ (第 197 页)
\Vert 双横线关系: \parallel (第 197 页)
*\vfil 输出无限可扩展的竖直粘连 (第 161 页)
*\vfill 输出比 \vfil 更加无限可扩展的竖直粘连 (第 161 页)
*\vfilneg 输出无限负向可扩展的竖直粘连 (第 162 页)
\vfootnote 输出带有指定参考标记的指定脚注, 但在文本中不输出参考标记 (第 148 页)
*\vfuzz 报告过满 vbox 的间隔阈值, 默认值为 0.1 点 (第 176 页)
\vglue 输出在分页时不消失的指定的竖直粘连 (第 160 页)
*\voffset 从页面顶部开始的相对于一英寸的竖直偏移 (第 143 页)
\vphantom 输出宽度为零, 但具有自然高度与深度的不可见公式 (第 174 页)
*\vrule 输出竖直标线; 只在水平模式下有效 (第 178 页)
*\vsize 页面高度, 默认值为 8.9 in (第 143 页)
*\vskip 输出指定的竖直粘连 (第 159 页)
*\vsplit 将指定的盒子寄存器的内容分隔为指定的高度 (第 152 页)
*\vss 输出可无限扩展与收缩的竖直粘连 (第 162 页)
*\vtop 输出竖直盒子, 它的基线是闭合的顶部盒子的基线 (第 166 页)
*\wd 指定盒子寄存器中盒子的宽度 (第 172 页)
\wedge 逻辑“和”运算符: \wedge (第 198 页)
\widehat 数学重音符, 比如 $\widehat{y+z+a}$ 中那样 (第 209 页)
\widetilde 数学重音符, 比如 $\widetilde{b+c+d}$ 中那样 (第 209 页)
*\widowpenalty 开始新页的单行的惩罚, 默认为 150 (第 141 页)
\wlog 在日志文件中 \write 指定的记号列表 (第 279 页)
\wp Weierstraß ‘p’ 符号: \wp (第 197 页)
\wr 圈积运算符: \wr (第 198 页)
*\write 将一行写入指定的输出流 (第 265 页)
*\xdef 等同于 \global\edef, 即, 定义全局宏, 立即展开替换文本 (第 246 页)
\xi 数学希腊字母 ξ (第 196 页)
\Xi 数学希腊字母 Ξ (第 196 页)
*\xleaders 输出剩余空白均匀分布于指引线盒子之间的指引线 (第 179 页)

- *\xspaceskip 如果非零, 且 $\spacefactor \geq 2000$, 覆盖正常的字间粘连 (第 107 页)
- *\year 以数值形式输出当前年份 (第 239 页)
- \zeta 数学希腊字母 ζ (第 196 页)



GNU 自由文档许可证

Version 1.3, 3 November 2008

Copyright © 2000,2001,2002,2007,2008 Free Software Foundation,
Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.

1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program

should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a worldwide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, L^AT_EX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”,

“Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may

add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the

Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the

Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting

from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

”Massive Multiauthor Collaboration Site” (or ”MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A ”Massive Multiauthor Collaboration” (or ”MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

”CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

”Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ”eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

11. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover

Texts.

A copy of the license is included in the section entitled ”GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the ”with...Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

索引

在这个索引的各项中，意大利体页码表示一个主要项或定义项。

\u, 3, 55
\u, 104, 340
!, 14
\!, 225, 340
\", 100, 340
#, 27, 48, 75, 340
 其类别码, 55
 在普通文本中, 15
\#, 15, 98, 341
\$, 16, 27, 92, 341
 其类别码, 55
 在普通文本中, 15
\\$, 15, 98, 341
 \$\$, 16, 62
%, 13, 27, 341
 其类别码, 55
 去掉不想要的空格, 289
 写出该字符, 315
 在普通文本中, 15
\%, 15, 98, 341
 &, 27, 47, 48, 341
 其类别码, 55
 在普通文本中, 15
\&, 15, 98, 341
 ', 197, 341
\', 100, 341
(, 60
*, 199, 341
\+, 47, 182, 341

\,, 225, 341
-, 15
 导致 \hyphenpenalty, 126
\-, 70, 127, 341
 在断行中, 74, 287
--, 15
---, 15
. , 14
\., 100, 341
/, 60
\/, 106, 341
\;, 225, 341
=, 200, 201
\=, 100, 341
\>, 225, 341
? , 14
[, 60
\, 58, 62, 341
 其类别码, 55
 在普通文本中, 15

^^J, 57
 ^^K, 55, 56
 ^^L, 55, 315, 341
 ^^M, 57, 105, 314, 341
 其类别码, 55
 -, 27, 206, 341
 其类别码, 55
 在普通文本中, 15
 _, 15, 98, 341
 \`_, 100, 341
 |, 198
 \|_, 197, 341
 ~, 13, 27, 55, 105, 341
 在断行中, 74
 在普通文本中, 15
 \~_, 100, 341

\AA, 97, 341
 \aa, 97, 341
 \above, 210, 342
 \abovemodecolumnspenalty, 332
 \abovedisplayshortskip, 229, 290, 342
 \abovedisplayskip, 228, 290, 342
 \abovewithdelims, 59, 211, 342

\accent, 101, 342
 \active, 46, 267, 342
 \acute, 209, 342
 \adjdemerits, 125, 342
 \advance, 261, 342
 \advancenopage, 145, 342
 \AE, 97, 342
 \ae, 97, 342
 \afterassignment, 244, 342
 \aftergroup, 243, 331, 342
 \aleph, 197, 342
 \allowbreak, 121, 139, 342
 \alpha, 196, 342
 \amalg, 198, 342
 $\mathcal{A}\mathcal{M}\mathcal{S}$ - \TeX , viii, 18
 \AMSTeX, 317
 \angle, 197, 342
 \approx, 200, 201, 342
 \arccos, 203, 342
 \arcsin, 203, 342
 \arctan, 203, 342
 \arg, 203, 342
 \langle argument \rangle, 4
 \Arrowvert, 223, 342
 \arrowvert, 223, 342
 ASCII, 51, 55, 56, 65
 \ast, 198, 342
 \asymp, 200, 201, 342

\atop, 210, 343
 \atopwithdelims, 59, 211, 343
 埃尔帕索, 91

\b, 209, 343
 \backslash, 197, 343
 \badness, 18, 176, 343
 \bar, 209, 343
 \baselineskip, 71, 136, 136–138, 343
 和 \smallskipamount 等, 159
 用于保留页尾, 286
 \batchmode, 269, 343
 \begingroup, 241, 343
 \begin{section}, 130, 343
 \belowdisplayshortskip, 229, 343
 \belowdisplayskip, 228, 343
 \beta, 196, 343
 beta, 197
 \bf, 103, 343
 \bf fam, 221, 343
 \bgroup, 343
 BIB \TeX , 19
 \BibTeX, 317
 \Big, 222, 343
 \big, 222, 343
 \bigbreak, 140, 343
 \bigcap, 204, 343
 \bigcirc, 198, 343
 \bigcup, 204, 343
 \Bigg, 222, 343
 \bigg, 222, 343
 \Biggl, 222, 343
 \biggl, 222, 343
 \Biggm, 222, 343
 \biggm, 222, 343
 \Biggr, 222, 344
 \biggr, 222, 344
 \Bigl, 222, 344
 \bigl, 60, 222, 344
 \Bigr, 222, 344
 \bigm, 198, 222, 344
 \bigodot, 204, 344
 \bigoplus, 204, 344
 \bigotimes, 204, 344
 \Bigr, 222, 344
 \bigr, 60, 222, 344
 \bigskip, 140, 158, 344
 \bigskipamount, 158, 344
 \bigsqcup, 204, 344
 \bigtriangledown, 198, 344
 \bigtriangleup, 198, 344
 \biguplus, 204, 344

\bigvee, 204, 344
\bigwedge, 204, 344
\binoppenalty, 126, 344
\blackbox, 317
\bmod, 203, 344
\bordermatrix, 216, 344
\bot, 197, 344
\botmark, 79, 147, 344
\bowtie, 200, 344
\box, 169, 170, 344
\box255, 84, 88, 147, 151
\boxmaxdepth, 54, 167, 168, 344
\brace, 210, 344
\bracevert, 223, 344
\brack, 210, 344
\break, 74, 121, 139, 344
 将行尾字符变为它, 296
 纠正断行, 287
\breve, 209, 344
\brokenpenalty, 141, 345
\buildrel, 212, 345
\bullet, 198, 345
\bye, 263, 345
 八进制数, 81
 版本号, 239
 帮助文本, 279
 帮助诊断, 270–278
 备选内容, 83
 本地信息, 8, 10
 闭符号, 201
 编程功能, 234
 编号列表, 322–324
 编组, 15–16, 55, 68–69, 241–244
 边距, 25, 78–79, 85
 变音符, *see* 重音
 标点, 14, 23
 标号尺寸, 62, 89
 标号样式, 91, 207, 221
 标记, 79, 147–148
 用于分割列表, 147
 用于页眉和页脚, 85
 标记文本, 79
 标识, 64
 标线, 89, 178–179, 291–292
 标线的厚度, 315
 表格, 47–49, *see* 阵列
 波浪符, 100, 209
 不完整的条件句, 297
 不需要的空格, 13
\c, 100, 345
\cal, 220, 345
\cap, 198, 345
\cases, 211, 345
\catcode, 46, 55, 62, 267, 345
\cdotp, 198, 206, 345
\cdotp, 206, 345
\cdots, 213, 345
\center, 331
\centereddisplays, 318
\centerline, 68, 108, 345
\char, 54, 57, 99, 345
(charcode), 4
\chardef, 247, 250, 345
\check, 209, 345
\chi, 196, 345
\choose, 210, 345
\circ, 198, 345
\cleaders, 72–73, 179, 345
\cleartabs, 183, 345
\closein, 64, 264, 345
\closeout, 64, 265, 345
 和 \immediate 一起, 266
 生成小玩意, 93
\clubpenalty, 141, 345
\clubsuit, 197, 345
 cmex10, 223
 cmtt10, 109
\colon, 206, 345
\cong, 200, 201, 345
\coprod, 204, 345
\copy, 169, 345
\copyright, 98, 345
\cos, 203, 345
\cosh, 203, 346
\cot, 203, 346
\coth, 203, 346
\count, 50, 258, 262, 346
\count0, 145
\countdef, 261, 346
\cr, 47–49, 186, 186, 187, 346
\crcr, 187, 346
\csc, 203, 346
\csname, 240, 248, 346
 用 \edef 规则展开, 246
\cup, 198, 346
 参考符号, 148
 参考文献, 14
 参量, 12, 50–51

- 参数, 86
 定界参数, 76–77
 非定界参数, 75
 给参数赋值, 51
 和 `\the` 一起使用, 249
 如同寄存器, 88
 用 # 指明, 55
 与参量对应, 50
 作为命令, 4, 12
 操作系统, 63, 64
 插入项, 70–71
 插入项的惩罚, 142
 用 `\newinsert` 预留编号, 260
 用 `\supereject` 强制排版, 140
 用于插入项的命令, 149–151
 肠道, 17, 49, 50, see `\TeX` 剖析
 陈列公式, 16, 62, 141, 290
 陈列公式的间隔参数, 228–229
 多行陈列公式, 218–220
 显示形式, 318
 作用到每个陈列公式, 230
 陈列数学模式, 79, 81
 陈列样式, 91, 204, 207
 惩罚, 87
 列表中最后一个惩罚项, 177
 在竖直列表中, 84
 在水平列表中, 74
 程序代码, 排版, 122
 尺寸, 61–62
 比较两个尺寸, 253
 负尺寸, 62
 盒子寄存器的尺寸, 172
 最大尺寸, 259
 尺寸寄存器, 87, 258
 用 `\newdimen` 预留, 260
 错误信息, 10, 278–280, 304–310
- `\d`, 100, 346
`\dag`, 98, 346
`\dagger`, 198, 346
`\dashv`, 200, 346
`\day`, 239, 321, 346
`\ddag`, 98, 346
`\ddagger`, 198, 346
`\ddot`, 209, 346
`\ddots`, 214, 346
`\deadcycles`, 151, 346
`\def`, 59, 2/5, 346
 变成全局的, 66
`\defaulthyphenchar`, 130, 346
`\defaultskewchar`, 224, 346
`\deg`, 203, 346
- `\delcode`, 60, 217, 268, 346
`\delimiter`, 60, 215, 346
`\delimiterfactor`, 215, 346
`\delimitershortfall`, 215, 346
`\Delta`, 196, 346
`\delta`, 196, 346
`\det`, 203, 207, 346
`\diamond`, 198, 346
`\diamondsuit`, 197, 347
 didôt point, 61
`\dim`, 203, 347
`\dimen`, 4, 294
`\dimen`, 258, 262, 347
`\dimendef`, 261, 347
`\discretionary`, 127, 347
`\displayindent`, 228, 347
`\displaylimits`, 205, 347
`\displaylines`, 218, 320, 347
`\displaysetup`, 319
`\displaystyle`, 91, 207, 347
`\displaywidowpenalty`, 141, 347
`\displaywidth`, 228, 347
`\div`, 198, 347
`\divide`, 262, 347
`\dot`, 209, 347
`\doteq`, 200, 347
`\dotfill`, 181, 347
`\dots`, 99, 347
`\doublehyphendemerits`, 126, 347
`\Downarrow`, 60, 202, 347
`\downarrow`, 60, 202, 347
`\downbracefill`, 223, 347
`\dp`, 172, 347
`\dump`, 65, 280, 347
`.dvi` 文件, 9
 created by `\TeX`'s intestines, 50
 接收 `\shipout` 的盒子, 151
 来自输出例行程序的素材, 83, 84
 其中的小玩意, 93
 用 `\special` 插入的素材, 93
 由 `\TeX` 的肠道生成, 17
 由驱动程序转换, 65
 在日志文件中记录盒子内容, 275
 作为结果文件, 63
 打印机, 9
 打字机字体, 109
 大点, 61
 大写转换, 103–104
 大写字母
 转换为大写字母, 103–104
 带子, 13, 74, 105
 单词间距, 52, 106, 107
 单词连在一起, 288

- 导言, 48, 184, 185
迪多点, 61
点, 61
点号, 99
调试, 270–278
叠印, 109
定界参量, 50
定界符, 59–61, 201–202, 214–216
 定界符的一部分, 223
 定界符高度, 215
 空定界符的间隔, 229
 增大定界符, 222
定界码, 60–61, 217, 268
定理, 33, 131
读取文件, 264
度量单位, 61, 62, 92
度量文件, 9, 60
 其中的默认连字符, 130
 其中的默认斜字符, 224
 其中的倾斜修正, 106
短音符, 100, 209
断行, 17, 50, 74–75, 121–130, 290
 不良断行, 287
 断行处的紧排, 161
 断行的劣度, 52
 断行的缺陷, 61
 断行与段落形状, 113–120
 鼓励或阻碍断行, 121–123
 删除断行, 13
 影响断行的参数, 123–126
 在数学公式中, 121
 追踪断行, 277
段落, 85–86
 段落间的粘连, 144
 段落末尾的粘连, 111
 结束段落, 13, 23
 开始段落, 110
 缩进, *see* 缩进
 形成段落, 110–120, 290–291
 窄段落, 25, 114
段落间距, 144
堆叠子公式, 210–213
对齐文本, 71
钝音符, 100, 209
多余空格, 275
- \edef, 245, 347
 变成全局的, 66
 在其中展开 \`c, 82
\egroup, 242, 347
\hrule, 315
\eject, 140, 285, 347
- \ell, 197, 347
\else, 58, 255, 347
em, 61
\emergencystretch, 18, 124, 124, 287, 347
\empty, 257, 347
\emptyset, 197, 348
en, 157
\end, 263, 348
\endcsname, 348
\endgraf, 110, 348
\endgroup, 241, 348
\endinput, 263, 348
\endinsert, 150, 348
\endline, 186, 348
\endlinechar, 268, 299, 348
\enskip, 157, 348
\enspace, 157, 348
\environment, 329
 explain.tex, 19, 314–336
\epsilon, 196, 348
\eq, 319
\eqalign, 218, 348
\eqalignno, 218, 320, 348
\eqdef, 328
\eqn, 319
\eqno, 217, 348
\eqprint, 328
\eqref, 328
\equiv, 200, 201, 348
\errhelp, 279, 279, 348
\errmessage, 278, 348
 用 \edef 规则展开, 246
\errorcontextlines, 18, 280, 309, 348
\errorstopmode, 269, 348
\escapechar, 62, 240, 240, 266, 348
\eta, 196, 348
\everycr, 186, 191, 348
\everydisplay, 230, 319, 322, 348
\everyfootnote, 333
\everyhbox, 169, 348
\everyjob, 281, 348
\everymath, 230, 348
\everypar, 86, 110, 113, 348
 用于设定 \looseness, 125
 用于悬挂缩进, 117
\everyvbox, 169, 348
\rule, 315
 ex, 61
\exhyphenpenalty, 126, 348
\exists, 197, 349
\exp, 203, 349
\expandafter, 245, 249, 349

- 额外间隔, 225
- \fam, 221, 349
\fi, 58, 255, 349
fil, 61, 68
\filbreak, 140, 285, 349
fill, 61, 68
filll, 61, 68
\finalhyphendemerits, 126, 349
\firstmark, 79, 147, 349
\fivebf, 102, 349
\fivei, 102, 349
\fiverm, 102, 349
\fivesy, 102, 349
\flat, 197, 349
\floatingpenalty, 142, 349
\flushleft, 331
\flushright, 331
\fmtname, 239, 349
\fmtversion, 239, 349
\folio, 146, 349
\font, 130, 224, 234, 317, 349
\fontdimen, 235, 317, 349
\fontname, 241, 349
\footline, 65, 85, 146, 292, 349
\footnote, 71, 148, 150, 349
\footnotemarkseparation, 333
\for, 316
\forall, 197, 349
 for 循环, 316
\frac, 317
\frenchspacing, 14, 106, 349
\frown, 200, 349
\futurelet, 248, 349
 反斜杠, 241
 放大率, 9, 62, 77–78
 非定界参量, 50
 分式, 210–213
 斜线形式, 317
 用 \over 生成, 210
分页, 84–85, 139–143
 不良分页, 284–286
 分页参数, 141–143
 分页处的紧排, 161
 分页处的粘连, 286
 分页的劣度, 52
 分页时的插入项, 70
 鼓励或阻碍分页, 139–141
 由 TeX 的胃插入, 17
 追踪分页, 276
分音符, 100
分支测试, 255
- 浮动体, 142, 149
辅助文件, 240
覆盖文件, 109
赋值, 51–52, 86
 给盒子赋值, 169
 给寄存器赋值, 258
- \Gamma, 196, 349
\gamma, 196, 349
\gcd, 203, 349
\gdef, 66, 243, 246, 349
\ge, 200, 201, 349
\generaldisplay, 319
\geq, 200, 201, 350
\gets, 202, 350
.gf 文件, 9, 65
\gg, 200, 350
\global, 66, 243, 243, 350
 global, 243
\globaldefs, 66, 243, 300, 350
\glue, 4
\gobble, 314
\gobblethree, 314
\gobbletwo, 314
\goodbreak, 140, 350
\grave, 209, 350
 感叹号, 14
 高度, 53, 69, 172
 格式文件, 65–66, 280
 公式编号, 218, 328
 孤行, 141
 寡行, 141
 关系符, 126, 200–201
 将公式放在其上, 212
 过满盒子, 287–288
- \H, 100, 350
- \halign, 47–48, 184, 186, 350
 本质上竖直的, 70
 在数学模式中不合法, 320
 阵列中的编组, 16
- \hang, 117, 290, 350
\hangafter, 117, 118, 350
\hangindent, 86, 117, 118, 290, 350
\hat, 209, 350
\hbadness, 124, 175, 350
\hbar, 197, 350
\hbox, 53, 164, 287, 350
 它导致的过满盒子, 288
- \headline, 69, 85, 146, 292, 350
\heartsuit, 197, 350
\hfil, 161, 163, 164, 288, 350

- \hfill, 161, 350
\hfilneg, 162, 350
\hfuzz, 124, 176, 287, 350
\hglue, 160, 161, 350
\hidewidth, 191, 350
\hoffset, 78, 85, 143, 350
\holdinginserts, 18, 152, 350
\hom, 203, 350
\hookleftarrow, 202, 350
\hookrightarrow, 202, 350
\phantom, 174, 350
\rule, 89, 178, 291, 350
 本质上竖直的, 70
\rulefill, 181, 350
\size, 78, 85, 113, 143, 350
 用 \magnification 设定, 237
\hskip, 66, 159, 351
\hss, 162, 288, 351
\ht, 172, 351
\hyphenation, 70, 128, 351
\hyphenchar, 129, 130, 250, 351
\hyphenpenalty, 126, 141, 351
 函数名称, 203–204
 毫米, 61
 盒子, 17, 53–54
 不可见盒子, 90
 测试空盒子, 254
 复制盒子, 169, 170
 过满盒子, 175–176, 287–288
 盒子的高度, 69
 盒子的基线, 52
 盒子的基准点, 87
 盒子的宽度, 93
 盒子的深度, 61
 盒子与粘连, 66
 幻影盒子, 173–175
 绘制盒子, 31
 空盒子, 175
 列表中最后一个盒子, 177
 提取盒子的内容, 170
 未满盒子, 175–176, 287–288
 移动盒子, 171–172
 盒子寄存器, 54, 87, 169–172
 用 \newbox 预留, 260
 盒子命令, 164–176
 横线号, 15, 23
宏, 75–77, 245–257
 定义宏, 245–247
 宏的参量, 50, 299
 宏的参数, 50, 55, 75–77, 299
 控制宏展开, 249–250
 全局宏, 66
 使宏易读, 299–300
使用 \begingroup 和 \endgroup, 241
外部宏, 82–83, 298
用 \bgroup 和 \egroup, 242
由活动字符命名, 46
在 TeX 的胃展开, 49
在辅助文件中, 8
追踪宏, 275
花括号, 212, 223
 不匹配的花括号, 293–294
 花括号后的空格, 289
花色, 27, 197
画线, see 标线
环境, 329
幻影, 173–175
换页符, 55
活动字符, 46, 55
\i, 101, 351
\ialign, 186, 351
\if, 250, 351
\ifcase, 255, 351
\ifcat, 251, 351
\ifdim, 253, 351
\isempty, 316
\ifeof, 254, 351
\ifeqno, 318
\iff, 202, 351
\iffalse, 255, 351
\ifhbox, 254, 351
\ifhmode, 253, 351
\ifinner, 253, 351
\ifleqno, 318
\ifmmode, 253, 351
\ifnum, 252, 351
\ifodd, 252, 351
\iftrue, 255, 351
\ifvbox, 254, 351
\ifvmode, 253, 351
\ifvoid, 150, 254, 351
\ifx, 251, 351
\ignorespaces, 268, 351
\Im, 197, 351
\imath, 101, 197, 351
\immediate, 64, 93, 265, 266, 266, 351
\in, 200, 351
\indent, 85, 111, 351
\inf, 203, 351
\infty, 197, 352
\initex, 66, 281
\input, 8, 49, 63, 263, 352
\inputlineno, 264, 352
\insert, 70, 71, 150, 352

- <inserted text>, 308
- \insertpenalties, 142, 142, 352
- \int, 204, 205, 352
 - 极限放在后面, 205
- \interlinepenalty, 141, 352
- \iota, 196, 352
- \it, 103, 352
- \item, 131, 352
- \itemitem, 131, 352
- \itfam, 221, 352
-
- \j, 101, 352
- \jmath, 101, 197, 352
- \jobname, 240, 352
- \jot, 226, 352
 - 基线, 29, 53, 52–53, 87
 - 基准点, 53–54, 87
 - 极限, 204
- 寄存器, 87–88, 258–262
 - 给寄存器赋值, 51
 - 和 \the 一起, 250
 - 寄存器中的算术运算, 261–262
 - 预留寄存器, 260–261
 - 作为参数, 4, 12, 86
- 计数寄存器, 87, 258
 - 用 \newcount 预留, 260
- 计算机程序, 排版, 295, 299
- 计算机现代字体, 35, 51, 64, 109
- 记号, 16, 92
 - 传给 T_EX 的胃, 49
 - 将字符组合为记号, 49
 - 显示记号的含义, 240
 - 用 \show 显示, 270
 - 作为命令, 57
- 记号寄存器, 87, 259
 - 用 \newtoks 预留, 260
- 假设, 131
- 尖角符, 209
- 间隔, 90
 - 不想要的间隔, 289–290
 - 产生间隔, 156–163
 - 单词间距, 66, 104–108
 - 丢失的间隔, 288
 - 数学公式中的间隔, 225–227
 - 行间距, 29
 - 用紧排调整, 71
 - 间隔因子, 107
 - 间距寄存器, 259
 - 用 \newskip 预留, 260
 - 箭头, 181, 202, 212
 - 交叉引用, 326
- 脚注, 23, 333
 - 在脚注中用 \textindent, 112
- 结果文件, 63
- 结束任务, 263
- 节标题, 130
- 紧排, 71–72, 160
 - 列表中最后一个紧排, 177
 - 数学公式中的紧排, 227
 - 用它生成间距, 90
 - 作为列表项目, 53
- 禁止的控制序列, 297
- 居右对齐, 33, 108–109, 331–332
- 居中对齐, 33, 108–109
- 居中文本, 68, 71, 331–332
- 居左对齐, 33, 108–109, 331–332
- 句号, 12, 14
- 矩阵, 216
- 均匀对齐, 108–109, 116
-
- \kappa, 196, 352
- \ker, 203, 352
- \kern, 160, 352
 - Knuth, Donald E., 18, 317
 - 开符号, 201
 - 可打印字符, 51
 - 可忽略符, 55
 - 可见空格, 104
 - 空白, 保留, 296
 - 空格, 12–13
 - 可见空格, 3
 - 空格符, 23, 317
 - 其类别码, 55
 - 控制 T_EX, 269
 - 控制词, 11, 58, 59
 - 控制符, 11, 58, 59
 - 控制空格, 11, 104, 289
 - 控制序列, 10–11, 58–59
 - 忽略空格, 11
 - 用 \let 定义, 247
 - 与命令对比, 11
 - 转换为字符串, 240
 - 作为记号, 16
 - 控制字符, 8, 51, 56
 - 宽波浪符, 209
 - 宽度, 53, 93, 172
 - 宽尖角符, 209

\L, 97, 352
\l, 97, 352
\Lambda, 196, 352
\lambda, 196, 352
 Lamport, Leslie, 18
\land, 198, 352
\angle, 60, 201, 352
\language, 18, 93, 129, 129, 352
\lastbox, 177, 177, 352
\lastkern, 177, 250, 352
\lastpenalty, 177, 250, 352
\lastskip, 177, 250, 352
 ^ATeX, viii, 18, 239
\LaTeX, 317
\lbrace, 60, 201, 352
\lbrack, 60, 98, 201, 352
\lccode, 103, 353
\lceil, 60, 201, 353
\ldotp, 206, 353
\ldots, 213, 353
\le, 200, 201, 353
\leaders, 72–73, 179, 353
\left, 59, 214, 353
\Leftarrow, 202, 353
\leftarrow, 202, 353
\leftarrowfill, 181, 353
\leftdowndown, 318–320
\leftharpoondown, 202, 353
\leftharpoonup, 202, 353
\lefthyphenmin, 18, 129, 353
\leftline, 68, 108, 353
\Leftrightarrow, 202, 353
\rightarrow, 202, 353
\leftskip, 71, 115, 290, 353
\leq, 200, 201, 353
\leqalignno, 218, 320, 353
\leqno, 217, 353
\let, 59, 247, 353
\letreturn, 314, 317
\letter, 314
\lfloor, 60, 201, 353
\lg, 203, 353
\lgroup, 214, 353
\li, 322
\lim, 203, 207, 353
\liminf, 203, 353
\limits, 204, 353
\limsup, 203, 353
\line, 109, 353
\linepenalty, 125, 353
\lineskip, 71, 136, 136–137, 354
\lineskiplimit, 71, 136, 136–137, 354
\listcompact, 322
\listing, 324
\ll, 200, 354
\llap, 109, 354
\lmoustache, 223, 354
\ln, 203, 354
\lnot, 197, 198, 354
\log, 203, 354
\loggingall, 315
\long, 246, 252, 354
\Longleftarrow, 202, 354
\longleftarrow, 202, 354
\Longleftrightarrow, 202, 354
\longleftrightarrow, 202, 354
\longmapsto, 202, 354
\Longrightarrow, 202, 354
\longrightarrow, 202, 354
\loop, 256, 354
\looseness, 86, 124, 286, 354
\lor, 198, 354
\lower, 53, 171, 354
\lowercase, 104, 354
\lq, 98, 354
类, 57, 221, 230
 定界符的类, 60
类别码, 54–56
 测试类别码, 251
 导致不想要的间隔, 289
 改变类别码, 299
 活动字符的类别码, 46
 用于原文文本, 296
 有用的类别码定义, 314
 在 \catcode 表格中, 267
 在读取时附加, 56
厘米, 61
连写, 73, 97–98, 101
连音符, 100
连字, 33, 70, 127–130, 141
 德语连字, 128
 连字的惩罚, 125–126
连字规则, 129
列表, 75
列举, 322–324
劣度, 52, 61
轮廓文件, 9, 65
罗马数字, 238

\mag, 62, 77, 237, 354
 \magnification, 237, 354
 \magstep, 78, 237, 237, 354
 \magstephalf, 78, 237, 238, 354
 \makeactive, 314
 \makeblankbox, 317
 \makecolumns, 332
 \makefootline, 293
 \makeheadline, 292
 \mapsto, 202, 354
 \mark, 79, 147, 147, 354
 \mathaccent, 209, 354
 \mathbin, 230, 354
 \mathchar, 99, 354
 \mathchardef, 247, 250, 354
 \mathchoice, 207, 355
 \mathclose, 230, 355
 \mathcode, 80, 215, 267, 268, 355
 \mathinner, 230, 355
 \mathop, 230, 355
 \mathopen, 230, 355
 \mathord, 230, 355
 \mathpalette, 208, 355
 \mathpunct, 230, 355
 \mathrel, 230, 355
 \mathstrut, 90, 173, 355
 \mathsurround, 229, 355
 \matrix, 216, 355
 \max, 203, 355
 \maxdeadcycles, 152, 355
 \maxdepth, 144, 153, 355
 \maxdimen, 259, 355
 \meaning, 240, 355
 \medbreak, 140, 355
 \medmuskip, 226, 355
 \medskip, 140, 158, 355
 \medskipamount, 158, 355
 \message, 278, 355
 用 \edef 规则展开, 246
 METAFONT, vii, 35
 \MF, 317
 \mid, 200, 355
 \midinsert, 71, 149, 150, 355
 \min, 203, 355
 \mit, 220, 355
 \mkern, 227, 355
 \models, 200, 355
 \month, 239, 321, 356
 \monthname, 321
 \moveleft, 54, 166, 171, 356
 \moveright, 54, 166, 171, 356
 \mp, 198, 356
 \mskip, 81, 227, 356
 \mu, 196, 356
 \mu, 81, 259
 \multiply, 262, 356
 \multispan, 188, 188, 356
 \muskip, 258, 262, 356
 \muskipdef, 261, 356
 描述列表, 31
 命令, 3, 10–11, 57–58
 命令的参量, 50
 与控制序列对比, 11
 模板, 48, 184–189
 模式, 17, 81
 模运算, 203
 蘑菇, 37
 目录, 324
 \nabla, 197, 356
 \narrower, 114, 290, 356
 \natural, 197, 356
 \ne, 200, 356
 \nearrow, 202, 356
 \neg, 197, 198, 356
 \negthinspace, 157, 356
 \neq, 200, 356
 \newbox, 54, 88, 260, 356
 \newcount, 88, 258, 260, 356
 \newdimen, 88, 260, 356
 \newfam, 63, 260, 356
 \newhelp, 279, 330, 356
 \newif, 256, 356
 \newinsert, 150, 260, 356
 \newlanguage, 18, 260, 356
 \newlinechar, 267, 279, 356
 \newmuskip, 88, 260, 356
 \newread, 64, 260, 356
 \newskip, 88, 260, 356
 \newtoks, 88, 260, 356
 \newwrite, 64, 260, 357
 \ni, 200, 357
 \noalign, 48, 187, 189, 357
 \noboundary, 18, 73, 101, 357
 \nobreak, 74, 121, 139, 285, 357
 \noexpand, 245, 357
 \noindent, 86, 111, 357
 \nointerlineskip, 138, 357
 \nolimits, 205, 357
 \nonfrenchspacing, 106, 357
 \nonscript, 227, 229, 357
 \nonstopmode, 269, 357
 \nopagenumbers, 145, 357
 \normalbaselines, 137, 357
 \normalbaselineskip, 137, 357

\normalbottom, 140, 357
\normallineskip, 137, 357
\normallineskiplimit, 137, 357
\not, 357
\notin, 200, 357
\nu, 196, 357
\null, 175, 357
\nulldelimiterspace, 61, 215, 229, 357
\nullfont, 102, 357
(number), 4
\number, 82, 238, 357
\numberedfootnote, 333
\numberedlist, 322
\numberedmarker, 323
\numbername, 318
\narrow, 202, 357
内部模式
 测试内部模式, 253
 内部竖直模式, 81, 92
内置字体, 235

\O, 97, 357
\o, 97, 357
\obeylines, 122, 296, 317, 358
\obeyspaces, 107, 122, 296, 317, 358
\obeywhitespace, 107, 296, 316
\odot, 198, 358
\OE, 97, 358
\oe, 97, 358
\offinterlineskip, 90, 137, 172, 358
\oint, 204, 205, 358
\oldstyle, 220, 358
\Omega, 196, 358
\omega, 196, 358
 omicron, 197
\ominus, 198, 358
\omit, 187, 358
\openin, 64, 264, 264, 358
\openout, 64, 265, 358
 和 \immediate 一起, 266
 生成小玩意, 93
\openup, 138, 189, 358
\oplus, 198, 358
\or, 358
\oslash, 198, 358
\other, 314
\otimes, 198, 358
\outer, 82, 246, 252, 298, 358
<\output>, 309
\output, 83, 151, 151, 358
\outputpenalty, 152, 358
\over, 210, 215, 358
\overbrace, 212, 223, 358
\overfullrule, 175, 358
\overleftarrow, 212, 358
\overline, 212, 358
\overrightarrow, 212, 358
\overwithdelims, 59, 211, 359
\owns, 200, 359
 欧洲语言, 73, 129
 欧洲语言字母, 97

\P, 98, 359
\pagedepth, 142, 359
\pagefillstretch, 143, 359
\pagefillstretch, 143, 359
\pagefilstretch, 143, 359
\pagegoal, 142, 143, 152, 359
\pageinsert, 71, 149, 150, 359
\pageno, 145, 146, 359
\pageshrink, 142, 359
\pagestretch, 143, 359
\pagetotal, 142, 359
 Palatino 字体, 35, 64
\par, 85, 110, 110, 359
 来自空行, 90, 105
 用于结束段落, 85
 在宏参量中, 246
 在修改段落形状时, 291
\parallel, 200, 359
\parfillskip, 111, 359
\parindent, 3, 86, 112, 359
 逐项列表的缩进, 131
\parshape, 86, 118, 118, 359
\parskip, 111, 112, 144, 286, 288, 359
\partial, 197, 359
 Patashnik, Oren, 19
\pausing, 269, 359
\penalty, 122, 139, 359
\percentchar, 315
\perp, 200, 359
\phantom, 173, 359
\Phi, 196, 359
\phi, 196, 359
\Pi, 196, 204, 359
\pi, 196, 359
.pk 文件, 9, 65
\plainoutput, 151, 359
 plain TeX, 3, 8, 9, 87
 其中的字体族, 63
\pm, 198, 359
\pmatrix, 216, 360
\pmod, 203, 360
\postdisplaypenalty, 141, 360

- \Pr, 203, 360
 \prec, 200, 201, 360
 \preceq, 200, 201, 360
 \predisplaypenalty, 141, 360
 \predisplaysize, 228, 360
 \pretolerance, 123, 360
 \prevdepth, 136, 137, 360
 \prevgraf, 120, 360
 \prime, 197, 360
 \proclaim, 131, 360
 \prod, 204, 360
 \proto, 200, 360
 \Psi, 196, 360
 \psi, 196, 360
 .pxl 文件, 9
 派卡, 61
 普通模式, 82
 普通竖直模式, 81, 92
 普通水平模式, 69, 81
- \quad, 157, 360
 \quad, 157, 360
 其他字符, 55, 57
 起点, *see* 页面起点
 倾斜修正, 106
 驱动, *see* 设备驱动
 全局的, 66
 缺陷, 61, 74
- \radical, 217, 360
 \raggedbottom, 140, 360
 \raggedright, 71, 116, 288, 360
 \raise, 53, 171, 360
 \rangle, 60, 201, 360
 \rbrace, 60, 360
 \rbrack, 60, 98, 201, 360
 \rceil, 60, 201, 360
 \Re, 197, 360
 \read, 63, 264, 360
 \readreffile, 327
 \readtocfile, 326
 (register), 4
 \relax, 10, 257, 360
 \relpenalty, 126, 361
 \repeat, 256, 361
 \rfloor, 60, 201, 361
 \rgroup, 214, 361
 \rho, 196, 361
 \right, 59, 214, 361
 \Rightarrow, 202, 361
 \rightarrow, 202, 361
 \rightarrowfill, 181, 361
- \rightarpoondown, 202, 361
 \rightarpoonup, 202, 361
 \righthyphenmin, 18, 129, 361
 \rightleftarpoons, 202, 361
 \rightline, 68, 108, 361
 \rightskip, 71, 115, 361
 \rlap, 109, 361
 \rm, 103, 361
 \rmoustache, 223, 361
 \romannumeral, 82, 238, 361
 \root, 217, 361
 \rq, 98, 361
 任意连字符, 126
 日期, 239, 321
 日期时间, 239, 321
 日志文件, 75, 340
 错误信息, 304
 用 \wlog 写入, 279
 用 \write 写入, 266
 在其中追踪统计信息, 277
 作为结果文件, 63
- 软音符, 100
 锐音符, 100, 209
- \S, 98, 361
 \sb, 206, 361
 \scriptfont, 63, 221, 361
 \scriptscriptfont, 63, 221, 361
 \scriptscriptstyle, 91, 207, 207, 361
 \scriptspace, 229, 361
 \scriptstyle, 91, 207, 207, 361
 \scrollmode, 269, 361
 \searrow, 202, 362
 \sec, 203, 362
 \setbox, 169, 362
 \setlanguage, 18, 93, 129, 362
 \setminus, 198, 362
 \settabs, 47, 182, 362
 \setuplistinghook, 324
 \sevenbf, 102, 362
 \seveni, 102, 362
 \sevenrm, 102, 362
 \sevensy, 102, 362
 \sfcode, 107, 362
 \sharp, 197, 362
 \shipout, 64, 83, 151, 151, 277, 362
 此时显示 \count 寄存器的值, 145
 \show, 269, 270, 298, 362
 \showbox, 270, 273, 278, 362
 \showboxbreadth, 270, 275, 278, 362
 \showboxdepth, 270, 275, 278, 362
 \showhyphens, 128, 362

- \showlists, 270, 273, 362
\showthe, 88, 270, 362
\Sigma, 196, 204, 362
\sigma, 196, 362
\sim, 200, 201, 362
\simeq, 200, 201, 362
\sin, 203, 362
\singlecolumn, 334
\sinh, 203, 362
\skew, 224, 362
\skewchar, 224, 224, 250, 362
\skip, 258, 262, 362
\skipdef, 261, 362
\sl, 103, 362
\slash, 74, 123, 362
\slfam, 221, 362
\smallbreak, 140, 363
\smallint, 204, 363
\smallskip, 140, 158, 363
\smallskipamount, 158, 363
\smash, 174, 363
\smile, 200, 363
\sp, 206, 363
\space, 104, 299, 363
\spacefactor, 107, 363
\spaceskip, 107, 363
\spacesub, 316
\spadesuit, 197, 363
\span, 188, 363
\special, 93, 266, 363
Spivak, Michael D., 18
\splitbotmark, 79, 147, 363
\splitfirstmark, 147, 363
\splitmaxdepth, 153, 363
\splittopskip, 153, 363
\sqcap, 198, 363
\sqcup, 198, 363
\sqrt, 217, 363
\sqsubseteq, 200, 201, 363
\sqsupseteq, 200, 201, 363
\ss, 97, 363
\star, 198, 363
\string, 240, 248, 249, 363
\strut, 49, 90, 172, 173, 363
\subset, 200, 201, 363
\subseteq, 200, 201, 363
\succ, 200, 201, 363
\succeq, 200, 201, 364
\sum, 204, 364
\sup, 203, 364
\supereject, 140, 364
\supset, 200, 201, 364
\supseteq, 200, 201, 364
\surd, 197, 364
\swallow, 202, 364
上标, 55, 206–207
上点符, 100, 209
上极限, 206
上线符, 209
设备驱动, 9, 65, 235
 它所知的页面起点, 143
 用 \special 指导它, 266
设备驱动器, 266
设计大小, 78
伸长量, 66–68, 90
深度, 53, 61, 172
省略号, 99
诗歌、排版, 122
十六进制数, 81
十六进制数字, 57
食道, 49, 49, 54
收缩量, 66–68, 90
受限模式, 88
 水平受限模式, 291
受限水平模式, 69, 81
输出例行程序, 83, 150–152
 \insertpenalties 的意义, 142
 plain TeX 的默认程序, 151
 处理插入项, 70
输出流, 63, 83
 打开输出流, 265
 关闭输出流, 265
 写入输出流, 265
 用 \newwrite 预留, 260
输出设备, 9
输出文件, 265–267
输入流, 70
 打开输入流, 264
 用 \newread 预留, 260
 用 \read 读取, 63, 264
输入文件, 9, 49, 263–265
 嵌入输入文件, 10
 准备输入文件, 10
输入行, 268
输入字符, 299
数, 81–82
 比较两个数, 252
 测试奇偶性, 252
数学, 16, 39, 41, 196–231
 数学重音, 209
数学单位, 81, 81
数学符号, 64, 208
数学公式中的标点, 206
数学间距寄存器, 259
 用 \newmuskip 预留, 260
数学开关, 55

- 数学扩展符号, 63
- 数学码, 80, 247
 - 数学码中的类, 57
 - 数学模式, 79, 81
 - 测试数学模式, 253
 - 数学粘连, 81
 - 数学粘连寄存器, 87
 - 数学字符, 247
 - 用数学码描述, 80
 - 数值
 - 转换为字符, 238
 - 竖直标线, 89, 178–179
 - 竖直盒子, 53, 92
 - 测试竖直盒子, 254
 - 竖直盒子的行间粘连, 136
 - 位于竖直模式中, 92
 - 由 `\hsize` 确定宽度, 113
 - 竖直间隔, 158–163
 - 保留首页空白, 286
 - 竖直间距, 159
 - 竖直列表, 53, 66, 92
 - 不能包含水平命令, 92
 - 插入到段落中, 120
 - 竖直列表中的标线, 89
 - 竖直列表中的惩罚, 87
 - 组成竖直盒子, 92
 - 竖直模式, 81, 92–93
 - 测试竖直模式, 253
 - 竖直模式中的标线, 178
 - 竖直粘连, 159, 160
 - 双倍行距, 137
 - 双点符, 209
 - 双栏, 334
 - 水平标线, 89, 178–179
 - 水平盒子, 53, 69
 - 测试水平盒子, 254
 - 处于水平模式中, 69
 - 控制断行, 74
 - 用 `\hbox` 构造, 164, 166
 - 水平花括号, 223
 - 水平间隔, 156–163
 - 水平间距, 159
 - 水平列表, 53, 66, 69
 - 不能包含竖直命令, 70
 - 水平列表中的标线, 89
 - 水平列表中的惩罚, 87
 - 组成水平盒子, 69
 - 水平模式, 69–70, 81
 - 测试水平模式, 253
 - 水平模式中的标线, 178
 - 水平粘连, 159, 160
 - 水平制表符, 55
 - 算术, 261–262
 - 缩点, 61
 - 缩放因子, 237
 - 缩进, 25, 111–120
 - `\t`, 100, 364
 - `\tabalign`, 182, 364
 - `\tabskip`, 48, 190, 364
 - `\tan`, 203, 364
 - `\tanh`, 203, 364
 - `\tau`, 196, 364
 - `\tenbf`, 102, 364
 - `\tenex`, 102, 364
 - `\teni`, 102, 364
 - `\tenit`, 102, 364
 - `\tenrm`, 102, 364
 - `\tensl`, 102, 364
 - `\tensy`, 102, 364
 - `\tentt`, 102, 364
 - `\testfileexistence`, 318
 - `\TeX`, 99, 364
 - The *TeXbook*, viii, 2, 46
 - TeX* *M_EX*, 91
 - TeX* 剖析, 17, 49–50
 - `\textfont`, 63, 221, 364
 - `\textindent`, 112, 364
 - `\textstyle`, 91, 207, 364
 - TeX* 用户组织, 18
 - `.tfm` 文件, 9, 65, 72, 234
 - `\the`, 249, 258, 259, 364
 - `the`, 240
 - `\Theta`, 196, 364
 - `\theta`, 196, 364
 - `\thickmuskip`, 226, 364
 - `\thinmuskip`, 226, 364
 - `\thinspace`, 156, 364
 - `\tilde`, 209, 364
 - `\time`, 239, 321, 364
 - `\times`, 198, 365
 - `\timestamp`, 321
 - `\timestring`, 321
 - `\to`, 202, 365
 - `<to be read again>`, 306
 - `\toks`, 258, 365
 - `\toksdef`, 261, 365
 - `\tokstostring`, 315
 - `\tolerance`, 123, 124, 125, 365
 - `\top`, 197, 365
 - `\topglue`, 18, 160, 160, 286, 365
 - `\topinsert`, 71, 149, 150, 365
 - `\topmark`, 79, 147, 365
 - `\topskip`, 144, 153, 365
 - `\tracingall`, 278, 365

\tracingboxes, 315
\tracingcommands, 273, 365
\tracinglostchars, 274, 365
\tracingmacros, 275, 365
\tracingonline, 270, 273, 273–278, 365
\tracingoutput, 275, 278, 365
\tracingpages, 276, 365
\tracingparagraphs, 277, 365
\tracingrestores, 277, 365
\tracingstats, 277, 365
\triangle, 197, 365
\triangleleft, 198, 365
\triangleright, 198, 365
true, 78
\tt, 103, 365
\ttfam, 221, 365
\traggedright, 116, 365
TUGBoad, 18
特殊符号, 63, 97
特殊字符, 98
填充, 181
条件测试, 58, 250–256
条目 (在行或列中), 47, 184–185

\u, 100, 365
\uccode, 103, 365
\uchyph, 128, 365
\unbox, 178
\uncatcodespecials, 314
\underbar, 169, 365
\underbrace, 212, 223, 365
\underline, 212, 366
\uhbox, 170, 366
\uhcopy, 170, 366
\unkern, 177, 366
\unorderedlist, 322
\unpenalty, 177, 366
\unskip, 177, 366
\unvbox, 170, 366
\unvcopy, 170, 366
\Uparrow, 60, 202, 366
\uparrow, 60, 202, 366
\upbracefill, 223, 366
\Updownarrow, 60, 202, 366
\updownarrow, 60, 202, 366
\uplus, 198, 366
\uppercase, 104, 366
\Upsilon, 196, 366
\upsilon, 196, 366

\v, 100, 366
\vadjust, 120, 285, 286, 366
\valign, 49, 185, 366
本质上水平的, 93
用于 \makecolumns, 332
阵列中的编组, 16
\varepsilon, 196, 366
\varphi, 196, 366
\varpi, 196, 366
\varrho, 196, 366
\varsigma, 196, 366
\vartheta, 196, 366
\vbadness, 175, 366
\vbox, 53–54, 92, 166, 366
它导致的过满盒子, 288
用它修正分页, 284
\vcenter, 225, 320, 366
\vdash, 200, 366
\vdots, 213, 366
\vec, 209, 367
\vee, 198, 367
\Vert, 60, 197, 198, 367
\vert, 60, 197, 367
\vfil, 161, 167, 288, 367
填充竖直盒子, 167
在 \eject 时需要, 140
\vfill, 161, 285, 367
\vfilneg, 162, 367
\vfootnote, 148, 150, 333, 367
\vfuzz, 176, 288, 367
\vglue, 160, 161, 367
virtex, 66, 280
\voffset, 78, 85, 143, 292, 293, 367
\phantom, 174, 367
\rule, 89, 178, 291, 367
本质上水平的, 93
\vsize, 78, 85, 143, 292, 293, 367
用 \magnification 设定, 237
\vskip, 67, 159, 367
\vsplit, 147, 152, 367
\vss, 162, 288, 367
\vtop, 53–54, 92, 166, 367

\wd, 172, 367
\wedge, 198, 367
\widehat, 209, 367
\widetilde, 209, 367
\widowpenalty, 141, 367
\wlog, 279, 367
用 \edef 规则展开, 246
\wp, 197, 367
\wr, 198, 367

- <write>, 310
- \write, 265, 367
 - 在 \shipout 时进行展开, 151
 - 和 \immediate 一起, 266
 - 生成小玩意, 93
 - 用 \edef 规则展开, 246
 - 用它写出%, 315
 - 用于输出流, 63
 - 在其中展开 \`c, 82
- \writetocentry, 324
 - 外部的, 82–83, 246
 - 外语, 17, *see* 欧洲语言
 - 未满盒子, 287–288
 - 胃, 16, 49, 49, *see* T_EX 剖析
 - 文本编辑器, 8
 - 文本尺寸, 62, 92
 - 文本样式, 91, 207, 221
 - 文件, 63–64, 263–267
 - 检测文件是否存在, 318
 - 文件名, 64
 - 文件尾测试, 254
 - 文内公式, 16, 92
 - 文内数学模式, 79, 81
 - 文内样式, 204
 - 问号, 14
 - 无点字母, 101
 - 无效字符, 55, 56
- \xdef, 66, 243, 246, 367
- \Xi, 196, 367
- \xi, 196, 367
 - 新 T_EX, 17, 160
- \xleaders, 72–73, 179, 367
- \xrdef, 326
- \xref, 326
- \xrefn, 326
- \xspaceskip, 107, 368
 - 希腊字母, 196–197, 220
 - 西塞罗, 61
 - 狭窄变体, 91
 - 下边距, 66
 - 下标, 55, 206–207
 - 下点符, 100
 - 下极限, 206
 - 下线符, 209
 - 像素文件, 65
 - 向量符, 209
 - 项目, 71
 - 小标号尺寸, 62, 89
 - 小标号样式, 91, 207, 221
 - 小数, 59, 82
 - 小数点, 82
- 小玩意, 93, 129
- 小写字母
 - 转换为小写字母, 103–104
- 斜线符号, 74, 123
- 写入文件, 265
- 信息发送, 278–280
- 行间距, 136
- 行间粘连, 54, 71, 136–137
- 行结束符, 55
 - 行宽, 113
 - 行尾, 105
 - 行尾符, 90
 - 匈牙利分音符, 100
 - 修订线, 31
 - 悬挂缩进, 117
 - 循环, 256–257
- \year, 239, 321, 368
 - 眼睛, 16, 49, 49, *see* T_EX 剖析
 - 扬抑符, 100
 - 样式, 91, 207–208
 - 页脚, 65, 85
 - 多行页脚, 292–293
 - 用于页脚的标记, 79
 - 页码, 88, 145–146
 - 页眉, 69, 85
 - 多行页眉, 292–293
 - 用于页眉的标记, 79
 - 页面, 17, 83–84
 - 在 T_EX 的胃中组装, 50
 - 页面布局, 85
 - 页面尺寸, 143
 - 页面构建器, 83, 85
 - 页面起点, 143
 - 抑扬符, 100, 209
 - 溢出的水平盒子, 124
 - 因子, 62
 - 音符, 27, 197
 - 引号, 14, 23
 - 引理, 33, 131
 - 英镑, 98
 - 英寸, 61
 - 有序列表, 33
 - 右边距, 66
 - 右对齐, 68, 71, 115
 - 预载入, 66
 - 原始的, 87
 - 原始控制序列, 59
 - 原始命令, 3, 50, 340
 - 原文文本, 295
 - 圆点, 213–214
 - 圆括号, 59, 214

- 源码文件, 63
浏览器, 9
运算符, 126, 198–200
 巨算符, 204–205
运行 TeX, 9–10, 269
- Zapf, Hermann, 35
`\zeta`, 196, 368
粘连, 17, 66–68, 227
 负粘连, 162
 可无限伸展的粘连, 161–162
 可无限收缩的粘连, 68
 列表中最后一个粘连, 177
 数学粘连, 81
 用它生成间距, 90
粘连寄存器, 87
展开记号, 16
长音符, 100
阵列, 47–49
 各行之间的间隔, 138
 其中的外部控制序列, 298
 用于阵列的命令, 182–192
 在其中使用 `\offinterlineskip`, 138
 阵列的制表符, 55
支架, 90–91, 138, 172–173
 在竖直阵列中, 185
指引线, 72–73, 179–181
制表, 55
制表符, 182
制表阵列, 182–184
终端, 270
重复操作, 256–257
重音, 27, 100, 101, 209
 对齐重音, 224
轴线, 225
逐项列表, 131
主竖直列, 83
- 注释, 13, 23, 55
转义符, 58, 59, 62
 用 `\escapechar` 表示, 240
 转义符的类别码, 55
追踪, 270–278
字符, 51, 56–57, 99–100
 特殊字符, 27
 用 `\chardef` 定义, 247
 字符的 ASCII 码, 82
 字符的类别码, 54
字符记号, 316
字母, 55, 57
字体, 27, 64–65, 102, 220
 混合字体, 294–295
 命名及修改字体, 234–238
 字体参数, 235
 字体的连字符, 129–130
 字体族, 62
字体风格, 103, 220
字体名称, 241
字体文件, 9
字体信息文件, 234
字形文件, 78
自定连字符, 70, 74, 127
 纠正不良断行, 287
 纠正过满盒子, 287
族, 62–63
 可变族, 80
 用 `\fam` 给出, 221
 用 `\newfam` 预留, 260
 族的标号尺寸, 89
 族的文本尺寸, 92
 族的小标号尺寸, 89
 作为数学码的一部分, 80
组合数记法, 211
嘴巴, 16, 49, 49, see TeX 剖析
左对齐, 68, 71, 116

作者简介

Paul W. Abrahams, Sc.D., CCP, is a consulting computer scientist and a past president of the Association for Computing Machinery. His specialties are programming languages, software systems design and implementation, and technical writing. He received his doctorate from the Massachusetts Institute of Technology in 1963 in mathematics, studying artificial intelligence under Marvin Minsky and John McCarthy. He is one of the designers of the first LISP system and the designer of the CIMS PL/I system, developed when he was a professor at New York University. More recently, he has designed SPLASH, a Systems Programming LAnguage for Software Hackers. Paul resides in Deerfield, Massachusetts, where he writes, hacks, hikes, hunts wild mushrooms, and listens to classical music.

Kathryn A. Hargreaves received her M.S. degree in computer science from the University of Massachusetts, Boston, in August 1989. Her specialities are digital typography and human vision. She developed a set of programs to produce high-quality, freely distributable digital type for the Free Software Foundation and also worked with Robert A. Morris as an Adjunct Research Associate. In 1986 she completed the Reentry Program in Computer Science for Women and Minorities at the University of California at Berkeley, where she also worked in the T_EX research group under Michael Harrison. She has studied letterform design with Don Adleta, André Gürler, and Christian Mengelt at the Rhode Island School of Design. A journeyman typographer, she has worked at Headliners/Identicolor, San Francisco, and Future Studio, Los Angeles, two leading typographical firms. She also holds an M.F.A. in Painting/Sculpture/Graphic Arts from the University of California at Los Angeles. Kathy paints watercolors, designs letterforms, plays piano, and reads feminist film criticism.

Like Kathy, Karl Berry received his M.S. degree in computer science from the University of Massachusetts, Boston, in August 1989. He also worked for the Free Software Foundation, did research with Morris, and has studied with Adleta, Gürler, and Mengelt. He has been working with T_EX since 1983 and has installed and maintained the

T_EX system at a number of universities. He was the maintainer of the Web2c system developed by Tim Morgan for a number of years, among other T_EX projects. He became the president of the T_EX Users Group in 2003.

书尾

This book was composed using Te_X (of course), developed by Donald E. Knuth. The main text is set in Computer Modern, also designed by Knuth. The heads of the original book were set in Zapf Humanist (the Bitstream version of Optima), designed by Hermann Zapf.

The paper was Amherst Ultra Matte 45 lb. The printing and binding were done by Arcadia Graphics-Halliday. The phototypeset output was produced at Type 2000, Inc., in Mill Valley, California. Proofs were made on an Apple LaserWriter Plus and on a Hewlett Packard LaserJet II.

Cross-referencing, indexing, and the table of contents were done mechanically, using the macros of 第 12 章 together with additional macros custom-written for this book. The production of the index was supported by an additional program written in Icon.

概念列表

活动字符 46	编组 68	页面 83
阵列 47	水平盒子 69	分页点 84
T _E X 剖析 49	页眉 69	页面构建器 85
参量 50	高度 69	页面布局 85
ASCII 51	水平列表 69	段落 85
赋值 51	水平模式 69	参数 86
劣度 52	连字 70	惩罚 87
基线 52	输入流 70	plain T _E X 87
盒子 53	插入项 70	原始的 87
类别码 54	行间粘连 71	基准点 87
字符 56	项目 71	寄存器 87
类 57	对齐文本 71	受限模式 88
命令 57	紧排 71	标线 89
条件测试 58	指引线 72	标号尺寸 89
控制序列 58	连写 73	小标号尺寸 89
控制符 59	断行点 74	收缩量 90
控制词 59	列表 75	间隔 90
小数 59	日志文件 75	伸长量 90
定界符 59	宏 75	支架 90
缺陷 61	放大率 77	样式 91
深度 61	边距 78	T _E X M _E X 91
尺寸 61	标记 79	文内公式 92
陈列公式 62	数学模式 79	文本尺寸 92
转义符 62	数学码 80	记号 92
族 62	数学单位 81	度量单位 92
文件 63	模式 81	竖直盒子 92
文件名 64	数学粘连 81	竖直列表 92
字体 64	数 81	竖直模式 92
页脚 65	普通模式 82	小玩意 93
格式文件 65	外部的 82	宽度 93
全局的 66	输出例行程序 83	
粘连 66	输出流 83	