

# 关于 L<sup>A</sup>T<sub>E</sub>X 的那些你想知道却从不敢问的问题

或者说，如何在不会使用 L<sup>A</sup>T<sub>E</sub>X 的情况下使用 L<sup>A</sup>T<sub>E</sub>X

Tout ce que vous avez toujours voulu savoir sur L<sup>A</sup>T<sub>E</sub>X sans  
jamais oser le demander

Ou comment utiliser L<sup>A</sup>T<sub>E</sub>X quand on n'y connaît goutte

ver. 1.5

Vincent Lozano 著

January 23, 2023



# Contents

<b>I</b>	<b>关于 L<sup>A</sup>T<sub>E</sub>X 的那些你想知道却从不敢问的问题</b>	<b>11</b>
<b>1</b>	<b>基本原则</b>	<b>13</b>
1.1	安装	13
1.2	“生产”周期	14
1.2.1	编辑	14
1.2.2	编译	16
1.2.3	显示	16
1.2.4	打印	17
1.3	源文件的结构	17
1.3.1	文档的类	18
1.3.2	文前部分	18
1.3.3	添加扩展	19
1.4	开始！	19
1.4.1	几个特殊字符	20
1.4.2	调用指令	21
1.4.3	变音符号	22
1.5	第一批工具	22
1.6	第一组报错	23
1.6.1	症状	23
1.6.2	诊断	24
1.6.3	一些消息	25
1.7	再说几句！	25
<b>2</b>	<b>需要了解的知识</b>	<b>27</b>
2.1	突出显示	27
2.1.1	族-风格-字重	28
2.1.2	意大利体校正	29
2.1.3	字号	29
2.1.4	几个建议	29
2.2	环境	30
2.2.1	居中和对齐	30
2.2.2	列表	32
2.2.3	制表	34
2.2.4	表格	34

2.2.5	模拟终端	36
2.2.6	引用语	36
2.3	页边注	38
2.4	标题	38
2.5	页面底部的注释	39
2.6	页眉和页脚	40
2.7	浮动环境	40
2.7.1	图 (figure) 和表 (table)	40
2.7.2	确定位置	41
2.7.3	图片列表	42
2.8	引用	42
2.8.1	原理	42
2.8.2	需要引用什么?	43
2.9	辅助文件	44
2.9.1	与引用的交互	44
2.9.2	与目录的交互	45
2.9.3	一些建议	46
2.10	断字的处理	46
2.10.1	控制断字	47
2.11	小结	49
<b>3</b>	<b>数学排版</b>	<b>51</b>
3.1	编写数学公式的两种方式	51
3.2	常用指令	52
3.2.1	上标和下标	52
3.2.2	分式和根式	52
3.2.3	符号	53
3.3	函数	55
3.3.1	标准函数	55
3.3.2	积分、求和和其他极限	56
3.4	重叠的符号	57
3.4.1	操作符 not	57
3.4.2	“变音符号”	57
3.4.3	向量	58
3.4.4	指令 stackrel	58
3.5	两个重要原则	58
3.5.1	数学模式的空格	59
3.5.2	数学模式中的文本	59
3.6	阵列 (array): 简单且高效	60
3.6.1	阵列的原理	60
3.6.2	阵列和定界符号	61
3.6.3	说话的方式简单点……	62
3.7	方程和环境	62
3.7.1	环境 displaymath	62
3.7.2	方程环境 equation	63
3.7.3	多行数学式	63

3.8	数学模式的风格	64
3.8.1	字体	64
3.8.2	符号的字号	65
3.8.3	创建新操作符	66
3.9	小结	67
<b>4</b>	<b>成为小魔仙</b>	<b>69</b>
4.1	计数器	69
4.1.1	可用的计数器	69
4.1.2	操作	70
4.1.3	显示	71
4.2	长度	72
4.2.1	单位	72
4.2.2	L <sup>A</sup> T <sub>E</sub> X 中的几个长度	73
4.2.3	操作长度	73
4.2.4	弹性长度	75
4.2.5	显示长度	76
4.3	空间	76
4.3.1	基本长度	77
4.3.2	预定义的空间	77
4.4	字盒	80
4.4.1	简单字盒	81
4.4.2	简单字盒的操作	83
4.4.3	段落字盒	85
4.4.4	小技巧	87
4.4.5	保留和复用	88
4.5	定义	89
4.5.1	指令	89
4.5.2	环境	91
4.5.3	重定义	92
4.6	然后呢?	93
<b>5</b>	<b>图表学</b>	<b>95</b>



# 序

男人的邪恶胜过女人的善良<sup>1</sup>。——《便西拉智训<sup>2</sup>》42:14

## 从前……

一切始于 1990 年年初。我当时正在 PC 286 计算机上使用称为 *WordPerfect* 的软件，以此入门人们所谓的“文字处理”。这款软件现在仍然存在，并且由 Corel 公司维护，运行在日后拥有响当当名头的 MS-DOS 中。MS-DOS 集成了用以粗略预览文档的接口，尤其允许用户“看到代码”，也就是借助一种标记语言将文档可视化，以灵活地控制。

稍晚些时间，随着 Windows 3.1 迅速风靡，人们突如其来地追求图形界面，我虽然仍情有所不甘，却逐渐说服了自己去使用那款在今天很出名的文字处理软件——的 2.0 版（后面还带个小小的字母，在当时那可真是重大的升级）……但我日后才知道，这个版本有个很有趣的“特性”：文件体积过大，超过了某个特定的值时，会出现保存失败的情况！这时，你既不能保存，也不能恢复文档。有些头铁的朋友尝试先删除几行再保存，但这种撞大运的解决方案并没能成功……

当时，大家毫不掩饰地嘲讽这些“你懂的”公司制作的软件<sup>3</sup>——这里就不点名了。我周围的大多数人躺平地选择了接受，认为使用这些堂而皇之的不给出警告的可悲的跟风之流是正常现象。软件的这种“特性”坚定了我的信念：我绝不使用这种软件。当时还在攻读工程师学位的我意识到，我今后的部分工作将会集中在起草文档和使用通用的信息系统上。为此，我需要足够健壮的工具。

我是在让·莫奈大学 (Université Jean Monnet) 和圣-埃蒂安高等矿业学校 (École des Mines de Saint-Étienne) 攻读 DEA (现在叫 master recherche)<sup>4</sup> 时相继接触 UNIX 和 Linux 的。那时 (1993~1994 年)，在我刚写论文的开头时，“拉泰克”(latèque) 这个词就开始围着我转。这里问题似乎是要找到一款能排出数学公式的软件，而说到撰写理科文档，L<sup>A</sup>T<sub>E</sub>X 似乎显然是避不开的唯一答

---

<sup>1</sup> 本书的章首引言来自《旧约》与《新约》，将它们引用在这里纯粹是我一手挑动的——有时，这些句子中带有一些与章标题相关的内容（译注：宗教相关内容按原文直译，不代表译者对任何宗教文献中任何语句的认可或否认。本书未标注“译注”字样的脚注均为原书脚注）。

<sup>2</sup> 译注：原文如此，但作为《诗歌智慧书》一部分的《便西拉智训》（天主教译为《德训篇》）似乎属于次经，即在一些教派中不被承认作为《圣经》的一部分出现。

<sup>3</sup> 这些被嘲讽的对象中，我们可以看到一些名场面：通用汽车公司老板对比尔·盖茨挑衅性言论的回应（译注：可能是指比尔·盖茨的观点，即如果汽车工业能够像计算机领域一样发展，那么一辆汽车只需要 25 美元就能买到，并且消耗 1 加仑汽油就能跑 1000 英里。作为回应，通用汽车方面罗列了一系列言论来嘲讽，例如“如果那样，那么想要汽车熄火，需要点击开始菜单”），以及罗伯特·迪·科斯莫 (Roberto Di Cosmo) 的“赛博空间中的陷阱”(piège dans le cyberspace)。

<sup>4</sup> 译注：DEA 即 diplôme d'études approfondies，法国教育体系下的一种学位。

案。说实话，找软件这种问题甚至都根本没出现过！

于是，我着手把这个叫做  $\text{\LaTeX}$  的“玩意儿”装在 Mac 系统（安装的发行版叫  $\text{\OzTeX}$ ）和另一个由古登堡（Gutenberg）协会支持的发行版系统——Solaris 上。为此，我还得去收买一个系统管理员，让他同意创建一个特权用户 `texadm`，用来管理那个发行版……

1994 年年初，我带着坚定的意志使用  $\text{\LaTeX}$  开始写论文。在 1995 年，在被我发现的种种技巧激起的兴趣的巨大感召下，我着手为同事和实验室起草用于入门  $\text{\LaTeX}$  的指导手册。这个手册就是本书的原型。在 1997 年，在练习了两年并一只脚踏入了排版领域后，我更坚定了自己的看法： $\text{\LaTeX}$  绝对是写严肃文件的首选软件：它有对版面（mise en page）的全面控制，有对参考文献的管理，支持索引（通用名称和作者名），能轻松操作文件。最重要的是，排版的结果很好看。从那时起，这就是支撑我使用  $\text{\LaTeX}$  的最强大而无可争辩的理由。

今天，作为国立圣-埃蒂安工程师学院（École Nationale d'ingénieurs de Saint-Étienne）的计算机高级讲师，我用  $\text{\LaTeX}$  来起草理科文档和教学材料。几年使用下来，我仍然在学习和发现，也仍然会对项目贡献者提出的各种扩展啧啧称奇。这些扩展使  $\text{\LaTeX}$  成为了充满宝藏的巴扎（bazar），成为了一款名副其实地朝着更高工效发展的<sup>5</sup>、始终以“产出优美的工作成果”为目标的卓越而独特的工具。

## 本书结构

本书是针对“使用  $\text{\LaTeX}$  进行文字处理”的介绍。它不是一本参考手册，但本书的写作目标是传授读者使用  $\text{\LaTeX}$  的基本知识，并在可能情况下，让读者对它感兴趣。读者可以在本书中找到开始使用  $\text{\LaTeX}$  的必要信息和起草文档的建议。为了提升阅读体验，我们“高明地”将本书分为了若干章节，并配有附录。本书首先介绍  $\text{\LaTeX}$  的基础知识：

**基本原则** 展示  $\text{\LaTeX}$  的基本原理。为了读懂本书的剩余部分，需要阅读本章。

**需要了解的知识** 展示标准工具。为了起草一篇简单的文档，需要了解这些知识。

**数学排版** 如何生成数学式。

然后有如下附录：

我们建议您先从第 1 章一路读到数学部分。其余的章节相对独立，可以根据需要阅读。再强调一遍，我们建议在熟练掌握了基础概念之后再去阅读本书的第 II 部分。文档最后的索引提供了查询所需内容的快捷入口。最后，正如同其他关于  $\text{\LaTeX}$  的答疑解惑的法文资料，我没有费神地将所有  $\text{\LaTeX}$  术语和计算机术语逐一翻译。

## 你需要知道的知识

本书适用于初学者阅读，不要求读者有关于  $\text{\LaTeX}$  的任何知识。然而，本书读者应当具有基本的、有关操作系统和计算机用户的知识。本书读者最好懂得如何从使用绘图软件或图片处理软件开

---

<sup>5</sup>并不是指那些诸如在菜单中添加一个功能入口、在弹出对话框时添加个提示音的“提效”。



始，创建一个封装在其计算机系统上的 PostScript 文件。

## 你不会通过本书学习到的知识

你正阅读的这本图书在令人称赞的同时也有以下知识面漏洞。

- 本书不含有关于  $\text{T}_\text{E}\text{X}$  或  $\text{L}^{\text{A}}\text{T}_\text{E}\text{X}$  生成字体原理的清晰解释。你不会找到关于“元字体” (METAFONT) 一词的知识。
- 你不会找到关于在 UNIX 系统下安装  $\text{L}^{\text{A}}\text{T}_\text{E}\text{X}$  发布版的知识。
- 你不会找到任何现有扩展包的“目录”或清单，无论扩展包是否实用、是否兼容。
- 本书回避了“先有鸡还是先有蛋”之类的问题，也避免讨论关于上帝和科学的问题。
- .....

① 不要对本书的内容抱有不切实际的幻想：本书书名着实是个不要脸的谎言。

## $\text{T}_\text{E}\text{X}$ 是什么？

唐纳德·欧文·克努特 (Donald Ervin Knuth) ——就是那个有着众多关于数学和算法的著作 [包括《计算机程序设计的艺术》(英: *The Art of Computer Programing*)] 的数学家——对 20 世纪 70 年代的技术条件下打印出来的文章的样子深感失望，产生了开发称为  $\text{T}_\text{E}\text{X}$  的文字处理系统的初步想法。20 世纪 80 年代初次公布的  $\text{T}_\text{E}\text{X}$  是由一个宏处理器 (processeur de macro ; 英: macro processsor) 和几个基元 (primitive) 组成的复杂系统。第一组预编译的宏很快以“普通格式” (*format plain*) 的名义出现。

注意， $\text{T}_\text{E}\text{X}$  既不是文字处理器 [克努特将其称为“typesetting system”，可以翻译成“排字系统” (système de composition)] 也不是一种编译后的编程语言。这是克努特关于  $\text{T}_\text{E}\text{X}$  的一些说明<sup>6</sup>：

“英文的‘technology’一词由希腊文词根‘ $\tau\epsilon\chi$ ...’演变而来，这个词根有时也指艺术和科学技术。 $\text{T}_\text{E}\text{X}$  由此而来，正是  $\tau\epsilon\chi$  的大写形式。”

关于  $\text{T}_\text{E}\text{X}$  中“X”的发音：

“……它的发音像德语单词 ach 中的‘ch’，或西班牙语中的‘j’……如果你对着电脑正确地发音，屏幕上会出现哈气。”

<sup>6</sup>出自  $\text{T}_\text{E}\text{X}$ book 的“The Name of the Game”一章。

你谦逊的仆人可能会更想让你读成“TeK”，来避开那种有气无力的感觉和没过几天就要给你擦一次电脑的麻烦工作。

最后，对于  $\text{T}_{\text{E}}\text{X}$  的标识设计，克努特强调字母 E 需要稍微错位一些，以提示人们这是关于排版的工具。对于确实会遇到的一些无法使字母 E 稍微错位的情况，他坚持道，需要将  $\text{T}_{\text{E}}\text{X}$  写成“ $\text{TeX}$ ”。

目前， $\text{T}_{\text{E}}\text{X}$  的最新版本号是 3.1415926（没错，它收敛于  $\pi$ ）。在  *$\text{T}_{\text{E}}\text{X}$ : the program* 一书的前言中，克努特估测上一个程序漏洞已于 1985 年 11 月 27 日发现并改正，并出价 20.48 美元来悬赏下一个漏洞。今天，这个十六进制的金额停留在 327.68 美元，如果有人喜欢 2 的幂，这个数字应该会让他满意……

## $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 是什么？

1985 年， $\text{T}_{\text{E}}\text{X}$  已经传播了一段时间，莱斯利·兰波特（Leslie Lamport）将宏组合起来，创造了一个视野更广的格式，称为  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ，版本号为 2.09。今天， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  已经成为了事实标准，只有一些生古的情况才会只支持  $\text{T}_{\text{E}}\text{X}$  而不支持  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 。然而， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  有点像  $\text{T}_{\text{E}}\text{X}$  的“镀层”，提供  $\text{T}_{\text{E}}\text{X}$  的宏的调用。有时，掌握  $\text{T}_{\text{E}}\text{X}$  中的部分概念有助于从困难的处境中脱身。兰波特在他的书中这样说 [10]：

“可以将  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  想象成一幢房子，它的构架和钉子就是由  $\text{T}_{\text{E}}\text{X}$  提供的。如果你只是在房子中生活，那么你不需要准备钉子、搭建构架，但如果想要为房子新增一个房间，那么你就会需要它们。”

他还说道：

“ $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  的出名是因为它允许作者从排版工作中抽离，并且专注在写作上。如果你在形式上花费了太多时间，那么你并没有很好地使用  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 。”

从 1994 年至今，一个由欧美成员组成的团队 [以弗朗克·米特尔巴赫（Frank Mittelbach）为核心] 着手  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  的开发。1994 年发布的  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  版本称作  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ 。团队的长期目标是孵化一个名为  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$  的系统。

## 使用许可

画重点： $\text{T}_{\text{E}}\text{X}$  和  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  属于自由软件——也因此是免费的。同时，自由软件（logiciel libre；英：free software）的标志是其开放性。因此， $\text{T}_{\text{E}}\text{X}$  也可以有其 Web 源码<sup>7</sup>。 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  的宏是以  $\text{T}_{\text{E}}\text{X}$  源码的格式发布的。对于大部分用户来说，获取程序的源码可能不是首要考虑的，但需要知道，正是这种不隐藏任何内容的性质，使得人们可以改进现有的扩展、创造新的扩展。

一款软件是自由软件，并不意味着我们可以使用它做任何想做的事情。自由软件属于其作者，所有的改动都需要被记录。同样，每次改动都需要以与具有与改动前不同的文件名体现。这样可以保证系统的严密和便携（关于  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$  的使用许可，请参阅 <ftp://ftp.lip6.fr/pub/TeX/CTAN/macros/latex/base/lppl.txt>）。

<sup>7</sup> 克努特孕育的 Web 语言被形容为一种“文学性的编程语言”。使用 Web 源码，可以生成程序的 Pascal 或 C 代码，也可以为代码生成  $\text{T}_{\text{E}}\text{X}$  文档。

## 不使用 L<sup>A</sup>T<sub>E</sub>X 的五个理由

在一些情况下，强烈建议不使用 L<sup>A</sup>T<sub>E</sub>X。具体来说，这些不使用 L<sup>A</sup>T<sub>E</sub>X 的理由如下。

1. 你只将文字处理器用于制作贺卡、写邮件、记录几个想法等用途。
2. 你十分喜欢鼠标（可能具有 1~3 个按键），并且认为输入方程的唯一方式就是频繁地使用鼠标点来点去。
3. 你觉得 UNIX 是一个“让人头痛”且“不易使用”的系统，或者你对所有的编程语言都有着强烈的反感。
4. 你认为以下情况是正常的：
  - (a) 新版软件不能读取其旧版本创建的文档；
  - (b) 要使用新版软件，必须换一个操作系统；
  - (c) 要使用新版操作系统，必须换一台计算机；
  - (d) 要使用新计算机，必须……
5. 你不知道键盘上的“\”键在哪里。

如果你的情况满足以上任何一条，最好在你现在的系统上知足常乐。

## 使用 L<sup>A</sup>T<sub>E</sub>X 的若干理由

说服本书读者使用 T<sub>E</sub>X 和 L<sup>A</sup>T<sub>E</sub>X 而不是其他系统似乎不成问题——毕竟，你都读这本书了，也就已经不知不觉被说服了。让我们看看 T<sub>E</sub>X 的设计者是怎么说的：

“在使用 T<sub>E</sub>X 起草文档时，你就是在指挥计算机如何准确地把你的稿件转化为几个页面，以媲美世界上最好的打印机能够实现的排版样式。”——D.E. 克努特，T<sub>E</sub>Xbook[9]

T<sub>E</sub>X 和 L<sup>A</sup>T<sub>E</sub>X 可以生成无与伦比的文档（并可以极细微地调整<sup>8</sup>），这显然归功于它们的以下能力：

- 仔细地绘制字体；
- 处理排版上的细节，包括连接号（tired）和合字，比如你可能没有观察过的以下情况：
  - “avez-vous — bien — regardé ces tirets (page 19–23)” 这句文字中的各种连接号；
  - fin 一词中的“fi”、souffle 一词中的“ffl”，以及 trèfle 一词中的“fl”；

---

<sup>8</sup>作为参考，L<sup>A</sup>T<sub>E</sub>X 内置的衡量单位是比例点（英：scaled point），在 T<sub>E</sub>Xbook 中记作 sp，合 1/65536 点；1 点合约 1/72 英寸；1 英寸合 2.54 厘米。比例点可以在大约 50 埃米的尺度上调整文档。目前打印机的分辨率对于这个尺度来说，实在是太充裕了。

- 性能良好的断字算法；
- 专门针对数学公式的呈现。

此外， $\text{\LaTeX}$  是少数瞄准科技文档的文字处理软件。这是因为，除了处理方程和公式之外外， $\text{\LaTeX}$  还有大量围绕起草文章、生成参考文献和索引的功能。

最后， $\text{\LaTeX}$  尤其针对大文件的生成做了适配。这不仅是由于处理  $\text{\LaTeX}$  文档本身占用的内存空间极小，也是因为宏和交叉引用（*référence croisée*；英：cross reference）可以让我们对文件有着全面而灵活的控制。

**交叉引用**  $\text{\LaTeX}$  允许以符号的形式于文档的任何位置引用有编号的对象。此外，标题、图片、表格、方程、参考文献、列表、定理等的序号都可以在文章的多个位置以简单的方式引用，不需要我们去关心具体的号码本身是多少。

**宏** 宏无疑是  $\text{\LaTeX}$  最强大的功能。要知道，生成文档的所有过程对视都是一系列指令或宏。因此，每个用户都可以文件中的宏来改变文件的生成情况。显然，我们可以很好地定义我们自己的宏，使得文档的一部分呈现出特殊的效果。围绕宏的一个很强烈的观点是，我们原则上可以将设置格式的部分从起草文档的过程中分离出去。

## 所见即所得的缺陷

$\text{\LaTeX}$  处于“所见即所得”<sup>9</sup> 的对立面，因为  $\text{\LaTeX}$  源码是包含文本内容本身和页面布局命令的本文档。兰波特将这种方法成为逻辑页面布局而不是视觉页面布局<sup>10</sup>。

然而，我们也可以说  $\text{\LaTeX}$  是“所见即所得”的，因为在编译后，我们就可以在屏幕上呈现出文档未来出现在纸面上的精确形态。

以下是一个能够说明“所见即所得”缺陷和逻辑页面布局优势的案例<sup>11</sup>：假设在一个文档中，某个具有两个参数的函数出现了一定次数。科学文档的一个精巧之处就是可以使用符号，我们可以定义一个宏 `\mafct` 来生成这个函数。如此一来，`\mafct{1}{2.5}` 和 `\mafct{x}{t}` 会分别生成  $\mathcal{F}_{\alpha,\beta}(1,2.5)$  和  $\mathcal{F}_{\alpha,\beta}(x,t)$ 。一旦我们想要改变符号，只需重新定义 `\mafct` 这个宏，以在需要的位置重新生成对应的符号（比如分别生成  $F^{\alpha,\beta}[1,2.5]$  和  $F^{\alpha,\beta}[x,t]$ ），就可以了！

另一个例子是：假设你的文件中有很多科技词汇，你想要用一种特殊的形式展示她们。因此你事先定义了宏 `\jargon`<sup>12</sup>，以将科技词汇设置为意大利体，并在在文档中写下 `\jargon{implémentation}` 之类的实现。你的文档中以这种方式提到了 235 个术语词汇。你如果改变了主意，想把它们从意大利体改成其他的格式，那么只需要重新定义宏 `\jargon`，而不需

<sup>9</sup>英：What you see is what you get，简称为 Wysiwyg。指软件允许用户在屏幕上看到即将在纸上获得的结果完全相同的内容。第一款“所见即所得”的文本处理器大概是 Bravo，它于 1974 年出现在施乐帕罗奥多研究中心（英：Xerox Palo Alto Research Center）的机器 Alto 上。

<sup>10</sup>说点不好听的，根据兰伯特在其关于  $\text{\LaTeX}$  的书中的描述，“所见即所得”的软件被柯尼汉 [译注：Kernighan，可能指 UNIX 开发者布莱恩·W. 克尼汉（Brian W. Kernighan）] 描述为“只能看到已经有的东西”。

<sup>11</sup>兰波特在它的手册中提到了一个类似的案例。

<sup>12</sup>译注：jargon 意为“术语”。

要逐个排查那 235 处术语词汇。经过一些练习以后，你甚至能使这个宏自动将术语插入文档的索引中……

以下是一个略微变形的例子：在稍前的名为“不……的五个理由”的章节标题中，我在源码中完全没有写“五<sup>13</sup>”这个字。标题是使用“……的`\ref{nbraisons}` 个理由”这样的句子写成的，它会自动将文中提到的不使用  $\text{\LaTeX}$  的理由的数量替换为对应的法文词汇。这样，如果还想在列表中再插入一条理由，我就不用重新统计一遍数量了<sup>14</sup>。

① 阅读本书的过程中，会有其他案例会为你逐一呈现“所见即所得”的缺陷。这个“说明”（nota）段落为你指出了一些重要的知识点，同时也是另一个案例。理由是：在作者输入这行文字的时候，展示“警示牌”般的版式是细枝末节的问题，但它只是一个 nota，作者是这样写的：

```
\begin{nota}
在阅读本书的过程中……
\end{nota}
```

作为对宏的总结，我们可以说，这是微软公司的著名软件——Word 中样式的推广。阅读本文档，尤其是它的第 II 部分，足以使你信服：宏可以比这些脍炙人口的样式走得更远……

对于那些对“所见即所得”模式上瘾的人，有团队发布了一个“所见即所表”（原文如此；*What you see is what you Mean (sic)*）版本的  $\text{\LaTeX}$ ，命名为 LyX。你可以访问 <http://www.lyx.org> 来了解它。

## 如何打印本书

准备打印机<sup>15</sup>，使用源码文档生成的“papier”版本文件，可以获得适用于 A5 型号纸张的打印预览。

## 你可以用本书做什么？

作者 Vincent Lozano

原书名 Tout ce que vous avez toujours voulu savoir sur  $\text{\LaTeX}$  sans jamais avoir osé le demander

日期 2013 年 11 月 22 日

版权开放 (Copyleft) 本书是开放图书，遵循开放作品许可 (Licence Art Libre, LAL):

<http://www.artlibre.org>

<sup>13</sup>即使在这里也没有写。

<sup>14</sup>译注：此例特指原版书。翻译时舍弃了原书源码中类似的自动化部分。

<sup>15</sup>啊——啊——[就像弗兰克·扎帕 (Frank Zappa) 说的那样] (译注：有人知道这是那首歌吗?)

总之，LAL 规定你可以复制本书，你同样可以在遵循以下条款的前提下传播本书：

- 注明其遵循 LAL；
- 注明原作者名 Vincent Lozano 及对本书做出修改的人名；
- 注明其源码可以通过 <http://cours.enise.fr/info/latex> 下载。

最后，你可以在满足以下条件的前提下修改本书：

- 满足以上传播协议；
- 注明你的作品是修改过的版本，以及如果可能，注明修改的内容；
- 以相同的使用协议或与本书的使用协议不冲突的协议规定下传播。

## 开始之前

就像很多非常强大的软件一样， $\text{\LaTeX}$  的使用从来就不简单。实际上，当我们在自己的方向上前进时， $\text{\LaTeX}$  经常让人感觉很舒适，我们可以借助它来避免过于纠结版式问题，就像兰波特所说的那样。如果我们想要改变行为时的解决方案仍然是选择另一条指令，那么一切都会顺利进行。然而，尽管  $\text{\LaTeX}$  给出的选择代表了优秀出版人采用的现行通用标准，我们仍然有一天会想要排出某种特殊的版式，而  $\text{\LaTeX}$  表面上做不到这一点。这种情况下，有几种解决方案供你选择。

- 包含一个可以解决你的问题的包（ $\text{\LaTeX}$  是一个开放系统，有大量或多或少标准化的包可供我们去实现多种多样、稀奇古怪的操作）。
- 找一个  $\text{\TeX}$  学家 ( $\text{\TeX}$ nician<sup>16</sup>) 来帮你排除问题。
- 如果前两个方案对你来说没有用，就不要在代码中埋头<sup>17</sup> 探寻蛛丝马迹、查找出错的指令并修改了。此刻你需要的是去了解这个系统的第一层，去了解  $\text{\TeX}$ 。这里，我们遇到了  $\text{\LaTeX}$  的一个缺点：如果说其他软件不能做到的都是很复杂的事情，那么有时让  $\text{\LaTeX}$  做一些简单的事情也是很困难的（在阅读过本书第 II 部分后，你可能会同意这一点）。

## 排版上的约定

为了让呈现效果更清晰，本书遵循了一些排版约定。文档中散布的  $\text{\LaTeX}$  代码的片段看起来是这样的格式：

%注意看

这样的`\emph{就是\LaTeX 代码了}`。

---

<sup>16</sup>也有人喜欢称作  $\text{\TeX}$ pert，但很少见。

<sup>17</sup>对于写起代码来“文思如尿崩” (pisser du code) 的人来说，这是最让人开心的解决方案了。



相关内容会使用 L<sup>A</sup>T<sub>E</sub>X 的“打字机”(machine à écrire) 字体显示。代码也会使用如下的形式展示，中间竖条上的数字偶尔会被引用<sup>18</sup>：

#### 清单 0.1

这样的就是 L<sup>A</sup>T<sub>E</sub>X 代码了。

%注意看

这样的\emph{就}是\LaTeX 代码了。

① 一些内容会以“补充说明”的形式给出，这是为了强调一个知识点。读者不需要第一时间阅读。

① 对于请读者务必阅读的内容，我们会使用这种形式来引起注意……

软件名或 L<sup>A</sup>T<sub>E</sub>X 中的包名会以本句所展示的形式展示。英文词汇会以这种形式（英： *like this*）展示。为了展示指令中通用的部分，我们会使用⟨这种形式⟩。例如：

这是\LaTeX 源码中的\emph{⟨强调文字⟩}。

偶尔出现的UNIX指令会以如下形式展示：

```
❑ grep -wi bidule /tmp/truc.dat | sort -n
```

在其中一个附录中，emacs指令会以如下形式展示：

```
❑ M-x doctor
```

最后，作为让人反感的精华，Makefile片段会以如下形式展示：

```
bidule : bidule.o truc.o
↳ gcc -o $$@ $^
```

## 致谢

本书的起草工作始于1995年，起初是作为给位于圣埃蒂安的图形信息和视觉工程实验室编写的内部指南。在此，作为这个研究团队曾经的一员，我要感谢团队成员的意见和鼓励。fr.comp.text.tex论坛的用户间接地为我提供了大量信息，这些信息丰富乐本书的内容。在此感谢这些用户。

我同样要感谢邦雅曼·巴亚尔（Benjamin Bayart）帮助我创造了一些本书用到的扩展，尤其是围绕章首“迷你”表的外框<sup>19</sup>，感谢纪尧姆·科南（Guillaume Connan）关于PDF格式的附录的建议和鼓励。

<sup>18</sup>译注：这个格式我不一定能调出来，暂时用清单编号代替了

<sup>19</sup>译注：暂时不装饰了，这个外框请看原版书。

特别感谢德尼·比图泽 (Denis Bitouzé) 的专心阅读, 他给出了一些珍贵的建议, 并且在本书第I部分给出关于数字的勘误。德尼救了我, 我搞错了关于`a4wide`和`eqnarray`的事情。这些类似的可怕错误足以把我钉在耻辱柱上。

特别感谢FramaSoft的迪迪埃·罗什 (Didier Roche) 和亚历克西·考夫曼 (Alexis Kauffmann), 他们同意我在FramaBook合集中新开一卷。特别感谢由樊尚 (Vincent; “Vim”) 带领的重读小组, 尤其是帕皮雷 (Papiray) 和安托万·布朗什 (Antoine Blanche) ——他们不仅找到了很多深藏在段落中的问题, 还纠正了一些恼人的重复问题。这里要特别感谢这些他们, 尤其是因为与他们的意见交换十分有效: *Genèse*一词中正确的变音符号、*nota*类、关于前置知识和标题的长时间讨论等, 此处不一一列举。

关于 $\text{T}_{\text{E}}\text{X}$ 和 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 的“名著”潜移默化地影响了本书的编写。本书设立“注意”栏<sup>①</sup>无疑是收到了克努特的 $\text{T}_{\text{E}}\text{X}$ book[9]的启发。古森斯 (Goossens)、米特尔巴赫 (Mittelbach) 和萨马兰 (Samarin) 的《 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 伴侣》(英: *L<sup>A</sup>T<sub>E</sub>X Companion*) 几乎是必读的图书, 它很大程度上影响了本书的内容和形式。最后, 一些在线手册也影响了我的选择 [例如《关于 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 的不简短介绍》(英: Not So Short Introduction to  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ ) 中有一章可以翻译成“需要了解的知识”] ……

在开始正式内容前, 我要指出, 虽然本书经历了几年时间, 已比较成熟, 但在风格上它绝对还有些问题。证据就是, 在我的计算机上运行如下指令:

```
grep -E -i 'on peut|permet' *.tex | wc -l
```

运行的结果是343 (比一页一次还要多), 可见我的写作风格依然贫乏<sup>20</sup>。

祝阅读愉快, 加油<sup>21</sup> !

<sup>20</sup>译注: 该指令是在统计全书中出现“on peut”和“permet”(均可以翻译成“我们可以”)的总次数。

<sup>21</sup>这行文字是逻辑排版成果的写照。无论本页剩余的空白还有多少, 它都会出现在其三等分的位置, 留下剩余的2/3。



## Part I

# 关于L<sup>A</sup>T<sub>E</sub>X的那些你想知道却从不敢问 的问题



# Chapter 1

## 基本原则

人若身患漏症，他因这漏症就不洁净了。——《圣经·利未记》15:2

本章介绍 $\text{\LaTeX}$ 的基本原理。你将会看到关于 $\text{\LaTeX}$ 安装的简介、使用 $\text{\LaTeX}$ 的基本“流程”(session)介绍、文章格式的结构、使用变音符号的注意事项，认识几个工具，以及了解面对编译错误消息时的态度。

### 1.1 安装

你想安装 $\text{\LaTeX}$ 吗？你将要安装的是 $\text{\LaTeX}$ 的其中一个发行版，具体的版本取决于你的操作系统<sup>1</sup>。发行版中带有可以自动安装和配置 $\text{\LaTeX}$ 、 $\text{\TeX}$ 和其他相关内容的程序。

**对于UNIX** 我们可以找到称为 $\text{te}\text{\TeX}$ 的发行版，虽然它的开发早在2006年就停止了。今天，我们一般安装 $\text{\TeX}$ Live (<http://www.tug.org/texlive>)。

**对于macOS** 建议安装的发行版是 $\text{Mac}\text{\TeX}$  (<http://www.tug.org/mactex>)。

**对于Windows** 最简单的方式无疑是选择 $\text{pro}\text{\TeX}$ t (<http://www.tug.org/protext>)。它会安装称为 $\text{MiK}\text{\TeX}$ 的发行版 (<http://www.miktex.org>) 和几个开发工具，其中包含一个查看PostScript文件的程序 ( $\text{gsview}$ )。

偶尔，需要在为发行版中搭配一款文字编辑器（如果其中没有包含），因为你很快就能看到，使用 $\text{\LaTeX}$ 就是在文件中输入文字和命令。

- UNIX中，推荐使用 $\text{emacs}$ 或 $\text{vi}$ ，即使前者明显比后者更高级，但二者用户之间无结果的恶性争吵仍在继续。

---

<sup>1</sup>如果你不知道操作系统是什么东西，那么你使用的是macOS；如果你不知道你的计算机用的具体是哪个操作系统，那么你在用Windows；否则，你在用UNIX……

- kile和texmaker是已集成的开发环境。依靠它们，初学的用户在入门时会觉得更轻松。它们的特点是将编辑、编译和可视化集成在一个界面。这两个环境也使通过菜单、对话框或其他标签来探索L<sup>A</sup>T<sub>E</sub>X指令称为可能（如图1.1a所示）。
- Windows中的对应产品是T<sub>E</sub>XnicCenter（如图1.1b所示）。
- macOS中的对应产品是T<sub>E</sub>Xshop和iT<sub>E</sub>Xmax。

你很快就会学到，用L<sup>A</sup>T<sub>E</sub>X制作文档是一个翻译（也称作编译）的过程——将编辑者创建的源文件转换为用于显示或印刷的格式<sup>2</sup>。因此，发行版中内置了或多或少的著名工具，可以将编译后的不同格式的文件显示出来。

**对于PDF格式** 除了著名的acrobat reader，UNIX中还有一些可以显示PDF文件，如xpdf、evince等。

**对于DVI格式** UNIX中的xdvi、kdvi和Windows中的yap都是可以显示这种L<sup>A</sup>T<sub>E</sub>X编译文件的程序。

**对于PostScript格式** ghostscript套件（在各平台下的名称可能有差异）可以显示PostScript文件。

① 需要注意，为了使你选用的发行版包含L<sup>A</sup>T<sub>E</sub>X的“法文”模式，以确保能够正确处理断字（césure；英：hyphenation），我们需要在编译文档是需要更改其“日志”（见1.6节）以使法文模式加载：

```
LaTeX2e <2005/12/01>
Babel <v3.8h> and hyphenation patterns for english, [...] dumylang, french,
loaded.
```

## 1.2 “生产”周期

即使L<sup>A</sup>T<sub>E</sub>X并不是通常意义上说的编译型语言，但我们仍然可以将制作一个L<sup>A</sup>T<sub>E</sub>X文档的周期与使用一款经典的编程语言开发软件的编辑—编译—执行周期进行类比。

### 1.2.1 编辑

一个L<sup>A</sup>T<sub>E</sub>X源文件是一个文本文件<sup>3</sup>。因此，对L<sup>A</sup>T<sub>E</sub>X文件的操作并不依赖于某个特定的软件，只需要一个经典的文本编辑器即可。因此，若要操作L<sup>A</sup>T<sub>E</sub>X文档，指令

```
| emacs <文件名>.tex &
```

或

```
| vi <文件名>.tex
```

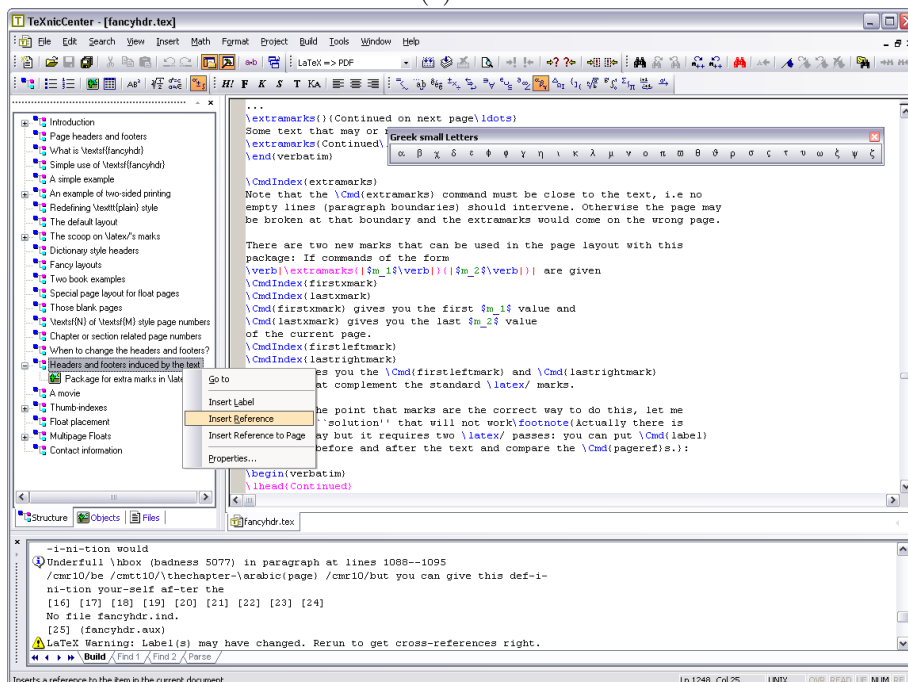
足以让你进入L<sup>A</sup>T<sub>E</sub>X文档这个充满野性和未知的世界。在Windows中，根据自己的喜好，我们可以选用一款文字编辑器。注意，对于L<sup>A</sup>T<sub>E</sub>X源文件，推荐使用.tex扩展名。

<sup>2</sup>本章会略微多介绍一些这个格式。

<sup>3</sup>即文件仅由组成其中符号的代码构成。



(a) Kile



(b) TeXnicCenter

Figure 1.1: 集成的两个开发环境：Linux中的Kile和Windows中的TeXnicCenter。它们将编辑、编译和可视化集成在一个界面中

### 1.2.2 编译

我们用如下指令开始编译：

```
❏ pdflatex <文件名>.tex
```

早晚有一天，你会看到编译会产出错误。这将是1.6节会处理的问题。总之，解决了编译问题后，我们会得到一个带有.pdf扩展名的文件，它代表便携文件格式（英： *portable document format*），这是一种由Adobe公司创造的著名格式。

① 历史上，编译 $\text{\LaTeX}$ 源文件会生成dvi文件，代表设备无关（英： *device independant*）。此类文件独不受输出环境（如屏幕、打印机等）的影响。这是一种包含了“图像”的 $\text{\LaTeX}$ 便携二进制文件，可以用于各种操作系统。随后，出现了一批用途各异的程序：

- 用于显示文档，即.dvi→点阵屏幕；
- 用于打印，即.dvi→打印机语言；
- 用于转换格式，即.dvi→PostScript文件。

图1.2表明了UNIX生成最终文件过程中参与流程的多种程序。

① 除了使用pdflatex外，也可以使用其他“编译器”来生成PDF文件。例如，xelatex和lualatex可以正确地处理以UTF-8编码的文件，是常用的替代选项。

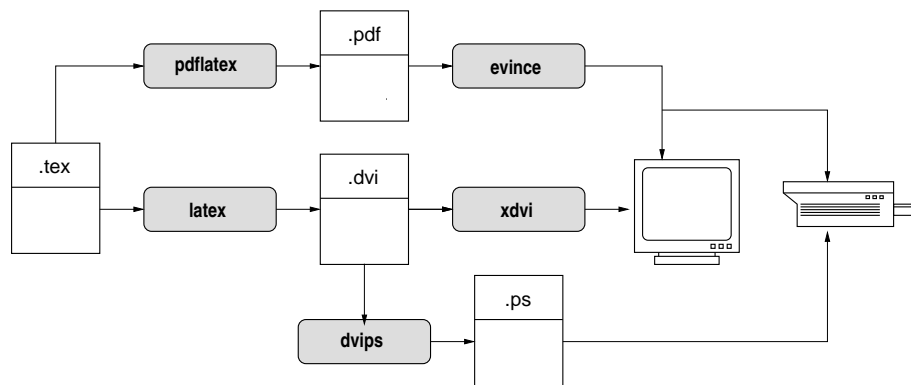


Figure 1.2: UNIX中参与生成过程的工具

### 1.2.3 显示

在编译后，可以简单地使用evince程序来完成显示步骤。输入以下指令：

```
❏ evince <文件名>.pdf &
```

这是一个linux下运行的十分直观的程序，能够给出一个方便阅读的文件预览。

❗ 注意，不必在每次编译后都重新运行evince，它显示的内容会自动刷新。

### 1.2.4 打印

对于pdf格式，如何打印它这一问题就丢给了你的操作系统。关于这一点，没有特殊的注意事项。你有了一个文件，可以自由地处置它，无论是直接打印，还是根据你所处的环境来发挥才艺。

❶ 从dvi到ps格式的转换需要调用dvips程序：

**|** `dvips <文件名>.dvi`

这可以生成一个PostScript格式的文件。这个格式也由Adobe创造，是一种打印机语言，可以看作pdf的祖先。目前的打印机出厂即可识别这种打印机语言。我们可以说，文件发送到打印机时，十有八九传送的是PostScript格式的参数。对于PostScript格式的文件，有大量可以显示、修改这种文件的工具。

## 1.3 源文件的结构

本节将介绍一种文档类型。实际上，所有L<sup>A</sup>T<sub>E</sub>X文档都具有相同的结构，形式如下：

```
\documentclass[<类选项1>,<类选项2>,...]{<类>}
\usepackage[<包选项1>,<包选项2>,...]{<包>}
...
<文前部分>
...
\begin{document}
...
<文本>
...
\end{document}
```

如此一来，所有的L<sup>A</sup>T<sub>E</sub>X文档都可以按以下方式拆解。

- 说明文档的<类>；
- 文前部分，包含以下内容：
  - 使用特定的<包>；
  - 多样的初始化和声明；
- 文档主体，即我们将要亲手输入的全部内容，出现在\begin{document}和\end{document}之间。

以下介绍各部分的细节。

### 1.3.1 文档的类

所谓类，就是提供给L<sup>A</sup>T<sub>E</sub>X的一个指示，可以帮助L<sup>A</sup>T<sub>E</sub>X决定如何为文档的特定部分排版。根据具体使用的类不同，允许使用与否的指令可能不同（如`\chapter`在book类中允许使用，在article类中不允许使用）。另一方面，根据所选择的类，给出的命令会具有特定的含义（标题、材料表……）。在入门时<sup>4</sup>，所有的L<sup>A</sup>T<sub>E</sub>X文档都必须以的指令开始——`\documentclass`接由花括号括住的类，包含以下几种：

- `article`，用于文章；
- `proc`，用于电气与电子工程师协会（英：Institute of Electrical and Electronics Engineers, IEEE）会刊（英：proceeding）风格的文章；
- `report`，用于几十页篇幅的报告；
- `book`，用于图书或论文；
- `letter`，用于信件；
- `slides`，用于演示文档。

我们当然也可以为文档定义自己的类。类的配置项用方括号括住，可以是以下内容之一：

- `11pt`，`12pt`，用于全局地更改文字字号；
- `twoside`，用于生成适合双面打印的文档；
- `draft`，用于以草稿模式生成文档。

例如，输入：

```
\documentclass{article}
```

以上命令可以将全部配置项配置为默认值（字号为10 pt，单列，单面……）。

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

以上命令将字号设置为12 pt（默认为10 pt）。再如：

```
\documentclass[twoside, draft]{report}
```

以上命令可以以草稿模式生成适合双面打印的报告。

### 1.3.2 文前部分

文前部分是指位于子句`\documentclass`和子句`\begin{document}`间的区域。在这个区域中，我们可以明确想要包含的扩展（请看下一小节）、初始化全局参数（如页边距等）、定义风格（如标题样式、序号等）、定义特殊的宏，等等。

<sup>4</sup>实际上，我们可以在`\documentclass`前添加更多神奇的“咒语”……



### 1.3.3 添加扩展

L<sup>A</sup>T<sub>E</sub>X命令`\usepackage`可以与C语言的指令`#include`类比。这一命令允许添加L<sup>A</sup>T<sub>E</sub>X中满足宏或环境形式的功能<sup>5</sup>。目前，只需记住，我们可以在一行之内包含多个包：

```
\usepackage{⟨包1⟩,⟨包2⟩,⟨包3⟩,...}
```

如果⟨包<sub>1</sub>⟩、⟨包<sub>2</sub>⟩、⟨包<sub>3</sub>⟩拥有共同的配置项⟨*opt1*⟩，我们可以输入：

```
\usepackage[⟨opt1⟩]{⟨包1⟩,⟨包2⟩,⟨包3⟩}
```

相反，如果⟨*opt1*⟩只涉及⟨包<sub>2</sub>⟩，那么我们只能像这样写成两行：

```
\usepackage{⟨包1⟩,⟨包3⟩}  
\usepackage[⟨opt1⟩]{⟨包2⟩}
```

下面是两个例子：

```
% 包graphicx带有配置项draft和xdvi  
\usepackage[xdvi, draft]{graphicx}  
% 包array和包subfig  
\usepackage{array, subfig}
```

① 根据定义，所有（类、包、命令的）的配置项参数都是**可选的**。因此我们可以这样记：L<sup>A</sup>T<sub>E</sub>X中所有由方括号括住的参数[...]都是非强制的。

## 1.4 开始！

在本节，我们将尝试从一个只含几个排版命令的文档开始，介绍L<sup>A</sup>T<sub>E</sub>X的基本原理。

### 清单 1.1

从你手中掉落的工具总是掉到最难够到的地方，或脆弱的物品上。  
这是墨菲定律（loi de Murphy）的一个体现。

```
\documentclass{article}
```

```
\begin{document}
```

从你手中掉落的工具  
总是掉到最难够到的地方，  
或脆弱的物品上。

这是\emph{墨菲}定律（loi de Murphy）的

<sup>5</sup>相关内容将在下一章讲解。

```
一个体现。
\end{document}
```

这个示例体现了 $\text{\LaTeX}$ 中的几个重要的原理，具体如下。

**空行代表跳转至下一段**  $\text{\LaTeX}$ 中的空行代表一段文字的结尾，因此在以上实例中，第一段从“从你”开始，直到“物品上。”结束。指令`\par`与空行等价，可以用来表示一段文字的起始。

**$\text{\LaTeX}$ 会忽略换行** 最终的文档中，换行并不由源文件中的换行决定。 $\text{\LaTeX}$ 会自动为各段文本打断、压缩、调节文字，除非你有特殊的要求。

**$\text{\LaTeX}$ 会忽略重复的空格** 输入1个或18784个空格是等价的，比如源码中`de`和`Murphy`前插入的空格那样。此规则也适用于跳转段落：输入一行或多行空格是等价的。

**“\”是转义字符** (*caractère d’échappement*; 英: *escape char*) “\”可以告诉 $\text{\LaTeX}$ 它后面的一系列字符是控制序列，也就是说，是最一般意义上的指令（或宏）。这里，它对“墨菲”一词生效，具体的效果由指令`\emph`控制。

**“{”和“}”** 它们是组的定界符，稍后会进一步解释它们。

### 1.4.1 几个特殊字符

就像符号“\”的出现所暗示的那样， $\text{\LaTeX}$ 中还有10个有特殊含义的符号，在此将其列出：

`\ $ & % # ^ _ { } ~`

以下是一个使用部分特殊字符的案例：

#### 清单 1.2

是下标:  $x_{i+1}$ ，还是上标:  $e^{i\pi}$ ；这是问题 1！

%毫无意义的段落

`\textbf{是}`下标:  $x_{i+1}$ ，

还是上标:  $e^{i\pi}$ ；

这是问题~1! %还是问题2?

目前，你需要知道：

- %会使得 $\text{\LaTeX}$ 忽略当前行的剩余部分，因此，它是表示注释的符号（与C中的`//`等价）；
- ~代表不可拆分的空格<sup>6</sup>，可以防止 $\text{\LaTeX}$ 在指定的位置断字。尽管有大量的情况需要插入这个符号来表示不可拆分（如所有形如“图~1”的情况），然而，对于此类符号的使用，并没有系统化的规则。

- `$`用于标记公式的开始和结束。L<sup>A</sup>T<sub>E</sub>X遇到一个`$`符号时，它会切换到► 数学模式，直到遇到下一个`$`符号。
- `_`和`^`分别代表将文本转化为下标和上标。**注意**，这两个符号只能在数学模式下使用。
- `{`和`}`分别表示组的开始和结束。本例中出现了两种组：一种出现在数学模式中，用于把将要放到下标或上标的“子公式”组合起来；另一种把将要设置成粗体的文字组合起来。

我们可以使用如下的指令来在让文档生成部分特殊字符：

```
\$ \% \& \# \{ \} \_
```

这串指令可以输出“\$ & % # { } \_”。2.2.5小节会解释如何使文档生成其余特殊字符（即`\` ~ `^`）。

### 1.4.2 调用指令

你已经知道了，要想调用指令或宏，需要输入转义字符，并紧接着输入你想使用的宏名。但是，L<sup>A</sup>T<sub>E</sub>X如何知道宏名的末尾在哪里呢？此处以用于生成T<sub>E</sub>X标识的`\TeX`为例来解释<sup>7</sup>。

#### 清单 TODO代码清单序号

```
TeXbook is for TEXhackers.
TeX has some powerful macros.
LATEX is a document preparation system
```

```
\TeX book is for \TeX hackers.
```

```
\TeX\  has some powerful macros.
```

```
\LaTeX{} is a document preparation system
```

① `\_`（其中`_`代表空格）称作控制空格（espace de contrôle）。这个空格不会被L<sup>A</sup>T<sub>E</sub>X忽略。因此，指令“`et\_\_\_hop!`”会生成“et hop !”。实际上，以`\{函数\}{参数}`的形式来调用宏是很好的习惯。因此，使用上例中的第三种方式比第二种方式更佳。这种形式可以避免空格被忽略的情况发生<sup>a</sup>。因此，我们将使用`the \tex{}book`来生成“the T<sub>E</sub>Xbook”，使用“`\LaTeX{}is a\_...`”来生成“L<sup>A</sup>T<sub>E</sub>X is a ...”。

<sup>a</sup>所以他为什么要跟我们说这些？！

<sup>6</sup>见2.10节。

<sup>7</sup>译注：此例涉及对西文行文中空格的处理，不宜翻译。

1.4.3 变音符号

法国人往往对于使用 $\text{\LaTeX}$ 这件事忧心忡忡，因为法文中带有变音符号。别怕！你不必像表1.1<sup>8</sup>中展示的那样输入带有变音符号的字符。然而，你需要知道：无论是什么种类的字符，包括大写字母，我们都可以为其添加变音符号。

变音符号	源码	效果
尖音符	<code>\`z</code>	Ẑ
钝音符	<code>\'z</code>	ẑ
长音符	<code>\^z</code>	ẑ
软音符	<code>\c{z}</code>	z
分音符	<code>\"z</code>	z̈

Table 1.1: 输入占7位 (bit) 的变音符号

注意！虽然我们可以输入带有变音符号的字符，但不要忘记：这需要我们调用编码。目前，编码可能只针对地球上的某个地区。在法国，我们使用ISO 8859编码，配合拉丁文1区拓展。这套编码允许我们操纵美丽的变音符号。在详细阅读本书中专为使用法文书写文档的情况准备的章节之前，我们建议你在你源码的文前部分添加以下指令，以“攻克”法文文档的难关：

◀第7章

```
%源文件编码
\usepackage[latin1]{inputenc}
%TeX字体编码
\usepackage[T1]{fontenc}
%针对法文文档
\usepackage[francais]{babel}
```

1.5 第一批工具

以下几个宏和合字在文档中很常用，因此应当了解它们。首先， $\text{\LaTeX}$ 会区分三种连接号：

- `-`，用于“Saint-Étienne”；
- `--`，用于“page 12-24”；
- `---`，在法文中用于插入语，如“une parenthèse — comme cela”。

引号应以如下方式输入：

- ```和`'`可以输入英文中的引号<sup>9</sup>——“English”。

<sup>8</sup>译注：该表与原书不完全相同。  
<sup>9</sup>译注：此处与原书不同。原书前文输入变音符号时亦直接使用了单弯引号```和`'`，但这两个符号作为命令参数时会导致编译错误，因此分别替换为反引号和直单引号。这可能是编译器的差异导致的。

- 如果键盘允许，对于法文，你可以输入«和»<sup>10</sup>。babel包的法文支持部分（参考第7章）允许我们通过\og和fg输入引号，由此，\og français\fg{}这类命令是允许的——« français »。

以下是其余几个使用的命令：

- \today可以生成（编译时的）日期——2013年11月22日。
- \S可以生成段落符号——§。
- \ldots可以在英文文档中生成省略号“...”。但在法文文档中，应当输入三个点“...”（第7章会介绍更多有关法文排版的内容）。

最后要记得，英文中的双成分标点（punctuation double；包括：; ! ?）前不加空格，但法文相反，它们的前面需要加空格。另外，在法国这个可爱的国家，我们几乎都靠右行驶。

## 1.6 第一组报错

① 接下来，我们会看看L<sup>A</sup>T<sub>E</sub>X编译你的文档时闹的“情绪”。当我们放出编译指令时，我们会直接在终端看到这些输出。为了能让L<sup>A</sup>T<sub>E</sub>X得到充分使用，我们鼓励你在自己的环境下找到属于你自己的检查L<sup>A</sup>T<sub>E</sub>X“日志”的方法。这些日志可以为你指明错误信息，以及编译过程中出现的其他警告。

### 1.6.1 症状

如果你与L<sup>A</sup>T<sub>E</sub>X打交道，那么早晚有一天，你会看到屏幕上显示出类似这样粗旷的信息：

```

1 This is TeX, Version 3.1415 (C version 6.1)
2 (erreur.tex
3 LaTeX2e <1995/12/01>
4 (/usr/local/lib/texmf/tex/cls/article.cls
5 Document Class: article 1995/11/30 v1.3p Standard LaTeX document class
6 (/usr/local/lib/texmf/tex/clo/size10.clo)) (erreur.aux)
7 ! Undefined control sequence.
8 1.5 paragraphe de ce \empha
9                               {document}
10 ?
```

几乎可以肯定，你看不懂这团内容。它是使用L<sup>A</sup>T<sub>E</sub>X处理文件erreur.tex后终端上显示的内容。以下是文件全文<sup>11</sup>：

<sup>10</sup>例如，在Linux系统下，分别按键盘的[Alt Gr]+[Z]和[Alt Gr]+[X]组合键（译注：适用于AZERTY键盘）。

<sup>11</sup>译注：文件正文意为：我觉得，在这样短的\empha{文档}的第一段就有一个错误。

```
\documentclass{article}

\begin{document}
Il me semble bien qu'il y ait une erreur dans le
premier paragraphe de ce \empha{document} somme
toute assez court.
\end{document}
```

### 1.6.2 诊断

我们可以以一种简单的方式来解释以上错误信息。

**第1行** 你使用的T<sub>E</sub>X版本为 $\pi \pm 10^{-4}$ 。

**第2行** 你想编译文件`erreur.tex`。

**第3行** 你使用了L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>，版本日期为1995年12月。

**第4行~第5行** 你使用了标准文档类`article`。

**第6行** 字号默认设置为10 pt。

**第7行** 错误信息本身。

**第8行~第9行** `erreur.tex`中造成错误的代码行和对应行号。

**第10行** T<sub>E</sub>X给出的短促且尤其让人焦虑的消息：？。

第8行和第9行之间的“裂缝”精准地表明了L<sup>A</sup>T<sub>E</sub>X崩不住了的地方。以下消息：

```
! Undefined control sequence.
```

向你说明了，L<sup>A</sup>T<sub>E</sub>X不认识你输入的指令。实际上，指令`\empha`不存在。

### 治疗方案

那么，当L<sup>A</sup>T<sub>E</sub>X给我们展示这个著名的“？”时，我们应该怎么回答它呢？这里有3种最简短的方式，可以用来与L<sup>A</sup>T<sub>E</sub>X轻度交流。

- 按回车键来忽略错误。
- 输入`x`来退出编译。
- 输入`r`让L<sup>A</sup>T<sub>E</sub>X继续编译，并忽略其他错误信息。

- 输入i以插入一条更正信息并继续编译，但新插入的更正信息并不会出现在源文档中。
- 输入h以获取更多关于该错误的信息。以下是本例中TeX提供给你的更多信息：

```
Undefined control sequence :  
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.
```

### 1.6.3 一些消息

TeX和LaTeX会根据不同的情况给出大量错误消息，其中有一些在第一次面对时并不能读懂。然而，大多数经常出现的消息有以下类型：

- LaTeX语法或保留字错误；
- 花括号没有成对出现；
- 在文本模式中出现了本应出现在数学模式中的内容；
- 数学模式没有关闭；
- 你忘记包含一个包；
- 编译过程无法结束；
- .....

## 1.7 再说几句！

现在，你知道了如何用LaTeX源码创建可打印的文件。同时，本章向你介绍了调用指令的原则。接下来，如果想要了解LaTeX语法中的不同功能，需要做的仅仅是去着手接下来的章节。





## Chapter 2

# 需要了解的知识

褻慢的人受刑罰，愚蒙的人就得智慧。智慧人受訓誨，便得知識。——《圣经·箴言》  
12:11

本章要研究使用 $\text{\LaTeX}$ 生成文档时的基本排版指令。我们将零散地处理用于突出显示、 $\text{\LaTeX}$ 标准环境、标题、页面下方的注释、页眉和页脚，以及浮动的环境。接下来，我们会介绍参考系统和 $\text{\LaTeX}$ 生成的辅助文件。最后，阅读到本章末尾的人将有机会读到一些关于断字的思考。

所有这些指令都将以其默认行为模式使用。也就是说，我们这里不介绍重新定义它们的方法。对应地，你将能够以传统的版式来生成文档。若要打出一篇更进阶的文章，你需要了解如何输入数学式（第3章）、一些关于科技文档的知识（第6章），以及包含图像的方法（第5章）。

## 2.1 突出显示

要了解 $\text{\LaTeX}$ 选用字体的机制，需要知道：我们通常通过4个参数来区别字体。

**族 (famille)** 指字体的整体形状。默认情况下， $\text{\LaTeX}$ 使用三种字体族：罗马体、无衬线体、打字机体。 $\text{\LaTeX}$ 中，以英文单词*family*来指代字体族。

**风格 (style)** 指字体体现出的体态（英文以*shape*指代），分为：意大利体、倾斜和小型大写字母风格（PETITES CAPITALES）<sup>1</sup>。

**字重 (graisse)** 指字体笔画的粗细（ $\text{\LaTeX}$ 中以*serie*指代）。默认情况下有两种粗细：中等和**加粗**。

**字号** 字体的大小。

---

<sup>1</sup>译注：本书翻译时，以楷体对应意大利体、以仿宋体对应倾斜排版，按原文翻译。中文字体的族和风格往往是并列的，很少有交叉叠加的情况，因此在展示叠加效果时酌情保留原文。中文中很少见到类似小型大写字母的突出显示方式。因此，除了特意展示西文的部分外，本书忽略小型大写字母格式，必要时以其他风格取代。

指令	声明	输出
<code>\textrm{...}</code>	<code>{\rmfamily ...}</code>	罗马 (roman)
<code>\textsf{...}</code>	<code>{\sffamily ...}</code>	非衬线 (sans sérif)
<code>\texttt{...}</code>	<code>{\ttfamily ...}</code>	打字机 (machine à écrire)
<code>\textup{...}</code>	<code>{\upshape ...}</code>	正 (droit)
<code>\textit{...}</code>	<code>{\itshape ...}</code>	意大利 ( <i>italique</i> )
<code>\textsl{...}</code>	<code>{\slshape ...}</code>	倾斜 ( <i>penché</i> )
<code>\textsc{...}</code>	<code>{\scshape ...}</code>	小型大写 (PETITES CAPITALES)
<code>\textmd{...}</code>	<code>{\mdseries ...}</code>	中等 (médium)
<code>\textbf{...}</code>	<code>{\bfseries ...}</code>	加粗 (gras)

Table 2.1: 更改字体的声明

2.1.1 族—风格—字重

有两种不同的宏可以设置族、风格、字重这三个变量：指令和声明（如表2.1所示）。指令以花括号的形式将其参数括住。声明可以打断行文，同时修改三个变量之一，直到新的命令出现。总体上的规则是，我们使用指令来突出显示一个词或一组词：

清单 2.1

`char`类型的变量总是被编码为8 位。

`\texttt{char}`类型的`\emph{变量}``\textsl{总是}`被编码为`\textbf{8 位}`。

注意，在上面的命令中，指令`\emph`（对应的声明是`\em`，可以以更优雅的方式突出显示一组词）。相较声明，强烈建议使用指令。当要修改文本的一部分时，使用指令是更明智的选择<sup>2</sup>：

清单 2.2

马格马 (*Magma*) 的音乐就像一面镜子，每个人都能看到他自己的倒影。

`{\em \bfseries 马格马 (Magma) \mdseries 的音乐`  
`就像一面镜子，每个人都能看到他自己的倒影。}`

接下来的例子展示了如何使用组。声明`\slshape`出现在一个组中，因此它只在组内发挥作用。此外，组会继承它外层的组的参数。这样一来，“silence”一词会使用非衬线体（根据外层的组），并且倾斜展示（根据内层的声明）：

清单 2.3

在爵士乐中，沉默 (*silence*) 永远是正确的。因此，这是一种充满万千可能的音乐。

<sup>2</sup>定义指令也是。

`\sffamily` 在爵士乐中，  
`{\slshape 沉默 (silence) \/}`永远是正确的。因此，  
这是一种充满万千可能的音乐。

2.1.2 意大利体校正

另一个推荐使用指令而不是声明的理由是，与声明不同，指令可以实现意大利体校正。所谓意大利体校正，是指在以意大利体显示的字符组后，有必要增加一个间距，使得这组字符不会“碰到”后面的词。这个间距与所涉及的字符有关<sup>3</sup>：

清单 2.4

le *chef* a toujours raison.  
le *chef* a toujours raison.  
le *chef* a toujours raison.

```
le {\em chef} a toujours raison.\par
le {\em chef\/} a toujours raison.\par
le \emph{chef} a toujours raison.\par
```

我们可以看到，指令`\emph`实现了校正，然而，若要使用声明，则需要明确地借助宏`\/`来实现相同的效果。

2.1.3 字号

表2.2中展示的宏可以修改行文的字号。这些宏都是声明。同时，对于每一个宏都存在同名的环境。

<code>\Huge</code>	宏大	<code>\normalsize</code>	正常
<code>\huge</code>	庞大	<code>\small</code>	小
<code>\LARGE</code>	特大	<code>\footnotesize</code>	加小
<code>\Large</code>	加大	<code>\scriptsize</code>	特小
<code>\large</code>	大	<code>\tiny</code>	微小

Table 2.2: 修改字号

2.1.4 几个建议

习惯上，我们应当尽量减少字体变化。实际上，如果字体变化得不合适宜或没有实际用处，尤其是喧宾夺主，影响了文档的可读性，看起来就会很低级。这里是有关使用字体变化的几条建议

<sup>3</sup>译注：此例展示不同情况下*f*和*a*间距的细微差距，宜保留原文。例句意为：首长永远是对的。

(仍然是习惯上的建议！)：

- 相较于使用其余指令，更多地使用`\emph`（默认会将字体改为意大利体）。
- 将**加粗**的机会留给特别重要的提示。
- 在法文文档中，几乎只在人名中使用小型大写字母（如Donald KNUTH）。
- 打字机字体族经常被用于生成编程语言的代码或类似的内容。

以下内容适用于能读懂其中道理的人……

除以上内容外，我们给出以下两个用于突出显示的情况，读者可以字形斟酌：改变字号、添加下划线（使用指令`\underline`）。

“也许那些希望<sup>小声说话</sup>的诗人会让图书的字体频繁变化，但目前，只有一些字体狂人（比如本手册的作者）才喜欢这样做。”——克努特，`TEXBook`[9]

“注意，出版行业认为，以添加下划线的形式来强调内容是不好的习惯。下划线只应用于一种情况——输出设备无法以其他方式突出显示内容，比如使用打字机。”——迈克尔·古森斯（Michel Goossens）等，《`LATEX`伴侣》（*L<sub>A</sub>T<sub>E</sub>X Companion*）[6]

## 2.2 环境

`LATEX`以环境格式提供了一系列工具，具体结构是一个代码块，语法如下：

```
\begin{环境名}
...
\end{环境名}
```

其中`<环境名>`需要替换为具体的环境名称。我们目前遇到的第一个环境是`document`。`\begin`和`end`间的文字会以特殊版式展示。

① 我们立刻注意到，环境中所有声明都是局部的。另外，当然可以在我们自己定义的环境中套用已经存在的环境。

### 2.2.1 居中和对齐

想要居中显示几行文字，我们使用环境`center`：

清单 2.5

……上一段的结尾。

完美居中的  
几行文字  
且前后带有间距

然后继续下一段……

……上一段的结尾。

```
\begin{center}
  完美居中的\\
  几行文字\\
  且前后带有间距
\end{center}
然后继续下一段……
```

同样，我们可以轻易地借助环境`flushright`，让一个段落右对齐排列：

#### 清单 2.6

……上一段的结尾。

两行右对齐  
的文字

然后继续下一段……

……上一段的结尾。

```
\begin{flushright}
  两行右对齐\\
  的文字
\end{flushright}
然后继续下一段……
```

注意，以上两个示例中，命令`\\`起到了换行的功能。除一些特殊场景（表格、文档的标题和作者、特意的居中或对齐处理）外，不要使用这个命令——如果想要换行，需要使用空行或命令`\par`。

一般情况下，我们使用环境`flushleft`时需要搭配命令`\\`。但我们可以使用这个环境来生成右侧不对齐的文档，将换行的麻烦工作留给 $\text{\LaTeX}$ ，就像这段文字一样。（译注：此处给出本段原文。请注意右侧的换行的不对齐处理。En général, on emploie l'environnement `flushleft` avec des commandes `\\`. Mais on peut l'utiliser pour produire un paragraphe comme celui-ci, non justifié à droite, en laissant à  $\text{\LaTeX}$  le soin d'insérer les sauts de lignes.)

❗ 绝大部分环境都会重启一行来插入内容。然而，重要的是：环境只是在插入内容的位置中  
断当前段落，而不是结束当前段落。你可以在前两个实例中看到，“然后继续下一段……”这句话前没有缩进。另外， $\text{\LaTeX}$ 贴心地在每个环境前后都留了一段空白。

我们可以注意到，前面的三个环境分别代表以下三个声明：

- `\centering`;
- `\raggedleft`;
- `\raggedright`。

例如，我们可以这样写<sup>4</sup>：

#### 清单 2.7

Emacs代表：

Emacs  
Makes  
A  
Computer  
Slow

Emacs代表：

```
{\centering Emacs\\Makes\\  
A\\Computer\\Slow\\}
```

### 2.2.2 列表

$\text{\LaTeX}$ 提供了三种呈现列表的基本环境：`itemize`、`enumerate`、`description`。如果它们都不能满足你，也可以定义属于你自己的►列表。但目前，先来看看标准的列表。

首先，`itemize`可以生成不编号的项目列表。在法文版本中，一级列表会使用连接号（—）标记；在其他版本中，会使用点（•）标记：

#### 清单 2.8

<sup>4</sup>译注：实际上，Emacs代表Editor MACroS。本例是对Emacs的调侃。

……一句话的结尾。

- 在复杂的计算中，分子的系数需要传递给分母；
- 不应当写逗号。

然后行文继续。

……一句话的结尾。

```
\begin{itemize}
\item 在复杂的计算中，
      分子的系数需要
      传递给分母；
\item 不应当写逗号。
\end{itemize}
然后行文继续。
```

环境`enumerate`遵循类似的规则，只不过项目会被编号。一个环境中可以套用另一个环境。下面的例子中，我们同时展示了 `enumerate`和`description`环境：

#### 清单 2.9

……还是一句话的结尾。

**T<sub>E</sub>X** T<sub>E</sub>Xbook；

**L<sub>A</sub>T<sub>E</sub>X** 两本重要的书：

1. 《L<sub>A</sub>T<sub>E</sub>X：一个文档准备系统》 (*L<sub>A</sub>T<sub>E</sub>X: A Document Preparation System*)。
2. 《L<sub>A</sub>T<sub>E</sub>X伴侣》。

跟之前一样，接下来段落继续……

……还是一句话的结尾。

```
\begin{description}
\item[\TeX] \TeX{}book;
\item[\LaTeX] 两本重要的书：
  \begin{enumerate}
\item 《\LaTeX{}：一个文档
        准备系统》 (\emph{\LaTeX{}:
        A Document Preparation System}) 。
\item 《\LaTeX{}伴侣》。
\end{enumerate}
\end{description}
```

```
\end{description}
```

跟之前一样，接下来段落继续……

至于环境`description`的列表，在我们习惯的文档处理中没有对应内容。不幸的是，对于 $\text{\LaTeX}$ 初学者，它最好的结果是被误用，最差的结果是被无视。

### 2.2.3 制表

环境`tabbing`可以用于打字机上使用的那种老式制表过程。我们可以使用指令`\=`来放置定位标记，并使用指令`\>`来在定位标记之间移动。此外，`\>`可以用来换行。

#### 清单 2.10

```
左倾 中间立场 右倾
      中立派
              保守派
              无观点
```

```
\begin{tabbing}
  左倾 \= 中间立场 \= 右倾 \\
  \> 中立派 \\
  \> \> 保守派 \\
  字字字字字 \= \kill
  \> 无观点
\end{tabbing}
```

我们可以从这个例子中看到两个规则：

- 可以在制表时插入一行“模板”，并且使用指令`\kill`来隐藏这一行；
- 如果已经存在定位标记，新的指令`\=`可以在逻辑上移除它们。

### 2.2.4 表格

$\text{\LaTeX}$ 中用于生成表格的环境称为`tabular`。表线的处理可能不太精细，但对于线条简单的表格，结果可以接受<sup>5</sup>：

#### 清单 2.11

嚯：	俩 五个	仨 六个
----	---------	---------

<sup>5</sup>附录B提供了一些提示，可以用来找到能够生成更复杂表格的包。



嚯：

```
\begin{tabular}{|r|c|}
\hline
  俩 & 仨 \\
  五 & 六 \\
\hline
\end{tabular}
```

通过这个例子，我们可以得到以下结论。

- 环境`tabular`会等待输入一个参数，应用指示表格的格式。每列都应该以一个定位字符表示。
  - `r`：右对齐。
  - `c`：居中。
  - `l`：左对齐。
- 字符`&`用于分隔不同列。
- 指令`\\`用于跳转至下一行。
- 布局字符串中的`|`表示插入纵向表线。
- 横向表线由指令`\hline`插入。

因此，我们可以自由地调整`\hline`和`|`的数量，以隐藏或显示表线。包`array`可以满足一些关于表格的幻想。

❗ 大多数环境都会另起一行，但`tabular`不会。`tabular`会紧接当前的文本生成表格。

此外，我们还可以使用参数来精确地指定表格竖直方向上的位置：

#### 清单 2.12



```
一个表：\begin{tabular}[b]{|c|}
  甲\\乙
\end{tabular}
，另一个表：\begin{tabular}[t]{|c|}
  丙\\丁
\end{tabular}
```

我们可以看出，参数`b`将表格“放置”在当前行上，参数`t`将表格“悬挂”在当前行下。如果没有参数，表格会在竖直方向上居中，就像前面的例子一样。

显然，表格可以不插入句子中，而可以单独成段，比如配合环境`center`被单独居中。

### 2.2.5 模拟终端

环境`verbatim`可以将其内容逐字插入文档。因此，无论什么字符，甚至是特殊字符，都可以使用它来插入，如插入一个C++代码片段：

#### 清单 2.13

```
class pixel {  
    int x, y;  
public:  
    pixel(int i=0, int j=0);};
```

```
\begin{verbatim}  
class pixel{  
    int x,y;  
public:  
    pixel(int i=0, int j=0);};  
\end{verbatim}
```

❗ 我们可以在环境`verbatim`中写任何内容，除了以下字符串——`\end{verbatim}!`

有两个指令可以像`verbatim`一样生成文本片段：`\verb`和`\verb*`。带有星号的版本可以将空格“ ”显示为“`\`”的形式。

这两个指令的参数不使用花括号括起，而夹在任何一种可以满足以下两个条件的符号之间：不能是特殊符号、不能在参数中包含：

#### 清单 2.14

使用`#include<stdlib.h>+`可以包含C语言标准库的原型。

使用`\verb+#include<stdlib.h>+`可以包含C语言标准库的原型。

❗ 在任何情况下，指令`\verb`都不能出现在其他指令的参数中，无论这个指令是什么。

### 2.2.6 引用语

环境`quote`和`quotation`可以在文本中引用内容。首先来看`quote`：

## 清单 2.15

……仍然是一句话的结尾。

万物相关。

爱因斯坦

有件事情是不确定的，那就是所有事情都是确定的。

帕斯卡

然后继续被打断的段落……

……仍然是一句话的结尾。

```
\begin{quote}
```

```
万物相关。 \hfill \textbf{爱因斯坦}
```

```
有件事情是不确定的，
```

```
那就是所有事情都是确定的。
```

```
\hfill \textbf{帕斯卡}
```

```
\end{quote}
```

然后继续被打断的段落……

指令\hfill可以插入一段在水平空间上►无限延伸的空白。环境quotation有一些细微的差别：

◀§4.2.4

## 清单 2.16

……仍然是一句话的结尾。

人有许多缺陷，但我们若能想到他们被创造的那个年代，就能对此感到宽容。

阿方斯·阿莱（Alphonse Allais）

然后继续被打断的段落……

……仍然是一句话的结尾。

```
\begin{quotation}
```

```
人有许多缺陷，但我们若能
```

```
想到他们被创造的那个年代，
```

```
就能对此感到宽容。 \par
```

```
\raggedleft 阿方斯·阿莱
```

```
(Alphonse Allais)
```

```
\end{quotation}
```

然后继续被打断的段落……

实际上，对于这两个环境，根据莱斯利·兰波特的介绍，一个（quote）是为引用一条或多条短的内容而准备的，另一个（quotation）是为引用一条长的内容而准备。

## 2.3 页边注

指令`\marginpar`可以在页边缘创建一个小段落，语法如下：

```
\marginpar{<文字>}
```

对于双面打印的模式，为了区分左侧和右侧的页面，我们可以使用如下指令<sup>6</sup>：

```
\marginpar[<左侧文字>]{<右侧文字>}
```

其中<左侧文字>和<右侧文字>会根据当前页码的奇偶性在页面左侧或右侧呈现。如此一来，以下指令：

```
\marginpar[这!]{嚯!}
```

这！

会生成你在本页页边缘看到的内容。

## 2.4 标题

表2.3给出了L<sup>A</sup>T<sub>E</sub>X种可用的章节命令。对于类型为`article`的文档，指令`\chapter`不可用；对于类型为`letter`的文档，任何标题类型都不可用。目前，你需要了解以下两点：

- 使用指令生成的每个标题都可以自动编号，并可以在必要时列入目录。
- 使用指令`\tableofcontents`可以生成目录，并将其插入该指令所在的位置。

<code>\part{...}</code>	<code>\chapter{...}</code>	
<code>\section{...}</code>	<code>\subsection{...}</code>	<code>\subsubsection{...}</code>
<code>\paragraph{...}</code>	<code>\subparagraph{...}</code>	

Table 2.3: 章节指令

① 此外，所有的标题指令都带有可以自定义的相关类型。总的来说，这些指令会自动生成标题前后的间距。如此一来，指令前后的任何空行都会被忽略。

### 清单 2.17

……一句话的结尾。

## 3.2 结论

最终，……

<sup>6</sup>译注：原书有误。此处已更正。

……一句话的结尾。

```
\section{结论}
```

最终，……

每种标题指令都带有一种“带星”形式（例如，`\section*`），可以插入不参与编号的标题。但需要注意，这种标题不会出现在► 目录中。与节（section）相关的标题指令中可以接受参数，用于明确目录中展示的内容。例如：

◀§2.9.2

```
\section[张三]{太帅了!}
```

可以在文档中插入节标题“太帅了！”，但在目录中将其展示为“张三”。

## 2.5 页面底部的注释

有一种方便的方式可以在页面底部插入注释：使用指令`\footnote{<文本>}`。注释会自动编号，且在默认情况下按章重新编号。如下是 $\text{\LaTeX}$ 可以实现的效果：

### 清单 2.18

无论如何，指令`\footnotea`可以在页面底部插入注释。

<sup>a</sup>正如其名。

无论如何，指令`\verb +footnote+ \footnote{正如其名。}`可以在页面底部插入注释。

在一些特殊情况中，`\footnote`不能生成我们想要的效果。此时，有必要分两步走：

1. 使用指令`\footnotemark`放置注释标记；
2. 当条件允许时，使用指令`\footnotetext`在页面底部输入文字。

例如，在表格中插入注释似乎有点棘手，此时可以这样写：

### 清单 2.19

甲 乙<sup>a</sup>

丙 丁

<sup>a</sup>一个注释。

```
\begin{tabular}{cc}
  甲 & 乙\footnotemark \\
  丙 & 丁
\end{tabular}
```

```
\end{tabular}\footnotetext{一个注释。}
```

## 2.6 页眉和页脚

L<sup>A</sup>T<sub>E</sub>X生成页眉和页脚的标准指令有些简陋，但这里也值得一提，因为这些指令对于一些特定情况已经足够了。

① 关于这些指令，在这里就不讲更多内容了，因为fancyhdr.dvi中介绍的fancyhdr包易用得多，也提供了比L<sup>A</sup>T<sub>E</sub>X的标准指令多得多的有趣选项。本书关于使用此包生成页眉页脚的详细解释在10.4节。

如果不调用其他包，可以在文档的文前部分使用指令\pagestyle来指定页眉页脚的风格：

```
\pagestyle{<风格>}
```

其中，参数<风格>可以从以下值中选择：

- empty，不显示页眉和页脚；
- plain，默认风格，页脚居中显示当前页码；
- headings，在页眉显示一定的内容，根据文档风格的不同而不同（例如，在双面的report文档中，会显示当前的章名或节名）；
- myheadings，自定义插入内容。

此外，使用指令\thispagestyle，可以变更或指定当前页的风格。

## 2.7 浮动环境

L<sup>A</sup>T<sub>E</sub>X为其优秀的用户提供了使用浮动环境的可能性。这种环境的特点是，其内容会被“漂浮地”渲染到正文中。也就是说，L<sup>A</sup>T<sub>E</sub>X通过一种算法，综合考虑一系列参数，在文档中为这种环境挑选位置。

① L<sup>A</sup>T<sub>E</sub>X中的环境figure和table并不仅仅可以用来插入图片和表格，这一点与它们的名字相反。实际上，这两种环境只是将其内容浮动起来，并且为添加图题或表题提供了可能性。实际上，这两种环境中可以是你喜欢的任何内容，也不一定要是图形。

### 2.7.1 图（figure）和表（table）

环境figure一般用来插入图像，而环境table一般用于插入表格。这两种环境都可以带有自己的小标题，使用语法如下：

**清单 2.20**

此段包含了浮动环境“figure”，其内容可以在页面中浮动。

三  
O丁O  
四  
Figure 1: 有个老丁头

此段包含了浮动环境“figure”，  
其内容可以在页面中浮动。

```
\begin{figure}
  \begin{center}
    三\\
    O丁O\\
    四\\
  \end{center}
  \caption{有个老丁头}
\end{figure}
```

注意到，使用指令\caption可以生成图题或表题。文本“Figure 1:”会自动生成，并给出对应该图片的编号。这种“风格”也是可以自定义的。

**2.7.2 确定位置**

我们可以在\begin后面用方括号中给出参数，来让L<sup>A</sup>T<sub>E</sub>X尝试依此放置浮动的内容，具体如下：

- h，放置在源码中出现的位置；
- t，放置在页面顶部；
- b，放置在页面底部；
- p，放置在单独的页面上。

注意，有时我们会为如何放置浮动环境而抓狂。为了不因此而烦躁，需要理解（并接受）下面这一点：L<sup>A</sup>T<sub>E</sub>X会综合考虑多个参数来安排环境figure和table的位置，其中包括：

- 放置在页面顶部和底部的浮动环境的最大数量；
- 顶部和底部放置有浮动环境的页面占全部页面的最大可接受比例；
- 浮动环境前后的空白。

如果你在放置图片时遇到了关于其位置的问题<sup>7</sup>，我们建议你遵循以下建议：

- 如果你尝试写“如下图示：”之类的话，并且希望紧接着放一个图片，不要使用环境`figure`！
- 尽量使用参考系统，例如写成“如图3所示”。
- 我们总是喜欢放一些超大图片——缩小它们！
- 如果你的表格太长，把它放到附录中去。无论如何，它会让读者很难受。
- L<sup>A</sup>T<sub>E</sub>X中的参数经过精心研究，目的是平衡文档中文字和图片的数量。所以如果你的文档是连环画，请做好最坏的准备……
- 只有到了**印刷你的最终版文档**时才有必要去纠结图片的位置。

### 2.7.3 图片列表

使用指令`\listoffigures`（对应地，`\listoftables`）可以在文档中插入全部图片（对应地，表格）的列表。列表会出现在指令所在的位置。此类指令会生成扩展名为`.lof`（对应地，`.lot`）的文件。此外，与插入节标题时使用参数指定目录中展示的内容类似，指令`\caption`中也可以带有一个参数，来控制其在列表中显示的内容。默认情况下，列表中会显示图题（对应地，表题）：

```
\caption[嚯]{你可以在这里尽情写些人生故事，因为无论你写多长，它都不会出现在图片列表中，因为我们已经指定了这个无比长的小标题显示成“嚯”……}
```

## 2.8 引用

L<sup>A</sup>T<sub>E</sub>X的引用系统以为对象编号的方式允许你以符号的方式操作文档中任何部分的序号，因此，没有必要去记忆一个图片究竟是图4还是图5。L<sup>A</sup>T<sub>E</sub>X通过这种方式为你减少了很多工作，并且这种方式可以通过几行文字就能解释清楚。

### 2.8.1 原理

为了在文档中成功使用引用，我们需要做两件事：第一，在文档中添加符号标记；第二，调用这个标记来进行引用，要么引用相应对象的序号，要么引用其出现的页码。这样简单的方式简化了工作：

1. 使用指令`\label`来添加标记：
 

```
\label{标记内容}
```

 其中(标记内容)是一个不带有特殊字符的字符串；

---

<sup>7</sup>并且你确定它真的成问题……



2. 使用指令`\ref`来引用相应对象的序号：

`\ref{<标记内容>}`

使用`\pageref`，可以引用相应页码：

`\pageref{<标记内容>}`

## 2.8.2 需要引用什么？

我们可以引用的内容如下：

- 各级标题；
- 浮动环境（figure、table……）；
- 数学式（参见第3章）；
- 带编号列表（如`enumerate`）的项；
- 等等。

如下的示例综合了三种引用指令：

### 清单 2.21

## 3.5 二次方程

二次方程即以下形式的方程：

$$ax^2 + bx + c = 0 \quad (2.12)$$

第13页3.5节中的方程2.12这么说完那么说……

```
\section{二次方程}\label{sec-2dg}
```

二次方程具有以下形式：

```
\begin{equation}
```

```
ax^2 + bx + c = 0 \label{equ}
```

```
\end{equation}
```

```
第\pageref{sec-2dg}页\ref{sec-2dg}节中的方程\ref{equ}这么说完那么说……
```

在这个示例中，我们引用了对象`\section`和另一个`\equation`。此外，我们还引用了这个关于方程的节所在页的页码。

⚠ 当在浮动环境中放置`\label`时，请一定将其放置指令`\caption`的后面。否则，引用会“指向”当前的节，而不是图形。

## 2.9 辅助文件

为了进一步理解引用的机制，我们需要检查以下 $\text{\LaTeX}$ 编译源文件时向你的硬盘中写入了什么。目前，你可以看到的文件有如下格式。

`dvi` 文档中的图片。

`log`  $\text{\LaTeX}$ 在最近一次编译时的“自言自语”。一般情况下，它多多少少代表了你编译时在终端上看到的内容。

`aux` 辅助文件，储存引用、页码、标题等信息。

`toc` 目录文件。

`lof` 图片列表文件。

### 2.9.1 与引用的交互

$\text{\LaTeX}$ 以如下的方式处理引用：在第一次编译时，在文件`<文件名>.aux`中储存引用，其中`<文件名>`是你的文档的文件名。我们可以在如下示例中看到其解决引用的问题时的机制和原理，在该示例中，我们尝试引用位于第35页的第3节中的一个名为`truc`的标记<sup>8</sup>。

1. 在第一次编译时， $\text{\LaTeX}$ 在`.aux`格式的辅助文件中存储该标记的号码（在本例中，存储其所在节的序号）和该标记出现页的页码（如图2.1所示）。

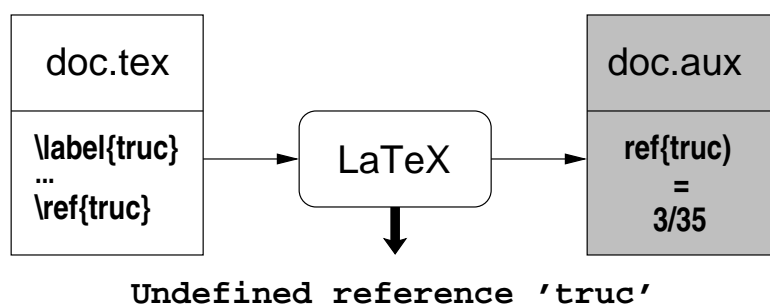


Figure 2.1: 使用`.aux`的第一次编译

因此，此次编译时，会 $\text{\LaTeX}$ 发送警告，指出标记`truc`未定义。



Figure 2.2: 使用 .aux 的第二次编译

2. 我们会进行第二次编译，这次编译会使用辅助文件的内容（如图2.2所示）。

在以下情况中的引用定义是不正确的。

1. 你插入了一个新的标记，并且在插入标记后第一次编译（引用未被定义）。此时，对于新插入的标记，会得到如下形式的消息：

```
Reference 'vlunch' on page 2 undefined on input line 17.
```

2. 你在文档中引入的变化无疑会改变页码或对象（图片、方程等）的位置，因此引用会被错误定义。在编译结束时，你会看到如下警告：

```
Label(s) may have changed.
Rerun to get cross-references right.
```

3. 你引用了一个不存在的标记。在这种情况下，再编译八百次也无济于事。

## 2.9.2 与目录的交互

对于目录，你会发现，其原理与引用是类似的。在文档中插入指令`\tableofcontents`意味着目录将会分两步创建，具体如下。

1. 第一轮遍历会收集文档中与表题相关的信息，并将其存入文件`(文件名).toc`中（如图2.3所示）。
2. 第二轮遍历会使最终的文档中包含`(文件名).toc`，因此，目录也会被包含（如图2.4所示）。

你可能会遇到这种情况：在写草稿时，文档已经包含了目录指令（`tableofcontents`），而你又添加了新的章节指令。在这种情况下，新的章节只有经过**两次**编译后才会显示在目录中。

---

<sup>8</sup>译注：“truc”意为“小东西”。

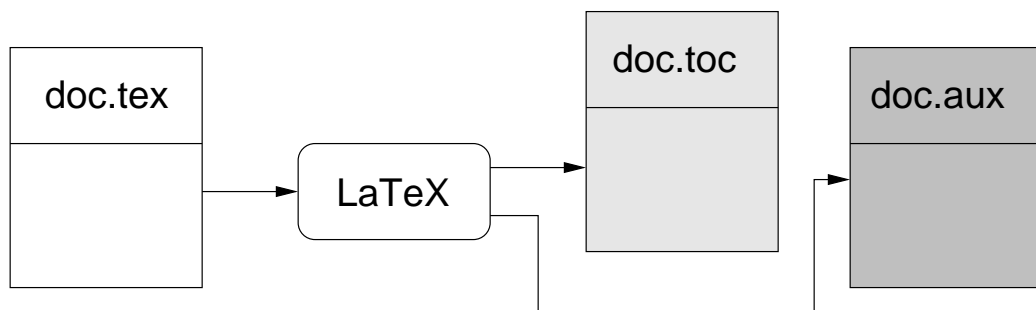


Figure 2.3: 使用 .toc 的第一次编译

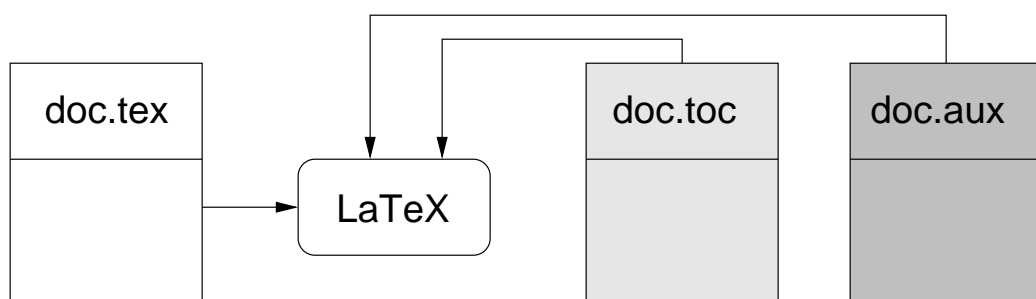


Figure 2.4: 使用 .toc 的第二次编译

### 2.9.3 一些建议

要养成为每个文档生成目录的习惯。实际上， $\text{\LaTeX}$  会围绕你的 `.tex` 文件生成多个文件<sup>9</sup>。另外，在起草文档时，不必过于关心目录是否时刻更新——它早晚会更新的！实际上，只有在印刷之前才有必要去确认所有的引用都是正确的。

最后，就像我们在不再能确定目标文件时会时不时运行一下 `make clean` 指令一样，在看起来一切都运行不正常时，删掉辅助文件并重新编译是个好习惯。

◀§B.2

## 2.10 断字的处理

$\text{\LaTeX}$  依靠  $\text{\TeX}$  对特定的语言来实现不同的断字效果。这种算法在  $\text{\TeX}$ Book 的附录 H 中有描述，也体现出  $\text{\TeX}$  最成功的一面。通过检查文档打断段落的方式，可以识别出文档是不是由  $\text{\LaTeX}$  生成的，因为其他很多软件都喜欢以在单词间添加更多空间的方式来处理此类问题。然而，也会有一些  $\text{\LaTeX}$  无法正确断字的情况。在这种情况下， $\text{\LaTeX}$  会以以下两种“吓人的”信息之一为你给出警告：

```
Underfull \hbox (badness 1810) detected at line 33
```

<sup>9</sup> 这还没有涉及参考文献、索引、术语目录等。

或

```
Overfull \hbox (14.24376pt too wide) detected at line 41
```

T<sub>E</sub>X在很底层用你的文档生成一系列字盒。每个字符都装入合适的字盒，字盒组合成词，再以类似的方式组合成行、成段，然后成页面。

这里以一种简单的方式总结和介绍原理。我们可以简单地理解为，T<sub>E</sub>X在水平模式下操纵\hbox来组合单词，在竖直模式下操纵\vbox来生成页面。此外，组合这些字盒时，T<sub>E</sub>X如果觉得结果不太美观，就会用以上两种信息警告你。信息的含义如下。

- `Underfull \hbox`表示字盒组合得有些稀疏。通过显示badness的值，T<sub>E</sub>X会告诉你它认为当前行“有多丑”。如果一行文字排列得很完美，该值为0。在最差的情况下，该值为10000。
- `Overfull \hbox`表示字盒有些太挤了。T<sub>E</sub>X可以以pt为单位显示文字越界深入边缘的长度。

如果一个页面过于稀疏，L<sup>A</sup>T<sub>E</sub>X会以\vbox代替\hbox显示类似的消息。表2.4展示了同一句话的不同疏密程度<sup>10</sup>排列效果。

效果	评价
Ô rage ! ô désespoir ! ô vieillesse ennemie !	稀疏
Ô rage ! ô désespoir ! ô vieillesse ennemie !	稀疏
Ô rage ! ô désespoir ! ô vieillesse ennemie !	稀疏
Ô rage ! ô désespoir ! ô vieillesse ennemie !	理想
Ô rage ! ô désespoir ! ô vieillesse ennemie !	过密
Ô rage ! ô désespoir ! ô vieillesse ennemie !	过密
Ô rage ! ô désespoir ! ô vieillesse ennemie !	过密

Table 2.4: 横向的不同疏密排列

① 可以在文档选项中启用draft，来使在出现Overfull \hbox问题的位置的侧栏显示一个黑色的方块，就像本段的侧栏一样。这个选项可以帮助快速定位导致问题的行。

### 2.10.1 控制断字

对于以下情况，L<sup>A</sup>T<sub>E</sub>X的断字处理可能遇到困难。

- 它不能识别需要打断的词——这是个极端情况。
- 不能打断的对象占据了需要打断的位置，例如\verb|...|或方程类型的对象。

这里提供以下几个控制断字的方法。

<sup>10</sup>译注：表中的例句出自法国古典主义戏剧大师高乃依（Pierre Corneille）的戏剧《熙德》（*Le Cid*），意为“哦，愤怒！哦，绝望！哦，宿敌！”。

① 如果以下方法都不能使你满意（如果你的句子中包含太多TeX不能打断的对象，就会产生这种情况），就只能想办法更换表达方式来规避问题了。

## 引导断字

我们可以通过在必要的位置插入指令\来指出可以断字的位置，从而帮助LaTeX实现断字。例如，如果LaTeX不能成功地打断“nonmaïçavapamieu”<sup>11</sup> 一词，我们可以输入：

```
non\ -mai\ -ça\ -va\ -pa\ -mieu
```

如果这个词频繁出现，为了避免反复像上面这样给出指示，可以在文前部分输入指令\hyphenation：

```
\hyphenation{non-mai-ça-va-pa-mieu}
```

这样就可以告诉LaTeX这个生词的断字方式。

## 强制断字

通过输入指令\linebreak[<数字>]，我们可以强制断字，但这样做可能带来灾难性的后果——如果你明白我的意思。参数<数字>可以调节指令\linebreak。你可以“腼腆”地给出指示——\linebreak[0]，或是给出不容置疑的命令——\linebreak[4]。

指令\pagebreak[<数字>]可以打断页面。另一方面，还有两个指令可以用来换页：

- \clearpage 完成当前页面，换页另起。
- \cleardoublepage 完成当前页面，并在双面模式下从奇数页另起。

这两条指令会强制LaTeX在布局过程中插入所有浮动的图像。

① 另外一种对某些情况很实用的手动介入方式是将当前页面纵向长度加长，需要调用如下指令：

```
\enlargethispage
```

指令需要给出尺寸，且其后需要插入一个空行：

```
\enlargethispage{10cm} ← 针对过短的面
```

```
[……一段过长的文字……]
```

```
\clearpage ← 明确延长10 cm的页面的结尾
```

<sup>11</sup> 译注：作者的生造词，形似句子“Non, mais ça ne va pas mieux.”，意为“不，没有变得更好”。

## 防止断字

有三种方式可以强制 $\text{\LaTeX}$ 不打断文本。

1. 通过~插入不可打断的空格。
2. 通过 $\text{\mbox{\{单词\}}}$ 将词放入一个字盒中<sup>12</sup>。
3. 对于防止换行使用指令 $\text{\nolinebreak}$ :

$\text{\nolinebreak}[(\text{数字})]$

同样, 为了防止换页, 可以使用如下指令:

$\text{\nopagebreak}[(\text{数字})]$

其中 $(\text{数字})$ 与 $\text{\linebreak}$ 或 $\text{\pagebreak}$ 中的作用一致。

## 2.11 小结

本章介绍了 $\text{\LaTeX}$ 的标准功能。你如果专心地阅读至此, 应该已经可以创建任何类型的简单文档了(目前还不能处理带有公式和图表的文档)。即使你还不能自由地定制你的文档, 你的文档的排版质量也会足够好, 不需要你提出很多形而上学的问题, 如多大的页边距才“理想”、标题和文字间留出多少空白才“合适”……实际上,  $\text{\LaTeX}$ 中的默认特性已经足够满足全世界范围内有关印刷的大部分实用性规则。

---

<sup>12</sup>这是因为 $\text{\TeX}$ 永远不会打断字盒。





## Chapter 3

# 数学排版

这十二使徒的名：头一个叫西门，又称彼得……——《圣经·马太福音》10:2

毫无疑问， $\text{\LaTeX}$ 最实用和有趣<sup>1</sup>的特性就是可以生成数学公式。它生成的公式自然、美观，并且不需要你做任何工作<sup>2</sup>。另外，如果你有使用关于某个特定的公式编辑器点来点去的糟糕记忆，现在就偷着乐吧：现在，编写公式不需要鼠标了！使用 $\text{\LaTeX}$ 生成公式是一个广大的领域，我们这里仅仅会介绍一些用于生成“常用”公式所需的基本知识。因此，本章仅仅包含操作 $\text{\LaTeX}$ 公式的简短介绍。

①  $\text{\LaTeX}$ 的标准指令足以生成大多数常见的数学方程。然而，建议使用美国数学学会（英：American Mathematical Society）发布的扩展`amsmath`和`amssymb`。可以在很多情况下，这两个扩展可以简化格式化过程。

### 3.1 编写数学公式的两种方式

$\text{\LaTeX}$ 可以识别两种数学公式。第一种是在文本中直接插入公式，就像这样： $ax + b = c$ ；另一种是将若干公式写在环境中，例如：

$$dU = \delta\mathcal{W} + \delta\mathcal{Q}$$

这两种模式都遵循一系列原则，涉及不同符号的字号和位置。如下示例使用了两种模式：

#### 清单 3.1

<sup>1</sup>没错，没错！真的有人排公式纯是为了玩！

<sup>2</sup>或者只需要你去做两三件小事情。

函数 $f(x)$ 定义如下：

$$f(x) = \sqrt{\frac{x-1}{x+1}}$$

若其导函数存在，求其导函数。

函数 $f(x)$ 定义如下：

```
\begin{displaymath}
  f(x)=\sqrt{\frac{x-1}{x+1}}
\end{displaymath}
```

若其导函数存在，求其导函数。

这个示例告诉我们，我们可以使用 $\$$ 符号来进入“内部”数学模式，并再次使用 $\$$ 符号退出。此外，这里使用了环境`displaymath`，这是最简单的生成数学式的方法。使用`\[`和`\]`也可以达成后者的效果（参见3.7.1节。）

3.7节会介绍 $\text{\LaTeX}$ 的不同环境。

## 3.2 常用指令

### 3.2.1 上标和下标

正如1.4.1小节提到的，指令`_`和`^`分别可以生成下标和上标。若需要这两条指令处理多个字符，需要将这些参数“打包”到一组花括号中。

$x_2$	$x_2$	$x_{2y}$	$x_{2y}$	$x_{t_0}$	$x_{t_0}$
$x^2$	$x^2$	$x^{2y}$	$x^{2y}$	$x^{t_0}$	$x^{t_0}$
		$x^{2y}_{t_0}$	$x^{2y}_{t_0}$	$x^{t_1}_{t_0}$	$x^{t_1}_{t_0}$

### 3.2.2 分式和根式

生成分式和根式的指令如下：

- 指令`\frac{⟨分子⟩}{⟨分母⟩}`可以生成分式，`⟨分子⟩`会排在分数线上方，`⟨分母⟩`会排在分数线下方；
- 指令`\sqrt[⟨n⟩]{⟨arg⟩}`可以生成分式，表示变量`⟨arg⟩`的`⟨n⟩`次方根。

注意，这两种指令在字间模式和方程模式下生成的效果不同。对于分式 $\frac{1}{\sin x + 1}$ 和根式 $\sqrt{3x^2 - 1}$ ，他们在方程模式下的显示效果为：

$$\frac{1}{\sin x + 1} \quad \sqrt{3x^2 - 1}$$

作为介绍这两条指令的结尾，我们来看看它们是如何套用的：

## 清单 3.2

$$\sqrt{\frac{1 + \sqrt[3]{3x+1}}{3x + \frac{1-x}{1+x}}}$$

```
\begin{displaymath}
\sqrt{\frac{1+\sqrt[3]{3x+1}}
{3x+\frac{1-x}{1+x}}}
\end{displaymath}
```

## 3.2.3 符号

## 常用符号

表3.1展示了部分生成你可能需要的符号的宏。

<code>\pm</code>	±	<code>\otimes</code>	⊗	<code>\cong</code>	≅	<code>\imath</code>	ι
<code>\mp</code>	∓	<code>\oslash</code>	⊘	<code>\subset</code>	⊂	<code>\jmath</code>	ℵ
<code>\div</code>	÷	<code>\odot</code>	⊙	<code>\supset</code>	⊃	<code>\ell</code>	ℓ
<code>\ast</code>	*	<code>\leq</code>	≤	<code>\subseteq</code>	⊆	<code>\aleph</code>	ℵ
<code>\times</code>	×	<code>\geq</code>	≥	<code>\supseteq</code>	⊇	<code>\nabla</code>	∇
<code>\bullet</code>	•	<code>\equiv</code>	≡	<code>\in</code>	∈	<code>\lvert</code>	∥
<code>\circ</code>	°	<code>\ll</code>	≪	<code>\ni</code>	∋	<code>\partial</code>	∂
<code>\star</code>	*	<code>\gg</code>	≫	<code>\emptyset</code>	∅	<code>\wedge</code>	∧
<code>\setminus</code>	\	<code>\sim</code>	~	<code>\forall</code>	∀	<code>\vee</code>	∨
<code>\oplus</code>	⊕	<code>\simeq</code>	≈	<code>\infty</code>	∞	<code>\cup</code>	∪
<code>\ominus</code>	⊖	<code>\approx</code>	≈	<code>\exists</code>	∃	<code>\cap</code>	∩

Table 3.1: 常用数学符号

① 我们盘点了`latexsym`和`amssymb`包中的近450个符号（参见附录C）。目的不是介绍它们！表3.1是标准符号中的一部分，我们认为它们可能是最常用的一批——除了完全偶然出现的ℵ<sup>a</sup>。也许这证明了作者的数学水平不太高。

<sup>a</sup>译注：aleph，希伯来文字母表的第一个字母。

## 省略号

为了节省篇幅，数学式中经常使用省略号。省略号有三种，指令`\dots`可以生成点“放置在”基线上的省略号：

**清单 3.3**

$C = \{\vec{c}_0, \vec{c}_1, \dots, \vec{c}_N\}$  为  $N$  个颜色的集合。

`$C=\{\vec{c}_0,\vec{c}_1,\dots,`  
`\vec{c}_N\}`  
 为  $N$  个颜色的集合。

指令 `\cdots` 生成的省略号圆点上下居中，就像等号一样：

**清单 3.4**

$\vec{\mu} = \frac{1}{N}(\vec{c}_0 + \vec{c}_1 + \dots + \vec{c}_N)$  为  $N$  个颜色的平均值。

`$\vec{\mu}=\frac{1}{N}`  
`(\vec{c}_0+\vec{c}_1+\cdots+\vec{c}_N)`  
 为  $N$  个颜色的平均值。

最后，指令 `\vdots` 和 `\ddots` 主要在矩阵中使用（参见3.6节、例3.15）。这两个指令分别可以生成  $\vdots$  和  $\ddots$  这两种省略号。

**箭头**

用于生成箭头的指令可以使用以下简单的方法来记忆：

- 所有指令均以 `arrow` 结尾；
- 必须带有前缀 `left` 或 `right`，表示方向；
- 可以带有前缀 `long`，表示加长；
- 指令的第一个字母可以改为大写，表示箭头使用双线；
- 可以连写 `left` 和 `right`，表示双向箭头。

综上：

<code>\rightarrow</code>	表示	$\rightarrow$	<code>\Longrightarrow</code>	表示	$\Longrightarrow$
<code>\Leftarrow</code>	表示	$\Leftarrow$	<code>\Longleftarrow</code>	表示	$\Longleftarrow$
			<code>\Longleftrightarrow</code>	表示	$\Longleftrightarrow$

**希腊字母**

可以以一种极简单的方式使用希腊字母：打出它们的名字。也就是说，`\alpha` 表示  $\alpha$ ，`\pi` 表示  $\pi$ 。将指令的第一个字母改为大写，表示将对应希腊字母改为大写：`\Gamma` 表示  $\Gamma$ 。注意，不是所有大写希腊字母都有对应的指令，如果要将  $\alpha$  改为大写，直接使用字母 A 即可（指令 `\Alpha` 不存在）。

## 实数集

科技文档的作者常常会面临一个“至关重要”的问题：“我们应当如何打出代表实数集的字母‘ $\mathbb{R}$ ’？”关于这个问题，这里分享几个观点。从历史上看，似乎最初的数学资料上将实数符号排版为加粗的形式（“令  $x \in \mathbf{R}$ ”），老师们会使用粉笔反复在字母“ $\mathbf{R}$ ”上描几遍，来代表这个符号。这种比较烦琐的方法促成了我们“更熟悉”的写法：“令  $x \in \mathbb{R}$ ”。因此，出现了不同的流派： $\mathbf{R}$ 、 $\mathbb{R}$ ，等等。如果你想自己选择，那么有如下包和指令供你选择：

- bbm提供的指令`\mathbbm{R}`可以生成 $\mathbb{R}$ 、指令`\mathbbbmss{R}`可以生成 $\mathbb{R}$ ，等等；
- bbold提供的指令`\mathbbbm{R}`可以生成 $\mathbb{R}$ ；
- amssymb提供的指令`\mathbb{R}`可以生成 $\mathbb{R}$ 、指令`\mathbf{R}`可以生成 $\mathbf{R}$ 。

## 3.3 函数

### 3.3.1 标准函数

要生成经典的数学函数（如对数函数、三角函数等），需要使用 $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 预装的函数来实现，这里是一个示例：

#### 清单 3.5

$$\sin^2 x + \cos^2 x = 1$$

```
$\sin^2x + \cos^2 x=1$
```

如果不使用 $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 函数：

#### 清单 3.6

$$\sin^2 x + \cos^2 x = 1$$

```
$\sin^2x + \cos^2x=1$
```

二者的区别在于， $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 会将字符串`cos`视为一系列变量（因此生成意大利体），而将函数`\cos`生成成为罗马体的“`cos`”。另一个区别是对可能存在的下标的处理（参见以下`\max`的示例）。以下函数都是 $\mathrm{L}^{\mathrm{A}}\mathrm{T}_{\mathrm{E}}\mathrm{X}$ 的标准数学函数。

- 各种三角函数：`\sin`、`\cos`、`\tan`。在前面加`arc`，可以得到对应的反函数。在后面加`h`，可以得到双曲三角函数。
- 自然对数和常用对数<sup>3</sup>分别使用函数`\ln`和`\log`。
- 函数`\sup`、`\inf`、`\max`、`\min`、`\arg`可以用于如下形式的数学式中：

<sup>3</sup>译注：指以10为底的对数，标准的写法为 $\lg$ 。本书以原书习惯为准，约定使用 $\log$ 。

**清单 3.7**

$$T = \arg \max_{t < 0} f(t)$$

```
\begin{displaymath}
  T=\arg \max_{t<0} f(t)
\end{displaymath}
```

注意搭配`\max`使用下标操作符`_`的结果。

**3.3.2 积分、求和和其他极限**

L<sup>A</sup>T<sub>E</sub>X使用一套简单的语法来生成积分、求和等内容，具体如下：

`\langle操作\rangle_{\langle下界\rangle}^{\langle上界\rangle}`

其中`\langle操作\rangle`可以是`sum`、`prod`、`int`、`lim`之一，`\langle上界\rangle`和`\langle下界\rangle`会排列在操作符号的周围。例如：

**清单 3.8**

对此等比数列求和：

$$\sum_{i=0}^n q^i = \frac{1 - q^{n+1}}{1 - q}$$

对此等比数列求和：

```
\begin{displaymath}
  \sum_{i=0}^n q^i =
  \frac{1 - q^{n+1}}{1 - q}
\end{displaymath}
```

类似地，使用指令`\prod`可以生成求积符号 $\prod$ 。以下是使用积分的示例：

**清单 3.9**

对于 $x > 0$ ，定义自然对数如下：

$$\ln(x) = \int_1^x \frac{1}{t} dt$$

对于 $x > 0$ ，定义自然对数如下：

```
\begin{displaymath}
  \ln(x) = \int_1^x \frac{1}{t}
  \, \mathrm{dt}
\end{displaymath}
```

```
\end{displaymath}
```

指令`\,`可以在“dt”前插入很小的空格（参见3.5.1小节）。你如果更喜欢线积分，可以使用`\oint`，这个指令可以生成符号 $\oint$ 。好了，这里会给出一个关于极限的示例，相信你可以看懂：

### 清单 3.10

$f(x)$ 在 $x_0$ 处存在极限 $\ell$ ：

$$\lim_{x \rightarrow x_0} f(x) = \ell$$

$f(x)$ 在 $x_0$ 处存在极限 $\ell$ ：

```
\begin{displaymath}
\lim_{x \rightarrow x_0} f(x) = \ell
\end{displaymath}
```

希望你能注意到示例中漂亮的 $\ell$ 。为了巩固一下关于两种数学模式的知识，这里给出相同的数学式，但它们这次会镶嵌在行文中：求和， $\sum_{i=0}^n q^i = \frac{1-q^{n+1}}{1-q}$ ；求积分， $\ln(x) = \int_1^x \frac{1}{t} dt$ ；求极限， $\lim_{x \rightarrow x_0} f(x) = \ell$ 。

## 3.4 重叠的符号

### 3.4.1 操作符`\not`

操作符`\not`可以生成特定关系的“否定”样式：

### 清单 3.11

令实数 $x \notin I$ ……

令实数 $x \not\in I$ ……

`\not`的输出结果就是在其下一个符号上加上一道“斜杠”。注意，这个操作符并不能总是呈现出完美的效果，例如`\not\longrightarrow`会显示为 $\not\rightarrow$ 。但对于宽度合适的符号，它给出的结果还能令人满意。

### 3.4.2 “变音符号”

对于特殊的数学概念，经常需要<sup>4</sup>在符号上加“变音”符号。以下是可用的符号示例：

<code>\hat{x}</code>	$\hat{x}$	<code>\check{x}</code>	$\check{x}$	<code>\breve{x}</code>	$\breve{x}$
<code>\acute{x}</code>	$\acute{x}$	<code>\grave{x}</code>	$\grave{x}$	<code>\tilde{x}</code>	$\tilde{x}$
<code>\bar{x}</code>	$\bar{x}$	<code>\dot{x}</code>	$\dot{x}$	<code>\ddot{x}</code>	$\ddot{x}$

<sup>4</sup>实际上，一些名副其实的大数学家很喜欢这种符号上面的小帽子。一些人甚至还喜欢在上面叠两层、三层……

### 3.4.3 向量

有两种<sup>5</sup>方式可以得到向量：

- 对于小些的符号，可以使用`\vec`，因为这个指令是用于添加“变音”符号的。
- 对于其他情况，可以使用`\overrightarrow`。

#### 清单 3.12

设 $\overrightarrow{AB}$ 在基底  $(\vec{i}, \vec{j})$  下定义。

设 $\overrightarrow{AB}$ 在基底  
 $(\vec{i}, \vec{j})$ 下定义。

注意， $\vec{AB}$ 会显示为 $\overrightarrow{AB}$ （关于`\!`的用途，参见3.5.1小节）。此外，指令`\imath`和`\jmath`分别可以生成不带点的字母 $i$ 和 $j$ ： $i$ 、 $j$ 。

### 3.4.4 指令`\stackrel`

指令`\stackrel`可以将两个符号叠放在一起：

`\stackrel{<符号_1>}{<符号_2>}`

`<符号_1>`会置于`<符号_2>`上方。例如：

`x\stackrel{f}{\longmapsto}y`

以上代码会生成 $x \xrightarrow{f} y$ 。

## 3.5 两个重要原则

为了掌握 $\text{\LaTeX}$ 生成数学式的方法，需要知道以下两个原则。

**空格**  $\text{\LaTeX}$ 会忽略数学式中夹带的空格，因此`$x+1$`和`$x + 1$`生成的结果是相同的。 $\text{\LaTeX}$ 会在它认为最合适的地方添加空格。

**文本** 任何的符号组都会被当作同一系列变量或函数对待，因此`$x=t$ avec  $t>0$` <sup>6</sup> 会生成“ $x = t$  avec  $t > 0$ ”，而不是你所期待的“ $x = t$  avec  $t > 0$ ”<sup>7</sup>。

在了解了两个原则后，来看看入门如何处理相关的问题。

<sup>5</sup>由埃迪·索德雷（Eddie Soudrais）开发的包`esvect`可以为向量生成更好看的箭头。

<sup>6</sup>译注：此问题几乎只在西文排版中出现，因此保留原文。`avec`可理解为“且其中”。

<sup>7</sup>数学式中夹带文本的问题只会在使用`displaymath`系列的环境时显现出来。毕竟，使用“`$x=t$ avec  $t>0$` ”总是可以的！



### 3.5.1 数学模式的空格

首先需要知道， $\text{\LaTeX}$ 选择添加空格的方式一般是正确的。然而，如果有一天你非要去吹毛求疵，表3.2可以帮助你数学式中插入空格。在该表格中，我们在两个 $\square$ 符号之间夹入不同的空格指令，来展示它们的效果。

$\backslash!$ $\square\square$	无指令 $\square\square$	$\backslash,$ $\square\square$	$\backslash:$ $\square\square$
$\backslash;$ $\square\square$	$\backslash_$ $\square\square$	$\backslashquad$ $\square\square$	$\backslashqqquad$ $\square\square$

Table 3.2: 数学模式中的空格

对于那些关注“毛”“疵”的人，要强调一下，本书在等比数列的示例（参见3.3.2小节）中偷偷地在分子上添加了一些空格，以让分式中的两个 $q$ 稍微对齐。如果按照默认的生成方式，结果会是这样的：

$$\sum_{i=0}^n q^i = \frac{1 - q^{n+1}}{1 - q}$$

不知道这个关于 $q$ 的故事是否为你带来了更敏锐的观察力。

### 3.5.2 数学模式中的文本

在数学式中插入文本，最简单的方法是将文本“装箱”，并适当地插入空格：

#### 清单 3.13

设数列 $(u_n), (v_n)$ :

$$u_n = \ln n \quad \text{且} \quad v_n = \left(1 + \frac{1}{n}\right)^n$$

设数列 $(u_n), (v_n)$ :

```
\begin{displaymath}
  u_n = \ln n \quad
  \mbox{\text{\textbf{且}}}\quad v_n = (1 + \frac{1}{n})^n
  \label{ex-maths-suite}
\end{displaymath}
```

你可以在4.4.1小节找到关于指令 $\text{\backslashmbox}$ 的细节。如果你已经在考虑使用包 $\text{\amsmath}$ ，相比于使用 $\text{\mbox}$ ，也可以考虑使用指令 $\text{\text}$ 。

## 3.6 阵列 (array): 简单且高效

阵列环境`array`可以满足生成大多数数学式的需求。正如其名, 它可以将对象排列成一行行、一列列的样子。实际上, 它和环境`tabular`对应。也正如`tabular`一样, `array`也不会换行。

### 3.6.1 阵列的原理

关于阵列环境的语法, 可以回忆一下`tabular`, 有:

```
\begin{array}[(垂直位置)]{<格式>} ... \end{array}
```

其中, `<格式>`指明各列的对齐方式: `c`表示居中, `l`表示左对齐, `r`表示右对齐。可选的参数`(垂直位置)`可以明确整个表格的垂直位置。与表格中相同, 我们使用以下指令:

- 使用`&`分隔不同列;
- 使用`\\`换行。

#### 清单 3.14

设  $A = \begin{array}{cc} -1 & 1 \\ 3 & 4 \end{array}$  为数字阵列……

```
设$A=\begin{array}{rc}
-1&1\\
3&4
\end{array}$为数字阵列……
```

以下示例使用了省略号:

#### 清单 3.15

$$A = \begin{bmatrix} a_{00} & \dots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{n0} & \dots & a_{nn} \end{bmatrix}$$

```
\begin{displaymath} A=\left[\begin{array}{ccc}
a_{00} & \dots & a_{0n}\\
\vdots & \ddots & \vdots\\
a_{n0} & \dots & a_{nn}
\end{array}\right]\end{displaymath}
```

### 3.6.2 阵列和定界符号

我们经常需要array来生成矩阵，这需要借助定界符号的辅助。定界符号是一类特殊的括号，可能是方括号、花括号等，可以将数学对象包裹其中。其语法如下：

`\left<定界1> <对象> \right<定界2>`

其中，<定界<sub>1</sub>>和<定界<sub>2</sub>>为定界符号，<对象>为其包裹的数学对象。

较常用的定界符号如下：

( 和 )	(\Pi)	[ 和 ]	[\Pi]
\{ 和 \}	\{\Pi\}	\lfloor 和 \rfloor	[\Pi]
\lceil 和 \rceil	[\Pi]	\langle 和 \rangle	\langle \Pi \rangle
	\Pi	\	\ \Pi\

使用定界符号的好处是，这种符号可以自动适应它包裹的对象的尺寸：

#### 清单 3.16

设  $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  为单位矩阵。

设  $I =$

```
\left[\begin{array}{cc}
1&0\\0&1
\end{array}\right]
```

为单位矩阵。

我们同样可以使用定界符号重写示例3.13，来改变括号的尺寸：

#### 清单 3.17

设数列  $(u_n), (v_n)$ ：

$$u_n = \ln n \quad \text{et} \quad v_n = \left(1 + \frac{1}{n}\right)^n$$

设数列  $(u_n), (v_n)$ ：

```
\begin{displaymath}
u_n=\ln n\quad\text{et}\quad
\quad v_n=\left(1+\frac{1}{n}\right)^n
\end{displaymath}
```

① 对于每一个指令`\left`，都应该有一个指令`\right`与其对应。然而，左侧和右侧分别使用的符号不必是配套的。

以下示例中使用了`\right.`，表示我们不需要右侧的符号：

清单 3.18

$$\text{设 } S_i = \begin{cases} -1 & \text{若 } i \text{ 为偶数,} \\ 1 & \text{否则。} \end{cases}$$

```

设$ S_i=\left\{\begin{array}{rl}
-1 & \& \mbox{若$i$为偶数, } \\
1 & \& \mbox{s否则。}
\end{array}\right.$

```

### 3.6.3 说话的方式简单点……

包`amsmath`中提供了两种环境——`pmatrix`（`p`代表`parenthèse`，即圆括号）和`bmatrix`（`b`代表英文的`braket`，即方括号），可以简单地插入矩阵：

清单 3.19

$$\bar{\bar{\sigma}} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$$

```

\begin{displaymath}
\bar{\bar{\sigma}}=\begin{bmatrix}
\sigma_{11} & \sigma_{12} \\
\sigma_{21} & \sigma_{22}
\end{bmatrix}
\end{displaymath}

```

## 3.7 方程和环境

本节会介绍 $\text{\LaTeX}$ 中可以生成数学式的三种标准环境。

### 3.7.1 环境`displaymath`

你如果已经阅读到这里，应该已经明白，`displaymath`可以打断当前段落，并居中显示一行公式。`\begin{displaymath}...\end{displaymath}`的一种简略的写法是：`\[...\]`。例如：

## 清单 3.20

色度距离:

$$\Delta E = \sqrt{\Delta L^2 + \Delta a^2 + \Delta b^2}$$

色度距离: \[

```
\Delta E=\sqrt{
\Delta L^{*2}+ \Delta a^{*2}
+\Delta b^{*2}} \]
```

## 3.7.2 方程环境equation

方程环境equation的作用与displaymath相同，但可以为数学式编号：

## 清单

请牢记，若 $a > 0$ 且 $b > 0$ ，有

$$\ln(ab) = \ln(a) + \ln(b) \quad (3.1)$$

请牢记，若 $a > 0$ 且 $b > 0$ ，有

```
\begin{equation}
\ln(ab)=\ln(a)+\ln(b)
\end{equation}
```

① 文档类中的选项`leqno`可以将方程的编号放在左侧。方程环境的选项`fleqn`可以将方程居左，而非居中显示。

## 3.7.3 多行数学式

① 在本书的某个旧版本中，作为介绍标准环境的结尾，我们曾引入环境`eqnarray`。这个环境可以生成多行数学式，但**要知道这是错误的**。关于这个话题，有一些资料（如[12]或[14]）会向你解释如何生成“正确”的文档。要坚定不移地相信，使用`eqnarray`（以及很多其他工具）**是在造孽**。无论如何，如果你没有禁住诱惑而向`eqnarray`让步，那么终有一天，审判会顺着搜索引擎降临，任何忏悔都将无济于事，任何赦免都将无法拯救你。勿谓言之不预也。

我们在此介绍包`amsmath`中的环境`align`：

- 使用`\\`换行；
- 每行数学式都会编号，除非指令`\nonumber`出现在该行中；
- 使用两个操作符<sup>8</sup>来对齐。

<sup>8</sup>因为共有三列。（译注：原文如此。）

清单 3.22

$$\begin{aligned}(a+b)^2 &= (a+b)(a+b) \\ &= a^2 + b^2 + 2ab\end{aligned}\tag{3.2}$$

```
\begin{align}
(a+b)^2 &= (a+b)(a+b)\nonumber\\
&= a^2+b^2+2ab
\end{align}
```

① 带有星号的环境`align*`可以使所有行都不编号。要想引用环境`align`中的多行，需要插入同样数量的`\label`，分别对应相应行。

若要为占用多行的方程编号，可以使用环境`split`（同样由`amsmath`提供）：

清单 3.23

$$\begin{aligned}(a+b)^2 &= (a+b)(a+b) \\ &= a^2 + b^2 + 2ab\end{aligned}\tag{3.3}$$

```
\begin{equation}
\begin{split}
(a+b)^2 &= (a+b)(a+b)\\
&= a^2+b^2+2ab
\end{split}
\end{equation}
```

## 3.8 数学模式的风格

### 3.8.1 字体

L<sup>A</sup>T<sub>E</sub>X支持多种用于在数学模式中切换字体的指令<sup>9</sup>。默认情况下，所有符号或字符序列（除了特定函数）在最终文档中都会以意大利体呈现。在一些情况下，强制切换字体风格会很有用。实现方法具体如下：

<sup>9</sup>译注：本节疑似原书作者的包和译稿使用的`xeCJK`都有冲突，仅针对使用法文书写的环境。此小节暂时按原文誊写。

设 $\mathit{A \in \Phi}$	设 $A \in \Phi$
设 $\mathrm{A \in \Phi}$	设 $A \in \Phi$
设 $\mathbf{A \in \Phi}$	设 $\mathbf{A} \in \Phi$
设 $\mathsf{A \in \Phi}$	设 $A \in \Phi$
设 $\mathtt{A \in \Phi}$	设 $A \in \Phi$
设 $\mathcal{A \in \Phi}$	设 $\mathcal{A} \in \Phi$

❗ 指令`\mathcal`只接受大写拉丁字母作为变量，否则结果会展示为乱码。例如：

`\mathcal{abcd\Gamma}`

上述指令的运行结果为 $\mathcal{abcd\Gamma}$ 。

### 3.8.2 符号的字号

L<sup>A</sup>T<sub>E</sub>X会区分四种数学式写作风格。L<sup>A</sup>T<sub>E</sub>X生成数学式时，会根据当前的“处境”选择模式。

**文本** 适用于行文间插入的数学式。

**方程** 适用于方程格式下的数学式。

**角标** 适用于角标。

**子角标** 适用于角标的角标。

每种模式都可以明确地使用以下声明激活：

- 使用`\textstyle`切换文本模式；
- 使用`\displaystyle`切换方程模式；
- 使用`\scriptstyle`切换角标模式；
- 使用`\scriptscriptstyle`切换子角标模式；

以下示例阐明了如何在方程模式中强制使用文本模式，以及相反的操作：

#### 清单 3.24

两种形式的积： $\prod_1^n f_i$  和  $\prod_1^n f_i$  相反操作：

$$\prod_1^n f_i \text{ 和 } \prod_1^n f_i$$

两种形式的积：`\prod_{1}^n f_i`

```
和 $\prod_{i=1}^n f_i$ 
相反操作:
 $\prod_{i=1}^n f_i$ 
 $\prod_{i=1}^n f_i$ 
```

### 3.8.3 创建新操作符

想象在一个场景中，你需要创建一个特殊的操作符，称作“burps”。只需要通过如下形式生成：

#### 清单 3.25

```

$$x = \text{burps}_i f(i)$$

\newcommand{\burps}{
\mathop{\text{trm}{burps}}}
$x=\text{burps}_i f(i)$
```

再看一个例子。为了让“反正弦函数”（默认显示为arcsin）依法国的习惯显示，可以按如下示例操作：

#### 清单 3.26

```

$$\theta = \arcsin x \quad \theta = \text{Arcsin } x$$


$$\theta = \arcsin x$$

\renewcommand{\arcsin}{%
\mathop{\text{trm}{Arcsin}}\nolimits}

$$\theta = \arcsin x$$

```

指令`\nolimits`可以使相关操作符不再将参数显示为上标或下标的形式，正如操作符`\lim`、`\int`等所做的那样<sup>10</sup>。此外，前文的两个示例使用了指令`\newcommand`和`\renewcommand`，相关问题请参见4.5节。

最后，还有一种方式可以达到类似的目的。如果你已经加载了`amsmath`包，并在文前部分进行了以下声明：

```
\DeclareMathOperator*{\vlunch}{vlunch} \DeclareMathOperator{\zirgl}{Zirgl}
```

那么可以实现以下操作：

#### 清单 3.27

<sup>10</sup>译注：此句疑似说反了。



$$x = \text{vlunch}_i f(\theta)$$

其中 $\theta = \text{Zirgl } y$ 。

```
\[x=\text{vlunch}_i f(\theta)\]
```

其中 $\theta = \text{zirgl } y$ 。

### 3.9 小结

本章介绍了用于生成数学式的基本方程。对于大多数科学文档，这些指令已经足够使用了。如果你不得不起草一份充斥这复杂数学式的文档，仅靠 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的宏可能不能满足需求。因此，著名的美国数学协会（英： *American Mathematical Society*）孵化了称为 $\mathcal{A}\mathcal{M}\text{S}^{\text{E}}\text{X}$ 的包（通过`\usepackage{amsmath}`使用），可以生成尤其“奇形怪状”的数学式。



## Chapter 4

# 成为小魔仙

羔羊揭开第七印的时候，天上寂静约有二刻……——《圣经·启示录》8:1

在继续探索 $\text{\LaTeX}$ 这个庞大且出色的系统前，有必要停下脚步来了解一些重要的概念。实际上，为了成为能在组成这个系统的大量文件中大展身手的“赫尔克里·波洛”<sup>1</sup>，掌握这些概念是基本的要求。在本章，我们介绍有关计数器、长度、空白和字盒的内容。如果你在使用 $\text{\LaTeX}$ 时，除了顺从地使用它提供给你的格式外，还想要其他版式，这4个概念会很有用。

① 本章涉及的概念有些微妙的细节，比较难以把握<sup>a</sup>。我们建议你实际上手体验一下，因为这里介绍的工具一方面可以为你生成最令人满意的结果，另一方面也能让你掉最多的头发，基本上跟直接在你头上薅差不多。

<sup>a</sup>作者本人也不能保证掌握了全部内容……

### 4.1 计数器

文档中所有带有编号的对象都由计数器生成。计数器可以递增、递减，也可以重设为0，等等。我们也可以根据自己的需要来创建计数器。

#### 4.1.1 可用的计数器

计数器主要与标题、页码、浮动环境（环境`figure`和`table`）、方程（环境`equation`）、页面底部的注释、编号列表（环境`enumerate`）息息相关。

表4.1列出了 $\text{\LaTeX}$ 中基础的计数器的名称。可以注意到，它们基本上都由相关对象的名称组成。计数器`enumi`~`enumiv`分别与环境`enumerate`中的第1层~第4层相关。`mpfootnote`是环境`minipage`中脚注的计数器，会在4.4.3小节提到。

<sup>1</sup>译注：Hercule Poirot，阿加莎·克里斯蒂所著系列侦探小说中的主角。

part	paragraph	figure	enumi
chapter	subparagraph	table	enumii
section	page	footnote	enumiii
subsection	equation	mpfootnote	enumiv
subsubsection			

Table 4.1: L<sup>A</sup>T<sub>E</sub>X的计数器

### 4.1.2 操作

在接下来的段落中，我们介绍几个操作计数器的基本工具。注意，计数器是全局变量，因此以下描述的三个指令也会在全局范围内生效。同时，也需要注意，其中的变量都是整数。

#### 创建计数器

可以通过以下指令创建计数器：

```
\newcounter{<计数>}[<父计数>]
```

该指令可以创建一个新的计数器，名为<计数>。参数<父计数>是非强制的，如果配置，则每次<父计数>递增时，<计数>都会归零。

#### 指定计数

可以通过以下方法为计数器指定一个值：

```
\setcounter{<计数>}{<值>}
```

其中，<计数>代表我们想要指定值的计数器，<值>即具体指定的值。

#### 增值

可以通过以下指令使计数器的值增加或减少：

```
\addtocounter{<计数>}{<值>}
```

其中，若<值>为正数（对应地，负数），则可以使计数器的值增加（对应地，减少）。为了展现该指令的效果，我们在本书的文档中添加以下一行指令：

```
\addtocounter{footnote}{357}
```

可以看到页面下方脚注的编号变化<sup>359</sup>了。为了接下来的脚注恢复正确的序号，我们在源码中插入以下指令：

```
\addtocounter{footnote}{-357}
```

---

<sup>359</sup> 虽然这个值变了这么多属实有些荒谬……

可以看到，现在的脚注序号恢复正常<sup>3</sup>了。

### 4.1.3 显示

使用如下指令可以将计数显示出来：

`\the<计数器名>`

实际上，所有可以显示计数器的指令或环境都调用了类似的指令。这样一来，我们有如下指令：

- `\thepage`，在此处可以生成“71”，会在每次换页时调用；
- `\thefootnote`，在此处可以生成“3”，会被`\footnote`调用；
- `\thesubsection`，在此处可以生成“4.1.3”，会被指令`\subsection`调用；
- .....

“`\the`”家族的指令通常借助以下格式指令定义：

- `\arabic{<计数器>}`；
- `\roman{<计数器>}`和`\Roman{<计数器>}`；
- `\alph{<计数器>}`和`\Alph{<计数器>}`。

以下是几个示例：

- `\arabic{page}`在此处可以生成“71”；
- `\alph{footnote}`在此处可以生成“c”，`\Alph{section}`在此处可以生成“A”；
- `\Roman{subsection}`在此处可以生成“III”，`\roman{page}`在此处可以生成“lxxi”；
- .....

为了自定义文档，重定义“`\the`”家族的指令的做法很常见。例如，在本书的文档中，指令`\thefigure`以如下形式重定义：

```
\arabic{chapter}.\arabic{figure}
```

这样一来，本书的图题会依次以如下形式编号：

1. 阿拉伯数字形式的章号；
2. 圆点；

---

<sup>3</sup>天灵灵，地灵灵！

### 3. 阿拉伯数字形式的图号。

可以以如下的形式重新定义图题编号的显示形式。例如，以如下的形式定义：

```
(\Roman{chapter}):\arabic{section}.\arabic{figure}
```

这样可以在图题中以另一种形式——也是一种相对不整洁的形式——来为图编号。这里，我们重定义了指令`\thefigure`，来生成新的编号方式：先在括号中以罗马数字形式显示章号，再以圆点分隔以阿拉伯数字形式显示的节号和图号。“FIG.”等字样、图号后的连接号等则需要在指令`\caption`的层级去定义。



FIGURE (IV):1.1: 特殊编号的图题

## 4.2 长度

如果说计数器需要依托文档中对象的序号而存在，那么长度就定义了实体占的空间。长度是 $\text{\LaTeX}$ 中的一种参数，旨在以某种方式来表述对象的尺寸。

### 4.2.1 单位

所有的尺寸都需要带有单位。刚性<sup>4</sup>的尺寸具有如下形式：

`<数字>(单位)`

其中，`<数字>`可以是正值或负值，在需要时也可以带小数；`<单位>`需要是 $\text{\LaTeX}$ 可以识别的单位。可以使用包括但不限于如下列出的单位：

**cm** 厘米；

**mm** 毫米；

**in** 英寸（约2.54cm）<sup>5</sup>；

**pt** 点，通常在排版领域使用，合 $\frac{1}{72.27}$ 英寸；






**em** 当前字体下字母“M”的宽度；

**ex** 当前字体下字母“x”的高度。

注意，单位`em`（对应地，`ex`）一般会在水平（对应地，竖直）方向上衡量尺寸时使用，且使我们得以根据当前字体字号来决定尺寸。

<sup>4</sup>我们稍后会看到弹性的尺寸。

<sup>5</sup>译注：目前，1英寸严格定义为2.54 cm，原文中的“约”不严谨。

1cm :   
 1in :   
 3mm :   
 2em :   
 10pt : 

### 4.2.2 L<sup>A</sup>T<sub>E</sub>X中的几个长度

在L<sup>A</sup>T<sub>E</sub>X和各个扩展中都存在一些预定义的长度。这些长度一般决定了文档中某个部分的尺寸，例如：

- `\parindent`为段首缩进，默认为15pt；
- `\textwidth`和`\textheight`分别定义文本的宽度和高度；
- `\baselineskip`定义了相邻两行间的基线距离（本书中为10pt<sup>6</sup>）；
- `\parskip`为段间距，初始化为0pt plus 1pt<sup>7</sup>；
- .....

需要理解，可以使用这些“内置”长度的函数来表示另一个长度，例如：

`0.5\textwidth`

这表示页面文本宽度的一半。再如：

`3\parindent`

这表示段间距的三倍。注意，对于长度`-1\baselineskip`，可以直接将其写成`-\baselineskip`。

### 4.2.3 操作长度

就像计数器，长度也可以使用一些指令来操作。

#### 创建长度

以下指令可以创建一个长度：

`\newlength{<尺寸>}`

其中<尺寸>是该长度的名称。长度会被初始化为0pt（参见示例4.1）。

① 注意，不论指令`\newlength`出现在哪里，它的定义都是**全局的**。此外，一个长度被定义两次会引发报错。相反，对长度的修改是局部的，只会在它出现的组（`{...}`）内生效。

<sup>6</sup>译注：本项及下一项的具体尺寸都指原书。

<sup>7</sup>关于plus的含义，参见弹性尺寸部分。

## 指定长度

可以使用以下指令来指定长度的值：

```
\setlength{⟨尺寸⟩}{⟨值⟩}
```

该指令可以将⟨值⟩赋给⟨尺寸⟩。

## 增量

可以使用以下方式来增加长度：

```
\alltlength{⟨尺寸⟩}{⟨值⟩}
```

该指令可以使⟨尺寸⟩的值增加⟨值⟩。

在你读得正爽的时候，我们使用如下指令，冒昧地将长度`\parindent`增加了30 pt：

```
\addtolength{\parindent}{30pt} 在你读得正爽的时候，我们……
```

这样可以向你展示为长度增量的效果。在这之后，我们又写下了如下指令：

```
\addtolength{\parindent}{-30pt}
```

这样，接下来的缩进就又回到之前的样子了。

## 获取对象尺寸

### ◀§2.10

▶ 之前有模糊地提到过，在 $\text{\TeX}$ 层面上，组成文档的不同对象会被打包成字盒。排列这些字盒时，实际上是将它们各自的参考点码放整齐。这些码放整齐的点组成了一条虚拟的线，也就是该行文字的基线。所有的字盒都可以从以下三个维度衡量。

- 宽度。
- 高度：从基线到字盒顶部的距离。
- 深度：从基线到字盒底部的距离。

例如，单词“Ingénierie”中，字盒的组合方式如下：



其中，符号“。”代表参考点。可以看到，除了带有字母“g”得字盒外，其他字盒的深度均为0。

但现在，我们还是先不接着说关于字盒的内容了！

借助以下指令，可以从对象（字母、单词、字盒等）中提取尺寸特征：

```
\settowidth{⟨尺寸⟩}{⟨对象⟩}
```

```
\settoheight{⟨尺寸⟩}{⟨对象⟩}
```

```
\settodepth{⟨尺寸⟩}{⟨对象⟩}
```

这三条指令分别将对象⟨对象⟩的宽度、高度、深度指派给⟨尺寸⟩。例如：



## 清单 4.1

- 鸡零狗碎
- 一地毛

```
\newlength{\malongueur}
\settowidth{\malongueur}{鸡零狗碎}
\begin{itemize}
\item 鸡零狗碎
\item \hspace{\malongueur}一地毛
\end{itemize}
```

长度`\malongueur`代表了文字“鸡零狗碎”的宽度，被用作空格（参见关于►空格的段落）。

◀§4.3

#### 4.2.4 弹性长度

到目前为止，我们介绍的尺寸都属于刚性尺寸<sup>8</sup>。然而，还有一种弹性（*élastiques*或*ressort*）长度。在 $\text{T}_\text{E}\text{X}$ 的层面，大量的尺寸都是以如下形式定义的：

⟨值⟩ plus ⟨增值⟩ plus ⟨减值⟩

这种语法可以定义一个尺寸⟨值⟩，但根据所处环境，它可以膨胀或缩小。也就是说，如果我们将创建的尺寸称为⟨尺寸⟩，则有

$$\langle \text{值} \rangle - \langle \text{减值} \rangle \leq \langle \text{尺寸} \rangle \leq \langle \text{值} \rangle + \langle \text{增值} \rangle$$

例如，长度`\parskip`用于分隔相邻的段落，它被定为：

0pt plus 1pt

这意味着，如果页面比较稀疏， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 就会在段落之间增加1 pt垂直方向的空白。若要细微地调整水平或竖直方向上的距离，就很适合使用这种尺寸。此外， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的用户会有个奇特的机会，来处理另一类同样有趣的弹性长度。这类长度由以下两个特点：

- 长度为零；
- 可以在特定力的作用下无限伸长。

只要指明弹性的强弱， $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ 的指令就可以生成相应的弹性长度：

`\stretch{⟨数字⟩}`

其中⟨数字⟩代表弹力。这个数字可以带有符号，也可以是小数。以下是一个示例：

---

<sup>8</sup>除了`\parskip`。

## 清单 4.2

○	三分点	—
---	-----	---

```

○\hspace{\stretch{1}}%
三分点\hspace{\stretch{2}}—

```

这段 $\text{\LaTeX}$ 代码在“○”和“三分点”、“三分点”和“—”之间依据弹性长度生成了空间<sup>9</sup>，其中第二个空间的“僵硬”程度是第一个空间的两倍，对应地，空白占据了两倍的空间。此外需要注意，这种弹力是相对但无限的，因此，“○”和“—”被“推”向了页面边缘。此外补充一点：`\fill`是`\stretch{1}`的简便写法。

## 4.2.5 显示长度

有时，显示长度的值十分有用。为了达成这个目的，可以借助指令`\showthe`。该指令可以打断编译过程，并显示作为参数传入的长度的值，即：

```
\showthe\linewidth
```

◀§1.6

该指令可以显示长度`\linewidth`的值，同时打断▶ 编译。我们可以在终端看到类似这样的信息：

```

[ ... ]           ← 开头的一些套话
Document Class:  book 2001/04/21 v1.4 Standard LaTeX document
(/usr/share/texmf/tex/latex/base/bk12.clo)) (./test.aux)
> 17.62482pt.      ← 长度的值
1.10 \showthe\parindent ← 要显示的长度
?

```

若编译是使用指令终端启动的，按<回车>键就可以回到编译过程。

① 正如1.6节所说，你使用的开发环境很可能不会显示来自 $\text{\LaTeX}$ 的消息。这种情况下，找到这些消息的任务就交给你了……

## 4.3 空间

我们将在文档不同位置插入的空白区域统称为空间。有很多指令可用于按预定义或由用户选择的长度插入空白。当然，此处的“长度”是指 $\text{\LaTeX}$ 中的长度。

<sup>9</sup>指令`\hspace`生成了水平方向的空间，空间的长度取决于传入的参数。

### 4.3.1 基本长度

以下形式的指令可以在对象间插入一段空间（une espace）<sup>10</sup>：

`\<方向>space{<尺寸>}`

其中，<尺寸>可以是刚性或弹性的长度，<方向>按以下规则取值：

- 对于竖直方向，为v；
- 对于水平方向，为h。

有：

#### 清单 4.3

空      出1cm

再跳过两行。

`空\hspace{1cm}出\texttt{1cm}`

`\vspace{2\baselineskip}`

再跳过两行。

①  $\text{T}_{\text{E}}\text{X}$ 会在某些情况下删除空间。因此，有必要使用“带星”版本的指令：`\hspace*`和`\vspace*`。会造成问题的情况如下：

- 页首和页末；
- 非段首或段末的行首和行末。

### 4.3.2 预定义的空间

此处给出数个空间指令，并依照水平或竖直模式分为两类。

#### 水平空间

刚性空间有如下示例：

<sup>10</sup>我们使用了单词*espace*的阴性形式，这可以指代印刷厂中用以分隔词或字母的小金属条。今天，该词作为阴性词依然在排版或印刷领域使用。

`\enspace` :  合0.5\quad

`\quad` :  合1em

`\qqquad` :  合2\quad

弹性空间有如下示例：

`\hfill` : 即`\hspace{\fill}`

`\hrulefill` : 与`\hfill`相似，但会画一条线

`\dotfill` : 与`\hfill`相似，但会画一条圆点线

以下几个示例使用了水平方向的空间。首先，需要注意，指令`\hspace`两侧的空格不会被忽略：

#### 清单 4.4



`\hspace{1cm}—\par`

`\hspace{1cm}—\par`

`\hspace{1cm} —\par`

接下来，展示L<sup>A</sup>T<sub>E</sub>X的弹性空间：

#### 清单 4.5



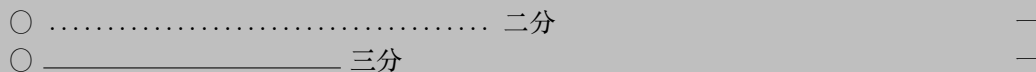
`\hfill{} —\par`

`\hrulefill{} —\par`

`\dotfill{} —\par`

最后，展示以下相对弹力：

#### 清单 4.6



`\dotfill{} 二分 \hfill{} —\par`

`\hrulefill{} 三分`

`\hspace{\stretch{2}} —\par`

现在，你应该已经明白了L<sup>A</sup>T<sub>E</sub>X预定义的“弹力”（即`\hfill`、`\hrulefill`、`\dotfill`）的“僵硬程度”为1。

## 竖直空间

以下是垂直空间类中有代表性的三条指令：

- `\smallskip`，用于垂直方向的小空间；
- `\medskip`，用于垂直方向的中等空间；
- `\bigskip`，用于垂直方向的大空间。

这些空间的用法就像指令`\vspace`一样。效果如下。

默认：	小：	中等：	大：
§下一段……	§下一段……	§下一段……	§下一段……

存在一个预定义的弹性空间：`\vfill`。它等价于

`\par\vspace{\fill}`

也就是说，它会切换自然段，并且插入尺寸为`\fill`的竖直空间：

### 清单



`\hrulefill{}`

高

`\vfill`

`\hfill{}`悬在中间`\hfill{}`

`\vspace{\stretch{2}}`

```
\hfill{}低
```

```
\hrulefill{}低
```

① 为了避免产生意料以外的结果，重要的是在将指令`\vspace`使用在**两段之间**。因此，建议养成在`\vspace`之前或之后切换段落，具体做法是插入一个空行或使用指令`\par`。

## 4.4 字盒

本章的最后一个主题是字盒。通过这个主题，你将明白本章的标题不是空口无凭的。正如我们之前大致了解过的，字盒是包含其他元素的实体（字盒也可以包含另一个字盒）。这种特殊的实体可以根据用户的幻想<sup>11</sup>来放置。

字盒有两种基本类型（在 $\text{T}_\text{E}\text{X}$ 的层面上，分类会更精巧一些），它们都有自己的特殊行为。我们可以以这样的称呼来区分它们：

- 简单字盒；
- 段落字盒。

我们将会看到一些精妙的操作，它们可以结合透明的概念，精益求精地实现一些实用的版式。

第一个示例实用了一些简单的字盒。我能肯定，它曾经让你的眼珠子瞪出来过—— $\text{T}_\text{E}\text{X}$ 的标识 $\text{T}_\text{E}\text{X}$ 。实际上，其中有三个字母：T、E、X。它们被“装盒”排列，并且在水平和竖直方向上错开：



注意，“E”的字盒向下错开，三个字盒彼此重叠。再举一个例子：



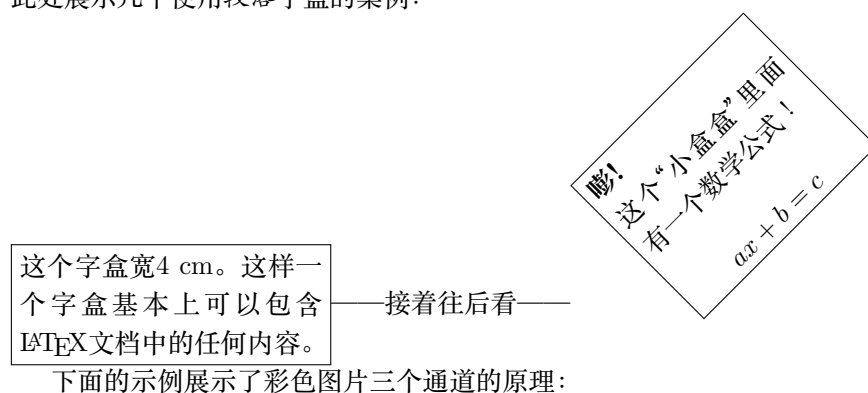
同时，为了避免争吵<sup>12</sup>：



<sup>11</sup>嗯……还有一点耐心和幽默感……

<sup>12</sup>译注：Citroën即雪铁龙，Renault即雷诺，二者都是法国的汽车品牌。此处是在用单词拼成两个品牌的车标，但雷诺的车标中间没有一横。

这里，每个单词都放在字盒中。字盒互相参考，实现一系列偏移和旋转。作为初级示例的收官，此处展示几个使用段落字盒的案例：



下面的示例展示了彩色图片三个通道的原理：



#### 4.4.1 简单字盒

第一类字盒是简单字盒，它可以像普通的单词一样出现在段落中。简单字盒有如下特点：

- 可以指定长度；
- 其高度取决于其中的内容；
- 不能包含段落跳转。

##### 不带边框

使用指令`\makebox`可以创建简单字盒：

`\makebox[⟨宽⟩][⟨位置⟩]{⟨内容⟩}`

其中，⟨宽⟩指定字盒宽度，⟨位置⟩指定字盒中⟨内容⟩的对齐方式（居中为c，左对齐为l，右对齐为r）。如下所示：

##### 清单 4.8

啪      嘹！      一个字盒  
 啪              嘹嘹！又一个字盒。

```

\makebox[2cm][c]{嚓!}一个字盒\par
\makebox[3cm][r]{嚓嚓!}又一个字盒。

```

参数<宽>和<位置>可以省略。如果省略，则字盒的宽度就是其中文本的宽度。此时可以使用：

```
\mbox{<文本>}
```

而不是`\makebox[] [] {<文本>}`。同时需要注意，在指令`\makebox`中使用选项`s`可以使内容伸展排列，使得指定的尺寸被完全填满：

#### 清单 4.9

好 累 啊！

```
\makebox[4cm][s]{好累啊!}
```

### 带边框

使用指令`\framebox`可以创建带边框的字盒。该指令与`\makebox`的语法相同：

```
\framebox[<宽>][<位置>]{<内容>}
```

与不带边框的字盒一样，带边框的字盒同样存在语法糖`\fbox{<文本>}`。例如：

#### 清单 4.10

好了 那个 嗒哒！  
然后 对对对 说得对  
还有 那啥

```

好了\framebox[1.5cm][c]{那个}嗒哒! \par
然后\framebox[2.8cm][r]{对对对}说得对
\par 还有\fbox{那啥}

```

可以调整两个长度，来修改`\framebox`的效果：

- `\fboxsep`，可以修改边框与文本的距离；
- `\fboxrule`，可以修改线条宽度。

#### 清单 4.11

红 绿 蓝  
青 品 黄



```
\setlength{\fboxrule}{5pt}
红 \framebox[2cm]{绿} 蓝\par
\setlength{\fboxrule}{0.4pt}
\setlength{\fboxsep}{8pt}
青 \framebox[2cm]{品} 黄
```

① 相较本章后文要提到的段落字盒，简单字盒有一个特点：不断字。这样一来，对于指令：  
`=== \framebox[3cm]{这句话再长也不会被打断} ===`  
 其会生成：  
`==这句话再长也不会被打断==`  
 我们可以进一步利用这个特性，来让文字重叠（参见4.2.2小节中宽度为0的字盒）

#### 4.4.2 简单字盒的操作

只要培养出一点习惯，我们就可以将字盒朝任何方向移动。

##### 竖直方向平移

平移可以通过如下指令实现。

```
\raisebox{⟨平移⟩}[⟨深⟩][⟨高⟩]{⟨文本⟩}
```

其中，⟨平移⟩是期望施加到⟨文本⟩的平移量。例如：

##### 清单 4.12

纽约  
 飘了，纽约在美国。

```
\raisebox{8pt}{纽约}飘了，
纽约在\raisebox{-1ex}{美国。}
```

两个参数⟨深⟩和⟨高⟩可以“骗过” $\text{\LaTeX}$ ，使其认为生成的字盒高度为⟨高⟩、深度为⟨深⟩。以下示例中，指令`\raisebox`选择性地使用了一些参数。

第1行：XXXXX\\

第2行：

```
XX\raisebox{0.8\baselineskip}{0}XX\\
```

第3行：XXXXX\\

第4行：XXXXX\\

第5行：

```
XX\raisebox{0.8\baselineskip}[1ex][2ex]{0}XX\\
```

第6行：XXXXX\\

代码运行结果为：

```
第1行: XXXXX
      O
第2行: XX  XX
第3行: XXXXX
第4行: XXXXX
      O
第5行: XX  XX
      |
第6行: XXXXX
```

我们将第2行中间的“O”提高了一些，边框显示出被提高的字盒的位置<sup>13</sup>。在第5行中间的“O”同样被提高，但这次我们指定了字盒（以边框展示）的尺寸。因此，L<sup>A</sup>T<sub>E</sub>X认为字盒的高度是1ex，而深度是2ex，生成的结果中，看上去O和字盒不在一行上。

### 水平方向平移

#### ◀§4.3

严格地说，水平方向的平移不是字盒的特性，因为通过插入合适的▶ 空间即可实现。如下所示：

#### 清单 4.13

Internet这个词不要 /打断// 。

```
Internet这个词不要\makebox[1.5cm]{打断}
\hspace{-1.5cm}\makebox[1.5cm]{////////}。
```

注意，这不是“划掉”单词的最佳方法，但它展示了如何在水平方向上移动字盒——借助长度为负值的\hspace。

### 宽度为0的简单字盒

宽度为0的字盒有时很有用。例如，对于需要叠加元素的情况，通过指令\makebox第一个可选的变量指定尺寸为0：

#### 清单 4.14

戴  
前 戴 后  
前 高  
乐  
前 乐 后

<sup>13</sup> 此处的边框出于帮助理解机制的目的而特别展示。

```
\newcommand{\grogra}{\huge\bfseries}
前前\makebox[0cm][c]{\grogra 戴}后后

前前\makebox[0cm][l]{\grogra 高}后后

前前\makebox[0cm][r]{\grogra 乐}后后
```

我们成功叠加了元素，但对齐方式似乎与我们想象的不太一样。实际上，变量`l`将内容放在了插入字盒位置的右侧，而变量`r`则会将其放在左侧。

## 旋转

有若干 $\text{\LaTeX}$ 扩展可以旋转文本中的元素。此处选择介绍指令`\rotatebox`，它属于第5章中会提到的扩展`graphicx`。语法如下：

```
\rotatebox{⟨角度⟩}{⟨文本⟩}
```

其中，⟨角度⟩指沿顺时针方向旋转的角度，⟨文本⟩即需要旋转的内容：

### 清单 4.15

当心转弯。

当心`\rotatebox{30}{转弯}`。

① 本书使用的`xdvi`版本不能成功显示带有旋转的对象<sup>a</sup>。这个缺陷（以及一些其他问题）在未来的版本中可能修复。目前的解决办法是使用`ghostview`或`gv`实现输出`PostScript`，或输出`Pdf`（参阅附录A）。

<sup>a</sup>对象可以显示，但不会旋转。

### 4.4.3 段落字盒

段落字盒的特点是其内容可以换行和换段（而简单字盒的内容不能）。有两种方法可以创建段落字盒，第一种方法是使用指令`\parbox`：

```
\parbox [⟨盒位置⟩][⟨高⟩][⟨文位置⟩]{⟨宽度⟩}{⟨内容⟩}
```

其中，⟨内容⟩是想要放入字盒的内容，⟨宽度⟩是要创建的字盒的宽度。选项⟨底位置⟩是可选的，可以精确地确定参考点。这个参数可以类比环境`tabular`中的参数。例如：

```
这是--- \parbox{2cm}{一个段落\\字盒}
---另一个--- \parbox[t]{2cm}{段落\\字盒}
---再来--- \parbox[b]{2cm}{一个段落\\字盒}
```

◀§2.2.4

该段指令的运行结果（为展示字盒，显示了边框）是：

这是 一个段落  
字盒 —另— 一个段落  
字盒 —再来— 一个段落  
字盒

为了在创建段落字盒时指定高度，我们可以选用参数`<高>`。`<文位置>`可以控制字盒中文字的垂直位置，默认与`<盒位置>`相同，但在需要时也可以由我们指定。一般地，它可以取`c`以表示居中、取`t`或`b`以分别表示靠上和靠下。此外，它还可以取`s`，表示将文字拉伸（英：stretch）填满整个字盒——此时，为文字安排位置的工作就落到了用户的头上。例如：

```
---\parbox[b][2cm]{2cm}{高\par 中\par 低}
\parbox[b][2cm][t]{2cm}{高\par 中\par 低}
\parbox[b][2cm][c]{2cm}{高\par 中\par 低}
\parbox[b][2cm][s]{2cm}{高\par
\vspace{\stretch{2}} 中\par\vfill 低}---
```

为清晰起见，配合`\fbox`使用该段指令。运行结果如下：

高 中 低	高 中 低	高 中 低	高 中 低
-------------	-------------	-------------	-------------

可以使用环境`minipage`创建段落字盒。该环境可以模拟创建一个页面，并且带有所需的元素，如页脚注释、表格、列表，等等<sup>14</sup>。`minipage`的语法和指令`\parbox`类似，只不过使用环境的形式：

```
\begin{minipage}[<盒位置>][<高度>][<文位置>]{<宽度>}
... \marg{文本} ...
\end{minipage}
```

如下所示：

由于三个因素 $x$ 、 $y$ 、 $z$ ，我们可能会想利用`minipage`来说些人生大道理，正如这个例子所示→

我将要说的话<sup>a</sup>不会很坦率：

- 也不会很有趣；
- 尤其也没什么必要。

……但我还是说了。

我本来还想说点别的，但忘了是什么了……

<sup>a</sup>这话还挺有分量的

在本例中，我们创建了一个`minipage`，它的宽度约是文本宽度的一半（55%），且其中包含了环境`itemize`和脚注`\footnote`。这样一来，创建的字盒会与“由于三个因素……”居中对齐，因为此处没有指定相关的`<位置>`参数：

```
\parbox{0.40\textwidth}{
```

<sup>14</sup>然而，该环境不能带有浮动元素。

```

……正如这个例子所示 $\longrightarrow$  $\hfill$ 
\begin{minipage}{0.55\textwidth}
  我将要说的话\footnote{这话还挺有分量的}
  不会很坦率：
  \begin{itemize}
    \item 也不会很有趣；
    \item 尤其也没什么必要。
  \end{itemize}
  ……但我还是说了。

  我本来还想说点别的，但忘了是什么了……
\end{minipage}

```

① 在段落字盒中，长度`\parindent`会被设置为0。正因如此，前面的示例中“我本来还想说点别的……”一句的段首没有缩进。此外，与使用`\parbox`的情况相反，当我们在`minipage`中参考长度`\textwidth`时，调用的是字盒的宽度，而非文本的宽度。

#### 4.4.4 小技巧

所有涉及字盒的函数都可以将以下尺寸作为参数。

- `width`: 内容文本的宽度。
- `\height`: 内容文本的高度。
- `\depth`: 内容文本的深度。
- `\totalheight`: 文本的高度和深度之和。

因此，可以借由所含文本精确指定字盒的尺寸。在一些情况下，这种方式很有用：

**清单 4.16**



这个`\framebox[.7\width]{字盒}`太挤了。

这个`\framebox[1.8\width]{字盒}`有些宽。

这个`\fbox{%  
 \parbox[c][3\height]{1cm}{%  
 字盒\\空荡荡。}}`

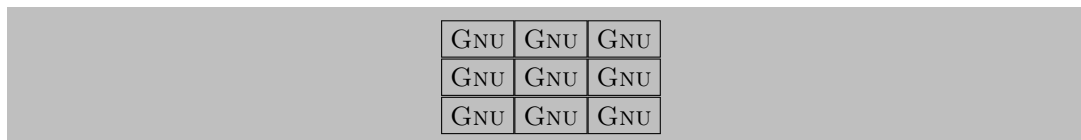
#### 4.4.5 保留和复用

可以将一段 $\text{\LaTeX}$ 源代码储存在字盒中，方便复用——例如对于需要大量 $\text{\LaTeX}$ 重要资源的代码。在这种情况下，可以通过三个步骤实现：

1. 使用指令`\newsavebox`声明一个字盒；
2. 使用`\sbox`或`\savebox`实现存储；
3. 使用`\usebox`实现复用。

例如，此处使用Gnu作为纹理铺贴：

清单 4.17



```
\newsavebox{\gnu}
\sbox{\gnu}{\fbox{\textsc{Gnu}}}
\begin{center}
```

```
\usebox{\gnu}\usebox{\gnu}\usebox{\gnu}\\
\usebox{\gnu}\usebox{\gnu}\usebox{\gnu}\\
\usebox{\gnu}\usebox{\gnu}\usebox{\gnu}
\end{center}
```

可以将指令`\sbox`和`\savebox`与指令`\mbox`和`\makebox`（参见§4.4.1）类比。

## 4.5 定义

专家又发话了：

“……这些宏应当被了解，因为它们的力量如此强大。简短的宏可以代表大量的内容。可以说，宏在某种程度上有一种宏观的效果。”——D.E. 克努特，*T<sub>E</sub>Xbook*

在文档中，我们会定义独立的“实体”。如果这个实体出现的次数多于“某个特定值”，就有必要去敏锐地判断是否应该将它做成宏。这个句子表述得有些模糊。总的来说，使用宏可以避免你将相同的动作重复做 $x$ 次。稍微从实践的角度来说，在情况变得越来越复杂时，可以方便地定义一些指令。

### 4.5.1 指令

指令`\newcommand`可以用来定义宏。该指令使用起来十分简单：

```
\newcommand{<宏名>}[<参数量>]{<LTEX代码>}
```

其中<参数量>是指参数的数量——所谓参数，是编程中的术语。此外，<L<sub>T</sub>E<sub>X</sub>代码>是指你的指令的定义。使用宏的示例如下，其中定义了测色法中表示空间的符号：

#### 清单 4.18

设空间 $L^*a^*b^*$ ……

```
\newcommand{\lab}{ $L^*a^*b^*$ }
设空间\lab{}……
```

注意，这个指令不接受参数，因此没有必要选用参数(参数量)。可以通过以下方式来改善该指令的使用方法：

#### 清单 4.19

对于空间 $L^*a^*b^*$ 及 $\vec{c} \in L^*a^*b^*$ ……

```
\newcommand{\Lab}{%
  \ensuremath{L^*a^*b^*}}
```

对于空间 $\backslash\text{Lab}\{\}$ 及 $\$\text{vec}\{c\}\backslash\text{in}\backslash\text{Lab}\$$ ……

指令 $\backslash\text{ensuremath}$ 确保指令需要在数学环境中使用，上下文可以是任意内容。正如上例所示。

①  $\text{\LaTeX}$ 的宏或指令并不完全承载编程语言中函数的意义，它更像是C中的`# define`，在这种意义上遵循扩展机制。如此一来，在前一次出现 $\backslash\text{Lab}$ 的示例中， $\backslash\text{Lab}$ 会“扩展”成 $\$L^*a^*b^* \$$ 。因此，我们就知道了为什么此时的 $\$...\backslash\text{Lab}\$$ 会造成编译错误。

如下指令使用了一个参数，可以用来代表键盘上的按键<sup>15</sup>：

#### 清单 4.20

先按 $\boxed{\text{Tab}}$ 键，再按 $\boxed{\text{Entrée}}$ 键。

```
\newcommand{\Touche}[1]{\Ovalbox{\#1}}
```

先按 $\backslash\text{Touche}\{\text{Tab}\}$ 键，

再按 $\backslash\text{Touche}\{\text{Entrée}\}$ 键。

可以看到，此处的指令会等待一个变量（这也是变量“[1]”的作用）。在指令的定义中，可以使用 $\#1$ 来引用这个变量。

如果希望定义带有多个参数的函数（最多9个），也完全OK：

#### 清单 4.21

$\frac{1}{2}$ 加 $\frac{3}{4}$ 等于 $\frac{5}{4}$

```
\newcommand{\fraction}[2]{%
  \raisebox{0.5ex}{\#1}%
  \slash\raisebox{-0.5ex}{\#2}}
\fraction{1}{2}加\fraction{3}{4}等于
\fraction{5}{4}
```

可以注意到以下内容：

- 宏 $\backslash\text{fraction}$ 可以接受2个变量；
- 可以使用 $\#n$ 来引用第 $n$ 个变量；
- 符号 $\%$ 看起来可能会有些突兀——它的作用是使得代码换行，同时避免在最终文档中插入空格（参见9.2.1小节）。

除此之外，也可以将指令的定义第一个参数为可选参数。语法如下：

```
\newcommand{<宏名>}[<参数量>][<默认值>]{<\LaTeX代码>}
```

其中， $\langle$ 参数量 $\rangle$ 依然指参数的数量，参数 $\#1$ 带有默认值 $\langle$ 默认值 $\rangle$ ， $\langle$ LaTeX代码 $\rangle$ 是指令的内容。以下示例再次实现了前文实现过的效果——显示键盘按键：

<sup>15</sup>其中调用了包`fancybox`中定义的指令 $\backslash\text{Ovalbox}$ 。



## 清单 4.22

先按 `\Key{Tab}` 键，再按 `\Key{Entrée}` 键。

```
\newcommand{\Key}[1][Entrée]{\Ovalbox{#1}}
```

先按 `\Key{Tab}` 键，

再按 `\Key{}` 键。

可以看到，参数#1不是强制的，它的默认值是“Entrée”。同时，我们可以注意到，使用非强制参数时，应该使用方括号而非花括号。

① 可以充分地想象这样一种情景：我们定义的指令带有一个非强制参数和多个强制参数。这种情况下，第一个强制参数应该是#2。此外，需要注意到，我们只能将**第一个参数**设为非强制参数。

## 4.5.2 环境

我们可以使用以下方式定义自己的环境：

```
\newenvironment{<环境名>}[<参数量>]{<begin条目>}{<end条目>}
```

其中，可以通过<环境名>定义环境名，<参数量>代表参数的数量，<begin条目>和<end条目>分别代表环境“开端”和“末尾”所需的处理。在定义环境时使用其他环境（如L<sup>A</sup>T<sub>E</sub>X的环境）是很实用的技巧<sup>16</sup>：

## 清单 4.23

男人从自然界得到了  
一把钥匙，  
可以每24小时  
为女人上一次发条。

```
\newenvironment{bonmot}%
{\small\slshape\begin{flushright}}%
{\end{flushright}\normalsize\upshape}
\begin{bonmot}
  男人从自然界得到了\\
  一把钥匙，\\
  可以每24小时\\
```

<sup>16</sup>译注：正如下文所说，译者也只能查到这句话来自维克多·雨果（Victor Hugo）的说法，但没有查到具体出处，只能照字面意义硬翻。

为女人上一次发条。

`\end{bonmot}`

的确，如果我们不指明来源，那么这句“名言”（bon mot）看起来疑点重重。通过为我们的环境添加一个参数，可以补救一下。参数可以使用#访问，但这种访问方式只能在begin条目中使用。将参数存到字盒中，就能规避这种限制，从而在end条目中复用▶ 参数：

◀§4.4.5

#### 清单 4.24

男人从自然界得到了  
一把钥匙，  
可以每24小时  
为女人上一次发条。  
维克多·雨果

```
\newsavebox{\auteurbm}
\newenvironment{Bonmot}[1]%
  {\small\slshape%
  \savebox{\auteurbm}{%
    \upshape\sffamily#1}%
  \begin{flushright}}
  {\[4pt]\usebox{\auteurbm}
  \end{flushright}\normalsize\upshape}
\begin{Bonmot}{维克多·雨果}
  男人从自然界得到了\
  一把钥匙，\
  可以每24小时\
  为女人上一次发条。
\end{Bonmot}
```

好吧，即使这样做了，这句“名言”的疑点也没有减少……

### 4.5.3 重定义

可以使用以下方式重新定义指令或环境。对于指令：

`\renewcommand{<指令名>}[<参数量>]{<TeX代码>}`

对于环境：

`\renewenvironment{<环境名>}[<参数量>]{<begin条目>}{<end条目>}`

我们可以通过重定义所需的指令来定制一些L<sup>A</sup>T<sub>E</sub>X的滑稽行为，也可以按照最自然的习惯行事。例如：

```
\renewcommand{\thepage}{\Roman{page}}
```

以上指令可以将页码改为罗马数字。

① 对 $\text{\LaTeX}$ 默认行为的修改是一个很广的话题，超出了本小节的讨论范围。然而需要明白，如果修改某个指令或某个环境时没有统筹考虑所有功能，那么最终的结果会很奇怪！本书第II部分会介绍一些重定义 $\text{\LaTeX}$ 指令的方法。

## 4.6 然后呢？

如果想要创建一个包含你的指令的文件，则需要添加一行指令。如果你使用`bash`，则需要添加在`.bash_profile`中：

```
export TEXINPUTS=$HOME/LaTeX/mesmacros//:
```

这行指令可以让 $\text{\LaTeX}$ 额外搜索目录`$HOME/LaTeX/mesmacros`（此处为一个示例目录）和其子目录。此时，使用指令`\usepackage{moncru}`即可使用你定义的指令集。 $\text{\LaTeX}$ 会搜索文件`moncru.sty`。另一个方法是使用指令`\input{moncru.sty}`。

对于绝大部分 $\text{\LaTeX}$ 发行版，文件`texmf.cnf`会包含一系列参数，用于配置 $\text{\LaTeX}$ 引擎，特别是搜寻文件的路径。借助以下指令，可以找到这个文件：

```
❏ kpsewhich texmf.cnf
```

此外，存在一种机制，可以使`lambda`用户在发布树外存储其指令。请看以下指令：

```
❏ kpsewhich -var-value=TEXMFHOME
```

在我的系统（ $\text{\TeX}$ Live Ubuntu）下，这条指令返回了以下信息：

```
/home/lozano/texmf
```

这条信息表明，系统会在我的私人目录下的`texmf`目录中寻找需要包含单文件。因此，可以将“古法酿造”的扩展放入目录`~/texmf/tex/latex`。

最后再给出一个建议：为了以一种舒适的方式定义属于你单指令或环境，我们建议你稍微看一看以下工具。

- 扩展`ifthen`►。借助它，可以使用“如果—则—否则”和“do-while”形式的控制指令。◀\$9.3
- 包`calc`，可以实现有关计数器和长度的算术。
- 环境`list`►。对于自定义列表种类的环境，它可以实现很好用的项目符号。◀\$9.5

这些扩展和它们的使用方式会在本书第II部分详细介绍。



## Chapter 5

# 图表学

不可为自己雕刻偶像，也不可作什么形像……不可跪拜那些像，也不可事奉它。——  
《圣经·申命记》5:8<sup>1</sup>

今天，在文档中插入画、照片或其他类型的图像看起来稀松平常。这是由于印刷技术的性能日渐强大、运行日渐稳定。与此不同，在20世纪80年代

---

<sup>1</sup>译注：本句实际出自《圣经·申命记》5:8和5:9。