TEX 主题选讲

TEX 技术员手册

Victor Eijkhout
翻译：LiYanrui, zoho, Liam0205

document revision 1.4, December 2013

This document is based on the book TEX by Topic, copyright 1991-2008 Victor Eijkhout. This book was printed in 1991 by Addison-Wesley UK, ISBN 0-201-56882-9, reprinted in 1993, pdf version first made freely available in 2001.

Cover design (lulu.com version): Joanna K. Wozniak (jokwoz@gmail.com)

目录

GNU 自由文档许可证

Version 1.2, November 2002

0. 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 world-wide, 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 in-

put 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, LaTeX 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.

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 for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

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.

前言

对于偶然了解到 TEX 的人来说，它并不是时下最先进的排版系统。它没有时髦的多级菜单，也没有对文本图片交互式的操作方式。然而，在更深层次，TEX 是一个精致的程序：首先是因为，TEX 内置的断行算法和数学公式排版算法十分精巧；其次是因为，它几乎是完备程序性的。这些因素使得 TEX 可以高度自动化地实现几乎所有能想到的排版布局。

不幸的是，这也意味着 TEX 有非常多的命令和参数；因此进行 TEX 编程远谈不上简单。每一个想要进行 TEX 编程的人，大概都需要两本书：一本新手引导，介绍 TEX 的基本内容；以及一本大而全的手册。本书属于后者，适合于入门后的 TEX 用户（无论阅读了市面上多少本入门书）。

本书将 TEX 划分为大约 40 个主题，每个主题一章。在每章开头，本书会列出和当前章节有关的控制序列。大多数章节都有内容介绍和示例参考。

大体上说，本书的结构如下。首先讲解基本机制，而后介绍有关文本和数学处理的内容，最后会有几章介绍 TEX 与外界的联系。借助示例、控制序列和主题，本书也包含了有关 TEX 命令、表格、索引的术语表。根据主题，大多数概念可以索引至具体一页；在这一页上，读者能找到有关该主题的大部分内容，也能了解去何处查阅相关信息。

本书没有涵盖任何具体的 TEX 宏包。有关 plainTEX 格式的内容，本书也只介绍其中最核心的部分：它们 LATEX 等格式的一部分。因此，大多数关于 plainTEX 格式的内容对于 LATEX 和其他格式也仍然适用。特别地，如果文中明确指出相关段落是对 plainTEX 的介绍，那么这些内容是有关纯 IniTEX 的，而与 LATEX 无关。为方便讲述，本书偶尔也会介绍一些 plainTEX 的非核心宏。

致谢

承蒙 Barbara Beeton、Karl Berry 以及 Nico Poppelier 阅读本书初稿。在他们的帮助下，本书的陈述有所提高。同时感谢以下讨论列表中的参与者：TEXhax、TEX-nl 以及 `comp.text.tex`。他们提出的问题和给出的回答给予我思考之食粮。最后，任何一本有关 TEX 的书，都应向 TEX 的发明者 Donald Knuth 致敬。本书也不例外。

Victor Eijkhout

厄巴纳, 伊利诺伊州, 1991 年 8 月
诺克斯维尔, 田纳西州, 2001 年 5 月
奥斯汀, 德克萨斯州, 2013 年 12 月

第 1 章　TEX 处理器的结构

本书涵盖了 TEX 诸多方面的内容；其中每一章都讲述一个相对较小且易于描述的主题。本章将简介 TEX 处理文档的基本流程。当然，本章会忽略很多细节，留待后续章节继续讨论。相应的，为了说明问题，本章会给出一些例子；这些例子在后续的这些章节中会继续讨论。

1.1　TEX 的 4 个处理器

TEX 处理输入的过程可分为 4 个阶段。你可以认为 TEX 处理器（也称 TEX 引擎）有 4 个独立的单元，每个单元接收前序步骤的输出，并将输出作为后续单元的输入。第一阶段的输入是 `.tex` 文件，最后一个阶段的输出是 `.dvi` 文件[1]。

通常而言，这样理解上述 4 个阶段方便且直观：每个阶段完整地接收上一个阶段的输出。然而，这样并不完全正确：4 个阶段的处理是同时进行的，阶段之间也有交互[2]。

4 个阶段列举如下（按 Knuth 最初的叫法，分别是「眼睛」、「嘴巴」、「胃」和「肠道」）：

1. 输入处理器（input processor）：负责从运行 TEX 的计算机的文件系统中读入输入，并处理为记号。记号是两种 TEX 内部对象的统称：构成排版文本的字符记号、待后续两个阶段作为命令处理的控制序列记号。
2. 展开处理器（expansion processor）：负责展开输入处理器输出的部分记号：宏、条件式、以及部分 TEX 原语。记号展开的过程会将一些（连续的）记号删除或替换为其它记号。
3. 执行处理器（execution processor）：不可展开的控制序列即是可执行的；执行处理器负责执行这些可执行的控制序列。

   执行处理器的部分活动涉及到 TEX 的内部状态之变化：赋值（包括宏定义）是其中典型。除此之外，执行处理器负责构造水平列表、垂直列表和数学列表。

[1] 译注（Liam0205）：此处讨论的 TEX 是 Knuth TEX。后续扩展的 pdfTEX 等不一定输出 `.dvi` 文件。
[2] 译注（Liam0205）：流式处理，上一阶段处理完多少，下一个阶段就接着继续处理这部分。

4. 可视化处理器（visual processor）：TEX 的最后一个阶段负责排版内容的可视化。可视化处理器将水平列表分行成段，将垂直列表分割成页，将数学列表构建成数学公式；最终输出 .dvi 文件[3]。可视化处理器的内部算法对于用户是不可见的；不过用户可以通过一系列参数来控制。

1.2 输入处理器

输入处理器是 TEX 的一部分，负责将 TEX 从输入文件中读入的内容翻译成记号。它的输出是记号流（stream of tokens）：记号组成的列表。大部分记号属于以下两类：字符记号和控制序列记号。还有一类记号是参数记号，留待后续章节讨论。

1.2.1 字符的输入

对于简单的输入文本，其中的字符被输入处理器翻译为字符记号。在某些情况下，TEX 会忽略一些输入字符；例如，连续空格通常等价于单个空格。此外，TEX 也会向记号流中插入一些与输入字符无关的记号。例如在每一行的末尾插入一个空格，以及在每一个空行末尾插入一个 \par 记号。

并非所有的字符记号都会被最终排版出来。事实上，在 TEX 中，字符被分类为 16 类 —— 每一类都有特定的功能 ——而只有两类最终会被排版出来。其它字符类，像 {、}、& 和 #，都不会被排版出来。也就是说，在 TEX 中，字符记号的含义由字符编码（通常是 ASCII 编码）和分类码这一对数字共同决定。而具体某一编码的字符对应的分类码是可以修改的。

当输入处理器遇到转义字符时（默认是 \），它会将转义字符之后的若干连续个字符拼在一起，形成一个控制序列记号。

在分类码的影响下，TEX 输入处理器可视作一个在不同内部状态间切换的机器。它有三种内部状态：N，新行；M，行内；S，忽略空格。第 2 章将讨论这些状态及其切换。

1.2.2 输入处理器的两个阶段

TEX 的输入处理器本身也可分为两个阶段。因为一些限制（来自终端、编辑器或者操作系统），某些字符可能很难输入。为此，TEX 设计了一套机制来表达这些字符：以两个连续的上标字符来获取所有可用字符。这套机制可视作输入处理器中相对独立的阶段；它运行于上述三状态机之前。

例如，由于 k 和 + 的 ASCII 码之差为 64，所以 ^^+ 会被输入处理器等价地转换为 k。由于这种替换机制发生在形成记号之前，所以输入 \vs^^+ip 5cm 与输入

[3]译注（zoho）：对于现代 TEX 处理器而言，还可以是 pdf 和 xml 等文件。

\vskip 5cm 是等价的。某些情况下，这种用法是很有用的。

注意，这是 TEX 输入处理器的第一阶段，它不考虑分类码，而只是一个从字符到字符的变换。分类码只在输入处理器的第二阶段才起作用：字符的编码和分类码组合在一起，变成一个字符记号。

1.3 展开处理器

TEX 的展开处理器接受记号流，并将记号流里的记号逐一展开，直到再无可展开记号为止。宏展开即是其中一例：如果一个控制序列是一个宏名，则根据定义，它将（和可能的参数记号一起）被替换。

展开处理器的输入主要来自输入处理器。自输入处理器而来的记号流被展开处理器展开成仅含有不可展开记号的记号流，最终流向执行处理器。

此外，在处理类似 \edef 及 \write 命令时，展开处理器也会参与工作。这些命令的参数记号会被直接展开，变成顶层的替换文本，而不是这些命令的参数。

1.3.1 记号的展开过程

记号展开的步骤如下：

1. 确认当前待处理的记号是否可展开。
2. 若当前待处理的记号不可展开，则直接将其加入正在构建的记号列表中，并读入下一个记号。
3. 若当前待处理记号可展开，则将其替换为其展开。对不带参数的宏以及一些 TEX 原语（例如 \jobname），该过程只需简单地进行记号替换即可。不过，通常 TEX 需要从输入流中读取一些参数记号，以便构造当前待处理记号的替换形式。举例说，对于带参数的宏，TEX 将读取足够的参数（parameters）记号，而后将这些参数记号作为宏的参数（arguments）。
4. 将展开结果的第一个记号作为待处理记号，继续展开。

宏的可展开性之判断很简单：所有的宏、活动字符、条件语句以及部分 TEX 原语（见第 ?? 页的列表）是可展开的，其他记号都不可展开。因此，展开处理器会将宏替换为其展开，并对条件语句求值，而后忽略条件为假的部分；诸如 \vskip、字符记号（包括美元符号和花括号）则原封不动地传到输出流。

1.3.2 几个特例：\expandafter、\noexpand 及 \the

如前所述，记号被展开后，TEX 会继续展开其展开后得到的记号[4]。

[4] 译注（Liam0205）：展开处理器负责「展开到底」，但展开结果中可能存在需要继续展开的记号。

\expandafter 是一个特例。初看之下，它只做一步展开，因而破坏了这个规则。例如说，

\expandafter⟨token$_1$⟩⟨token$_2$⟩

会被替换为

⟨token$_1$⟩⟨*expansion of* token$_2$⟩

但实际上，这个替换结果还会被展开处理器再次处理[5]。

\noexpand 是另一个特例。当 TEX 遇到 \noexpand，则其下一个记号会被展开处理器当做是不可展开的。因此，展开处理器会把 \noexpand 的下一个记号当做是 \relax 那样，直接将其加入正在构建的记号列表。

举例说，以下宏定义中，其替换文本 \noexpand\b 会在宏定义时被展开：

```
\edef\a{\noexpand\b}
```

\noexpand 展开后，其后的记号暂时会被当做是 \relax 的意思。于是，当展开处理器处理下一个记号 \b 时，认为它是不可展开的，并将其直接加入正在构建的记号列表中。因此 \b 就是这个宏的替换文本。

\the 也是一个特例。在 \edef 宏定义中，\the⟨token variable⟩ 的展开结果不会被继续展开。

1.3.3 展开处理器中的花括号

之前提到，展开处理器将花括号视为不可展开的字符记号。通常而言，这没错。例如说 TEX 在展开下列 \romannumeral 控制序列时，会一直展开，直到遇见花括号：

```
\romannumeral1\number\count2 3{4 ...
```

因此，如果 \count2 的值是 0，那么这个控制序列的展开结果是 103 的罗马数字表示。又例如说，展开处理器对待

```
\iftrue {\else }\fi
```

的方式与对待

\iftrue *a*\else *b*\fi

的方式完全一致；即：此时 { 只是一个字符记号，与它的分类码无关。

不过，在宏展开的上下文中，展开处理器会识别和处理花括号。首先，配对的花括号内可用于将其内的一组记号标记为宏的参数。例如，下面定义的单参数宏：

```
\def\macro#1{ ... }
```

你可以单传一个参数来调用它：

```
\macro 1 \macro \$
```

[5] 译注（Liam0205）：因而不算完全破坏了规则。

也可以使用配对的花括号包裹一组记号作为宏的参数：

```
\macro {abc} \macro {d{ef}g}
```

其次，对于带参数的宏，其参数中不得包含未配对的花括号。例如，如下定义的 \a 带有一个参数：包含从 \a 之后开始，直到遇见第一个花括号外的 \stop 位置的全部记号。

```
\def\a#1\stop{ ... }
```

对于以下用法，\a 的参数应当是 bc{d\stop}e；因为宏的参数中花括号必须是配对的。

```
\a bc{d\stop}e\stop
```

1.4 执行处理器

执行处理器构建水平列表、竖直列表和数学列表。与之相应，执行处理器有三种工作模式：水平模式、竖直模式、数学模式。这三种模式又分别有「内部」和「外部」两种类型。除了构建列表，执行处理器还需要处理一些与模式无关的操作，例如赋值。

执行处理器的输入是展开处理器输出的不可展开记号组成的流。在执行处理器看来，这条记号流中有两种类型的记号：

- 触发赋值操作（包括宏定义）的记号，以及触发与模式无关的其他操作的记号（例如 \show 和 \aftergroup）。
- 构建列表的记号：字符、盒子、粘连。执行处理器对它们的处理方式根据所处模式的不同而改变。

有些记号在所有模式下都可用；比如盒子可用于水平、竖直、数学三种模式。当然，这些记号的作用与效果，根据所处的模式不同而不同。也有记号只在特定模式下可用。例如说，字符记号（确切说是分类码为 11 和 12 的字符记号）则与水平模式密切相关：当执行处理器在竖直模式中遇到字符记号时，会切换到水平模式继续工作。

并非所有字符记号都是可排版的。执行处理器可能遇到数学模式切换标志（默认是 $）、分组起止符（默认是 { 和 }）。当执行处理器遇到数学模式切换标志时，它会进入或退出数学模式；而分组起止符则让执行处理器进入或退出新的一层分组。

控制序列 \relax 有些特别：该控制序列是不可展开的，同时执行时啥也不做。为说明 \relax 的作用，可与 \empty 进行比较。\empty 的定义如下：

```
\def\empty{}
```

使用 \relax 时，因其不可展开，故而下例中计数寄存器被赋值为 1：

```
\count0=1\relax 2
```

使用 \empty 时，因 \empty 的展开结果为空，故而下例中计数寄存器被赋值为 12：

```
\count0=1\empty 2
```

1.5 可视化处理器

TEX 的可视化处理器使用了若干用户不可直接控制的算法：断行、切齐、分页、数学排版以及 dvi 文件生成算法。尽管用户不可直接控制这些算法，但可通过一些参数间接地控制它们。

这些算法当中，部分算法的输出结果可被执行处理器继续处理。例如说，分段成行得到一系列行组成的列表；这些行会被加入主竖直列表当中；而每一行又是由若干水平盒子及其中的行间粘连和惩罚组成的。又例如说，分页算法将其结果保存在 \box255 当中；而后交由输出例程继续处理[6]。其余算法的输出结果则不然，例如数学公式不可以分解，又例如输出至 dvi 文件的盒子也是不可逆的。

1.6 示例

1.6.1 被忽略的空格

被忽略的空格可以反映数据在 TEX 各处理器之间的流动情况。例如：

```
\def\a{\penalty200}
\a 0
```

展开的结果不是（设置惩罚项为 200，并排版数字 0）

```
\penalty200 0
```

而是

```
\penalty2000
```

这是由于输入处理器会忽略 \a 后的空格，因此展开处理器的输入流中的内容是：

```
\a0
```

1.6.2 内部量值及其表示

TEX 使用了多种内部量，比如说整数和尺寸。这些内部量的外部表示是同一的：字符组成的字符串。例如 4711 和 91.44cm。

内部量与外部表示之间的转换发生在执行处理器或展开处理器中。具体来说，外部表示向内部量的转换发生在执行处理器中：

```
\pageno=12 \baselineskip=13pt
\vskip 5.71pt
```

内部量向外部表示的转换发生在展开处理器中：

[6] 译注（Liam0205）：最简单的是 \output{\shipout\box255}。

```
\number\pageno \romannumeral\year
\the\baselineskip
```

最后再举一例。假设 \count2=45，则下列代码

```
\count0=1\number\count2 3
```

首先将 \number\count2␣ 被展开为字符串 45，注意 \count2 后的空格被用于界定计数器编号而已被展开。因此，下一级处理时，执行处理器看到并执行的是：

```
\count0=1453
```

第 2 章　分类码与内部状态

读取字符时，TEX 的输入处理器会为字符分配分类码。根据读取到字符的分类码，输入处理器会在三种内部状态之间切换。本章讨论 TEX 是如何读取字符的，以及字符的分类码是如何影响读取行为的。本章还将讨论空格及行尾[1]的相关问题。

`\endlinechar` 该命令是输入处理器添加至输入行尾的行终止符的字符编码。在 IniTEX 中默认是 13。

`\par` 该命令结束当前自然段，并使 TEX 进入竖直模式。输入处理器会将（连续或单个的）空行转换成它。

`\ignorespaces` 该命令展开时读取连续的空格，直到遇见非空格记号（non-⟨space token⟩）后停止展开。

`\catcode` 该命令用于查询或者设置分类码。

`\ifcat` 该命令用于检测两个字符的分类码是否相同。

`\␣` 控制空格。该命令插入一个空白，其宽度与 `\spacefactor` = 1000 时空格记号之宽度相等。

`\obeylines` 该 plain TEX 宏将使源文件中的行尾符的换行效果体现在排版结果中。

`\obeyspaces` 该 plain TEX 宏将使源文件中（大多数）空格体现在排版结果中。

2.1　概述

TEX 的输入处理器从文件或终端扫描输入行，而后将读取到的字符转换成记号。输入处理器可视作一简单的有限状态自动机，其具有三种内部状态；根据输入处理器所处内部状态的不同，其扫描行为有所不同。本章将分别从内部状态和控制内部状态转换的分类码两个角度来考察该自动机。

[1] 译注（Liam0205）：在本文的翻译中：行尾（line ends）是一行末尾及相关问题的统称，行终止符（end-of-line character）是 TEX 的输入处理器主动添加在输入行末尾的字符，行尾符是操作系统中用于标识一行结尾的字符，例如：回车符（carriage return）与换行符（line feed）。

2.2 初始化处理

TEX 逐行处理来自文件的输入（也可能是来自终端的输入，但实际甚少使用，故不再提及）。此处首先讨论在 TEX 语境下，到底什么是「输入行」。

不同计算机系统对输入行的具体定义有所不同。最常见的方式是用回车符（carriage return）紧跟换行符（line feed）作为行尾符，有些系统单用换行符作为行尾符，一些有定长存储（块存储）的系统则根本不使用行尾符。因此，TEX 在结束一行输入时有自己特定的处理方式。

1. 从输入文件读取一行输入行（不包含可能的行尾符）。
2. 移除行尾空格（这是针对块存储系统设计的，同时避免了因行为空格不可见而导致的混乱）。
3. 将编码为 `\endlinechar` 的行终止符（默认是 ASCII 编码为 13 的 ⟨return⟩）添加在行尾。若 `\endlinechar` 的值为负或大于 255（在低于 TEX3 的版本中则是 127；**??** 页介绍了更多差异），则输入处理器不会添加任何行终止符；在输入行尾添加注释字符也有相同的作用。

不同计算机在字符编码方面也存在差异（最常见的是 ASCII 和 EBCDIC）。因此，TEX 有必要将从文件读入的字符转换为其内部编码。这些编码仅在 TEX 内适用，因此 TEX 在任何操作系统上的行为都保持一致。更多内容详见第 3 章。

2.3 分类码

256 个字符编码（0–255）中的每一个都关联了一个不尽相同的分*类码*。TEX 的分类码共有 16 个，从 0 开始编号至 15。在扫描输入流的过程中，TEX 会生成由字符编码和分类码组成的字符编码-分类码配对（character-code–category-code pairs）；而后，基于这些配对，输入处理器将它们处理成字符记号、控制序列记号和参数记号。这些记号随后被传给 TEX 的展开处理器和执行处理器。

字符记号是简单的字符编码-分类码配对，它们会直接被传给展开处理器。控制序列记号则由转义字符引导，后接一个或多个字符组成。关于控制序列记号和参数记号的介绍详见下文。

以下就这些分类做简单说明，详细的阐述则散布在本章其他位置及后续章节当中。

0. 转义字符；用于标记控制序列的开始。IniTEX 默认使用反斜线 \ 作为转义字符（ASCII 码为 92）。
1. 分组开始符；TEX 遇到此类字符时，会进入新的一层分组。在 plain TEX 中，默认的分组开始符是左花括号 `{`。

2. 分组结束符；TEX 遇到此类字符时，会关闭并从当前分组中退出。在 plain TEX 中，默认的分组开始符是左花括号 }。
3. 数学切换符；此类字符是数学公式的左右定界符。在 plain TEX 中，默认的数学切换符是美元符号 $。
4. 制表符；在 \halign（\valign）制作的表格中，作为列（行）间分隔符。在 plain TEX 中，默认的制表符是与符号 &。
5. 行终止符；TEX 用来表达输入行结束的字符。IniTEX 将回车符 ⟨return⟩（ASCII 编码为 13）作为默认的行终止符。这就是为什么 IniTEX 中，\endlinechar 的值是 13（详见前文）。
6. 参数符；用于表示宏的参数。在 plain TEX 中，默认的参数符是井号 #。
7. 上标符；在数学模式中表示上标；也可用于在输入文件中表示无法直接输入的字符（详见后文）。在 plain TEX 中，默认的上标符即是 ^。
8. 下标符；在数学模式中表示下标。在 plain TEX 中，默认的下标符是下划线 _。
9. 被忽略字符；此类字符将被 TEX 自输入流中清除，因此不会影响后续处理。在 plain TEX 中，默认将空字符 ⟨null⟩（ASCII 编码为 0）设置为被忽略字符。
10. 空格符；TEX 对待空格符的方式较为特殊。IniTEX 将空格 ⟨space⟩（ASCII 编码为 32）作为默认的空格符。
11. 字母；IniTEX 默认只将 a ... z 和 A ... Z 分为此类。在宏包中，某些「隐秘」字符（例如 @）会被暂时分为此类[2]。
12. 其他字符；IniTEX 将所有未归于其他类的字符归于此类。因此，数字和标点都属于此类。
13. 活动字符；活动字符相当于一个无需转义字符前导的 TEX 控制序列。在 plain TEX 中，只有带子 ~ 是活动字符，表示不可断行的空格（参见第 ?? 页）。
14. 注释符；TEX 遇见注释符后，会将从注释符开始到输入行尾的所有内容视作注释而忽略。在 IniTEX 中，默认的注释符是百分号 %。
15. 无效字符；该分类包含了不应在 TEX 中出现的字符。IniTEX 将退格字符（ASCII 编码为 127）⟨delete⟩ 归于此类。

用户可使用 \catcode 命令修改字符编码到分类码的映射（见第 ?? 章对诸如 ⟨equals⟩ 等概念的解释）：

\catcode⟨number⟩⟨equals⟩⟨number⟩.

该语句中，第一个参数可用如下方式给出：

`⟨character⟩ 或 `\⟨character⟩

[2]译注（Liam0205）：此类字符的分类码在普通用户文档中和在宏包中会有不同。在宏包中暂时分为此类，以起到「提示当前控制序列是内部的」这样的作用。

两种写法都表示该字符的字符编码（见第 38 页和 ?? 页）。

plain TEX 格式使用 \chardef 命令（将在第 40 页和 ?? 页介绍）将 \active 定义为：

```
\chardef\active=13
```

因此上述语句可写成这样：

```
\catcode`\{=\active
```

LATEX 格式定义了如下两个控制序列，用于开启或关闭「隐秘字符」@（详见下文）：

```
\def\makeatletter{\catcode`@=11 }
\def\makeatother{\catcode`@=12 }
```

使用 \catcode 命令查询字符编码对应的分类码，可得到一个数字：

```
\count255=\catcode`\{
```

在例中，{ 的分类码被保存在第 255 号 \count 寄存器中。

下列语句可用于检测两个记号的分类码是否相等：

\ifcat⟨token$_1$⟩⟨token$_2$⟩

无论 \ifcat 后有什么，TEX 都会将其展开，直至发现两个不可展开的记号；而后，TEX 将比较这两个记号的分类码。控制序列的分类码被视为 16；因此，所有控制序列的分类码都是相等的，而与所有字符记号的分类码都不相等。条件式的详细介绍见第 ?? 章。

2.4 从字符到记号

从文件或用户终端扫描输入行后，TEX 的输入处理器会将其中的字符转换为记号。记号共有三种。

- 字符记号：字符记号会被打上相应的分类码，而后直接传给 TEX 的后续处理器。
- 控制序列记号：严格来说，控制序列记号分为两种。其一是*控制词* —— 分类码为 0 的字符后紧跟一串字母（分类码是 11）。其二是*控制字符* —— 转义字符后紧跟单个非字母字符（分类码不是 11）。在无需区分控制词和控制字符的场合，它们统称为*控制序列*。

  由转义字符与一个空格字符 \␣ 构成的控制序列，称为*控制空格*。
- 参数记号：由一个参数符 —— 分类码为 6，默认为 # ——和一个紧跟着的 1..9 中的数字构成。参数记号只能在宏（见第 ?? 章）的上下文中出现。

  连续两个参数符（字符编码不一定相同）会被替换为单个字符记号。该字符记号的分类码是 6（参数符），字符编码则与上述连续两个参数符中后者的字符编码相同。最常见的情形是 ## 会被替换为 #$_6$。此处下标表示分类码。

2.5 输入处理器是有限状态自动机

TEX 的输入处理器有三种内部状态，可看做是一个有限状态自动机。这也就是说，在任意瞬间，TEX 的输入处理器都处于并且只能处于三种状态的一种；并且在状态切换完成后，TEX 的输入处理器对先前的状态没有任何记忆。

2.5.1 状态 *N*：新行

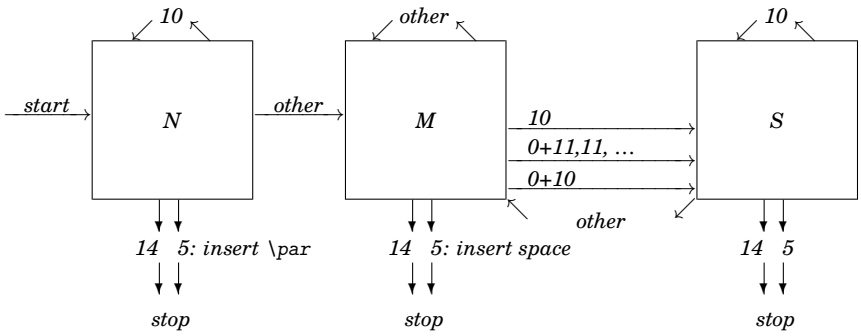
当且仅当遇到新的输入行时，TEX 会进入状态 *N*。在该状态下，TEX 遇到空格记号（分类码为 10 的字符）即会将之忽略；遇到行终止符则会将之替换为 `\par` 记号；遇到其它记号，则会切换到状态 *M*。

2.5.2 状态 *S*：忽略空格

在状态 *M* 下遇到空格记号，或在任意状态下遇到控制词或控制空格之后（注意其他控制字符不在此列），TEX 会进入状态 *S*。在该状态下，TEX 遇到空格记号或行终止符都会忽略。

2.5.3 状态 *M*：行内

显然，状态 *M* 是输入处理器最常见的状态，它表示「处理到输入行的中间」（middle of line）。当输入处理器遇到分类码为 1–4、6–8 以及 11–13 的字符或者控制字符（不包括控制空格）之后，就会进入该状态。在该状态下，TEX 会将行终止符替换为空格记号。



2.6 访问整个字符集

大体上，TEX 的输入处理器可以认为是一个有限状态自动机，但严格来说它并不是。输入处理器在扫描输入行期间，为了让用户能够输入一些特殊字符，而设计了这

样的机制：两个相同的上标符（分类码为 7）以及一个字符编码小于 128 的字符（暂称原字符）组成的三元组会被替换为一个新的字符。该字符的编码位于 0 – 127 之间，并且与原字符的编码相差 64。

这种机制可用于访问字体中难以输入的字符。例如 ASCII 中的控制符号 ⟨return⟩（ASCII 编码为 13）和 ⟨delete⟩（ASCII 编码为 127）可分别使用 `^^M` 和 `^^?` 进行访问。当然，由于 `^^?` 是无效字符（分类码是 15），故而在访问前需要先修改其分类码。

在 TEX3 中，该机制被扩展为可以访问 256 个字符：四元组 `^^xy` 会被替换为一个编码在 0 – 255 之间的字符；其中 x 和 y 是小写十六进制数字 `0–9, a–f`，而 `xy` 正是该字符编码的十六进制表示。这一扩展也给先前的机制带来了一些限制：例如 `^^7a` 会被输入处理器替换为 `z`，而不是 `wa`[3]。

这种机制一方面使得 TEX 的输入处理器在某种意义上比真正的有限状态自动机更为强大，另一方面还不会影响其余的扫描过程。因此，在概念上，可以简单地假装认为这种对 `^^` 引导的三元组或四元组的替换是提前进行的。不过，在实践中这样做是不可能的。这是因为，在处理输入行的过程中，用户可能将其他字符分类为第 7 类，从而影响后续处理[4]。

2.7 内部状态切换

现在我们来讨论特定分类码的字符对 TEX 输入处理器内部状态的影响。

2.7.1 0：转义字符

遇到*转义字符*后，TEX 开始构建控制序列记号。取决于转义字符后面的字符之分类码，所得的控制序列记号有三种类型。

- 若转义字符后的字符之分类码为 11，即字母，则 TEX 将转义字符和之后连续的分类码为 11 的字符构建成一个控制词，而后进入状态 *S*。
- 若转义字符后的字符之分类码为 10，即空格，则 TEX 将它们构建成名为控制空格的控制字符，而后进入状态 *S*。
- 若转义字符后的字符之分类码不是 10 也不是 11，那么 TEX 将它们构建成控制字符，而后进入状态 *M*。

控制序列名字的所有字符必须在同一输入行之中；控制序列的名字不能跨行，即使当前行以注释符结尾或者没有行终止符（通过将 `\endlinechar` 设置为 0 – 255 之外的值）。

[3] 译注（Liam0205）：`w` 和 7 的 ASCII 编码之差为 64。由于 `7a` 可被理解为是一个十六进制数，所以 TEX 贪婪地将四元组看做一个整体替换为 `z`。

[4] 译注（Liam0205）：也就是说，如果没有其他字符被分类为第 7 类，则这个假设在实践中也是可行的。

2.7.2 1–4, 7–8, 11–13：非空字符

分类为 1–4、7–8 及 11–13 的字符会被转换为字符记号，而后 TEX 进入状态 *M*。

2.7.3 5：行终止符

遇到行终止符时，TEX 的行为取决于输入处理器当前的状态。但不论处于何种状态，TEX 会忽略当前行[5]，而后进入状态 *N* 并开始处理下一行。

- 处于状态 *N*，即当前行在此前只有空格，TEX 将插入 \par 记号；
- 处于状态 *M*，TEX 将插入一个空格记号；
- 处于状态 *S*，TEX 将不插入任何记号。

此处「行终止符」指得是分类码为 5 的字符。因此，它的字符编码不一定是 \endlinechar，也不一定非得出现在行尾。详见后文。

2.7.4 6：参数符

在宏定义的上下文中，*参数符* —— 通常为 # —— 可跟 1..9 中的数字或另一个参数符。前者产生参数记号，而后者产生单个参数字符记号待后续处理。在这两种情形中，TEX 都会进入状态 *M*。

单独出现的参数符也被用于阵列的模板行（见第 ?? 章）。

2.7.5 7：上标符

TEX 对上标符的处理和大多数非空字符一样，仅在上述替换机制中有所不同：连续两个字符编码相同的上标符及其后字符组成的三元组或四元组会按规则被替换为其它字符。

2.7.6 9：被忽略符

分类码为 9 的字符会被忽略，且不会影响 TEX 的状态。

2.7.7 10：空格符

在状态 *N* 和状态 *S* 中，不论字符编码是多少，空格记号 —— 分类码为 10 的记号 —— 都会被忽略；同时 TEX 的状态保持不变。在状态 *M* 中，TEX 会向正在构建

[5] 译注（Liam0205）：指源文件中的当前行。

的记号序列中插入 $\sqcup_{10}$（ASCII 编码中的空格，编码为 32），并进入状态 S。这意味着空格记号的字符编码可能与输入字符的编码不同[6]。

2.7.8 14：注释符

TEX 遇到注释符后，会忽略当前行之后包括注释符本身在内的所有内容。特别地，TEX 会忽略行终止符。因此，哪怕是在状态 M 下，TEX 也不会插入额外的空格记号。

2.7.9 15：无效字符

TEX 遇到无效字符时会报错，而 TEX 自身会停留在之前的状态。不过，在控制字符的上下文中，无效字符是合法的。因此，`\^^?` 不会触发报错。

2.8 分类码中的字母与其他字符

大部分编程语言的标识符可由字母与数字构成（还可能包含其他诸如下划线之类的字符）。但是，在 TEX 中，控制序列的名字只能由第 11 类字符（即字母）组成。而通常，数字和标点的分类码是 12，即其他字符。

此外，TEX 可以产生一些由第 12 类字符组成的字符串，哪怕其中的字符原本并非都是第 12 类字符。此类字符串可由 `\string`、`\number`、`\romannumeral`、`\jobname`、`\fontname`、`\meaning` 以及 `\the` 等命令生成。若这些命令产生的字符串包含空格字符（ASCII 编码为 32）[7]，则在输出的字符串中，该字符的分类码为 10。

在极个别情况下，控制序列的展开中可能会包含十六进制数字；因此，除了通常表示字母的 $\mathtt{A}_{11}-\mathtt{F}_{11}$ 之外，TEX 中还有表示十六进制数字的 $\mathtt{A}_{12}-\mathtt{F}_{12}$。

举例来说，

`\string\end` 得到四个字符记号 $\backslash_{12}\mathtt{e}_{12}\mathtt{n}_{12}\mathtt{d}_{12}$

注意，此处输出中有*转义字符* $\backslash_{12}$ 的原因是宏 `\escapechar` 的值是反斜线的字符编码。而若将 `\escapechar` 的值改为其它字符的编码，则 `\string` 将输出另一个字符。有关 `\string` 命令的详细内容参见第 3 章。

通过一些特殊技巧，空格也可以出现在控制序列的名字当中：

```
\csname a b\endcsname
```

[6] 译注（Liam0205）：不论输入的是哪一个分类码为 10 的字符，输入处理器都会将其替换为字符编码为 32 的 ASCII 空格。

[7] 译注（Liam0205）：注意，此处说的是空格字符，而非是 TEX 的空格记号。前者讨论的是字符编码，而后者讨论的是分类码。

是一个控制序列记号，其名称由三个字符组成，并且其中之一是空格符。将这个控制序列转化为字符串

```
\expandafter\string\csname a b\endcsname
```

可得 $\backslash_{12}a_{12}\sqcup_{10}b_{12}$。

举个更加实用的例子。假设有一系列输入文件：file1.tex、file2.tex，而我们希望写一个宏来输出当前正在处理的文件的序号。第一种解法是：

```
\newcount\filenumber
\def\getfilenumber file#1.{\filenumber=#1 }
\expandafter\getfilenumber\jobname.
```

宏定义中，参数文本中的 file（见第 ?? 节）会吸走 \jobname 中的 file 部分，从而留下文件编号作为宏的参数。

但这段代码有些小问题。\jobname 输出的 file 四个字符，其分类码为 12。但在 \getfilenumber 的定义中，file 四个字符的分类码是 11。为此，需要对上述代码进行以下修正：

```
{\escapechar=-1
 \expandafter\gdef\expandafter\getfilenumber
       \string\file#1.{\filenumber=#1 }
}
```

此处，\escapechar=-1 让 \string 忽略反斜线；因此 \string\file 的结果会是 $f_{12}i_{12}l_{12}e_{12}$ 四个字符。为了在宏定义是得到分类码为 12 的四个字符，我们使用 \expandafter 命令让 \string\file 在宏定义之前先行展开；而由于 \escapechar 的设定被放在分组内部，所以我们需要使用 \gdef 进行宏定义。

2.9 \par 记号

在遇到空行之后，也就是在状态 *N* 遇到行终止符（分类码为 5）之后[8]，TEX 会向输入中插入一个 \par 记号。具体来说，由于 TEX 遇到任何非空格字符，都会从状态 *N* 转移走，因此空行只能包含分类码为 10 的字符。特别地，空行不能以注释符结尾[9]。因此，若输入文件中因格式美观需要保留空行，则可以在该行中放一个注释符。这算是 TEX 这一特性的常见用法。

两个连续的空行产生两个连续的 \par 记号，而实际上它们等同于一个 \par 记号：在遇见第一个 \par 记号之后，TEX 会进入竖直模式，而在竖直模式中，\par 只是充当 TEX 页面构建器，起到清空段落形状参数的作用。

TEX 于非受限水平模式（unrestricted horizontal mode）遇到竖直命令（⟨vertical

[8] 译注（Liam0205）：此处说的是 TEX 添加的行终止符，而不是输入文件中的行尾符。输入文件中的行尾符已在初始化处理中被移除并替换成了行终止符。

[9] 译注（Liam0205）：此时，TEX 添加的行终止符位于注释之后，故而该行终止符会被 TEX 忽略。

command⟩)时，也会向输入插入一个 \par 记号。当该 \par 被读取和展开后，上述竖直命令会被重新处理(详见第 ?? 章和 ?? 章)。

\end 命令[10]也会向输入插入 \par 记号，而后结束 TEX 的运行；见第 ?? 章。

值得注意的是，遇到空行时 TEX 通常的行为(结束当前自然段)完全取决于 \par 记号的默认定义。重定义 \par 后，空行和竖直命令的行为可能就完全两样了；因此，我们可以借此实现一些特别的效果。在这种情况下，为了使用正常的 \par 的功能，plain TEX 提供了其同义词 \endgraf。详见第 ?? 章。

除非宏被声明为 \long 的，不然 \par 记号不能出现在宏的参数当中。对于非 \long 声明的宏，若其参数中包含 \par 记号，则 TEX 会给出「runaway argument」的报错。不过，使用 \let 定义的与 \par 同义的控制序列(例如 \endgraf)是允许出现在这些宏的参数之中的。

2.10 空格

这一节讨论输入处理器中有关空格字符和空格记号的一些内容。有关文本排版中的空格，留待第 ?? 章讨论。

2.10.1 被忽略的空格

在上述有关输入处理器内部状态的讨论中，我们不难发现，有些空格在输入处理器中就被抛弃了，因此永远不会被输出：输入行开头的空格以及在让 TEX 进入状态 *S* 的字符之后的空格。另一方面，行终止符尽管不在输入中(而是由 TEX 添加的)，但能产生可输出的空格。除此之外，还有第三种空格：它们可以通过输入处理器，甚至干脆由输入处理器产生，但也不会被输出。那便是非强制空格(⟨optional spaces⟩)。在 TEX 的语法中，很多地方都会出现此类空格。

2.10.2 非强制空格

TEX 语法中有所谓非强制空格与单个非强制空格的概念：

⟨one optional space⟩ ⟶ ⟨space token⟩ | ⟨empty⟩
⟨optional spaces⟩ ⟶ ⟨empty⟩ | ⟨space token⟩⟨optional spaces⟩

通常单个非强制空格(⟨one optional space⟩)允许出现在数字和粘连说明之后；而非强制空格(⟨optional spaces⟩)允许出现在数字或粘连中任意允许出现空格的地方(比如负号与数字之间，又比如 plus 和 1fil 之间)。此外，根据 ⟨equals⟩ 的定义，非强制空格允许出现在 = 之前。

[10]译注(Liam0205)：注意这里说的不是 LATEX 中结束环境的 \end{⟨env-name⟩} 命令，而就是 \end 这个 plain TEX 命令。

以下是有关非强制空格的一些例子：

- ⟨one optional space⟩ 可用于界定数字的范围。这有助于避免一些意外情况（见第 ?? 章），同时能加速 TEX 的处理过程——这是因为借助单个非强制空格，TEX 能更容易地界定当前正在读入的 ⟨number⟩ 于何时结束。注意，并非每个「数值」都是 ⟨number⟩。例如说，`\magstep2` 中的 `2` 就不是数字，而是作为 `\magstep` 的参数的单独的字符记号。因此，在其后加上空格或行终止符是有意义的。此外，宏参数中的数字，例如 `#1`：因为一个宏最多允许有 9 个参数，故而只需在参数符后扫描一位数字即可[11]。
- 根据 TEX 的语法，关键字 `fill` 及 `filll` 由 `fil` 与若干单独的 `l` 字符记号组成（详见第 ?? 页）；因此此处允许非强制空格。据此，例如`fil␣L␣l` 是合法的关键字[12]。这里有一些潜在的问题，可能导致莫名其妙的情况。大多数情况下，在关键字后面加上一个 `\relax` 即可避免此类问题。
- TEX 原语 `\ignorespaces` 会吃掉其后的非强制空格；故此可将其插入宏定义的末尾，以避免将参量右花括号后的空格无意带入输出当中。例如说：

```
\def\item#1{\par\leavevmode
    \llap{#1\enspace}\ignorespaces}
\item{a/}one line \item{b/} another line \item{c/}
yet another
```

此处，`\ignorespaces` 吃掉了第二、第三两次调用之后的空格，而这些空格是不希望被排版输出的。不过，在 `\ignorespaces` 之后的空行仍然会插入 `\par` 记号。

2.10.3 被忽略和被保留的空格

TEX 会忽略控制词之后的空格。不过这不是因为控制词之后的空格是非强制空格，而是因为 TEX 在遇到控制词之后会进入状态 *S*，从而忽略空格。类似地，控制词之后的行终止符也会被忽略。

数字由单个非强制空格界定，但是

```
a\count0=3  b
```

的输出是 'ab'。这是因为 TEX 在第一个空格记号[13]之后会进入状态 *S*，从而第二个空格会被 TEX 的输入处理器忽略，永远不会变成空格记号。

当 TEX 处于新行状态 *N* 时，空格也会被忽略。另一方面，当 TEX 处于竖直模式工作时，空格记号（也就是在一开始未被忽略的空格）会被忽略。例如说，下例第一个盒子之后由行终止符生成的空格记号会被忽略[14]。

[11] 译注（Liam0205）：而不需要单个非强制空格来辅助界定数字的范围。
[12] 译注（Liam0205）：TEX 的关键字不区分大小写，并且在关键字前允许有非强制空格。
[13] 译注（Liam0205）：第一个空格记号是单个非强制空格，界定了单个数字。
[14] 译注（Liam0205）：此处 `\hbox{⟨ 内容物 ⟩}` 并不会使 TEX 从由 `\par` 记号引入的竖直模式中切换回

```
\par
\hbox{a}
\hbox{b}
```

plain TEX 和 LATEX 格式都定义了名为 \obeyspaces 的宏。该宏能使每个空格都是有意义的:在一个空格之后,连续的空格会被保留。两种格式中,\obeyspaces 的基本形式是一致的。

```
\catcode`\ =13 \def {\space}
```

不过,对于 \space 的定义,两种格式有所区别。在 plain TEX 中,\space 的定义如下

```
\def\space{ }
```

在 LATEX 中,同名的宏则定义为

```
\def\space{\leavevmode{} }
```

在 \obeylines 的上下文中,比较容易看出这两种定义的区别。使用 \obeylines 后,每个行终止符都会被转换成一个 \par 命令。因此 TEX 开始处理每一行时,都处于竖直模式。在 plain TEX 中,活动空格被展开为空格记号,因此在垂直模式中会被忽略。但在 LATEX 中,首先会离开竖直模式并进入水平模式,因此每个空格就都是有意义的了。

2.10.4 空格被忽略的其他情形

还有三种情形下,TEX 会忽略空格记号:

1. 在寻找未被花括号定界的宏参数时,TEX 会忽略所有空格记号,而将第一个非空记号(或分组)作为参数。详见第 **??** 章。
2. 在数学模式中,所有的空格记号会被忽略(详见第 **??** 章)。
3. 在阵列制表符之后,空格记号会被忽略(见第 **??** 章)。

2.10.5 空格记号:⟨space token⟩

在 TEX 中,空格总是表现得与众不同。举例来说,\string 会将所有字符的分类码设置为 12,唯独空格的分类码是 10 。此外,如前文所述,在状态 *M* 中,TEX 的输入处理器会将所有分类码为 10 的字符转换为真正的空格:字符编码会被设置为 32。于是,任何分类码为 10 的字符记号称为空格记号(⟨space token⟩)。字符编码不是 32 的空格记号称为滑稽空格。

例子:将字符 Q 的分类码设置为空格字符之后,如下定义

```
\catcode`Q=10 \def\q{aQb}
```

可得

水平模式。

```
\show\q
macro:-> a b
```

这是因为输入处理器改变了宏定义中滑稽空格的字符编码。

字符编码不为 32 的空格记号可以用 \uppercase 等命令生成。不过,「由于字符编码不同的空格记号的行为是一致的,所以纠缠于这类细节是没有意义的」。详见 [8] 第 377 页。

2.10.6 控制空格

控制空格命令 \␣ 给出一个与 \spacefactor 等于 1000 时空格记号宽度一样的空格。控制空格不能被当做是空格记号,也不能理解为会展开成一个空格记号的宏(例如 plain TEX 中的 \space)。举例来说,TEX 会忽略所有输入行开头的空格,但是控制空格是一个水平命令(⟨horizontal command⟩),故而 TEX 在遇到它之后会从竖直模式切换到水平模式(并插入一个缩进盒子)。有关 \spacefactor 的介绍,详见第 ?? 章;有关水平模式和竖直模式的介绍,详见第 ?? 章。

2.10.7 可见空格:␣

在 Computer Modern 的打字机字体中,字符编码为 32 的字符是显式空格符号'␣'。不过,简单地使用 \tt 命令[15]是无法将其打印出来的。这是因为空格在输入处理器中有特别的处理。

使空格字符 ␣ 显形的一种方法是将空格字符的分类码设置为 12:

```
\catcode`\ =12
```

此时,TEX 会将空格字符作为编码为 32 的字符排版出来。此外,连续的空格不会被忽略。这是因为状态 *S* 只是在遇到分类码为 10 的字符后才会进入。类似地,控制序列之后的空格也会因分类码的改变而显形。

2.11 有关行尾的更多知识

TEX 从输入文件中获取文本行,但不包括输入行中的行尾符。因此,TEX 的行为不依赖操作系统以及行尾符究竟是什么(CR-LF、LF 抑或是在块存储系统里根本就不存在行尾符)。而后,TEX 会移除输入行末尾的空格。这样处理是有历史原因的:TEX 必须能够适应 IBM 大型计算机的块存储模式有关。对于由计算机的不同而造成的有关行尾符的问题,详见 [1]。

此后,字符编码为 \endlinechar 的行终止符会被追加在文本行的末尾;除非 \endlinechar 中保存的数值为负数或大于 255。注意,改行终止符的分类码不一定

[15] 译注(Liam0205):在 LATEX 中是 \ttfamily 命令。

非得是 5。

2.11.1 保持各行

有时候会期望会希望输入文本中的行尾符能与排版输出的行尾一一对应。下面的代码可以可以解决这一问题：

```
\catcode`\^^M=13 %
\def^^M{\par}%
```

这里，\endlinechar 成为活动符，其含义变为 \par。上述代码中的注释符用于阻止 TEX 看到代码末尾的行终止符，以防它将其作为活动字符而展开。

需要注意的是，在将上述代码嵌入宏的展开文本中需要特别小心。例如说下列代码会让 TEX 误解：

```
\def\obeylines{\catcode`\^^M=13 \def^^M{\par}}
```

具体来说，TEX 将丢弃第二个 ^^M 之后的所有字符。这是因为，在宏展开的过程中，\catcode 命令尚未执行，因而此时 ^^M 分类码为 5，而非 13。也就是说，这一行实际上变成了：

```
\def\obeylines{\catcode`\^^M=13 \def
```

要修正上述问题，需要为 ^^M 营造一个可作为活动字符使用的环境：

```
{\catcode`\^^M=13 %
 \gdef\obeylines{\catcode`\^^M=13 \def^^M{\par}}%
}
```

这样解决了上面提到的问题，但仍有缺陷。这是因为，该 \obeylines 仍然不能保留输入文本中的空行——连续两个 \par 记号会被当成是一个。为此，我们需要对上述定义稍作改进：

```
\def^^M{\par\leavevmode}
```

这样，输入文本中的每一行都会开启一个新段落，空行则开启一个空段落。

2.11.2 改变 \endlinechar

某些情况下，你会希望改变 \endlinechar 的值或者 ^^M 的分类码，以达成一些特殊效果。例如说，可以用行终止符作为宏的参数的定界符。具体可参考第 ?? 页给出的例子。

在这些常识中，通常会有一些陷阱。我们来看以下写法：

```
{\catcode`\^^M=12 \endlinechar=`\^^J \catcode`\^^J=5
...
... }
```

这段代码的输出不符合预期：由于第一行和最后一行的行终止符，TEX 将输出字符码

为 13(^^M)和 10(^^J)的字符。[16]

在第一行和最后一行末尾加上注释符可以解决此问题，但还有另一种方法是将第一行拆成下面两行[17]：

```
{\endlinechar=`\^^J \catcode`\^^J=5
\catcode`\^^M=12
```

当然，在多数情况下没必要将行终止符替换为另一个字符；设置

```
\endlinechar=-1
```

就等同于各行都以注释符结尾。

2.11.3 行终止符的更多注记

TEX 对所有字符一视同仁，包括追加到输入行末尾的行终止符。考虑到它特别的分类码，通常大家都不会注意行终止符。但是有一些方法可以特别地处理行终止符。

例子：假定 *\endlinechar* 保持默认值为 *13*，那么，把 ^^ 置于文本行的末尾，将输出字符 *'M'*。因为它是编码为 *13+64* 的 ASCII 字符。

例子：如果 \^^M 有定义，此时可称为「控制换行」。则在输入行中用反斜线结尾将执行此控制换行命令。例如，在 *plain TeX* 中定义

```
\def\^^M{\ }
```

将使得控制换行与控制空格等价。

2.12 输入处理器的更多知识

2.12.1 输入处理器作为独立过程

TEX 处理器的各个阶段都是同时运行的，但是在概念上它们常被视为依次独立运行，前者的输出是后者的输入。关于空格的小戏法很好地展现了这一点。

考虑以下宏定义：

```
\def\DoAssign{\count42=800}
```

及其调用：

```
\DoAssign 0
```

[16] 译注（Liam0205）：在第一行执行之前，输入处理器的初始化处理将会添加 ^^M 作为行终止符，但由于其分类码被修改为 12，即其他字符，故而会被输出。最后一行执行之前，输入处理器的初始化处理将会添加 ^^J 作为行终止符，因为此时 \endlinechar 的值是它的字符码。但由于分组结束，^^J 的分类码已恢复，故此时添加在输入行尾的行终止符 ^^J 也会输出。

[17] 译注（Liam0205）：这样可以避免第一行的行终止符带来的问题，但无法避免最后一行的行终止符带来的问题。

作为构建记号的部分，TEX 的输入处理器在扫描此次调用时，会忽略控制序列之后和零之前的所有空格。因此，此次调用的展开为：

```
\count42=8000
```

不要认为：「\DoAssign 首先被读入，而后展开，而后空格作为分隔符分割了 800，于是 800 被赋值给计数器，并打印出数字零。」不过，需要注意的是，如果数字零出现在下一行，情况就不一样了。

再举一个让非强制空格出现在忽略空格阶段之后的例子：

```
\def\c.{\relax}
a\c. b
```

会被展开为：

```
a\relax b
```

其输出是：

‘a b’

这是因为，「输入处理器忽略控制序列 \relax 之后的空格」这一现象仅出现在该行被首次读取之时，而非在其被展开之时。另一方面，这个例子：

```
\def\c.{\ignorespaces}
a\c. b
```

则会被展开为：

```
a\ignorespaces␣b
```

执行 \ignorespaces 时会删除所有接续其后的连续空格记号。因此，输出是：

‘ab’.

在上述两个例子中，\c 之后的西文句号均为定界符，用于保护控制序列之后的空格不被输入处理器吃掉。

2.12.2 输入处理器不作为独立过程

将 TEX 的记号化的过程视作独立过程是一个便利的做法，但有时这种做法会引起困惑。例如

```
\catcode`\^^M=13{}
```

将行终止符设为活动字符；因此，该行自身的行终止符将报错：「未定义的控制序列（undefined control sequence）」。这表明，执行行内的命令有时会影响对同一行的扫描过程。

另一方面，下面例子则不会报错：

```
\catcode`\^^M=13
```

这是因为，在 TEX 扫描数字 13 时就读入了行终止符，此时，分类码的赋值过程尚未执行；而此时，行终止符被转换成了非强制空格，作为数字的定界符。

2.12.3 输入处理器的递归调用

前文中，将参数符和数字替换为参数记号的过程被描述得与将字母捆绑成控制序列记号类似。但实际情况要复杂得多。TEX 的记号扫描机制不仅在扫描文件输入时起作用，在扫描记号列表输入时同样会起作用：例如在处理宏定义时。前文提到的内部状态变化的机制，仅仅适用于前一种情况。

在两种情况下，输入处理器对参数符的处理方式都是相同的。否则 TEX 便无法处理下面这样的宏定义：

```
\def\a{\def\b{\def\c####1{####1}}}
```

见第 ?? 页对这种嵌套定义的解释。

2.13 @ 约定

读过 plain TeX 或是 LATEX 格式的源码就会发现其中有很多包含符号 @ 的控制序列。这种包含 @ 的控制序列不能被普通用户直接使用。

格式文件的起始处附近有命令

```
\catcode`@=11
```

它将 @ 的分类从「其他字符」变为「字母」，从而可以用于组成控制序列。而在格式文件的末尾处附近有命令

```
\catcode`@=12
```

它将 @ 的分类恢复为其他字符。

那么，为什么用户不能直接调用带有 @ 字符的控制序列，却能调用定义中包含此类控制序列的宏呢？原因在于，在宏定义时，带有 @ 的控制序列已被 TEX 内部处理过了，此后，这些控制序列就变成了记号而不是字符串了。在宏展开的过程中，TEX 只需要操作记号，因此，彼时记号内字符的分类码就不影响宏展开的过程了。

第 3 章　字符

TEX 在其内部使用字符编码来表示字符；本章讨论字符编码及相关命令。

`\char` 显式指定所要排版的字符。

`\chardef` 将一个控制序列定义为一个字符编码的同义词。

`\accent` 放置重音符号的命令。

`\if` 测试字符编码是否相等。

`\ifx` 测试字符编码与分类码是否都相等。

`\let` 将一个控制序列定义为一个记号的同义词。

`\uccode` 对于给定的字符编码，查询或设置其对应的大写变体的编码。

`\lccode` 对于给定的字符编码，查询或设置其对应的小写变体的编码。

`\uppercase` 将普通文本（⟨general text⟩）转换为大写形式。

`\lowercase` 将普通文本（⟨general text⟩）转换为小写形式。

`\string` 将一个记号转换为一个字符串。

`\escapechar` 将控制序列转换为一串字符记号时，用作转义字符的字符之编码。在 IniTEX 中默认为 92（\）。

3.1　字符编码

在概念上说，认为 TEX 内部直接对字符进行处理是最简单的；但是实际上 TEX 内部处理的是整数，即*字符编码*。

不同系统中，字符的编码方式可能不同。因此，TEX 使用其自身定义的字符编码方案。任何从文件或用户终端读入的字符，都会依据编码表转换成字符编码。之后，TEX 会根据这一字符编码为它们分配分类码（见第 2 章）。编码表基于 7-位 ASCII 编码表，可编码 128 个字符（见第 **??** 节）。

反引号（`，或称左引号）可将字符（准确说是字符记号）转换为其字符编码。在 TEX 中，所有需要 ⟨number⟩ 的地方，都可以通过这种方式将一个字符记号的编码传

给 TEX: 反引号加字符或是反引号加单字符控制序列。例如 \count`a 和 \count`\a 都表示 \count97。详见第 ?? 章。

第二种方式在某些情况下是有必要的，例如在下例中，若去掉 \，就会让 TEX 产生误解。

```
\catcode`\%=11   or   \def\CommentSign{\char`\%}
```

例如在下例中，=11 会被认为是注释。

```
\catcode`%=11
```

单字符的控制序列可以由任何分类码的字符组成。[1]

当字符记号被转换为字符编码后，字符编码就与字符的形状没有关联了。当然，对于大多数字符来说，在视觉上，其排版输出与输入是一致的。例如输入 a 会排版输出字符「a」。不过，即使在常见的符号中，也有例外情况。在 Computer Modern Roman 字体中，就没有小于号和大于号。输入 '<>' 得到的输出是 '¡¿'。

为了使 TEX 在输出端也不依赖具体的机器，dvi 文件格式也是用这种字符编码：操作码 $n = 0 \ldots 127$ 表示：「从当前字体中取出第 n 个字符。」在 [6] 中可以找到 dvi 文件的操作码的完整定义。

3.2 字符相关的控制序列

有许多种方式可以用控制序列来表示字符。例如说，\char 命令可以指定一个用于打印的字符；\let 命令可引入一个与指定字符记号同义的控制序列——包括其字符编码和其分类码。

3.2.1 表示要排版的字符：\char

字符可以通过数值来表示，比如 \char98。该命令会将当前字体中编码为 98 的字符添加到正在构建的水平列表中。

相较而言，用八进制或十六进制表示字符编码可能比用十进制更方便。在数值前加上单引号，即可使用八进制数：\char'142；在数值前使用双引号，即可使用十六进制数：\char"62。注意，连续使用两个单引号，比如 \char''62 是错误的写法——将两个单引号替换为一个双引号的工作是由可视化处理器承担的，而不是数字扫描操作（执行处理器）。

由于可用反引号获取字符的编码，在当前字体的编码与 ASCII 兼容时，用 \char`b 或 \char`\b 也可以得到 'b' 这个字符。

表面上看，\char 命令与 ^^ 替换机制（第 2 章）类似：两种机制都是通过间接的方式来访问字符。但是，^^ 替换机制发生的非常早——在 TEX 的输入处理器为字符

[1] 译注（Liam0205）：特别地，可以是无效字符。

分配分类码之前；而 \char 命令则在可视化处理器中起作用。从效果上看，\char 是告诉 TEX：「排版字体中第 n 个字符。」

\chardef 命令可定义一个控制序列定义作为一个字符编码的代替品。它的用法是

\chardef⟨control sequence⟩⟨equals⟩⟨number⟩,

此处的 ⟨number⟩ 可以是显式给出的数字，也可以是计数器值，也可以是用反引号提取得到的字符编码（见上文；关于 ⟨number⟩ 的完整定义，参见第 ?? 章）。在 plain TeX 中，就有类似下面的用法：

```
\chardef\%=`\%
```

它等价于下面的定义方式：

```
\chardef\%=37
```

如此定义之后，控制字符 \% 就变成了 \char37 的同义词。也就是说，它可以用于排版第 37 个字符（通常是百分号）。

用 \chardef 命令定义的控制序列可作为 ⟨number⟩ 使用。在诸如 \newbox 的分配类命令中，就用到了这一特性。（见第 ?? 章和第 ?? 章）用 \mathchardef 命令定义的记号同样可以这样使用。

3.2.2 隐式字符记号：\let

另一种定义表示字符的控制序列的方式是使用 \let 命令，并且将字符记号置于可选等号的右边：

\let⟨control sequence⟩⟨equals⟩⟨token⟩

如此，得到的控制序列称为隐式字符记号。（见第 ?? 页对 \let 的进一步讨论。）

在 plain TEX 中有对左右花括号定义同义控制序列：

```
\let\bgroup={ \let\egroup=}
```

如此，得到的控制序列被称为「隐式花括号」（见第 ?? 章）。

通过 \let 和 \chardef 将字符赋值给控制序列是有区别的。被 \let 定义的控制序列是字符编码与分类码这一组合的替代品。

例如在下例中，\b 关闭了 \m 定义所在的分组，因此 TEX 会提示错误：「未定义的控制序列」。

```
\catcode`|=2 % make the bar an end of group
\let\b=|  % make \b a bar character
{\def\m{...}\b \m
```

又例如在下例中，\b 的定义在分类码修改之前，因此它无法承担分组结束符的作用，只能表示一条竖线（或是其他任何在当前字体中编号为 124 的字符）。因此，这里构

造的是一个不闭合的分组。

```
\let\b=| % make \b a bar character
\catcode`|=2  % make the bar character end of group
{\def\m{...}\b \m
```

前一个例子表明，即是花括号本身被重定义了（例如在排版 C 语言代码时被定义成活动字符），分组开始和结束的功能依然可以由控制序列 \bgroup 和 \egroup 来承担。

隐式字符记号的行为与真实字符记号非常相似。举例来说，在如此定义之后，

```
\catcode`|=2 \let\b=|
```

以下两个真值测试均为真。

```
\if\b|
```

```
\ifcat\b}
```

再举一例。在 plain TEX 中，有如下对上标和下标符号的定义：

```
\let\sp=^ \let\sb=_
```

如此一来，即使键盘上没有这两个符号，也可以通过这两个控制序列得到数学环境的上标和下标。

```
x\sp2\sb{ij}  gives  x\sp2\sb{ij}
```

若是编写格式的人的键盘上也没有这些键，那么就要用更深的技巧了：

```
{\lccode`,=94 \lccode`.=95 \catcode`,=7 \catcode`.=8
\lowercase{\global\let\sp=, \global\let\sb=.}}
```

至于它为什么能起作用，可详见后文关于 \lowercase 的介绍。由于我们无法键入 ^，因此 TEX2 中的 ^^ 表示法无法在此解决问题。（见第 25 页）考虑到上标和下标的字符编码以十六进制表示分别是 5e 和 5f，利用 TEX3 中扩展的表示法可以依照如下方式解决问题：

```
{\catcode`\,=7
\global\let\sp=,,5e \global\let\sb=,,5f}
```

使用 \meaning 命令可以查看使用 \let 定义的命令究竟代表哪个字符。例如下列代码将给出 ‘the character 3’。

```
\let\x=3 \meaning\x
```

3.3 重音

重音符号可用水平命令（⟨horizontal command⟩）\accent 给出。

\accent⟨8-bit number⟩⟨optional assignments⟩⟨character⟩

其中，⟨character⟩ 是分类码为 11 或 12 的字符，或是形如 \char⟨8-bit number⟩ 的

命令，抑或是一个由 \chardef 定义的记号。此时，重音符号会被放置在该字符的上方。若 ⟨character⟩ 不是上述四种情况之中的任意一种，则重音符号会被 \char 命令直接处理，从而给出一个「悬在半空」的重音符号。可选的 ⟨optional assignments⟩ 可用于改变重音和字符的字体。

\accent 命令后必须紧跟 ⟨character⟩。这一规定避免了将重音符号置于连字之上或是置于另一个重音符号之上。在诸如印度语及越南语的少数语言中，确实存在双重音符号的情况；因此这种规定有时可能会令人不爽。不过，将重音符号置于另一个之上可以在数学模式中实现。

添加重音符号不会改变字符的宽度。对于重音符号摆放的位置，TEX 假设字体文件中，重音符号摆放的高度与字母 x 的高度相适应；对于其他高度的字母，TEX 则可以通过向上或向下移动恰当的高度来调整重音符号的位置。

在 TEX 中没有真正的下重音，而是将它们视作位置较低的上重音实现的。更准确的做法是：编写一个宏，测量紧跟着的字符的尺寸，而后相应地升高或降低重音符号的位置。plain TEX 中的变音宏 \c就是这样做的。不过，对于有下伸部分的字母，它不会降低重音符号的位置。[2]

重音符号的水平位置由 \fontdimen1（每点倾斜，*slant per point*）控制。为此，TEX 会引入铅空以校正重音符号的水平位置。[3]注意，尽管这些铅空是自动插入的，但这些铅空隶属显式铅空。因此，它们会抑制铅空前后的连字符。

以下代码能将水平列表转储出来，作为铅空和重音符号的示例。

```
\setbox0=\hbox{\it \`l}
\showbox0
```

其结果是

```
\hbox(9.58334+0.0)x2.55554
.\kern -0.61803 (for accent)
.\hbox(6.94444+0.0)x5.11108, shifted -2.6389
..\tenit ^^R
.\kern -4.49306 (for accent)
.\tenit l
```

注意，TEX 首先放置重音符号，以保证最后一个字符的倾斜校正仍然有效。

3.4 字符真值测试

使用 \if 可检测两个字符记号的字符编码是否相等：

\if⟨token$_1$⟩⟨token$_2$⟩

[2]译注（Liam0205）：所谓下伸部分，指的是个别小写字母比其它小写字母更低的部分。比如「p」、「q」的「小尾巴」。

[3]译注（Liam0205）：铅空是活字印刷排版时代，排版工人用于调整字符水平位置而使用的空白铅块。

TEX 遇到 \if 后，会对其后的记号持续展开，直至遇见两个不可展开的记号。而后，不管分类码如何，若这两个记号的字符编码相等，则真值测试为真。

对于 TEX 来说，不可展开的控制序列的字符码是 256 而分类码为 16（因此控制序列的字符编码只与控制序列的字符编码相等），除非控制序列被 \let 定义成一个非活动字符记号——如前所述，此时该控制序列的字符编码和分类码与相应的字符相同。

第 ?? 章提到了用于检验分类码的 \ifcat 命令；真值测试

\ifx⟨token$_1$⟩⟨token$_2$⟩

则检验字符编码和分类码是否均相等。特别注意的是，TEX 不会展开 \ifx 之后的记号。不过，若它们是宏，则 TEX 会检验它们的展开是否相等。

对于 \chardef 定义的数量，则可以用 \ifnum 来检验。

```
\chardef\a=`x \chardef\b=`y \ifnum\a=\b % is false
```

这是因为由 \chardef 定义的记号可被当做是数字来使用（见第 ?? 章）。

另见第 ?? 节。

3.5 大写和小写

3.5.1 大写和小写的编码

每个字符编码都有对应的*大写编码*和*小写编码*。（更多代码见下文）它们可分别用

\uccode⟨number⟩⟨equals⟩⟨number⟩

和

\lccode⟨number⟩⟨equals⟩⟨number⟩

来指定。在 IniTEX 中，`a..`z 及 `A..`Z 的大写编码为`A..`Z 而小写编码为 `a..`z。其余所有的字符的大写编码和小写编码均为零。

3.5.2 大写和小写命令

\uppercase{...} 和 \lowercase{...} 命令会遍历参数记号列表，而后将所有对应大写编码或小写编码不为零的显式字符替换成相应的大写编码或小写编码，分类码则维持不变。

\uppercase 和 \lowercase 命令的参数是普通文本（⟨general text⟩），其定义如下（其中 ⟨filler⟩ 的定义见第 ?? 章）：

⟨general text⟩ ⟶ ⟨filler⟩{⟨balanced text⟩⟨right brace⟩

此处，左花括号可以是隐式字符记号，但右花括号必须是分类码为 2 的显式字符记号。在寻找左花括号时，TEX 会进行宏展开。

与 \number 和 \string 不同，大小写转换不是宏展开，而是在执行处理器中执行的。下列尝试得到 \A 的代码会因 \uppercase 不可展开而报错——TEX 将在 \uppercase 之前插入 \endcsname。[4]

```
\expandafter\csname\uppercase{a}\endcsname
```

正确的写法如下：

```
\uppercase{\csname a\endcsname}
```

下例对 \uppercase 的使用是正确的，它可用于检测一个字符是否为大写：

```
\def\ifIsUppercase#1{\uppercase{\if#1}#1}
```

大写检测也可以这样做：\ifnum`#1=\uccode`#1。

首字母大写（首字母的字符编码与其 \lccode 不同）的单词是否允许截断连字取决于 \uchyph 参数。若改参数为正数，则允许对首字母大写的单词进行截断连字。另见第 ?? 章。

3.5.3 关键字的大写与小写形式

TEX 关键字可以是大写形式也可以是小写形式。比如 pt 还可以写作 pT、Pt 以及 PT。此处，TEX 不使用 \uccode 及 \lccode 表来确定小写形式；而是直接在大写字母的字符编码上加 32 来得到其小写形式——ASCII 编码中，同一字母大小写形式之差为 32。这种做法与 TEX 在处理非罗马字符时的具体实现方式有些关联；详见 the TEX book [8] 第 370 页。

3.5.4 巧用 \uppercase 和 \lowercase

利用 \uppercase 和 \lowercase 不会改变分类码的特性，可以构造出一些其它方式难以构造的字符编码-分类码配对；比如第 ?? 章中对宏 \newif 的讲解；又比如第 41 页上的例子。

再将一个稍有些不同的例子。考虑这一问题（已由 Rainer Schöpf 解决）：给定一个计数器 \newcount\mycount，如何将其值作为字符编号输出到终端？解法如下。

```
\lccode`a=\mycount \chardef\terminal=16
\lowercase{\write\terminal{a}}
```

此处，\lowercase 命令有效地将 \write 命令的参数，从「a」转换成目标值。

[4] 译注（Liam0205）：这里的表述比较奇怪。事实上，TEX 要求 \csname 和 \endcsname 之间的内容展开到底后不能有不可展开的原语；而 \uppercase 正是一个不可展开的原语，故而报错。这里原作者提出：「TEX 在 \uppercase 之前插入 \endcsname」，原因可能是 TEX 在发现问题「尝试修复的过程中」，会做这样的操作。

3.6 字符相关编码

每个字符编码都有一系列与之相关的 ⟨codename⟩。这些整数的取值范围各不相同；它们决定了在不同场合下，TEX 会如何处理该字符，或是决定了在不同场合下，该字符会如何影响 TEX 自身。

这些编码的名字罗列如下：

`\catcode` 4 位数字（⟨4-bit number⟩，0–15）；字符的分类码。见第 2 章。

`\mathcode` 15 位数字（⟨15-bit number⟩，0–"7FFF 或 "8000）；它决定 TEX 在数学环境下如何处理该字符。见第 ?? 章。

`\delcode` 27 位数字（⟨27-bit number⟩，0–"7 FFF FFF）；它决定 TEX 在数学环境下如何处理出现在 `\left` 和 `\right` 之后的该字符。见第 ?? 页。

`\sfcode` 整数；它决定 TEX 如何处理位于该字符之后的空距。见第 ?? 章。

`\lccode`、`\uccode` 8 位数字（⟨8-bit number⟩，0–255）；如前所述，这是字符的小写及大写的编码。

3.7 将记号转换为字符串

`\string` 命令能将其下一个记号展开为由若干字符组成的字符串。例如，下例将输出 `\control`。

```
\tt\string\control
```

又例如下例将输出 `$`。

```
\tt\string$
```

注意，字符串操作位于记号化之后。因此下例不能输出注释符号 `%`，因为早在输入处理器中，注释符号就被删除了。于是，下例中的 `\string` 会将下一行的第一个记号转换为字符串。

```
\tt\string%
```

`\string` 命令由展开处理器执行。因此，除非显式地禁止，它必然会被展开。详见第 ?? 章。

3.7.1 输出控制序列

上例使用了打字机字族[5]。这是因为，在 Computer Modern Roman 字体中没有反斜线这一字符。不过，TEX 也可以使用其它字符来打印控制序列：具体来说，TEX 使

[5] 译注（Liam0205）：`\tt` 是 plain TEX 风格的字体选择命令。在 LATEX 中，应当使用 `\ttfamily` 或者 `\texttt` 代替。

用 \escapechar 保存的字符编码对应的字符来打印控制序列。在使用下列命令时，也会用到 \escapechar 这个宏保存的值：\write、\message、\errmessage、\show、\showthe 以及 \meaning。若 \escapechar 的值为负或者大于 255，则转义字符不会显示出来；此外，在 IniTEX 中 \escapechar 的默认值是 92，即反斜线的字符编码。

在 \write 语句中，有时可用 \noexpand 代替 \string。详见第 ?? 页。

3.7.2 \string 输出输出字符串的分类码

\string 命令输出的字符串中，各个字符的分类码均为 12，唯独被字符串化的控制序列中的空格，其分类码是 10。由于控制序列内部不存在分类码的说法，\string 输出的所有空格必定只是空格字符，也就是说，它的字符编码是 32。由于 TEX 的输入处理器会将所有空格记号的字符编码设置为 32，「滑稽空格」出现在控制序列中的机会相当渺茫。

在分类码方面，其他命令的行为与 \string 表现一致：\number、\romannumeral、\jobname、\fontname、\meaning 以及 \the。

第 4 章　字体

在文本模式下，TEX 会从「当前字体」中取用字符。本章讨论 TEX 是如何识别不同字体的，以及字体都有哪些属性。

\font 该命令可声明一个用于指定字体的控制序列。

\fontname 字体的外部名字。

\nullfont 在特殊情况下，TEX 会使用空字体。这是空字体的名字。

\hyphenchar 字体中连字符的序号。

\defaulthyphenchar 字体装载时 \hyphenchar 的默认值。在 plain TEX 中，默认是 `\-（即 - 的字符编码）。

\fontdimen 该命令可访问多种字体参数。

\/ 倾斜校正原语。

\noboundary 忽略隐式边界字符。

4.1　字体

在 TEX 语境下，字体（font）这一术语指的是保存在外部文件中的一系列字符的集合。TEX 在运行时，会决定从哪一个字体中选取字符。而具体决策方式在文本模式和数学模式中又是不同的。

TEX 处理普通文本时会从「当前字体」中选取字符。通过字体声明，TEX 会将外部字体文件与字体选择命令联系起来。例如，下列声明会使 TEX 装载名为 myfont10.tfm 的字体文件。

```
\font\MyFont=myfont10
```

此后，通过下列命令，我们就可以选择使用该文件定义的字体了。

```
\MyFont
```

当前字体的状态是可查的：下列命令会产生当前字体对应的控制序列。

```
\the\font
```

数学模式下，TEX 会忽略当前字体；转而考察当前「字族」，其中有三个字体：正文字体（text style），角标字体（script style），双重角标字体（scriptscript style）。详见第 ?? 章。

关于字体（font）与字形（typeface）的一贯术语，请参阅 [9]。

从不同物理字体文件中分别选取部分字符，可以组成的「虚拟字体」（见 [7]）。在 TEX 看来，这种虚拟字体就像是真实字体一样。更多相关内容详见第 ?? 页。

4.2 字体声明

TEX 或 IniTEX 在执行时，必然在某处会建立内部字体选择命令与外部字体文件名的联系。字体声明的语法如下：

`\font`⟨control sequence⟩⟨equals⟩⟨file name⟩⟨at clause⟩

这里，⟨at clause⟩ 指的是：

⟨at clause⟩ ⟶ `at` ⟨dimen⟩ | `scaled` ⟨number⟩ | ⟨optional spaces⟩

注意，字体声明仅在当前分组内生效。

用户可以通过 ⟨at clause⟩ 指定字体的放大版本。⟨at clause⟩ 有两种形式：如果指定缩放比例 `scaled` f，则 TEX 会将该字体中的所有字符尺寸倍乘 $f/1000$；如果指定目标大小 `at` f`pt`，而字体本身的设计大小是 d`pt`，则 TEX 会将该字体中的所有字符尺寸倍乘 f/d。注意，⟨at clause⟩ 不会影响 TEX 从外部字体文件（`.tfm` 文件）中读入字体的过程；它只是在尺寸上乘了一个固定的倍数。

这样声明字体后，使用定义得到的控制序列会将当前字体设置为控制序列相应的字体。

4.2.1 字体与 `tfm` 文件

TEX 所需的外部字体文件是 `tfm` 文件（TEX 字体尺寸文件，TEX Font Metrics File），这些文件不受 `\font` 声明中 ⟨at clause⟩ 的影响。如果 `tfm` 事先已装载好（比如在构造格式文件时由 IniTEX 装载），则再次声明同一字体时，不依赖且无需重复载入 `tfm` 文件。

字体本身的设计大小由字体尺寸文件给出。例如，字体 `cmr10` 的设计大小是 10 点（point）。但实际上，字体中没有多少字符大小是 10 点：左右圆括号恰好是 10 点，但大写字母明显会小一点。

4.2.2 查询当前字体及字体名

前文提到过，当前字体的字体选择命令可通过 \the\font 得到。这是下列语法的一个特例。

\the⟨font⟩

此处，⟨font⟩ 表示：

⟨font⟩ ⟶ \font | ⟨fontdef token⟩ | ⟨family member⟩
⟨family member⟩ ⟶ ⟨font range⟩⟨4-bit number⟩
⟨font range⟩ ⟶ \textfont | \scriptfont | \scriptscriptfont

⟨fontdef token⟩ 是由 \font 定义的控制序列，或是预定义的 \nullfont。至于 ⟨family member⟩ 的概念，则只与数学模式有关。

此外，外部字体文件名可用以下方式取得：

\fontname⟨font⟩

该命令以字符串的形式（其中的字符记号分类码均为 12，而空格记号的分类码为 10）给出当前字体的文件名；若有相应的 ⟨at clause⟩ 则也一并给出。

例子：在如下声明之后，

```
\font\tenroman=cmr10 \tenroman
```

\the\font 和 \the\tenroman 会给出 \tenroman；\fontname\tenroman 则给出 cmr10。

4.2.3 \nullfont

TEX 将没有任何字符的字体与定义为 \nullfont。如果没有指定任何字体，或是在数学模式中某一所需的字族成员未定义，则 TEX 会从空字体中取用字符。该控制序列属于 ⟨fontdef token⟩：它与其它字体选择命令的行为类似；但不与任何外部 tfm 文件相关联。

4.3 字体信息

TEX 执行时所需的主要字体信息包括尺寸与字符。TEX 会在字体尺寸文件中寻找这些信息，这些文件的扩展名通常是 .tfm。此类文件包含

- 全局信息：主要是 \fontdiem 参数，还有其他一些信息
- 字符的尺寸以及倾斜校正（italic corrections）
- 字符的连字（ligature）及挤压（kerning）指令

字体的设计大小也由 tfm 文件指定。tfm 文件格式的定义，参见 [6]。

4.3.1 字体尺寸参数

文本字体至少得有 7 个 `\fontdimen` 参数来描述字体尺寸（对于未指定的参数，TEX 会以零作为默认值）；数学符号以及数学扩展字体还有更多的参数（详见第 **??** 页）。文本字体的 7 个参数描述如下：

1. 每点倾斜（slant per point）：该参数用于确定重音符号正确的水平位置。
2. 词间空距（interword space）：如果用户没有显式设定 `\spaceskip`，则该参数会被当做词间空距的默认值；参见第 **??** 章。
3. 词间拉伸（interword stretch）：该参数是词间距离的拉伸值部分。
4. 词间收缩（interword shrink）：该参数是词间距离的收缩值部分。
5. x-高度（x-height）：该参数是 TEX 内部单位（⟨internal unit⟩）`ex` 的值，通常是小写字母「x」的高度。
6. 方块铅空[1]宽度（quad width）：该参数是 TEX 内部单位 `em` 的值，它大约是大写字母「M」的宽度。
7. 额外空距（extra space）：如果用户没有显式设定 `\xspaceskip`，则该参数是在句子末尾处加在词间空距之上的额外空距（当 `\spacefactor` $\geqslant 2000$）。

第 1 个和第 5 个参数纯粹是字体的信息，因此改变它们没什么意义。其余参数则可以用来调整相关空距；第 **??** 章有关于第 2, 3, 4, 7 个参数的相关示例。

字体尺寸参数可用字体赋值语句（⟨font assignment⟩）来修改，这种赋值语句属于全局赋值语句（⟨global assignment⟩，参见第 **??** 页）：

`\fontdimen`⟨number⟩⟨font⟩⟨equals⟩⟨dimen⟩

⟨font⟩ 的定义可见上文。

4.3.2 字符挤压

通常，每个字符的边界盒子（bounding box）彼此邻接。但对于部分字符组合来说，它们之间的距离应该比边界盒子邻接的情况更近一些。这种情况称为字符挤压（*kerning*）。对于字体来说，字符挤压的设计与字符形状的设计同等重要。

举个例子：

有字符挤压的效果：Vo　没有字符挤压的效果：Vo

在 TEX 中，字符挤压由 `tfm` 文件中的字体信息控制，因此不受用户的影响。不过，`tfm` 文件是可以修改的，详情参见第 **??** 章。

`\kern` 命令与字符挤压现象没什么关系；它将在第 **??** 章中介绍。

[1] 译注（Liam0205）：英文单词「quad」是传统铅印术中的术语，它来自意大利语单词「quadratone」，意思是「大方块」。

4.3.3 倾斜校正

TEX 原语 \/ 会向前一个字符或连字后面插入倾斜校正（*italic correction*）。考虑到边界盒子（*bounding box*）的定义，这种修正是很有必要的：边界盒子总是有竖直的边，并且 TEX 将两条竖直边之间的距离当做是字符的宽度。然而，为了在斜体或意大利体中得到恰当的空距，字符会显著突出其边界盒子。当这种向外突出的字符后边紧跟着一个非倾斜的字符时，就需要插入倾斜校正。

对比以下两例：

无倾斜校正：*TEX* has　有倾斜校正：*TEX* has

这里，有倾斜校正的版本的代码如下。

```
{\italic\TeX\/} has
```

各个字符倾斜校正的具体值由字体尺寸文件中的字体信息决定；对于 Computer Modern 字体来说，该值大约是字符向外突出边界盒子距离的一半（详见 [4]）。倾斜校正与每点倾斜（`\fontdimen1`）是不一样的；后者仅用于放置重音符号。

倾斜校正仅在 TEX 刚处理完一个字符或者连字时才会插入其后。因此，由于在下例中，在遇到倾斜校正原语之前 TEX 刚处理完行间胶水（而不是字符或者连字），下面对罗马字族文本做的倾斜校正是不会生效的。

```
{\italic Some text {\/\roman not} emphasized}
```

4.3.4 连字

将字符序列替换为连字（*ligatures*）的过程是由字体 `tfm` 文件中的信息控制的。连字由字符（⟨character⟩）命令组成：在某些字体里，诸如 `fi` 的字符序列会替换为「fi」。

传统上，还有其它的连字，比如 `ff`, `ffi`, `fl` 以及 `ffl`；在更古老的字体里，可能还有 `ft` 以及 `st`，以及类似 `fl` 的连字：`fk` 和 `fb`。

在 TEX 中，连字可由显式字符记号、`\char` 命令以及 ⟨chardef token⟩ 组成。例如，如果字体里有这样的连字定义，`\char`f\char`i` 会被替换成连字「fi」。

我们有多种方式来限制不希望出现的连字。例如halflife 中不希望出现的连字可以这样避免：

`half{}life`, `half{l}ife`, `half\/life`, 以及 `half\hbox{}life`

注意，使用倾斜校正原语避免不希望出现的连字的方案与其它方案的机理不完全一样。

4.3.5 边界连字 (Boundary ligatures)

在 TEX3 中还有单词边界的概念：每个单词都由左右两个边界字符界定。这样的设定使 TEX 能够实现一些语言中特殊的现象[2]，例如在希腊语中位于词尾的西格玛（σ）和位于其他位置的西格玛是不一样的。在单词的前后添加 `\noboundary` 命令，可以抑制相应的边界连字机制。

大致上，TEX3 的连字机制比 TEX2 要复杂得多；参见 [5]。

[2] 译注（Liam0205）：这里指的是通过边界连字来实现一些语言中特殊的现象。

第 5 章　盒子

TEX 中的水平与竖直盒子分别是水平与竖直列表内容的容器。盒子可保存在盒子寄存器当中。本章讨论盒子寄存器，以及关于盒子尺寸、盒子内容相互位置的问题。

`\hbox` 构造一个水平盒子。

`\vbox` 构造一个竖直盒子，其参考点（reference point）是最后一个元素。

`\vtop` 构造一个竖直盒子，其参考点是第一个元素。

`\vcenter` 构造一个在竖直方向于数学轴（math axis）居中的竖直盒子；该命令仅在数学模式下使用。

`\vsplit` 将竖直盒子顶部部分分离。

`\box` 使用盒子寄存器，而后清空它。

`\setbox` 将盒子赋值给一个盒子寄存器。

`\copy` 使用盒子寄存器，但保留其中内容而不清空。

`\ifhbox` `\ifvbox` 判断盒子寄存器中保存的是水平盒子还是竖直盒子。

`\ifvoid` 判断盒子寄存器是否为空。

`\newbox` 分配新的盒子寄存器。

`\unhbox` `\unvbox` 将保存有水平盒子或竖直盒子的寄存器解包，并将其中内容追加到当前的水平列表或竖直列表中，而后清空寄存器。

`\unhcopy` `\unvcopy` 与 `\unhbox`、`\unvbox` 的行为相同，但不清空寄存器。

`\ht` `\dp` `\wd` 返回及设置盒子寄存器中盒子的高度、深度、宽度。

`\boxmaxdepth` 该命令表示 TEX 所允许的盒子的最大深度。在 plain TEX 中，它的默认值是 `\maxdimen`。

`\splitmaxdepth` 该命令表示 TEX 所允许的由 `\vsplit` 生成盒子的最大深度。

`\badness` 刚刚构造的盒子的劣度。

`\hfuzz` `\vfuzz` 该命令表示水平或竖直盒子的尺寸超出指定尺寸的阈值；当超过该阈值时，TEX 会考虑将其认定为溢出的（overfull）盒子。

\hbadness \vbadness 该命令表示一个阈值；当盒子的劣度超过该阈值时，TEX 会将相应盒子认定为欠满的（underfull）或者溢出的。

\overfullrule 当水平盒子溢出时，TEX 会打印一条标尺，以表明有溢出的水平盒子；该命令表示上述标尺的宽度。

\hsize 该命令表示在水平盒子内文本排版可用的行宽。

\vsize 该命令表示页盒子（page box）的高度。

\lastbox 若最后追加进当前列表的内容是一个盒子，则该寄存器会保存上述盒子。

\raise \lower 这两个命令用于在水平模式中调整盒子的垂直位置。

\moveleft \moveright 这两个命令用于在竖直模式中调整盒子的水平位置。

\everyhbox \everyvbox 这两个命令分别用于在每个水平盒子或竖直盒子开头处插入的记号列表。

5.1 盒子

本章我们讨论盒子。TEX 中的水平盒子与竖直盒子分别是盒子是水平与竖直列表内容的容器。需要多次使用的盒子可以保存在盒子寄存器当中。

当 TEX 需要接受一个盒子（⟨box⟩）时，TEX 可接受以下形式：

- \hbox⟨box specification⟩{⟨horizontal material⟩}
- \vbox⟨box specification⟩{⟨vertical material⟩}
- \vtop⟨box specification⟩{⟨vertical material⟩}
- \box⟨8-bit number⟩
- \copy⟨8-bit number⟩
- \vsplit⟨8-bit number⟩to⟨dimen⟩
- \lastbox

其中 ⟨box specification⟩ 的定义如下

⟨box specification⟩ ⟶ ⟨filler⟩
| to ⟨dimen⟩⟨filler⟩ | spread ⟨dimen⟩⟨filler⟩

而 ⟨8-bit number⟩ 则是 0–255 范围内的数字。

包围盒子内容的花括号定义了一个分组；它们可以是分类码为 1 和 2 的显式字符，也可以是通过 \let 定义的控制序列（隐式字符）；见下文。

除下文提到的 \lastbox 之外，盒子通常可以用在水平、竖直和数学模式。盒子与模式之间的关系将在第 ?? 中讨论。

由 \vcenter 命令（仅可在数学模式下使用）产生的盒子不是通常意义的盒子

(⟨box⟩)。特别地，它不能赋值给 \setbox；详见第 ?? 章。

\vsplit 命令将在第 ?? 章详细讨论。

5.2 盒子寄存器

共有 256 个盒子寄存器，它们从 0 开始编号至 255。盒子寄存器要不然是空的(void)，要不然包含了一个水平盒子或一个竖直盒子。本节只讨论盒子寄存器；盒子的尺寸、内部内容的排列方式则在后续小节中讨论。

5.2.1 使用 \newbox 分配盒子寄存器

plain TEX 中的 \newbox 宏可用于分配一个未被使用的盒子寄存器：

```
\newbox\MyBox
```

此后，我们可以这样给盒子寄存器赋值：

```
\setbox\MyBox=...
```

或者这样使用盒子寄存器里的内容：

```
\box\MyBox
```

连续调用该命令会生成的盒子寄存器的编号也是连续的；这样，宏集合(macro collections)可以各自分配盒子寄存器，而无需担心与其它宏中分配的寄存器冲突。

盒子寄存器的编号是通过 \chardef 赋值的(见第 ?? 章)。这说明，\MyBox 等价于一个数字(⟨number⟩)，并且能被当做是数字使用。控制序列 \newbox 是一个外部(\outer)宏。新分配的盒子寄存器初始时是空的。

5.2.2 \setbox、\box 和 \copy 的用法

我们可以将一个盒子(⟨box⟩)赋值给盒子寄存器：

\setbox⟨number⟩⟨equals⟩⟨box⟩

此处的 ⟨box⟩ 可以是显式的盒子，比如

```
\setbox37=\hbox{...}  or  \setbox37=\vbox{...}
```

也可以是另一个盒子寄存器：

```
\setbox37=\box38
```

一般来说，盒子寄存器的编号由 \newbox 命令分配。

通过 \box 和 \copy，我们可以将盒子寄存器中保存的盒子追加在 TEX 正在构建的列表当中。例如，下列命令会将编号为 38 的盒子追加到当前列表中：

```
\box38
```

为节省内存，盒子寄存器在使用后会被清空：TEX 假设当你使用 \box*nn* 之后，你就不会再需要其中的内容了，于是便会清空它。如果你确实需要重复使用盒子寄存器的内容，你可以用 \copy*nn* 命令。它与 \box*nn* 的效果相同，但不会清空盒子寄存器中的内容。

通过 \unhbox 和 \unvbox 及其拷贝版本 \unhcopy 和 \unvcopy 解包盒子寄存器中的内容是可行的。不同于 \box 可以在任何模式下使用，解包盒子的操作只能用在恰当的模式下——因为解包盒子会形成一个部分的水平列表或竖直列表（参见第 **??** 章）。后文有关于解包盒子寄存器的更多信息。

5.2.3 关于盒子的条件判断：\ifvoid、\ifhbox 及 \ifvbox

通过 \ifvoid 可以判断盒子寄存器是否为空。当 \box⟨number⟩ 为空时，下式返回真：

\ifvoid⟨number⟩

注意，盒子寄存器为空与盒子寄存器保存了空盒子是两码事。空盒子要么是水平盒子要么是竖直盒子；空的盒子寄存器既可以作为水平盒子使用，也可以作为竖直盒子使用。

\ifhbox 和 \ifvbox 则分别用于判断一个盒子寄存器中的盒子是否为水平盒子或竖直盒子：

\ifhbox⟨number⟩

\ifvbox⟨number⟩

注意，若寄存器为空，则两个判断都为假。

5.2.4 \lastbox

当 TEX 构建完成一个部分列表（partial list）后，此列表中的最后一个盒子可通过 \lastbox 来访问。该命令的行为与盒子寄存器相似；因此，你可以将 \lastbox 赋值给其他盒子寄存器，以将其从列表中删除。如果列表中最后一个元素不是盒子，则 \lastbox 的行为与空寄存器相似。你无法通过 \lastbox 访问主竖直列表中的最后一个元素；在这种情况下，\lastbox 总是空寄存器。

举例来说，下面的语句将当前列表中的最后一个盒子赋值给编号为 0 的盒子寄存器，并将其从当前列表中删除：

```
{\setbox0=\lastbox}
```

由于赋值过程位于分组之内，在分组结束后，寄存器会被清空。因此，在一个自然段开始之时使用这一语句，可以删除缩进盒子（见第 **??** 章）。在第 69 页也有关于 \lastbox 的例子。

由于 \lastbox 在外部竖直模式中总是为空，它不能访问追加到输出页上的盒子。但当页面保存在 \box255 中时，我们可以在输出阶段解包这个盒子，而后取得 \lastbox[1]：

```
\vbox{\unvbox255{\setbox0=\lastbox}}
```

值得一提的是，如果竖直模式中的盒子被 \moveright 或 \moveleft 调整过位置，或是水平模式中的盒子被 \raise 或 \lower 调整过位置，则使用 \lastbox 访问列表中最后一个元素时，这些位置调整信息都会丢失。

5.3 盒子天然的尺寸[2]

5.3.1 已创建的水平盒子的尺寸

\hbox 内所有的元素都依次排在一行，并且它们的参考点会在齐盒子的基线（baseline）上对齐，除非显式使用 \raise 或者 \lower 调整它们竖直方向上的位置。

盒子的宽度是盒子内所有元素宽度之和。因此下面盒子的宽度为正

```
\hbox{\hskip1cm}
```

而下面盒子的宽度为负

```
\hbox{\hskip-1cm}
```

具体举例来说：

```
a\hbox{\kern-1em b}--
```

的输出是

ba

这表明，水平盒子可以有负的宽度。

\hbox 的高度和深度是盒子内所有元素在盒子基线上方和下方的最大值。水平盒子的高度和深度都是非负的。

\raise 和 \lower 命令是在 \hbox 内调整元素水平位置的唯一方法（当然，除了在 \hbox 中嵌套一个新的 \vbox）；TEX 不允许在水平盒子中使用竖直命令（⟨vertical command⟩，例如 \vskip）；此外，虽然我们可以在水平盒子中使用 \par，但它却不会有任何作用。

[1] 译注（Liam0205）：关于这一技巧，这里有一个完整的示例：https://gist.github.com/91fd658069b0d3dfb7dbd2cba69b856a

[2] 译注（Liam0205）：这里指的是没有其他拉伸、收缩的影响下，盒子自身的尺寸。

5.3.2 已创建的竖直盒子的尺寸

在 \vbox 内，竖直元素的参考点与盒子的参考点在竖直方向上对齐，除非显式使用 \moveright 或者 \moveleft 调整它们水平方向上的位置。

竖直盒子的参考点总是在盒子的左边界上。竖直盒子的宽度是盒子内所有元素的右边界相对参考点向右超出距离的最大值。盒子内元素相对参考点向左超出的距离则不被计算在盒子的宽度之中。因此，与前例相对应，下列代码的结果

```
a\vbox{\hbox{\kern-1em b}}--
```

是

ba–

由 \vbox 和 \vtop 生成的竖直盒子在计算高度和深度时会有所不同。基本原则是，\vbox 的参考点位于最后一个元素的基线之上，而 \vtop 的参考点位于第一个元素的基线之上。一般来说，只要最后一个（第一个）元素是盒子或者标尺（rule），则 \vbox（\vtop）的深度（高度）非零。

\vbox 的高度是除最后一个元素之外所有元素高度与深度之和，再加上最后一个元素的高度；\vbox 的深度则是最后一个元素的深度。\vtop 的高度是第一个元素的高度；\vtop 的深度则是除第一个元素之外所有元素高度与深度之和，再加上第一个元素的深度。也就是说，对于 \vtop 来说，其深度是所包含所有元素的高度与深度之和，再减去第一个元素的高度。

如果 \vtop（\vbox）的第一个元素（最后一个元素）不是盒子或标尺，那么实际规则还要更复杂一点。如果 \vbox 的最后一个元素是铅空（kern）或伸缩胶（glue），则盒子的深度为零；而若 \vtop 的第一个元素是铅空或伸缩胶，则盒子的高度为零。（注意这些定义中的不对称性；可参考后续示例）

关于竖直盒子，还有一个限制：如果 \vbox 或者 \vtop 的深度超出 \boxmaxdepth，则盒子的参考点会下移这一超出的数值。具体来说，超出限制的那一部分深度，会被加在盒子自身的高度之上。如果盒子有 to 或者 spread 声明符，则相关的伸缩胶都会被重新设置，以将追加的高度考虑在内。

通常，\boxmaxdepth 的值是 TEX 能表示的最大尺寸。但在一些情况下，例如，在 plain TEX 的输出阶段的一些计算中，它的值会被修改得小很多。详见第 ?? 章。

5.3.3 例子

水平盒子相较而言简单直接一些。水平盒子的宽度是盒子两条竖直边界之间的距离：从开始到结束，因此，水平盒子的宽度不一定是正数。例如在下例中，\box1 的宽度为零。

```
\setbox0=\hbox{aa} \setbox1=\hbox{\copy0 \hskip-\wd0}
```

具体来说

/\box1/ gives '/a̸a'

水平盒子的高度和深度则不能为负。例如在下例中，\box1 的深度是 0pt 而高度是 15pt。

```
\setbox0=\hbox{\vrule height 5pt depth 5pt}
\setbox1=\hbox{\raise 10pt \box0}
```

竖直盒子相较而言就要麻烦不少。我们首先来讨论其宽度。首先，如下定义的水平盒子的宽度是 10pt。

```
\setbox0=\hbox{\hskip 10pt}
```

则如下定义的竖直盒子宽度是 5pt。

```
\setbox1=\vbox{\moveleft 5pt \copy0}
```

这是因为，在竖直盒子中，位于参考点左边的内容不计入其宽度。同样的，如下定义的竖直盒子宽度是 15pt。

```
\setbox2=\vbox{\moveright 5pt \copy0}
```

如果 \vbox 的最后一个元素是盒子，则其深度是最后一个元素的深度。因此，如下定义的 \vbox 高度为 10pt 深度为 5pt。

```
\vbox{\vskip 5pt \hbox{\vrule height 5pt depth 5pt}}
```

如下定义的 \vbox 的高度则是 0pt 而深度为 5pt。

```
\vbox{\vskip -5pt \hbox{\vrule height 5pt depth 5pt}}
```

如果最后一个元素是铅空或伸缩胶，则竖直盒子的深度为零。因此，如下定义的 \vbox 高度为 15pt 深度为 0pt。

```
\vbox{\hbox{\vrule height 5pt depth 5pt}\vskip 5pt}
```

下列 \vbox 的高度为 5pt 深度则是 0pt。

```
\vbox{\hbox{\vrule height 5pt depth 5pt}\vskip -5pt}
```

与 \vbox 的深度对应，\vtop 的高度基本上就是其第一个元素的高度。重复之前的例子，则如下定义的 \vtop 高度为 0pt 深度为 15pt。

```
\vtop{\vskip 5pt \hbox{\vrule height 5pt depth 5pt}}
```

如下定义的 \vtop 高度为 0pt 深度为 5pt。

```
\vtop{\vskip -5pt \hbox{\vrule height 5pt depth 5pt}}
```

如下定义的 \vtop 高度为 5pt 深度为 10pt。

```
\vtop{\hbox{\vrule height 5pt depth 5pt} \vskip 5pt}
```

如下定义的 \vtop 高度为 5pt 深度为 0pt。

```
\vtop{\hbox{\vrule height 5pt depth 5pt} \vskip -5pt}
```

5.4 继续讨论盒子的尺寸

5.4.1 预先指定尺寸

盒子的尺寸可以通过尺寸说明符(⟨box specification⟩)来指定;具体语法可见上文。盒子中的伸缩胶会依需求缩放,使得盒子的尺寸达到上述设定值。预先指定盒子的尺寸的方法是:

`\hbox to` ⟨dimen⟩ `{...}`, `\vbox to` ⟨dimen⟩ `{...}`

如果盒子中有可以拉伸或收缩的伸缩胶,则这些伸缩胶会被拉伸或收缩,以将盒子的尺寸调整至指定的大小。与伸缩胶伸缩相关的是所谓的劣度值(见第 **??** 章)。如果伸缩胶能够提供的拉伸或收缩值不足所需,则得到的盒子会欠满或溢出。后文有关于欠满和溢出的报错的相关讨论。

还有一个命令可使盒子实际的尺寸异于其天然尺寸。

`\hbox spread` ⟨dimen⟩ `{...}`, `\vbox spread` ⟨dimen⟩ `{...}`

这种情况下,TEX 会使盒子内的伸缩胶拉伸或收缩,以在盒子天然尺寸的基础上缩放⟨dimen⟩。

与 `\vtop` 配合使用尺寸说明符相对来说比较复杂。TEX 构造 `\vtop` 时会先构造一个 `\vbox`,同时会将尺寸说明符传递给该 `\vbox`;而后再根据上述规则计算盒子的高度和深度。

伸缩胶相关设置将在第 **??** 章中讨论。

5.4.2 改变盒子的尺寸

我们可以用 `\ht`、`\dp` 和 `\wd` 来分别获得盒子寄存器的高度、深度和宽度。举例来说,`\dp18` 会给出 18 号盒子的深度。盒子的尺寸不光可读,还可写。通过给盒子的尺寸赋值,我们可以在某种意义上愚弄 TEX,使得在 TEX 看来某个盒子的尺寸不同于其真实值。不过,修改盒子尺寸并不会修改其中的内容;特别地,不会修改伸缩胶设置的方式。

不少格式(format)都定义了 `\smash` 宏:

```
\def\smash#1{{\setbox0=\hbox{#1}\dp0=0pt \ht0=0pt \box0\relax}}
```

在宏定义中,`\smash` 命令将内容保存在盒子中,但是将盒子的高度和深度都值为零;这也就是说,最终盒子会被输出,但是后续有关盒子尺寸的计算中,TEX 都会将高度和深度认作是零。

改变盒子尺寸只能通过为其赋值来实现:这些盒子尺寸(⟨box dimen⟩s)能且只能通过盒子尺寸赋值语句(⟨box size assignment⟩)来设置,而不能用诸如 `\advance`

[4] 译注(Liam0205):这里指的是没有其他拉伸、收缩的影响下,盒子自身的尺寸。

的命令来修改。

注意，盒子尺寸赋值总是全局赋值。赋值的效果会超出任意限定它的分组（参见第 ?? 章）。因此下列代码的输出为：「ab–」。

```
\setbox0=\hbox{---} {\wd0=0pt} a\box0b
```

上述有关被创建的盒子的尺寸限制并不适用于显式修改盒子尺寸的情况。例如，对水平盒子设置 \dp0=-2pt 是合法的。

5.4.3 移动盒子

在水平盒子中，所有元素的参考点与盒子自身的参考点都位于同一条水平线上。水平盒子中的盒子可用 \raise 或 \lower 来上移或下移。

相似地，在竖直盒子中，所有元素的参考点与盒子自身的参考点都位于同一条竖直线上，且各个元素从上到下依次排列。竖直盒子中的盒子可用 \moveleft 或 \moveright 来左移或右移。

注意，上述命令只能用来移动盒子，特别地，这些命令不能用于字符或标尺。

5.4.4 盒子的尺寸与盒子的摆放

在确定水平列表和竖直列表中元素的摆放位置时，TEX 需要维护两个变量：当前参考线以及在参考线上的当前位置。对于水平列表来说，参考线是包围当前内容的 \hbox 的基线；对于竖直列表来说，参考线是穿过包围当前内容的 \vbox 的参考点的竖线。

在水平模式中，TEX 放置元素的逻辑如下。初始时，参考线上的当前位置即是包围当前内容的从水平盒子的参考点。此后，TEX 会依如下步骤操作。

1. 如果元素被 \raise 或 \lower 上下移动了位置，则相应地调整元素的位置。
2. 如果元素是水平盒子，则对其内容递归地调用该算法；如果元素是竖直盒子，按盒子的高度向上移动当前位置，而后按照后文所述的有关竖直盒子的算法放置盒子内竖直列表中的元素。
3. 按当前元素的宽度，沿着参考线向右移动当前位置。

对于竖直盒子中的竖直列表，TEX 的会将当前位置初始设置在盒子的左上角，参考线则是穿过该点的竖线——这条竖线同样也会穿过盒子的参考点。竖直盒子中的元素依照如下逻辑摆放。

1. 如果元素被 \moveleft 或 \moveright 上下移动了位置，则相应地调整元素的位置。
2. 将元素的左上角与当前位置对齐。
3. 如果元素是竖直盒子，则对其内容递归地调用该算法；如果元素是水平盒子，按

盒子的高度向下移动当前位置，而后按照前文所述的有关竖直盒子的算法放置盒子内水平列表中的元素。

4. 按当前元素的高度与深度之和，沿着参考线向下移动当前位置。

注意，上文中没有描述 TEX 是如何构建盒子的。对于竖直盒子，构建盒子的过程还要包括插入行间距伸缩胶（行间胶，baselineskip glue）的过程。上文介绍的过程描述的是 TEX 会如何输出一个已经构建完成的盒子中的内容。

5.4.5 盒子与反向伸缩胶

默认情况下，盒子在一条参考线上依次摆放。但有时，让盒子互相重叠也是很有用的。让盒子互相重叠，最简单的办法是使用反向行间胶。在水平模式中，下列代码会摆放编号为 8 的盒子，但不会移动当前位置。

```
{\dimen0=\wd8 \box8 \kern-\dimen0}
```

更实用的是使用 \llap 和 \rlap 命令。它们的定义如下：

```
\def\llap#1{\hbox to 0pt{\hss #1}}
\def\rlap#1{\hbox to 0pt{#1\hss}}
```

这两个命令允许其中的内容基于当前位置向左或向右突出。这里，\hss 伸缩胶等价于 \hskip 0pt plus 1fil minus 1fil，它能吸收 \llap 或 \rlap 的参数中的任意正负宽度。

例子：下列代码

```
\llap{\hbox to 10pt{a\hfil}}
```

等价于

```
\hbox{\hskip-10pt \hbox to 10pt{a\hfil}}
```

它们的宽度都是 0pt。

5.5 盒子的欠满与溢出

如果盒子有尺寸说明符则 TEX 会拉伸或收缩其中的伸缩胶。对于仅有有限伸缩能力的伸缩胶，TEX 会计算其伸缩过程中的劣度（见第 ?? 章）。在 TEX3 中，用户可通过 \badness 宏来检查刚刚构建的盒子的劣度。劣度的取值范围是 0–10 000；而如果盒子溢出了，则其劣度为 1 000 000。

如果 TEX 认为劣度过大，则会给出一条诊断信息。我们首先考虑对水平盒子的报错。

若 \hbadness $\geqslant$ 10 000，则 TEX 不会对需要拉伸其中伸缩胶的水平盒子报错；否则，当水平盒子中的伸缩胶拉伸的劣度超过 \hbadness 时，TEX 会提示盒子欠满

（underfull）。

伸缩胶的收缩可能引发盒子溢出的报错：如果盒子内伸缩胶最大的收缩量不足以使盒子的宽度满足指定的宽度，则 TEX 认为该盒子溢出了。当且仅当收缩量不足的量大于 \hfuzz 或者 \hbadness < 100 时（这会使得所有可用的收缩的劣度为 100），TEX 才会报告盒子溢出的问题。

例子：如果盒子内伸缩胶的最大收缩量不足以使盒子的宽度减小到指定的尺寸，则若设置 \hfuzz=1pt，TEX 会忽略上述不足的部分小于 1pt 的盒子而不报错。

```
\hbox to 1pt{\hskip3pt minus .5pt}
\hbox to 1pt{\hskip3pt minus 1.5pt}
```

这里只有第一个盒子会报错：它比目标尺寸大了 1.5pt；但是第二个盒子不会报错，因为它只比目标尺寸大了 0.5pt，小于 \hfuzz。

如果一个盒子虽不至变成溢出盒子但也被收缩了，也有可能被 TEX 报告问题：如果一个盒子比较紧凑，也就是说，如果这个盒子使用了至少一半的收缩能力，则在劣度（介于 13 到 100 之间）超过 \hbadness 时，TEX 会报告这个问题。

对水平盒子与竖直盒子来说，相关警告的逻辑几乎完全相同。对于竖直盒子来说，有 \vbadness 以及 \vfuzz 来控制相关报错。区别之处在于，如果一个水平盒子溢出了，则 TEX 会在盒子右侧绘制一个高度与盒子相同宽度为 \overfullrule 的溢出标尺。此处有一个例外，在 \halign 中，如果 \tabskip 伸缩胶无法提供足够的收缩量时，TEX 不会绘制溢出标尺。

5.6 盒子的开始与结束

盒子的内容以花括号定界。其中，左右花括号既可以是显式字符（分类码为 1 和 2 的字符），也可以是使用命令 \let 定义的隐式字符。TEX 会将记号 \everyhbox 或者 \everyvbox 插入在左花括号之后。如果盒子位于 \setbox 赋值语句中，则记号 \afterassignment 会在「everybox」记号之前插入在左花括号之后。

例子：

```
\everyhbox{b}
\afterassignment a
\setbox0=\hbox{c}
\showbox0
```

的结果是：

```
> \box0=
\hbox(6.94444+0.0)x15.27782
.\tenrm a
.\tenrm b
```

```
.\kern0.27779
.\tenrm c
```

借助隐式花括号，我们可以通过宏来开始和结束盒子。比如下面的代码就用到了这一特性：

```
\def\openbox#1{\setbox#1=\hbox\bgroup}
\def\closebox#1{\egroup\DoSomethingWithBox#1}
\openbox0 ... \closebox0
```

利用这一机制，我们可以提取整个自然段的内容[5]：

```
\everypar{\setbox\parbox=
   \vbox\bgroup
        \everypar{}
        \def\par{\egroup\UseBox\parbox}}
```

此处，\everypar 命令[6]开始了一个盒子，并且将自然段的文本内容保存在盒子中。假设段落的开头是：

```
Begin a text ...
```

这就等价于[7]

```
\setbox\parbox=\vbox{Begin a text ...
```

由于在盒子中，\par 被重定义了，所以，假设段落的结尾是[8]：

```
... a text ends.\par
```

等价于

```
... a text ends.}\Usebox\parbox
```

在这个例子中，\UseBox 命令只能将盒子作为一个整体对待；如果想要分别处理盒子中的元素，则需要使用新的技巧。在下列代码中，\HandleLines 可以访问自然段竖直列表中连续的元素：

```
\everypar{\setbox\parbox=
 \vbox\bgroup\everypar{}%
      \def\par{\endgraf\HandleLines
               \egroup\box\parbox}}
\def\HandleLines{ ... \lastbox ... }
```

第 69 页还有一个相关例子。

[5] 译注（Liam0205）：这是一个在 plain TEX 下的示例，而非是 LATEX 下的。因此，这里的 \parbox 不是 LATEX 里的段落盒子。关于这一技巧，此处有一完整示例：https://gist.github.com/1d8b9bd2f401eb0ea970a47693723ab1。

[6] 译注（Liam0205）：\everypar 命令的参数，会被插入在每个开启自然段的水平列表的头部，位于缩进盒子之后。

[7] 译注（Liam0205）：注意这里忽略了盒子开头位置的 \everypar 以及对 \par 的重定义。

[8] 译注（Liam0205）：注意这里的 \par 可以是手工插入的，也可以是输入处理器在状态 *S* 遇见行尾符自动插入的。

5.7 将盒子解包

使用 \unhbox 和 \unvbox 可将盒子解包；\unhcopy 和 \unvcopy 是对应的拷贝版本。由于它们会构建部分的水平列表和竖直列表，它们分别是水平命令和竖直命令（见第 ?? 章）。注意，不能用 \unhbox 去解包一个寄存了 \vbox 的盒子寄存器；反过来也一样不行。不过，空寄存器可以用 \unhbox 也可以用 \unvbox 解包。

解包操作会将盒子寄存器中盒子的内容取出，而后追加在盒子所处的列表中；所有的伸缩胶会被重新设置。因此下列代码

```
\setbox0=\hbox to 1cm{\hfil} \hbox to 2cm{\unhbox0}
```

与下列代码完全相同

```
\hbox to 2cm{\hfil}
```

且与下列代码不同

```
\hbox to 2cm{\kern1cm}
```

\unhbox 的这一本质特性被用来定义 \leavevmode：

```
\def\leavevmode{\unhbox\voidb@x}
```

该命令会使 TEX 从竖直模式切换到水平模式，而不向水平列表中添加任何额外的东西。当然，由模式切换出发的 \indent 会在水平列表中添加一个缩进盒子。在水平模式中，\leavevmode 没有任何作用。注意，此处没必要使用 \unhcopy，因为寄存器本身就是空的。

还有一个细节需要注意：在竖直模式中解包竖直盒子的操作不会在前序内容与盒子内容之间添加任何行间伸缩胶。此外，\prevdepth 的值也不会发生改变，因此，盒子内容之间的伸缩胶以及盒子内容与后续内容之间的伸缩胶仅在盒子前面有东西存在的时候才会生效；并且，行间伸缩胶的只取决于之前内容的深度。相似地，在水平模式中解包水平盒子的操作不会改变 \spacefactor。

5.8 盒子中的文本

水平盒子与竖直盒子里都能装文本，但它们处理文本的方式有所区别。在水平盒子中，文本被摆放在一条直线上，并且盒子的宽度在原则上是文本（及盒子内其它元素）的自然宽度。水平盒子中不允许有竖直命令（⟨vertical command⟩），此外 \par 在水平盒子里没有任何作用。

竖直盒子相对来说就要复杂得多了。在竖直盒子中，一旦遇到字符或其他任何水平命令（⟨horizontal command⟩，见第 ?? 页），TEX 就会开始在非受限水平模式构建段落；也就是说，仿佛这一段落是直属于页面的一部分。而当遇到竖直命令（⟨vertical command⟩，见第 ?? 页）或盒子结束时，TEX 会使用当前的参数值（例如 \hsize）将

该段落分段成行。

因此，下列代码的不会输出宽度为 3 厘米的段落（如果 \hsize > 3cm，则水平盒子会溢出）：

```
\hbox to 3cm{\vbox{some reasonably long text}}
```

下列代码才会输出宽度为 3 厘米的段落：

```
\vbox{\hsize=3cm some reasonably long text}
```

竖直盒子中的成段文本会被分段成行，而后每行都会被打包进一个水平盒子中。这些水平盒子会堆叠在内部竖直模式当中，盒子之间有 \baselineskip 和 \lineskip 分割（见第 ?? 章）。这也是页面当中文本内容摆放的过程；只不过水平盒子是在外部竖直模式当中堆叠起来的。

如果内部竖直列表是空的，则 TEX 不会在段落开始处插入 \parskip 伸缩胶。

由于水平盒子中的文本不会被分段成行，受限和非受限水平模式还有这样一个区别：在受限水平模式中，TEX 不会插入 discretionary nodes 和会改变当前语言值的无名项目（whatsit items）。如果文本随后被解包形成段落的一部分，则这可能引发一些问题。

第 ?? 章有关于这些内容的解释，[2] 则提供了解决办法。

5.9 杂记

5.9.1 忘掉 \box

在 \newcount\foo 之后，我们就可以单独用 \foo 来访问这个计数器了。但是，对于盒子来说则必须使用 \box\foo 来访问相应的盒子。其原因在于，TEX 中不存在单独的 \boxdef 命令，而是用 \chardef 命令来代替（详见第 ?? 章）。

例子：假设 \newbox\foo 命令分配了编号为 *25* 的寄存器，则 \foo 与 \char25 等价。

5.9.2 特殊用途的盒子

有些盒子寄存器有特殊用途。

- 编号为 255 的盒子 \box255 在 TEX 内部参与页面输出的例程。
- \voidb@x 是一个数字，对应的盒子寄存器在 plain.tex 中分配；我们可以总是假设它是一个空寄存器。它被用于定义诸如 \leavevmode 等宏。
- 使用 plain TEX 宏 \newinsert 创建新的 \insert 之后，TEX 会预留编号相同的若干寄存器，且编号从 254 开始向前倒数。这些寄存器是：\count、\dimen、\skip 以及 \box。

5.9.3 水平模式中竖直盒子的高度

在水平模式中，竖直盒子的参考点与包围它的水平盒子的参考点在竖直方向上对齐。TEX 会从竖直盒子的左上角，即竖直盒子参考点上方与参考点距离等于竖直盒子高度的点，开始遍历竖直盒子中的内容。改变盒子的高度会改变盒子中内容摆放的高度。

考虑以下代码

```
\hbox{a\setbox0=\vbox{\hbox{b}}\box0 c}
```

其效果是

abc

再考虑以下代码

```
\hbox{a\setbox0=\vbox{\hbox{b}}\ht0=0cm \box0 c}
```

其效果是

a c
b

相对的，在竖直模式中改变水平盒子的宽度不会影响水平盒子摆放的位置。

5.9.4 竖直盒子的更多细节

有两种竖直盒子：\vbox 和 \vtop。嵌套使用两种盒子的结果可能会令人困惑。例如

```
\vtop{\vbox{...}}
```

完全等价于

```
\vbox{...}
```

如前所述，若 \vbox 的最后一个元素是铅空或伸缩胶，则其深度为零；若 \vtop 的第一个元素是铅空或伸缩胶，则其高度为零。前面的几个例子已经通过用铅空作为第一个元素或者最后一个元素演示了这一点，但是，如果 \vtop 的第一个元素不是铅空或伸缩胶，人们有可能忽视其对 \vtop 的影响。举例来说，尽管下列代码中，第一个元素不是铅空或伸缩胶，但其高度为零。这是因为在 \write 指令中包含了一个无名项目，它会被放在竖直列表当中。

```
\vtop{\write16{...}...}
```

修正的办法是通过 \leavevmode 离开竖直模式，切换到水平模式并开启新的自然段，而后将无名项目放在自然段的起始处，而非是自然段的上方。

```
\vtop{\leavevmode\write16{...}...}
```

摆放数值列表中的元素有时会比较棘手。例如，竖直列表中的水平盒子与竖直盒

子的处理方式是不一样的。考虑以下例子，在 \offinterlineskip 之后[9]，第一个例子

```
\vbox{\hbox{a}
      \setbox0=\vbox{\hbox{(}}
      \ht0=0pt \dp0=0pt \box0
      \hbox{ b}}
```

结果是

a
(b

稍稍修改之后

```
\vbox{\hbox{a}
      \setbox0=\hbox{(}
      \ht0=0pt \dp0=0pt \box0
      \hbox{ b}}
```

结果变成了

a
(b

造成这一区别的原因在于，水平盒子的摆放位置由其参考点确定，而竖直盒子的摆放位置由其左上角确定。

5.9.5 将 \lastbox 还回列表

通过 \lastbox 你可以将（内部）竖直列表中最后一个盒子取出。但是，如果你想把取出的盒子换回去，则竖直间距可能会出问题。比如说，如果你这样将它换回去，则行间伸缩胶会有两份。

```
\setbox\tmpbox=\lastbox
\usethetmpbox \box\tmpbox
```

而如果你在换回去之前使用 \unskip 命令，看上去解决了问题，但实际上错得更加微妙：

```
\setbox\tmpbox=\lastbox
\unskip \usethetmpbox \box\tmpbox
```

此时，内部距离（⟨internal dimen⟩）\prevdepth（控制行间伸缩胶；见第 ?? 章）的值会因最后一个盒子而改变，但正确的行间伸缩胶应该由上一个盒子来决定。正确的方法不是使用 \unskip，而是使用 \nointerlineskip：

```
\setbox\tmpbox=\lastbox
\usethetmpbox \nointerlineskip \box\tmpbox
```

[9] 译注（Liam0205）：\offinterlineskip 会禁止 TEX 在行与行之间自动地插入行间伸缩胶；因此，在 \offinterlineskip 之后，每行的 \hbox 会没有缝隙地堆叠在一起。

5.9.6 使用 \lastbox 切分自然段

反复调用 \last... 和 \un... 可以将自然段切分开。这里给出一个例子。

> In typesetting advertisement copy, a way of justifying paragraphs has become popular in recent years that is somewhere between flushright and raggedright setting. Lines that would stretch beyond certain limits are set with their glue at natural width. This single paragraph is but an example of this procedure; the macros are given next.

```
\newbox\linebox \newbox\snapbox
\def\eatlines{
   \setbox\linebox\lastbox    % check the last line
   \ifvoid\linebox
   \else                      % if it's not empty
   \unskip\unpenalty          % take whatever is
   {\eatlines}                % above it;
                              % collapse the line
   \setbox\snapbox\hbox{\unhcopy\linebox}
                     % depending on the difference
   \ifdim\wd\snapbox<.98\wd\linebox
         \box\snapbox % take the one or the other,
   \else \box\linebox \fi
   \fi}
```

它的使用方式是

```
\vbox{ ... some text ... \par\eatlines}
```

或者也可以用 \everypar 自动插入；参见 [3]。

宏 \eatlines 会从竖直列表取得 \lastbox。如果列表为空，则 \ifvoid 将返回真。这些盒子都是水平盒子，每个盒子包含了段落中的一行：如果使用 \ifhbox 来判断盒子分类的话，将返回真。

索引

参考文献

[1] B. Beeton. Controlling <ctrl-M>; ruling the depths. *TUGboat*, 9:182–183, 1988. 33

[2] M.J. Downes. Line breaking in \unhboxed text. *TUGboat*, 11:605–612. 66

[3] V. Eijkhout. Unusual paragraph shapes. *TUGboat*, 11:51–53. 69

[4] D.E. Knuth. *Computer Modern Typefaces*. Addison-Wesley. 51

[5] D.E. Knuth. The new versions of TEX and Metafont. *TUGboat*, 10:325–327. 52

[6] D.E. Knuth. *TEX: the Program*. Addison-Wesley, 1986. 39, 49

[7] D.E. Knuth. Virtual fonts: more fun for grand wizards. *TUGboat*, 11:13–23, 1990. 48

[8] D.E. Knuth. *The TEX book*. Addison-Wesley, reprinted with corrections 1989. 33, 44

[9] R. Southall. Designing a new typeface with metafont. In *TEX for scientific documentation, Lecture Notes in Computer Science 236*. Springer Verlag, 1984. 48

版本历史

Vection 1.4

Reinstated a couple of figures (baseline distance, plus one one paragraph shape that didn't make it into the original book.)

Version 1.3

Finally managed to reinstate the tables chapter.

Starting to add more concepts to the index.

Version 1.2

Added chapter references to glossary.

Fixed a bunch of typographic accidents.

Version 1.1

Small remark about `\afterassignment` after macro definitions.

Trouble with indexing macros fixed, I hope.

Separate letter and a4 versions.

Better intro for the chapter **??** on spacing.