

第2章

组织你的文本

从这一章开始，我们将要深入 \LaTeX 编辑的各个方面。首先来看 \LaTeX 文档中文本的编辑和组织，在探究中，我们会从一个空格、一个标点开始分析，但深入细节的同时也不要忘记整个文档的结构和组织性，要见树木更见森林。

2.1 文字与符号

2.1.1 字斟句酌

简单正文的输入没有太多特别之处， \TeX 传统上使用扩展 ASCII 字符集^①，较新的 \TeX 引擎使用 UTF-8 编码。在接受的字符集之内，除了个别特殊符号，大部分字符可以直接录入。

2.1.1.1 从字母表到单词

\LaTeX 可以从标准键盘上直接打出 26 个字母的大小写形式，当然，从字母表到单词只有一步之遥，即用空格和标点把字母分开。但事实上总免不了要遇到一些稀奇古怪的词汇或人名，试试输入下面的词：

café Gödel Antonín Dvořák Øster Vrå Kırkağaç

或者这些：

^① 旧的中文处理方式使用 CJK 宏包，它的 GBK 等编码支持是实际也是在扩展 ASCII 字符集上工作的，只是使用特殊的方式把两个扩展 ASCII 字符拼接为一个汉字。

χαϊδεύης Крюкова

上面的例子中，前一组词都是来自拉丁字母，但明显增加了许多符号；后一组则是希腊语词汇和俄语人名。我们常用的字母表包括扩展的拉丁字母、希腊字母和西里尔字母（Cyrillic alphabet），这比标准键盘上所能直接输入的 52 个字母要多得多。不过不用担心，数以万计的汉字我们也搞定了，区区几个字母也不在话下。

解决输入超过 ASCII 码范围的字母的问题有传统和现代两类方案，先来看一下现代的方案。

现代的方案就是使用 UTF-8 编码直接输入。在 $\text{X}\text{\LaTeX}$ 这样的排版引擎下，UTF-8 编码是原生的，不需要任何多余的设置：

% UTF-8 编码

café\quad Gödel\quad Antonín Dvořák

χαϊδεύης\quad Крюкова

café Gödel Antonín Dvořák

χαϊδεύης Крюкова

2-1-1

不过，要想正确地输入、显示并输出所有这些符号，仍然不是一件简单的事。

输入特殊的字母需要计算机键盘布局或输入法的支持，例如在法语键盘中（一般可以在操作系统中设置），标准键盘 7890 位置上的字符分别是 è _ ç à，这种键盘布局对需要大量录入法语的人来说特别有用。如果不方便使用特殊键盘来设置，操作系统或编辑器往往还提供了“字符映射表”一类的程序或插件，可以用鼠标选取一些字符输入；中文输入法的软键盘也可以用来输入一小部分特殊的外文字母。

显示 ASCII 以外的字母表需要编辑器所使用的字体的支持。大部分西文字体都支持扩展拉丁字母，如 è、ç，但不是所有字体都有希腊字母和西里尔字母，即使是一些罕用的拉丁字母（如 ř）也有可能缺失。编辑器常用的等宽字体中，Windows 和 Mac 系统都预装的 Courier New，Windows 下的 Consolas、Lucida Sans Typewriter 等都能显示上面所列的所有字母，Linux 下常用的 Bitstream Vera Sans Mono、Inconsolata 则只支持到扩展拉丁字母，因此配置编辑器时也需要仔细选择。

最后但却最重要的是， $\text{T}\text{\LaTeX}$ 系统输出时所使用的字体，也必须要能直接显示这些字母。 $\text{L}\text{\LaTeX}$ 默认的 Computer Modern 在一个字体中只覆盖很小的字符集， $\text{X}\text{\LaTeX}$ 下默认的 Latin Modern 字体要好得多，一般都支持到拉丁字母（这通常就足够了），但希腊、西里尔等字母需要更换其他字体。要完整显示其他语言的字母，就必须频繁地更换不同的字体，或在 $\text{T}\text{\LaTeX}$ 中使用覆盖更大字符集的字体（更换字体参见 2.1.3 节）。比如，为了能正确显示前面例子中的三种字母，我们在前面已经设置了 Windows 系统预装的 Times New Roman 字体。

而传统的解决方案是使用特殊的命令，给字母加上重音的标记，使用特殊字母或者整体更换字母表。

所谓重音 (accents)，是指加在字母上的标记，实际包括抑音符、锐音符、抑扬符等等多种符号。 \LaTeX 支持的重音记号及其命令见表 2.1，命令名称大多取象形的符号。将重音加之于普通的拉丁字母上，可以得到一大批扩展的拉丁字母。

表 2.1 \LaTeX 中的重音命令，以字母 o 为例

ò \‘o	ó \’o	ô \^o	ö \~o
õ \~o	ō \=o	ô \.o	ő \u{o}
ö \v{o}	ő \H{o}	oo \t{oo}	õ \r{o}
q \c{o}	q \d{o}	o \b{o}	

使用命令可以输入的 \LaTeX 字母，见表 2.2。其中 i, j 就是字母 i, j，只是在加上重音命令时需要去掉上面的点。ij, IJ, SS 是字母连写，在默认的字体中没有区别。

表 2.2 \LaTeX 中的特殊字母

Å \AA	å \aa	Æ \AE	æ \ae
Œ \OE	œ \oe	SS \SS	ß \ss
IJ \IJ	ij \ij	L \L	l \l
Ø \O	ø \o	i \i	j \j

如果还要输入希腊字母和西里尔字母，默认的字体就不够用了，需要更换其他编码和字体。为此， \LaTeX 提供了 babel 宏包，可以方便同时访问多种语言的字母表。babel 宏包可带有一个或多个语言的可选参数，支持不同的语言，如

2-1-2

```
\usepackage[greek,english]{babel}
```

将使用英语和希腊语，其中最后一个参数的英语是默认语言，此时希腊语就可以用 ASCII 字符代替：

```
\textgreek{abcde}
```

上述代码输出 αβγδε，需要少量俄文的西里尔字母，可以换用 OT2 编码的字体（参见 2.1.3 节）得到，例如：

2-1-3

```
% 导言区 \usepackage[OT2,OT1]{fontenc}  
\fontencoding{OT2}\selectfont ABCabc
```

АБЦаБц

第 2 章 组织你的文本


53

如果排版全文是俄文的文章，也可以用 `russian` 参数使用 `babel` 宏包，不过输入时就不使用 ASCII 码，而使用俄文专用的编码。由于这些语言较少使用，这里不做更多说明，可参见 [31、245、278]。

使用 pdfTeX 这样的传统排版引擎也可以使用 UTF-8 编码输入文字，此时需要使用 `inputenc` 宏包并选用 `utf8` 选项，它会将 UTF-8 的输入编码自动转换为当前字体编码所对应的符号，字体编码的设置仍然与原来一样，如：

```
1 % coding: utf-8
2 % pdflatex 命令编译
3 \documentclass{article}
4 \usepackage[OT2,T1]{fontenc}
5 \usepackage[utf8]{inputenc}
6 \begin{document}
7 café \quad Gödel
8 {\fontencoding{OT2}\selectfont Крюкова}
9 \end{document}
```

2-1-4

 \LaTeX 在排版中会将单词中的一些字母连写为一个符号，即连字 (ligature)。连字的有无和多少一般是由使用的字体决定的，在默认的 Computer Modern 或 Latin Modern 字体中，小写字母组合 `ff` , `fi` , `fl` , `ffi` , `fl` 都有连字^①：

`differ find flight difficlut ruffle``differ find flight difficlut ruffle`

2-1-5


本书中主要使用的 Times 字体则只有 `fi` 和 `fl` 的连字 (`fi` , `fl`)，而一些专业字体可能使用的更多：

`fb fh fj fkffb ffh ffj fflk ct st sp Th`

偶尔出于意义或美观的考虑，需要取消连字。此时可以使用空的分组，或借用 `\` 命令 (参见 2.1.3 节)：

`shelfful shelf{}ful shelf\ful``shelfful shelfful shelfful`

2-1-6

 使用 XTeX 引擎、OpenType 字体时，可以方便地使用 `fontspec` 宏包的 `Ligature` 字体选项选择连字的有无和程度，参见 [212]。

^① 这也是排版中最为基本的五种连字。大部分连字都出现在字母 `f` 之后，这是因为字母 `f` 的实际内容通常会超出自己的字符边界，不使用连字时可能会与相邻字母挤在一起。



练习

2.1 使用 \LaTeX 输入前面特殊拉丁字母的例子：

café Gödel Antonín Dvořák Øster Vrå Kırkağaç

2

2.1.1.2 正确使用标点

在键盘上，可以直接使用的标点符号有 16 种：

, . ; : ! ? ‘ ’ () [] - / * @

标点 , . ; : ! ? 用来分隔句子或部分句子，在每个标点之后应该加上空格，以保证正确的距离和换行。在特殊情况下，这些标点与空格还有一些更微妙的关系，参见 2.1.1.3 节。

引号在 \LaTeX 中用 ‘ 和 ’ 两个符号表示。单引号就用一遍，双引号用两遍。如果遇到单引号与双引号连续出现的情形，则在中间用 \, 命令分开：

2-1-7

‘\, ‘A’ or ‘B?’\,’ he asked.

“ ‘A’ or ‘B?’ ” he asked.

这里 \, 命令产生很小的间距，注意 \LaTeX 并不会忽略以符号命名的宏前后的空格，所以在它前后都不要加多余的空格。符号 ’ 同时也是表示所有格和省字的撇号（apostrophe，如 “It’s Knuth’s book.”）。

引号和括号通常要在前后加空格分隔单词。逗号、句号等标点何时放在引号和括号内，何时放在引号和括号外，可参见英文写作的格式指导 [12、282]。

除了在数学模式中表示减号，符号 - 在 \LaTeX 正文中也有多种用途：单独使用时它是连字符（hyphen）；两个连用（--），是 en dash，用来表示数字范围；三个连用（---），是 em dash，即破折号^①：

2-1-8

An inter-word dash or, hyphen, as in X-ray.

A medium dash for number ranges, like 1--2.

A punctuation dash---like this.

An inter-word dash or, hyphen, as in X-ray.

A medium dash for number ranges, like 1-2.

A punctuation dash—like this.

^① en 和 em 指符号的宽度，详见 2.1.5 节。

第2章 组织你的文本

55

不过，按中文写作的习惯，表示数字范围也常使用符号 \sim （数学模式的符号 \sim ），有时也用汉字的全角破折号或半个汉字破折号。

西文的省略号（ellipsis）使用`\ldots`或`\dots`命令产生，相比直接输入三个句号，它所略微拉开的间距要合理得多：

Good: One, two, three`\ldots`

Bad: One, two, three...

Good: One, two, three...

Bad: One, two, three...

2-1-9

2

`\ldots`与`\dots`命令在正文中是等价的，它们会在每个点后面增加一个小的间距，因而直接在`\ldots`后面再加逗号、句号、叹号等标点，也能得到正确的间距。西文省略号的用法在不同的格式手册中往往有详细规定^[12、35]，通常在句中使用时，前后要加空格，而在句末使用则应该使用4个点。这是因为`\ldots`的后面也有间距，所以使用`H\ldots.`能得到正确的“H...”，但直接使用`H \ldots\ H`却将得到错误的间距“H ... H”（后一个间距比前一个大）。解决的办法是把省略号放进数学模式：

She \ldots she got it.

I've no idea`\ldots`.

She ... she got it.

I've no idea....

2-1-10

标准键盘上不能直接录入的标点符号有10个，它们占据了主键盘上面一排的一大半：

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

它们都有特殊作用，其中的许多我们已经熟知：数学模式符号`$`、注释符`%`、上标`^`、分组`{ }`、宏命令`\`。剩下的符号中，`~`是带子，`#`用在宏定义中，`&`用于表格对齐，而`_`表示数学模式的下标，我们也将会在后面的章节中陆续遇到。要在正文中使用这些符号，大部分是在前面加`\`，只有个别例外：

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

`# $ % & { } _ \`

2-1-11

可以用没有字母的重音`\~{}`和`\^{}{}`输出`~`和`^`，但这两个符号一般不直接在普通正文中出现，而出现在其他地方：可以是重音符号；可以出现在程序代码中（就像现在看到的`~`和`^`），参见2.2.5节使用抄录；此外还有一个数学符号 \sim ，参见4.3.3节。

符号

| < > + =

虽然可以接受,但它们一般用在数学公式中,其文本形式的效果不好或有错,一般并不使用它们。键盘上的双引号"一般也极少使用在正文中,而常被另外定义移作他用。


中文使用的标点与西文标点不同,中文写作使用全角标点:

句号	。	或.	逗号	,	顿号	、	分号	;
冒号	:		问号	?	感叹号	!	间隔号 ^①	·
单引号	‘ ’		双引号	“ ”	单书名号	〈 〉	双书名号	《 》
括号	()	[]						
省略号	……		破折号	——				

在计算机中使用中文输入法录入全角标点是通常是很直接的。特别需要说明的是破折号(——)和省略号(……),它们都占两个中文字符,在大部分输入法中可以使用 Shift 键加减号 - 和数字 6 键得到。

在科技文章中,为与数字、字母区分,中文的句号一般也使用一个圆点表示,此时应该使用全角的“。”而非混用西文句号。这个标点在大部分中文输入法下可能不易输入,可以先统一使用句号“。”,最后统一替换。

也有一种科技文章的写作风格,是中文与西文统一使用西文的标点,只有顿号、破折号和省略号仍用中文标点。但这样做可能造成标点大小、位置与汉字对不准,以及字体风格的不统一,应该小心使用。

 **LaTeX** 并不会自动处理好汉字标点的宽度和间距,甚至不能保证标点的禁则(如句号不允许出现在一行的开始)。使用 **X_YTeX** 作为排版引擎时,中文标点一般是由 **xeCJK** 宏包控制的。**xeCJK** 提供了多种标点格式^②,默认是全角式,即所有标点占一个汉字宽度,只在行末和个别标点之间进行标点挤压。还支持其他一些标点格式,可以使用 `\punctstyle` 命令修改:

^① 在较早版本的 **xeCJK** 中间隔号被看做是西文标点,需要用 `\xeCJKsetcharclass{‘·’}{‘·’}{1}` 命令把它设置为汉字符号。参见 2.2.5 节。

^② 旧的中文处理方式使用 **CJK** 宏包处理汉字,**CJKpunct** 宏包处理标点的禁则和间距。**CJKpunct** 也可以使用 `\punctstyle` 命令设定标点的格式,用法和 **xeCJK** 宏包类似。



西文的逗号、句号、分号等标点后面应该加空格，这不仅能保证正确的间距，也能保证正确的换行。这是因为标点后如果没有空格，就不能换行。 \LaTeX 在西文句末（包括句号、问号？和叹号！）后面使用的距离会比单词间的距离大些，这在上面的例子中已经可以看到。更确切地说， \LaTeX 把大写字母后的点看做是缩写标记，把小写字母后的点看做是句子结束，并对它们使用不同的间距；但偶尔也有大写字母结束的句子，或小写字母的缩写，这时就必须明确告诉 \LaTeX 使用普通单词间的空格 $\backslash\,$ ，或用 $\backslash@.$ 指明，是大写字母后的句末。例如：

2-1-15

A sentence. And another.

U.S.A. means United States Army?

Tinker et al. _made_the_double_play.

Roman number XII\@. Yes.

A sentence. And another.

U.S.A. means United States Army?

Tinker et al. made the double play.

Roman number XII. Yes.

有时也需要整体禁止这种在标点后的不同的间距，法语排版的习惯就是如此。此时可以使用 `\frenchspacing` 命令来禁止标点后的额外间距。

汉字后的空格会被忽略^①。使用 `xelatex` 编译中文文档时，汉字和其他内容之间如果没有空格，`xeCJK` 宏包会自动添加^②：

2-1-16

中文和English的混排效果并不依赖于「space」的有无。

中文和 English 的混排效果并不依赖于 space 的有无。

个别时候需要忽略汉字与其他内容之间由 `xeCJK` 自动产生的空格，这时可以把汉字放进一个盒子里面：

2-1-17

`\mbox{条目}-a` 不同于条目-b

条目-a 不同于条目 -b

① 旧的中文处理方式使用 CJK 宏包。它的 CJK* 环境忽略汉字后面的空格，而 CJK 环境则保留。

② 使用 CJK 方式处理中文时, 汉字与其他内容之间的间距必须手工控制, 常用的办法是使用 CJK* 环境, 并用 `\CJKtilde` 命令改变符号 `~` 的意义, 表示汉字与其他内容间的距离, `TeX` 中的带子改用 `\nbs` 命令表示。可以手工在合适的地方加 `~` 表示间距, 或用命令行工具 `cctspace` 添加。这是 `ctex` 宏包及文档类不使用 `XLaTeX` 时的默认设置。另一种较好的方式是使用 `CJKspace` 宏包, 它忽略汉字间的空格, 保留其他空格, 从而可以用较自然的方式书写, 但这类方式都不如在 `XLaTeX` 引擎下方便。

⚠ 有时需要完全禁用汉字与其他内容之间的空格（例如在本书所有 \TeX 代码中），这时可以使用 `\CJKsetecglue` 手工设置汉字与其他内容之间的内容为空（默认是一个空格）：

```
\CJKsetecglue{}  
汉字word
```

汉字word

2-1-18

⚠ 在空格之中，最神奇的一种可能就是被称为幻影（`phantom`）的空格。幻影命令 `\phantom` 有一个参数，作用是产生与参数内容一样大小的空盒子，没有内容，就像是参数的一个幻影一样。偶尔可以使用幻影完成一些特殊的占位和对齐效果：

```
幻影\phantom{参数}速速隐形  
幻影参数速速显形
```

```
幻影    速速隐形  
幻影参数速速显形
```

2-1-19

类似地有 `\hphantom` 和 `\vphantom`，分别表示水平方向和垂直方向的幻影（在另一个方向大小为零）。

空行，即用连续两个换行表示分段，段与段之间会自动得到合适的缩进。任意多个空行与一个空行的效果相同。

⚠ 分段也可以用 `\par` 命令生成，这种用法一般只在命令或环境定义的内部使用，而普通行文中不宜出现。与连续的空行类似，连续的 `\par` 命令也只产生一次分段效果。

⚠ 除了分段，也可以让 \TeX 直接另起一行，并不分段。有两种相关的命令：`\\` 命令直接另起一行，上一行保持原来的样子；而 `\linebreak` 则指定一行的断点，上一行仍按完整一行散开对齐：


```
这是一行文字\\另一行  
这是一行文字\linebreak 另一行
```

```
    这是一行文字  
    另一行  
      这 是 一 行 文 字  
    另一行
```

2-1-20

`\\` 一般用在特殊的环境中，如排版诗歌的 `verse` 环境（2.2.2 节），特别是在对齐（2.2.6 节）、表格（5.1.1 节）和数学公式（4.4 节）中使用广泛，但很少用在普通正文的行文中；`\linebreak` 命令则用在个别不太合适分行的手工精细调整上。为了完成精细调整分行的功能，`\linebreak` 可以带一个从 0 到 4 的可选参数，表示允许断行的

程度，0 表示不允许断行，默认的 4 表示必须断行。类似地，也有一个 `\nolinebreak` 命令，只是参数意义与 `\linebreak` 相反。注意在正常的行文中，这两个命令都不会被用到。

 `\` 命令可以带一个可选的长度参数，表示换行后增加的额外垂直间距。如 `\[2cm]`。因此必须注意在命令 `\` 后面如果确实需要使用方括号（即使括号在下一行），则应该在 `\` 后面加空的分组以示分隔，否则会发生错误，这种情况在数学公式中非常常见：

2-1-21

```
\begin{align*}
[2 - (3+5)]\times 7 &= 42 \\\{}
[2 + (3-5)]\times 7 &= 0
\end{align*}
```

$$[2 - (3 + 5)] \times 7 = 42$$

$$[2 + (3 - 5)] \times 7 = 0$$


练习

2.2 \LaTeX 用命令 `\#` `\$` `\%` 表示符号 `#` `$` `%`，为什么给反斜线 `\` 用了 `\textbackslash` 这么复杂的名称？


2.1.2 特殊符号

除了一般的文字，有时还需要使用一些特殊的符号，其中在正文中最为常用的如表 2.3 所示。

表 2.3 正文常用的部分特殊符号

<code>\\$</code>	<code>\S</code>	<code>\dagger</code>	<code>\dag</code>	<code>\ddagger</code>	<code>\ddag</code>	<code>\P</code>	<code>\P</code>
<code>\copyright</code>	<code>\copyright</code>	<code>\textregistered</code>	<code>\textregistered</code>	<code>\texttrademark</code>	<code>\texttrademark</code>	<code>\pounds</code>	<code>\pounds</code>
<code>\textbullet</code>	<code>\textbullet</code>						

上面的几种符号中，不带 `\text` 前缀的是在文本模式和数学模式通用的。特殊符号依赖于当前使用的字体，上面所列举的是在 \LaTeX 默认字体设置下的效果。

\LaTeX 定义了一批常用的符号命令，但实际使用时仍然捉襟见肘。在不同的字体包中，也定义了大量的符号，例如最常用的 \LaTeX 的基本工具宏包 `textcomp` 就定义了大量用于文本的符号，例如欧元符号 `\texteuro` (`€`)、千分符 `\textperthousand` (`‰`) 等；`tipa` 宏包提供了国际音标字体的访问（比如 `[lat&k]`）；`dingbat`、`bbding`、`pifont` 等宏包提供了许多指示、装饰性的小符号，如 ，等等。在这里一一枚举所有这些宏包和

第2章 组织你的文本

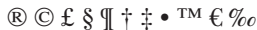
61

命令既冗长无味，也没有必要，Pakin 编写的“ \LaTeX 符号大全”^[192]是查找各种古怪符号的绝佳参考，它收集了约 70 个宏包的近 6000 个文本或数学符号，日常使用的各种符号一般都能在上面找到，同时这个文档也讲述了符号字体的一些一般知识及制作新符号的办法。

使用宏包来调用特殊符号时，需要注意的是，有些宏包只提供符号命令（如 `bbding`），可以随意使用；有些宏包提供一套符号字体的选择方式（如 `tipa`），可以通过与此符号对应的 ASCII 符号或符号的数字编码来使用符号；有些宏包则实际上是完整的正文字体包（如 `fourier`），使用它会整体改变全文的默认字体，同时提供一些额外的符号。“符号大全”^[192]对这些情况并没有仔细区分，可能需要再查看相应符号宏包的文档才能进一步了解它们的用法和注意事项。


`xunicode` 宏包重定义了 \XeTeX 的 UTF-8 编码下（EU1 字体编码）的大量符号的命令，使得在新编码下，`\textbullet` 之类的命令也能正常工作，而 `\'e` 这样的复合重音标记也会自动转为字体中带重音的字母。`xunicode` 宏包会自动被 `fontspec` 或 `ctex` 宏包载入，但在旧的系统下可能需要手工完成。不过，当字体本身缺少需要的符号时，需要的符号就无法显示，即使使用的是 `\'e` 这样的复合形式，你也可能需要更换其他字体或切换编码（参见 2.1.3 节）来显示这些符号。


其实，使用 \XeTeX 引擎时，输入特殊符号最简捷的办法就是在 UTF-8 编码下直接输入：



2-1-22

当然，与输入特殊字母一样，这种方式也要求输入法、编辑器字体和输出字体的共同支持。通过更换字体（参见 2.1.3 节），可以用输入任何可以找得到的符号。

 即使并非使用 \XeTeX 引擎，依然可能直接输入标准 ASCII 编码以外的特殊符号。这依然需要使用比 ASCII 更大的输入编码。`inputenc` 宏包^[116]允许使用诸如 `ascii`（默认值，ASCII）、`latin1`（ISO 8859-1）、`utf8`（UTF-8）等参数表示使用的输入编码。用这种方式也可以直接输入特殊字母与特殊符号，不过这里的 UTF-8 编码也只支持西方的拼音文字，无法支持汉字。

 使用 \XeTeX 或其他引擎直接指定字体输入特殊符号时，一个很大的问题是，并非所有符号都可以用键盘输入，即使借助特殊输入法、字符映射表等工具输入了这些符号，在源代码编辑器中仍然可能无法显示。为此， \LaTeX 提供了命令 `\symbol` 来直接用符号在字体中的编码来输入符号。`\symbol` 命令带有一个参数，就是一个表示字符编码的数字。在 \TeX 中数字除了普通的十进制，也可以用八进制、十六进制或编码对应的字符本身来表示，其语法形式如表 2.4 所示。其中字符形式中的字符如果是特殊字

符，需要在前面加 `\` 转义，如用 `\symbol{'\%}` 就得到 `%` 本身，与 `\%` 类似。

表 2.4 \TeX 中不同的数字表示形式

表示法	语法形式	例
十进制	<code><数字></code>	90
十六进制	<code>"<数字></code>	<code>"5A</code>
八进制	<code>'<数字></code>	<code>'132</code>
字符形式	<code>'<字符></code>	<code>'Z</code>

要得到字符的编码，对于传统的 \TeX 字体，可以参见字体的符号码表，许多字体宏包（如 `txfonts`）在文档中都列出了很长的字体符号码表，选定字体后，就可以利用 `\symbol` 命令输入用键盘难以输入的符号，例如：

2-1-23

```
\usefont{T1}{t1xr}{m}{n}
\symbols{"DE}\symbol{"FE}
```

Ðþ

而对于 \XeTeX 调用的 OpenType、TrueType 字体，一般编码均为 Unicode，可以查通用的 Unicode 码表或利用字符映射表等外部工具，查看符号的编码，输入对应的符号，例如：

2-1-24

```
\symbol{28450}\symbol{35486}
```

漢語

使用这种方式，就可以只使用 ASCII 编码的源文件，排版出任何字体中的任何符号。

2.1.3 字体

2.1.3.1 字体的坐标

当我们说“换一个字体”时，指的是什么呢？可能是想换掉文字的整体感觉，如 `TEXT` 与 `TEXT`；可能是想把直立的文字改为倾斜的，如 `TEXT` 与 `TEXT`；可能是想把细的文字改为粗的，如 `TEXT` 与 `TEXT`；可能是想把包含一些符号的字体改为包含另一些符号的，如 `TEXT` 与 `0eχ0`；还可能只是想改变字体的大小，如 `TEXT` 与 `TEXT`。——尽管所使用的文字内容都是一样的（`TEXT`）。

上面说的五种不同的性质，在 \LaTeX 中一起决定了文字的最终输出效果。字号（font size），即文字大小，常常被独立出来，看做不同于字体的单独的性质（参见 2.1.4 节）；

字体编码 (font encoding), 即字体包含的符号也较少直接设定。使用最多的是其他的三种性质, 即字体族 (font family)、字体形状 (font shape)、字体系列^① (font series)。

L^AT_EX 提供了带参数和命令和字体声明两类修改字体的命令, 前者用于少量字体的更换, 后者用于分组或环境中字体的整体更换。例如:

```
\textit{Italic font test}
```

Italic font test

```
{\bfseries Bold font test}
```

Bold font test

2-1-25

2

预定义命令的字体族有 3 种: 罗马字体族 (roman family)、无衬线字体族 (sans serif family) 和打字机字体族 (typewriter family)。其命令为:

字体族	带参数命令	声明命令	效果
罗马	<code>\textrm{<文字>}</code>	<code>\rmfamily</code>	Roman font family
无衬线	<code>\textsf{<文字>}</code>	<code>\sffamily</code>	Sans serif font family
打字机	<code>\texttt{<文字>}</code>	<code>\ttfamily</code>	Typewriter font family

正文默认使用罗马字体族。字体族一般对应一组风格相似, 适于一起使用的成套字体。

预定义命令的字体形状有 4 种: 直立形状 (upright shape, 也称为 roman shape)、意大利形状 (italic shape)、倾斜形状 (slanted shape)、小型大写形状 (small capitals shape)。其命令为:

字体形状	带参数命令	声明命令	效果 (Latin Modern Roman 字体族)
直立	<code>\textup{<文字>}</code>	<code>\upshape</code>	Upright shape
意大利	<code>\textit{<文字>}</code>	<code>\itshape</code>	<i>Italic shape</i>
倾斜	<code>\textsl{<文字>}</code>	<code>\slshape</code>	<i>Slanted shape</i>
小型大写	<code>\textsc{<文字>}</code>	<code>\scshape</code>	SMALL CAPITALS SHAPE

正文默认使用直立字体形状。注意其中倾斜形状和意大利形状的区别, 倾斜形状差不多是直接对符号倾斜产生的, 而通常所说的“斜体”往往是指意大利形状, 它类似于更加圆滑的手写体。因为数学公式中的字体一般使用意大利形状, 因而与数学混排时倾斜形状不会与公式中的字母混淆; 在标题、参考文献中也有使用倾斜形状的。并非所有字体族都有这么多形状, 除了 L^AT_EX 默认的 Computer Modern 和 Latin Modern, 大多数字体都只有意大利形状与倾斜形状中的一种 (比如本书使用的 Times 字体); 很多字体也可能缺少小型大写字母的符号。另一方面, 一些其他字体可能也会提供更多的

^① 通常指字体的重量 (weight, 即粗细) 和宽度 (width)。

字体形状，如 Venturis ADF 系列字体^[206] 就提供倾斜的小型大写（italic small capitals）、空心（outline）等形状。

预定义命令的字体系列有中等（medium）和加宽加粗（bold extended）两类：

字体系列	带参数命令	声明命令	效果
中等	<code>\textmd{<文字>}</code>	<code>\mdseries</code>	Medium series
加宽加粗	<code>\textbf{<文字>}</code>	<code>\bfseries</code>	Bold extended series

正文默认使用中等字体系列。两个命令表示的意义对不同套的字体可能有所区别，如命令 `\textbf` 和 `\bfseries` 对默认的字体选择加宽加粗的字体系列，但对一些字体则是选择加粗（bold）或半粗（demi-bold）字体系列。

字体的这三种性质有如确定字体的三维坐标，同一维度内的性质不能重叠，但不同类的性质则可以叠加。三种性质的组合效果见表 2.5。

表 2.5 字体的坐标：族、形状和系列。字体族、形状和系列用声明形式的字体命令表示。这里以 Latin Modern 字体为例，这是 \LaTeX 的字体默认值。表中实际共有 17 种不同的字体，如果使用其他的字体，表中的缺项可能会有所不同

字体族	\rmfamily		\sffamily		\ttfamily	
系列	\mdseries	\bfseries	\mdseries	\bfseries	\mdseries	\bfseries
形状	\mdseries	\bfseries	\mdseries	\bfseries	\mdseries	\bfseries
<code>\upshape</code>	Text	Text	Text	Text	Text	Text
<code>\itshape</code>	<i>Text</i>	<i>Text</i>	同 <i>slanted</i>	同 <i>slanted</i>	<i>Text</i>	同 <i>slanted</i>
<code>\slshape</code>	<i>Text</i>	<i>Text</i>	<i>Text</i>	<i>Text</i>	<i>Text</i>	<i>Text</i>
<code>\scshape</code>	TEXT	缺	缺	缺	TEXT	缺

除了上面列举的这些字体命令，还有 `\textnormal{<文字>}` 和 `\normalfont` 命令用来把字体设置为“普通”的格式。默认情况下，普通字体相当于 `\rmfamily\mdseries\upshape` 的效果。普通字体特别适用于在复杂的字体环境中恢复普通的字体，尤其是在宏定义这类不知道外部字体设置的情况下，如：

2-1-26

```
\sffamily
\textbf{This is a paragraph of bold and
\textit{italic font, sometimes returning
to \textnormal{normal font} is necessary.}}
```

This is a paragraph of bold and italic font, sometimes returning to normal font is necessary.

第2章 组织你的文本

65

使用斜体声明 (`\itshape`、`\slshape`) 时, 最后一个倾斜的字母会超出右边界, 使得后面的文字与它相距过紧, 而用带参数的命令 (`\textit`、`\textsl`) 就可以自动修正这个距离, 也可以手工使用 `\/` 命令进行这种倾斜校正 (italic correction), 如:

```
{\itshape M}M  
  
\textit{M}M  
  
{\itshape M\/}M
```

*MM**MM**MM*

2-1-27

2

倾斜校正命令 `\/` 会在字母后面加上一个小的距离, 其大小由具体的字体和符号来决定^[126, Chapter 4]。倾斜校正一般只用在声明形式的斜体命令, 但偶尔也使用它取消连字 (参见 2.1.1.1 节), 也可以用来校正一些粗体字母和引号之间的距离 (当然最好还是使用带参数的命令形式):

```
Bold '{\bfseries leaf}'  
  
Bold '{\bfseries leaf\/}'  
  
Bold '\textbf{leaf}'
```

Bold ‘leaf’**Bold ‘leaf’****Bold ‘leaf’**

2-1-28

在很少的情况下, `\textit` 自动加入的倾斜校正是不必要的, 此时可以使用 `\nocorr` 命令禁止校正, 如:

```
\textit{M}M  
  
\textit{M\nocorr}M
```

*MM**MM*

2-1-29

也可以使用 `\renewcommand` 重定义 `\nocorrlist` 命令设置不对特定字符校正, \LaTeX 默认定义不对句号和逗号校正, 相当于已经定义了:

```
\newcommand\nocorrlist{,.}
```

中文字体通常没有西文字体那样复杂的成套的变体, 各个字体之间一般都是独立的, 只有少数字体有不同重量的成套字体。因此, 对于中文字体, 一般只使用不同的字体族进行区分。`xeCJK` 和 `CJK` 宏包机制下, 中文字体的选择命令和西文字体是分离的, 选择中文字体族使用 `\CJKfamily` 命令, 如:

2-1-30

```
{\CJKfamily{hei}这是黑体}
```

```
{\CJKfamily{kai}这是楷书}
```

这是黑体
这是楷书

2

中文的字体族，根据不同的系统和使用方式各有不同。在 `ctex` 宏包及文档类下有一些预定义，在默认情况下（`winfonts` 选项）针对 Windows 常用字体配置了的四种字体族：`song`（宋体）、`hei`（黑体）、`kai`（楷书）、`fs`（仿宋）^①；如果使用了其他选项，则可能会有不同的字体^②，为了方便使用，`ctex` 宏包提供了简化的命令：

2-1-31

```
{\songti 宋体} \quad {\heiti 黑体} \quad  
{\fangsong 仿宋} \quad {\kaishu 楷书}
```

宋体 黑体 仿宋 楷书

`ctex` 宏包及文档类（如 `ctexart`）另外定义了一些组合字体，可以让中文也具备使用粗体（`\bfseries`）和意大利体（`\itshape`）的功能，并且重定义 `\rmfamily` 使它同时对中文起作用。默认的中文字体族是 `rm`，其正常字体是宋体，粗体是黑体，意大利体是楷体，如：

2-1-32

```
% ctex 宏包下默认相当于 \CJKfamily{rm}  
% \rmfamily 或 \textrm 也会同时设置此字体  
中文字体的\textbf{粗体}与\textit{斜体}
```

中文字体的**粗体**与斜体

类似地，`\sffamily`（对应 `sf` 中文字体族）和 `\ttfamily`（对应 `tt` 中文字体族）也可以同时作用于西文和中文，分别相当于幼圆和仿宋体。

❖ \LaTeX 的这种利用相互正交的几种性质来区分字体的方式，称为新字体选择方案^[142]（New Font Selection Scheme, NFSS）。当然，NFSS 最早于 1989 年发布，现在也并不新了；它是针对 Plain \TeX 和旧版本的 \LaTeX 2.09 中直接指定具体字体旧方案而说的。在旧的字体选择方式中，一个字体命令对应一个单一的字体，甚至有些字体命令对应的字体的大小也是确定的，如在 Plain \TeX 和旧的 \LaTeX 2.09 格式中，命令 `\rm`、`\bf`、`\it` 分别表示使用 Computer Modern 的普通罗马字体、粗罗马字体和意大利体，但使用 `\bf\it` 并不能得到加粗的意大利体，而仍然只是一个 `\it` 的效果。NFSS 改变了


① 对 CJK 方式还有 `li`（隶书）以及 `you`（幼圆）。

② 本书描述 `ctex` 宏包 1.02c 版，使用 \XeTeX 方式处理中文时的字体选择。除了默认的 `winfonts` 选项，还有 `adobefonts` 和 `nofonts` 选项。在 `adobefonts` 选项下有宋、黑、楷、仿宋四种字体可用；而 `nofonts` 选项则没有预定义的字体族和命令，只有自行定义后才能使用，参见 2.1.3.2 节。

第2章 组织你的文本

67

这种不便的使用方式。现在，出于对旧文档的兼容考虑，现在的版本 \LaTeX 2_ϵ 在标准文档类中也保留了 `\rm`, `\bf`, `\it`, `\sc`, `\sl`, `\sf`, `\tt` 等命令（以及数学模式下的 `\mit`, `\cal` 命令），请不要在文档中使用它们。

 NFSS 为字体划分了编码、族、系列、形状、尺寸等多个正交属性，这些属性各自可以用一个简短的符号来表示，如字体编码有 OT1, T1, OML, OMS, OMX, U 等；字体族有 cmr, cmss, cmtt, cmm, cmsy, cmex 等；字体系列有 m, b, bx, sb, c 等；字体形状有 n, it, sl, sc 等，由具体的字体可以有不同的定义，常用的标准定义可参见 NFSS 的标准文档 [142]、Adobe PostScript 字体文档 [229, § 8]。 \LaTeX 提供了更原始的命令：

```
\fontencoding{编码}
\fontfamily{族}
\fontseries{系列}
\fontshape{形状}
\fontsize{大小}{基本行距}    （纯数字，单位是 pt）
```

通过这些命令来使用这些基本属性，需要在后面加 `\selectfont` 命令使它们生效，如：

```
\fontencoding{OT1}\fontfamily{pzc}
\fontseries{mb}\fontshape{it}
\fontsize{14}{17}\selectfont
PostScript New Century Schoolbook
```

*PostScript New Century
Schoolbook*

2-1-33

也可以使用

```
\usefont{编码}{族}{系列}{形状}
```

一次性选择某个字体，如：

```
\usefont{T1}{pbk}{db}{n}
PostScript Bookman Demibold Normal
```

**PostScript Bookman
Demibold Normal**

2-1-34

2.1.3.2 使用更多字体

使用 `\rmfamily`, `\bfseries`, `\itshape` 或 `\kaishu` 这类命令，只能选择预定义的少数几种字体。对于西文，就是 Computer Modern 或 Latin Modern 系列的几款成套的字体；对于中文，就是 6 种预定义的 Windows 字体。这些对于简单的文章并没有问题，

但如果想要改变文章的字体风格 (For example, Palatino), 或者缺少预设的字体文件 (如中文 Linux 用户就不会有 Windows 预装的中文字体), 那么就需要用到预设之外的更多字体。

选择非默认的字体有两类方法: 当不使用 $\text{X}\text{\LaTeX}$ 引擎时, 可以通过字体宏包或 $\text{L}\text{\LaTeX}$ 底层字体命令调用在 $\text{T}\text{\LaTeX}$ 系统中预先安装好的字体; 当使用 $\text{X}\text{\LaTeX}$ 引擎时, 可以调用操作系统安装的中西文字体。由于质量可靠的中文字体几乎都是商用的, 因此 $\text{T}\text{\LaTeX}$ 系统中只安装了几种质量较差的中文字体, 一般总要调用操作系统所安装的字体来使用 $\text{X}\text{\LaTeX}$ 引擎的功能。

一个 $\text{T}\text{\LaTeX}$ 发行版通常都同时安装了大量西文字体, 直接使用 $\text{L}\text{\LaTeX}$ 底层命令 (参见 2.1.3.1 节末尾) 指定字体通常比较难用, 特别是一些字体同时还包括对应的数学字体和符号命令的设置, 非常烦琐, 因此很多西文字体都做成了方便调用的字体宏包, 可以直接更换整套的西文字体 (或数学字体)。其中, 最为常见的要求是把整套字体换为 Times Roman 的衬线字体 (罗马体) 或 Helvetica 的无衬线字体, Times 字体也能与中文宋体能很好地配合。有好几个宏包可以达到这个目的, 最简单的是 times 宏包^[229], 只更换正文字体, 没有更换配套的数学字体, 很少使用; mathptmx 在 times 宏包^[229] 的基础上增加了数学字体的支持; 效果最好的免费字体则是 txfonts 宏包^[220], 对整套西文字体和数学符号给出了完整的解决方案。使用字体宏包非常简单, 通常只要导入此宏包即可:

```
\documentclass{article}
\usepackage{txfonts}
\begin{document}
Test text
\end{document}
```

2-1-35

txfonts 的效果见图 2.1。

Theorem 1 (Cauchy's Theorem) Let f be holomorphic on a closed disc $\overline{D}(z_0, R)$, $R > 0$. Let C_R be the circle bounding the disc. Then f has a power series expansion

$$f(z) = \sum_{n=0}^{\infty} \frac{(z - z_0)^n}{2\pi i} \int_{C_R} \frac{f(\zeta)}{(\zeta - z_0)^{n+1}} d\zeta.$$

图 2.1 Times 字体效果 (txfonts)

为文章的不同部分选用字体应该协调配合, 因此通常带有配套数学字体的字体包是最为常用的, 但有时也需要分别定义正文字体和与之配套的数学字体, 此时就必须手工指定不同的字体包。例如使用高德纳的 Concrete 正文字体与 Zapf 的 Euler 数学字

体配合时（这正是 Concrete 字体设计时的用法^[87]），就需要综合使用字体包 `ccfonts` 和 `euler`，效果见图 2.2：

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage{ccfonts,eulervm}
\begin{document}
Test text
\end{document}
```

2


2-1-36

Theorem 1 (Cauchy's Theorem) Let f be holomorphic on a closed disc $\overline{D}(z_0, R)$, $R > 0$. Let C_R be the circle bounding the disc. Then f has a power series expansion

$$f(z) = \sum_{n=0}^{\infty} \frac{(z - z_0)^n}{2\pi i} \int_{C_R} \frac{f(\zeta)}{(\zeta - z_0)^{n+1}} d\zeta.$$

图 2.2 Concrete 和 Euler 字体效果（T1 编码，ccfonts, euler）

这一字体组合比较清晰，它与无衬线字体包（如 `arev`, `cmbright` 等）都比较适合于在幻灯片演示中使用。

在使用 Concrete 与 Euler 字体时，我们使用了一个新的宏包 `fontenc` 来选择字体的  编码。`fontenc` 宏包可以包含多个选项，表示文档所使用的正文字体编码，最后一个选项的编码是文档默认使用的编码。字体的编码决定字体包含的符号，同一族的字体可能会有不同编码的版本。除了 \LaTeX 使用的 Unicode 编码（在 NFSS 中一般是 EU1），传统的 \LaTeX 字体编码有一般正文字体 OT1、扩展正文字体 T1、数学字母 OML、数学符号 OMS、数学符号扩展 OMX 等。需要手工设置的通常只有正文字体的编码，默认的正文字体编码是 OT1，而扩展编码 T1 则包含更多的符号，特别是带重音的拉丁字母等。许多字体包都要求使用 T1 编码，才能获得最好的效果，在字体包的说明文档中一般都会提及。注意字体编码是指符号在字体中位置的编码，与输入文档时使用的文档编码不是一回事。

初用 \LaTeX 的最大的问题往往是并不知道 \LaTeX 中有哪些字体可用，因为作为一门排版语言， \LaTeX 并没有把可用的字体放在一个下拉菜单中等待选择，因此必须自己查看 \TeX 发行版的字体安装目录^①或综述性的字体文档。一个带有说明、例子、使用方法和详细功能比较的综述性字体文档是 Hartke [100]，里面介绍了 20 多种带有数学支持

^① 一般是 \TeX 安装目录下的 `fonts` 目录，不过除了 OpenType 和 TrueType 格式的字体，其他字体很不好认。Type 1 格式的字体的名称可参见 Berry [22]，但使用方法仍然需要另外查看说明。

2

的免费 $\text{T}_{\text{E}}\text{X}$ 字体。该文档在 $\text{T}_{\text{E}}\text{X}$ Live 和 $\text{C}_{\text{T}}\text{E}_{\text{X}}$ 套装中都有收录，是使用成套字体的很好的参考。一个收录更为全面的 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 字体目录是 Jørgensen [118]，它展示了 Linux 源中的 $\text{T}_{\text{E}}\text{X}$ Live 所能使用的近 200 种西文字体族。

下面来看现代的方法，使用 $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 来选择字体。在 $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 中，主要使用 `fontspec` 宏包的机制来调用字体。最基本的是设置正文罗马字体族、无衬线字体族和打字机字体族的命令：

```
\setmainfont[(可选选项)]{(字体名)}  
\setsansfont[(可选选项)]{(字体名)}  
\setmonofont[(可选选项)]{(字体名)}  
( 可用的可选选项参见 fontspec 文档 Robertson and Hosny [212] )
```

例如：

```
% 在导言区设置全文字体为 Windows 提供的  
% Times New Roman, Verdana, Courier New 字体  
\usepackage{fontspec}  
\setmainfont{Times New Roman}  
\setsansfont{Verdana}  
\setmonofont{Courier New}
```

2-1-37

此时 `\rmfamily`、`\sffamily` 和 `\ttfamily` 就分别对应设置的三种字体，而且 `fontspec` 会自动找到并匹配对应的粗体、斜体等变体，尽量使 `\bfseries`、`\itshape` 等命令也有效。

也可以定义新的字体族命令：

```
\newfontfamily(命令)[(可选选项)]{(字体名)}
```

例如为 Java 运行库附带的 Lucida Sans 字体定义一个命令 `\lucidasans`：

```
% 导言区使用  
\newfontfamily\lucidasans{Lucida Sans}  
% 正文使用  
{\lucidasans This is Lucida Sans.}
```

2-1-38

This is Lucida Sans.

$\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 下中文字体的设置使用 `xeCJK` 宏包^[310]（`ctex` 宏包或文档类会自动调用它）。`xeCJK` 提供了与 `fontspec` 对应的中文字体设置命令：

```
\setCJKmainfont[<可选选项>]{<字体名>}  
\setCJKsansfont[<可选选项>]{<字体名>}  
\setCJKmonofont[<可选选项>]{<字体名>}  
\setCJKfamilyfont{<中文字体族>}[<可选选项>]{<字体名>}
```

这些命令对于 Linux 等系统下的中文用户特别有用，因为 `ctex` 宏包默认是针对 Windows 的字体配置的，可以用 `nofonts` 选项禁用预定义的中文字体设置而来自定义字体。例如使用文鼎公司免费发布的字体：

```
% 在导言区设置全文字体为“文鼎 P L 报宋二GBK”  
% 并设置中文的 kai 字体族为“文鼎 P L 简中楷”  
\setCJKmainfont{AR PLBaosong2GBK Light}  
\setCJKfamilyfont{kai}{AR PL KaitiM GB}
```

此后使用 `\CJKfamily{kai}` 就会使用文鼎的楷体。

`fontspec` 所能使用的字体是 `fontconfig` 库所能找到的所有字体，也就是在 `TEX` 发行版和操作系统中所安装的字体，通常是 `OpenType` 和 `TrueType` 的字体，也包括一些 `PostScript` 格式字体，需要注意的就是要使用正确的字体名。字体的列表可以在命令行下使用 `fc-list` 命令来列出，通常列出的字体信息会非常多，不能在一屏内完全显示，Windows 下还会因编码不统一而出现乱码，此时可以利用命令重定向操作符把结果输出到文件中：

```
fc-list > fontlist.txt
```

`fc-list` 命令输出的结果类似这样（这里只选取了一小部分，并稍做重新排列）：

```
Minion Pro:style=Bold  
Minion Pro:style=Bold Italic  
Minion Pro:style=Italic  
Minion Pro:style=Regular  
Times New Roman:style=cursiva,kurzíva,kursiv,Πλάγια,Italic,  
Kursivoitu,Italique,Dólt,Corsivo,Cursief,kursywa,Itálico,  
Курсив,Ítalik,Poševno,nghiêng,Etzana  
Times New Roman:style=Negreta cursiva,tučné kurzíva,  
fed kursiv,Fett Kursiv,Έντονα Πλάγια,Bold Italic,  
Negrita Cursiva,Lihavoitu Kursivoi,Gras Italique,  
Félkövér dólt,Grassetto Corsivo,Vet Cursief,Halvfet Kursiv,
```

```
Pogrubiona kursywa, Negrito Itálico, Полужирный Курсив,
Tučná kurzíva, Fet Kursiv, Kalın İtalik, Krepko poševno,
nghiêng đậm, Lodi etzana
Times New Roman:style=Negreta, tučné, fed, Fett, Έντονα, Bold,
Negrita, Lihavoitu, Gras, Félkövé, Grassetto, Vet, Halvfet,
Pogrubiona, Negrito, Полужирный, Fet, Kalın, Krepko, đậm, Lodia
Times New Roman:style=Normal, obyčejné, Standard, Κανονικά,
Regular, Normaali, Normál, Normale, Standaard, Normalny, Обычный,
Normálne, Navadno, thường, Arrunta
宋体, SimSun:style=Regular
黑体, SimHei:style=Normal, obyčejné, Standard, Κανονικά, Regular,
Normaali, Normál, Normale, Standaard, Normalny, Обычный,
Normálne, Navadno, Arrunta
文鼎 P L 报宋 二 GBK, AR PLBaosong2GBK Light:style=Regular
文鼎 P L 简中楷, AR PL KaitiM GB:style=Regular
```

fc-list 列出的是字体名（对应于字体族）和同族字体变体（对应于 L^AT_EX 的字体系列和形状）。在冒号前面的是字体族名称，style= 后面的是字体的变体，用逗号分隔开的是同一个字体（或变体）在不同语言下的名称。如这里的 Minion Pro 字体就有 **Regular**（对应于 \mdseries\upshape）、**Italic**（对应于 \mdseries\itshape）、**Bold**（对应于 \bfseries\upshape）、**Bold Italic**（对应于 \bfseries\itshape）这 4 种变体，而几种中文字体都没有变体。通常 fontspec 宏包会自动处理好字体的变体，因此只要知道前面的字体名称。

fc-list 可以带许多参数来控制输出的格式。例如，我们通常只需要字体族名，而不需要变体的名称，就可以加 -f "%{family}\n" 选项只输出字体族；又如，使用 :lang=zh 选项可以只输出中文字体，而 :outline 则可以只显示矢量字体。下面的命令就可以将所有中文字体族的列表输出到 zhfont.txt 中：

```
fc-list -f "%{family}\n" :lang=zh > zhfont.txt
```

fontspec 宏包为字体，尤其是 OpenType 字体提供了多种字体性质的选项。例如 Minion Pro 字体使用带斜线的数字 0，以与字母 O 区分：

```
\newfontfamily\minion[Numbers=SlashedZero]{Minion Pro}
\minion 100, OK.
```


100, OK.

字体选项的选择可参见 Goossens and Rahtz [86]、Robertson and Hosny [212]，OpenType 字体提供的性质可通过命令行工具 `otfinfo` 或 `opentype-info` 宏包查看。

对中文字体来说，尤其有用的一组 `fontspec` 命令选项是设置斜体、粗体、粗斜体的选项 `ItalicFont`, `BoldFont`, `BoldItalicFont`，这几个选项可以把字体的变体用另一种字体来代替。如可以设置正文字体为宋体，其粗体为黑体、斜体为楷体、粗斜体为隶书，以弥补中文字体一般没有一族成套变体的问题：


```
\setCJKmainfont[BoldFont=SimHei,ItalicFont=KaiTi,
BoldItalicFont=LiSu]{SimSun}
```

`ctex` 宏包默认设置 Windows 字体时正是采用类似的方法。如果不指定中文字体的变体形式，`ctex` 调用 `xeCJK` 时会增加选项使用伪粗体和伪斜体代替。不过中文排版没有斜体的概念，因此在非 Windows 环境一般也都需要设置这种中文的复合字体，避免斜体的使用。

 `fontspec` 主要用于设置正文字体，通常不用来设置数学字体，不过一些数学字体（如 `\mathrm`）默认与正文字体一致，则正文字体的设置会影响到数学字体。`fontspec` 的 `\setmathrm`, `\setmathsf`, `\setmathtt` 等命令可以用来以与正文同样的方式设置这些受影响的数学字体，它们的用法和 `\setmainfont` 等命令类似。此外，`fontspec` 宏包的 `no-math` 选项可以禁用其对数学字体的所有影响（包括 `\setmathrm` 等命令的作用），在加载许多数学字体包时，这个选项都会自动生效，对于不自动加载 `no-math` 选项的宏包，我们可以自己加上这个选项。为了使用正确的字体编码，`fontspec` 应该放在数学字体包的后面。当然，`fontspec` 几乎总会令字体宏包原来的正文字体设置失效，可以使用 `fontspec` 的方式再另行设置搭配的字体，例如：

```
\documentclass{article}
\usepackage{ccfonts}% 公式使用 Concrete 系列字体
\usepackage[no-math]{fontspec}% \mathrm 等也使用 Concrete 系列字体
\setmainfont{Latin Modern Mono Prop}% 正文使用 Latin Modern Mono 字体
```

2-1-40

 处理中文时的情况可能更复杂一些。如果使用 `xeCJK` 或 `ctex/ctexcap` 宏包，可以在它们之前使用数学字体包，需要时可以带 `no-math` 选项加载 `fontspec`，用法和前面西文的文档类似。使用 `ctex` 中文文档类时，如果需要，`fontspec` 的 `no-math` 选项可以传递给文档类。当 `fontspec` 是由文档类加载时，就不可能在这之前加载数学字体包了，但由于个别字体包会改变字体编码，此时可能需要在数学字体包之后以 EU1 编码为最后一个选项加载 `fontenc` 宏包，以保证使用正确的字体编码，例如：

```
\documentclass[no-math]{ctexart}
\usepackage[utopia]{mathdesign}% 数学字体使用 mathdesign 与 Utopia
```

2-1-41

```
\usepackage[EU1]{fontenc}% 恢复正文字体的 Unicode 编码
\setmainfont{Utopia Std}% 正文字体使用 OpenType 格式的 Adobe Utopia
```

在使用数学字体包时同时加载对应的 OpenType 或 TrueType 格式的正文字体，是在 \LaTeX 下使用复杂字体的一般方法。

一种更方便的方式是使用传统的字体包设置全文的字体（数学或正文字体），只把其他字体（如汉字）留给 `fontspec` 和 `xeCJK`。也就是说，我们要将旧的字体选择机制和新的字体选择机制混合使用，这同样可以做到，与前面的区别仅仅是使用 `fontenc` 宏包将传统的非 Unicode 字体编码设置为默认的编码，即最后一个选项^①。这种方法不影响使用 `fontspec` 中 `\newfontfamily` 声明的字体和 `xeCJK` 声明的汉字字体。但 `\setmainfont`、`\setsansfont` 和 `\setmonofont` 这三个命令会因字体编码而失效，需要时在文档中使用 `\fontencoding` 命令临时切换编码，或者直接重定义 `\rmfamily`、`\sffamily` 和 `\ttfamily`，让它们增加切换编码的功能。下面给出一个在设置字体方面比较复杂的例子，以西文的 T1 编码为默认编码调用 `fourier` 字体包，西文用 `fontspec` 的两种方式定义其他字体，汉字也使用 OpenType 的字体：



```
\documentclass[UTF8]{ctexart}
% 西文正文和数学字体
\let\hbar\relax % 解决 xunicode 与 fourier 的符号冲突
\usepackage{fourier}
% 设置默认编码为 T1，以支持 fourier 宏包
\usepackage[T1]{fontenc}
% 定义新的西文 Times 字体族
\newfontfamily\times{Times New Roman}
% 设置西文等宽字体，并重定义 \ttfamily 来切换到 EU1 编码
\setmonofont{Consolas}
\let\oldttfamily\ttfamily
\def\ttfamily{\oldttfamily\fontencoding{EU1}\selectfont}
% 设置中文字体
\setCJKmainfont{Adobe Kaiti Std}
\begin{document}
Utopia text and  $\sum$  math fonts.
```

^① 这种方法在旧版本的 `fontspec` 宏包中会失效，如果遇到问题，最好更新 \TeX 系统。

汉字楷书与 `{\times Times New Roman}` 字体。

```
\texttt{Consolas 0123}  
\end{document}
```

2-1-42

 **X_YLaTeX** 通常不影响符号字体包的使用，但偶尔会因为字体编码不同而造成问题；
 注意 **L^AT_EX** 在同一时刻只能使用一种字体编码，多种字体编码要手工切换。例如，通常由 **fontspec** 自动加载的 **xunicode** 宏包会把国际音标字体包 **tipa**^[314] 的功能对应到 Unicode 编码字体上，但这会使原来的 `\textipa` 命令改用 **fontspec** 管理的 Unicode 编码字体。因此默认的正文字体 **Latin Modern** 因为符号不全就不能用于国际音标输出，而应该改用包含音标符号的字体，如 **Linux Libertine O**、**Times New Roman** 等。例如：



```
% XeLaTeX 编译  
\documentclass[UTF8]{ctexart}  
% 不需要 tipa 宏包，xunicode 已经实现其功能  
\setmainfont{CMU Serif} % Computer Modern Roman 的 Unicode 版本  
\begin{document}  
\LaTeX{} 读音为 \textipa{["lA:tEx]}。  
\end{document}
```

2-1-43

如果不愿意使用 **OpenType** 或 **TrueType** 格式的 Unicode 编码国际音标字体，仍然可以在 **X_YLaTeX** 下使用 T3 编码的 **tipa** 的功能：

```
% XeLaTeX 编译  
\documentclass[UTF8]{ctexart}  
\usepackage{tipa}% 宏包已经正确加载 fontenc  
% \mytipa 的定义参考原来的 \textipa 的旧定义，手工切换编码  
\newcommand\mytipa[1]{\fontencoding{T3}\selectfont#1}  
\begin{document}  
\LaTeX{} 读音为 \mytipa{["lA:tEx]}。  
\end{document}
```

2-1-44

  有时调用一个字体并没有对应的字体包，就必须通过文档了解字体在 **NFSS** 下的坐标，手工进行选定。在 2.1.3.1 节中我们已经见到了这一点，如使用 **Computer Modern** 的 **Fibonacci** 字体^[169]，就可以直接设置字体族：

```
\fontfamily{cmfib}\selectfont  
Computer Modern Fibonacci Roman
```

Computer Modern Fibonacci
Roman

2-1-45

而如果要 *将 Fibonacci 字体设置为全文的默认字体*，并且让 `\rmfamily` 和 `\textrm` 都指向 *Fibonacci 字体*，就需要在导言区重定义默认的罗马字体族 `\rmdefault`：

```
% 导言区修改全文默认字体 Computer Modern Fibonacci 字体族
\renewcommand{\rmdefault}{cmfib}
```

2-1-46

2

类似地，可以重定义 `\sfdefault` 和 `\ttdefault` 来修改 `\rmfamily` 和 `\ttfamily` 表示的字体族。进一步，如果希望让全文的默认字体是无衬线的字体族，那么还可以重定义 `\familydefault`，如：

```
% 导言区修改全文默认字体为无衬线字体族 phv (Helvetica)
\renewcommand{\familydefault}{\sfdefault}
\renewcommand{\sfdefault}{phv}
```

2-1-47

这正是 *helvet* 宏包的主要工作。当然，在有对应宏包的情况下，还是应该尽量使用宏包提供的功能，这样可能有针对特定字体的更详细的设置。

类似地，较新版本的 *xeCJK* 也提供了 `\CJKrmdefault`、`\CJKsfdefault`、`\CJKttdefault` 和 `\CJKfamilydefault` 命令，其用法与 *NFSS* 中的几个命令类似。在排版中文档时，如果重定义了 `\familydefault` 设置全文为无衬线字体族，那么也应该同时设置中文：

```
\renewcommand{\CJKfamilydefault}{\CJKsfdefault}
```

2-1-48

最后介绍一个有用的宏包 *fonttable*^[290]，可以用它输出字体的符号表，这对于使用 `\symbol` 命令时查找符号代码特别有用。在导言区使用

```
\usepackage{fonttable}
```

之后，就可以使用命令

```
\fonttable{<原始字体名>}
\xfonttable{<编码>}{<族>}{<系列>}{<形状>}
```

得到字体的符号表。例如，我们可以列举出 OT1 编码下 *Latin Modern Roman* 的字体符号表：

```
\xfonttable{OT1}{lmr}{m}{n}
```

2-1-49

	'0	'1	'2	'3	'4	'5	'6	'7	
'00x	Γ ₀	Δ ₁	Θ ₂	Λ ₃	Ξ ₄	Π ₅	Σ ₆	Υ ₇	"0x
'01x	Φ ₈	Ψ ₉	Ω ₁₀	ff ₁₁	fi ₁₂	fl ₁₃	ffi ₁₄	ffl ₁₅	
'02x	ı ₁₆	ı ₁₇	` ₁₈	' ₁₉	˘ ₂₀	˙ ₂₁	˚ ₂₂	˛ ₂₃	"1x
'03x	ˆ ₂₄	ß ₂₅	æ ₂₆	œ ₂₇	ø ₂₈	Æ ₂₉	Œ ₃₀	Ø ₃₁	
'04x	˘ ₃₂	! ₃₃	" ₃₄	# ₃₅	\$ ₃₆	% ₃₇	& ₃₈	' ₃₉	"2x
'05x	(₄₀)) ₄₁	* ₄₂	+ ₄₃	, ₄₄	- ₄₅	. ₄₆	/ ₄₇	
'06x	0 ₄₈	1 ₄₉	2 ₅₀	3 ₅₁	4 ₅₂	5 ₅₃	6 ₅₄	7 ₅₅	"3x
'07x	8 ₅₆	9 ₅₇	: ₅₈	; ₅₉	ı ₆₀	= ₆₁	ı ₆₂	? ₆₃	
'10x	@ ₆₄	A ₆₅	B ₆₆	C ₆₇	D ₆₈	E ₆₉	F ₇₀	G ₇₁	"4x
'11x	H ₇₂	I ₇₃	J ₇₄	K ₇₅	L ₇₆	M ₇₇	N ₇₈	O ₇₉	
'12x	P ₈₀	Q ₈₁	R ₈₂	S ₈₃	T ₈₄	U ₈₅	V ₈₆	W ₈₇	"5x
'13x	X ₈₈	Y ₈₉	Z ₉₀	[₉₁	" ₉₂] ₉₃	^ ₉₄	· ₉₅	
'14x	' ₉₆	a ₉₇	b ₉₈	c ₉₉	d ₁₀₀	e ₁₀₁	f ₁₀₂	g ₁₀₃	"6x
'15x	h ₁₀₄	i ₁₀₅	j ₁₀₆	k ₁₀₇	l ₁₀₈	m ₁₀₉	n ₁₁₀	o ₁₁₁	
'16x	p ₁₁₂	q ₁₁₃	r ₁₁₄	s ₁₁₅	t ₁₁₆	u ₁₁₇	v ₁₁₈	w ₁₁₉	"7x
'17x	x ₁₂₀	y ₁₂₁	z ₁₂₂	- ₁₂₃	— ₁₂₄	" ₁₂₅	~ ₁₂₆	¨ ₁₂₇	
'20x	Ǻ ₁₂₈	Ą ₁₂₉	Ć ₁₃₀	Č ₁₃₁	Ď ₁₃₂	Ě ₁₃₃	Ę ₁₃₄	Ǧ ₁₃₅	"8x
'21x	Ł ₁₃₆	Ł ₁₃₇	Ł ₁₃₈	Ń ₁₃₉	Ń ₁₄₀	Đ ₁₄₁	Ŏ ₁₄₂	Ř ₁₄₃	
'22x	Ř ₁₄₄	Ś ₁₄₅	Š ₁₄₆	Ş ₁₄₇	Ť ₁₄₈	Ţ ₁₄₉	Ů ₁₅₀	Ű ₁₅₁	"9x
'23x	Ÿ ₁₅₂	Ž ₁₅₃	Ž ₁₅₄	Ž ₁₅₅	IJ ₁₅₆	İ ₁₅₇	đ ₁₅₈	§ ₁₅₉	
'24x	ǻ ₁₆₀	ą ₁₆₁	ć ₁₆₂	č ₁₆₃	ď ₁₆₄	ě ₁₆₅	ę ₁₆₆	ǧ ₁₆₇	"Ax
'25x	í ₁₆₈	ı ₁₆₉	ı ₁₇₀	ń ₁₇₁	ň ₁₇₂	ŋ ₁₇₃	ő ₁₇₄	ı ₁₇₅	
'26x	ř ₁₇₆	ś ₁₇₇	š ₁₇₈	ş ₁₇₉	ť ₁₈₀	ţ ₁₈₁	ů ₁₈₂	ű ₁₈₃	"Bx
'27x	ÿ ₁₈₄	ž ₁₈₅	ž ₁₈₆	ž ₁₈₇	ij ₁₈₈	· ₁₈₉	" ₁₉₀	£ ₁₉₁	
'30x	À ₁₉₂	Á ₁₉₃	Â ₁₉₄	Ã ₁₉₅	Ä ₁₉₆	Å ₁₉₇	« ₁₉₈	Ç ₁₉₉	"Cx
'31x	È ₂₀₀	É ₂₀₁	Ê ₂₀₂	Ë ₂₀₃	Ì ₂₀₄	Í ₂₀₅	Î ₂₀₆	Ï ₂₀₇	
'32x	Ð ₂₀₈	Ñ ₂₀₉	Ò ₂₁₀	Ó ₂₁₁	Ô ₂₁₂	Õ ₂₁₃	Ö ₂₁₄	» ₂₁₅	"Dx
'33x	‰ ₂₁₆	Ù ₂₁₇	Ú ₂₁₈	Û ₂₁₉	Ü ₂₂₀	Ý ₂₂₁	Þ ₂₂₂	ŠŠ ₂₂₃	
'34x	à ₂₂₄	á ₂₂₅	â ₂₂₆	ã ₂₂₇	ä ₂₂₈	å ₂₂₉	— ₂₃₀	ç ₂₃₁	"Ex
'35x	è ₂₃₂	é ₂₃₃	ê ₂₃₄	ë ₂₃₅	ì ₂₃₆	í ₂₃₇	î ₂₃₈	ï ₂₃₉	

'36x	ð ²⁴⁰	ñ ²⁴¹	ò ²⁴²	ó ²⁴³	ô ²⁴⁴	õ ²⁴⁵	ö ²⁴⁶	÷ ²⁴⁷	"Fx
'37x	ø ²⁴⁸	ù ²⁴⁹	ú ²⁵⁰	û ²⁵¹	ü ²⁵²	ý ²⁵³	þ ²⁵⁴	„ ²⁵⁵	
	"8	"9	"A	"B	"C	"D	"E	"F	

2.1.3.3 强调文字

现在回到我们最初认识的第一个字体命令：`\emph`。`\emph` 命令表示强调，用于把直立体改为意大利体，把意大利体改为直立体：

2-1-50

```
You \emph{should} use fonts carefully.  
  
\textit{%  
You \emph{should} use fonts carefully.}
```

You *should* use fonts carefully.
You *should use fonts carefully.*

与其他字体命令一样，`\emph` 也有一个声明形式，可以用在分组或环境中：

2-1-51

```
This is {\em emphasized\} text.
```

This is *emphasized* text.

但注意此时要在合适的地方使用倾斜校正命令 `\/`。

在西文中通常使用意大利体表示夹在正文中的强调词句^[35]，这种轻微的字体变化不像粗体那样显眼突兀，因而与正文可以良好的切合。不过，有时仍然使用大写、小型大写或粗体进行更醒目的强调，比如在参考文献的一些项目、书籍索引中的部分页码，或其他类似的内容。假设我们需要用粗体表示比 `\emph` 更强烈的强调，就可以为此定义一个新的 `\Emph` 命令，实际内容就是 `\textbf`：

2-1-52

```
\newcommand\Emph{\textbf}  
This is \Emph{emphasized} text.
```

This is **emphasized** text.

下画线是另一种颇具手稿风格的强调方式， \LaTeX 命令 `\underline` 可以给文字或公式加下画线：

2-1-53

```
\underline{Emphasized} text and  
\underline{another}.
```

Emphasized text and another.

不过 `\underline` 的一个很大的缺点是下画线的部分不能换行，如果仔细看上面的例子还会发现下画线与文字的距离不整齐。`ulem` 宏包^[17] 的 `\ulem` 命令解决了这些问题，使用并且把默认的 `\emph` 命令也改为使用下画线的方式：

```
% 导言区用 \usepackage{ulem}
\uline{Emphasized} text and \uline{another}.

A \emph{very very very very very very very
very very very very very} long sentence.
```

Emphasized text and another.
A very very very very very very
very very very very very very
long sentence.

2-1-54

如果不希望用下画线代替标准的 `\emph` 命令定义，可以给 `ulem` 宏包加 `normalem` 参数，或使用 `\normalem` 和 `\ULforem` 命令切换两种强调。

除了下画线，`ulem` 宏包也提供了其他一些修饰文字的命令：

```
\uuline{urgent}\quad \uwave{boat}\quad
\sout{wrong}\quad \xout{removed}\quad
\dashuline{dashing}\quad \dotuline{dotty}
```

urgent boat wrong
~~removed~~ ~~dashing~~ ~~dotty~~

2-1-55

`CJKfntef` 宏包^[244] 对汉字也提供了类似的功能，同时也进行了一些扩充^①：

```
\CJKKunderdot{汉字，下加点} \
\CJKKunderline{汉字，下画线} \
\CJKKunderdblline{汉字，下画线} \
\CJKKunderwave{汉字，下画线} \
\CJKKsout{汉字，删除线} \
\CJKKxout{汉字，删除线}
```

汉字，下加点
汉字，下画线
汉字，下画线
汉字，下画线
汉字，删除线
汉字，删除线

2-1-56

此外，`CJKfntef` 还提供了指定宽度，让汉字分散对齐的环境：

```
\begin{CJKfilltwosides}{5cm}
汉字，分散对齐
\end{CJKfilltwosides}
```

汉 字， 分 散 对 齐

2-1-57

使用 `CJKfntef` 宏包后 `\emph` 命令也被改为下画线的格式，同样可以用 `\normalem` 改回原来的意大利体定义。在 `ctex` 宏包及文档类中，可以使用 `fntef` 选项调用 `CJKfntef`，此时 `\emph` 的定义不会被改变为下画线格式。同时也可以使用 `\CTEXunderline` 等以 `\CTEX` 开头的命令代替以 `\CJK` 开头的命令，如：

^① 新版本的 `CJKfntef` 还提供了自定义符号和自定义距离的功能，参见其源代码。

2-1-58

```
\emph{汉字，强调}\  
\CTEXunderdot{汉字，加点}
```

汉字，强调
汉字，加点



练习

2



2.3 soul 宏包^[81] 提供了 `\hl` 命令实现对参数的强调，当没有彩色时它与 `ulem` 一样用下划线强调，而有彩色支持时它的效果则是荧光高亮显示。但是，`soul` 宏包的实现机制较为脆弱，与现在各种中文支持方式都无法一起正常使用。查看文档 Arseneau [17]，试利用 `ulem` 宏包实现用黄色高亮强调的命令 `\hl`。（提示：利用下划线）



从 METAFONT 到 OpenType

使用各种字体是任何排版软件都应具备的重要功能。今天 $\text{T}_{\text{E}}\text{X}$ 引擎可以使用很多格式的字体，如 METAFONT, PostScript, TrueType, OpenType 等，这些又都是些什么呢？

高德纳教授最初设计 $\text{T}_{\text{E}}\text{X}$ 系统时，也同时为 $\text{T}_{\text{E}}\text{X}$ 设计了配套的字体格式和字体描述语言，这就是 METAFONT。 $\text{T}_{\text{E}}\text{X}$ 默认 Computer Modern 系列字体就是使用 METAFONT 描述的。METAFONT，顾名思义，就是“元字体”，它是一门通过曲线路径描述字体信息的宏语言^[125]，把字符用坐标绘图的方式“画出来”。例如 `cryst` 字体中的一个箭头符号 \rightarrow 就是用下面的代码画出来的：

```
beginchar(9,120u#,68u#,0);  
"2, 0 Grad";  
z1=(0u,37u); z2=(88u,37u); z3=(88u,31u); z4=(0,31u);  
z5=(116u,34u); z6=(73u,17u); z7=(72u,19u);  
z8=(84u,34u); z9=(72u,49u); z10=(73u,51u);  
fill z10{dir -28}..{dir -15}z5..z5{dir -165}..{dir -152}z6  
--z7--z8--z9--cycle;  
fill z1--z2--z3--z4--cycle;  
labels(range 1 thru 6);  
endchar;
```

METAFONT 使用坐标作图的命令描述字符的曲线，然后把曲线计算为三次 Bézier 样条，再转换为光栅格式的通用字体文件（generic font）.gf，同时生成 $\text{T}_{\text{E}}\text{X}$ 字体度量文件（ $\text{T}_{\text{E}}\text{X}$ font metric）.tfm。 $\text{T}_{\text{E}}\text{X}$ 引擎从 .tfm 文件中读

第2章 组织你的文本

81

入字符尺寸信息，生成 DVI 文件；驱动程序在打印输出或屏幕显示时从 .gf 文件（一般转换为压缩的 .pk 文件）中读入字符图形信息，生成最终打印或显示的结果。尽管 METAFONT 在描述字符时使用的是曲线方式，但生成的结果却是光栅点阵形式的，在字体放大和使用电子文档时质量不够好；而且这种用数学坐标的方式描述符号很不直观，因此，现在 METAFONT 格式的字体使用得越来越少了。倒是 METAFONT 这种绘图方式，催生了绘图语言 METAPOST 的产生。

Adobe 公司推出 PostScript 时，也定义了 PostScript 字体的格式，Type 1 和 Type 3 是其中的两个类型^[4]。很多高质量的商业字体都是 PostScript Type 1 格式的，它支持字体信息的微调（hinting），后来当 Adobe 开放 Type 1 格式的使用后，T_EX 中原来用 METAFONT 描述的字体也都纷纷转换为 Type 1 字体了。Type 3 格式不支持微调，不过也可以表示点阵字体，当输出 PostScript 或 PDF 文件时，METAFONT 生成的 PK 字体就被转换为 Type 3 格式。Type 1 字体一般使用 .pfb 后缀作为字体扩展名，用 .afm 作为字体度量文件的扩展名，不过 T_EX 使用时仍然要使用转换得到的 .tfm 度量文件。目前输出 PostScript 或 PDF 文件的 T_EX 驱动和引擎都支持 PostScript 字体。

TrueType 是苹果公司与 Adobe 竞争而于 1991 年发布的字体格式，微软得到授权后，它已成为 Windows 操作系统中最为常见的字体格式；OpenType 是微软公司和 Adobe 公司于 1996 年发布的字体格式，它继承了 PostScript 字体的许多特性。TrueType 字体以 .ttf 和 .ttc 为后缀，OpenType 字体以 .otf 和 .ttf 为后缀。新的驱动和引擎 dvipdfmx, pdfT_EX, X_YT_EX 和 LuaT_EX 都支持 TrueType 和 OpenType 字体格式，不过 dvipdfmx 和 pdfT_EX 使用 TrueType 和 OpenType 时仍然需要事先生成 .tfm 文件并进行配置，功能上也有一些限制，而 X_YT_EX 和 LuaT_EX 则支持直接读取系统字体及全部 OpenType 字体特性。

2.1.4 字号与行距

字号指文字的大小，它原本是字体的性质，也被 NFSS 作为字体的坐标之一。例如 Computer Modern Roman 字体族的中等直立体就有 5, 6, 7, 8, 9, 10, 12, 17 点共 8 种大小的不同字号，其中点（point, pt）是长度单位，为 1/72.27 英寸（inch）。不过当可缩放的矢量字体流行起来以后，字体只有一款，通过缩放达到不同的尺寸，字号也就经常作为独立于字体的单独性质了。

基本的 L^AT_EX 提供了 10 个简单的声明式命令调整文字的大小：

<code>\tiny</code>	tiny	<code>\Large</code>	larger
<code>\scriptsize</code>	script size	<code>\LARGE</code>	even larger
<code>\footnotesize</code>	footnote size		huge
<code>\small</code>	small	<code>\huge</code>	largest
<code>\normalsize</code>	normal size		
<code>\large</code>	large	<code>\Huge</code>	

例如：

2-1-59

The text can be `{\Large larger}`.

The text can be **larger**.

字号命令表示的具体尺寸随着所使用的文档类和大小选项的不同而不同（见表 2.7）。在标准 L^AT_EX 文档类 `article`, `report` 和 `book` 中，可以设置文档类选项 `10pt`, `11pt` 和 `12pt`，全局地设置文档的字号，默认为 `10pt`，即 `\normalsize` 的大小为 `10pt`，这个数值表示字体中一个 `\quad` 的长度，通常也就是整个符号所占盒子的高度（对汉字来说，一般也相当于汉字宽度）。

上述字号命令除了会修改文字大小外，同时对 `\normalsize`、`\small` 和 `\footnotesize` 也会修改文字的基本行距（默认是文字大小的 1.2 倍）、列表环境的各种间距（参见 2.2.3.3 节）、显示数学公式的垂直间距（参见 4.5.3 节）。因此这些命令比原始的 `\fontsize`、`\zihao` 更方便使用。

中文字号可以使用同样的命令设置（字号命令不区分中西文）。不过为了明确字号的具体大小，也可以使用 `ctex` 宏包或 `ctexart` 等文档类提供的 `\zihao` 命令设置。`\zihao` 命令带一个参数，表示中文的字号，它影响分组或环境内命令后面所有的文字。`\zihao` 命令可用的参数见表 2.6。

类似地，`ctexart`, `ctexrep` 和 `ctexbook` 文档类也提供了两个选项 `c5size` 和 `cs4size` 影响全文的文字大小和 `\normalsize` 等命令的意义（见表 2.7）。`c5size` 和 `cs4size` 分别表示 `\normalsize` 为五号字和小四号字，默认为 `c5size`。

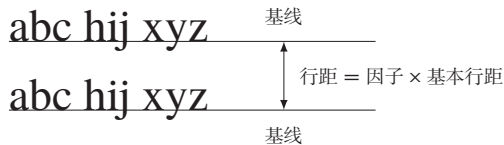
L^AT_EX 中的行距是与字号直接相关的。在设置字号时，同时也就设置了基本行距为文字大小的 1.2 倍。行距一词指的是一行文字的基线（base line）到下一行文字的基线的距离。

表 2.6 中文字号

命令	大小 (bp)	意义
<code>\zihao{0}</code>	42	初号
<code>\zihao{-0}</code>	36	小初号
<code>\zihao{1}</code>	26	一号
<code>\zihao{-1}</code>	24	小一号
<code>\zihao{2}</code>	22	二号
<code>\zihao{-2}</code>	18	小二号
<code>\zihao{3}</code>	16	三号
<code>\zihao{-3}</code>	15	小三号
<code>\zihao{4}</code>	14	四号
<code>\zihao{-4}</code>	12	小四号
<code>\zihao{5}</code>	10.5	五号
<code>\zihao{-5}</code>	9	小五号
<code>\zihao{6}</code>	7.5	六号
<code>\zihao{-6}</code>	6.5	小六号
<code>\zihao{7}</code>	5.5	七号
<code>\zihao{8}</code>	5	八号

表 2.7 不同文档类选项下的字号命令。这里给出不同字号命令对应的字号尺寸。正文的默认字号是 `\normalfont`。西文文档类默认选项是 10pt，字体尺寸以 pt 为单位；`ctex` 文档类的默认选项是 `c5size`，使用中文字号


命令	10pt	11pt	12pt	c5size	cs4size
<code>\tiny</code>	5	6	6	七	小六
<code>\scriptsize</code>	7	8	8	小六	六
<code>\footnotesize</code>	8	9	10	六	小五
<code>\small</code>	9	10	10.95	小五	五
<code>\normalsize</code>	10	10.95	12	五	小四
<code>\large</code>	12	12	14.4	小四	小三
<code>\Large</code>	14.4	14.4	17.28	小三	小二
<code>\LARGE</code>	17.28	17.28	20.74	小二	二
<code>\huge</code>	20.74	20.74	24.88	二	小一
<code>\Huge</code>	24.88	24.88	24.88	一	一



可以使用命令

`\linespread{(因子)}`

来设置实际行距为基本行距的倍数^①。与许多其他底层字体命令一样，它也在 `\selectfont` 命令（或等效的字体变更）后生效。对 `article` 等标准文档类来说，默认值为 1，即行距是字号大小的 1.2 倍；而对 `ctexart` 等中文文档类来说，默认值为 1.3，即行距是字号大小的 1.56 倍。

 `setspace` 宏包^[265] 提供了一组命令和环境，用于在修改行距因子的同时保证数学公式、浮动体、脚注间距的值也相对合理。基本的命令是 `\setstretch{(因子)}`，大致相当于使用了 `\linespread` 后再加 `\selectfont` 生效。除此之外，`setspace` 也提供了几个环境用来产生宏包定义的单倍、一倍半和双倍行距，不过要注意其“倍数”的意义并不是 `\setstretch` 的行距因子，而是指行距相比字号尺寸的倍数，这也与其他一些软件（如微软的字处理器 Word[®]）不同，见表 2.8。

^① 一些较早的书籍会使用重定义 `\baselinestretch` 命令的方式设置行距因子，这种方法并非 `LaTeX 2ε` 推荐的方式，我们这里使用更容易掌握的方式。

^② Word 中段落设置 n 倍行距的概念与标准 `LaTeX` 的 `\linespread` 是一样的，也是加之于基本行距的因子。

表 2.8 setspace 提供的命令和环境

命令形式	环境形式	意义
<code>\setstretch</code>	<code>spacing</code>	与 <code>\linespread</code> 功能相当
<code>\singlespacing</code>	<code>singlespace</code>	正常行距
<code>\onehalfspacing</code>	<code>onehalfspace</code>	基线间距是字号大小的 1.5 倍
<code>\doublespacing</code>	<code>doublespace</code>	基线间距是字号大小的 2 倍

可以使用 2.1.3.1 节的 `\fontsize` 直接指定文字的字号和基本行距，注意这里字号和基本行距是 NFSS 坐标中的一部分，只有在使用了 `\selectfont` 后才能生效。

除了设置两行基线间的距离， \TeX 还提供了两个基本尺寸，用来设置两行文字内容间的距离。 \TeX 中行距默认由基线间的距离 `\baselineskip` 控制（ \LaTeX 的 `\fontsize` 和 `\linespread` 间接控制 `\baselineskip`）。`\lineskiplimit` 是一个界限值，当前一行盒子的底部与后一行盒子的顶部距离小于这个界限时，行距改由 `\lineskip` 控制，它将设置行距，使得前一行底到后一行顶的距离等于 `\lineskip`。在文档中设置合适的 `\lineskiplimit` 和 `\lineskip` 值可以避免两行因包含太高的内容（如分式 $\frac{1}{2}$ ）而距离过紧，如本书的设置为：

```
\setlength{\lineskiplimit}{2.625bp} % 五号字 1/4 字高
\setlength{\lineskip}{2.625bp}
```

2-1-60

即要求两行间至少有 $\frac{1}{4}$ 五号字字号高度的距离。

2.1.5 水平间距与盒子

2.1.5.1 水平间距

现在我们来继续 2.1.1.3 节有关空格的话题，来说说如何正确地产生和使用水平间距。

说到长度，就不能不说说单位。在 \TeX 中，可以使用长度单位有以下几种：

- pt point 点（欧美传统排版的长度单位，有时叫做“磅”）
- pc pica（ $1\text{ pc} = 12\text{ pt}$ ，相当于四号字大小）
- in inch 英寸（ $1\text{ in} = 72.27\text{ pt}$ ）

- bp big point 大点（在 PostScript 等其他电子排版领域的 point 都指大点，
1 in = 72 bp）
- cm centimeter 厘米（2.54 cm = 1 in）
- mm millimeter 毫米（10 mm = 1 cm）
- dd didot point（欧洲大陆使用，1157 dd = 1238 pt）
- cc cicero（欧洲大陆使用，pica 的对应物，1 cc = 12 dd）
- sp scaled point（ \TeX 中最小的长度单位，所有长度都是它的倍数，
65 536 sp = 1 pt）
- em 全身（字号对应的长度，等于一个 $\backslash\text{quad}$ 的长度，也称为“全方”。
本义是大写字母 M 的宽度）
- ex x-height（与字号相关，由字体定义。本义是小写字母 x 的高度）

一个长度必须数字和单位齐全，正确的长度如下所示：

2cm 1.5pc -.1cm + 72.dd 1234567sp

在正文中可以使用下面的命令表示不可换行的水平间距：

$\backslash\text{thinspace}$ 或 $\backslash,$	0.1667 em	$\rightarrow\! \! \leftarrow$
$\backslash\text{negthinspace}$ 或 $\backslash!$	-0.1667 em	$\rightarrow\! \! \leftarrow$
$\backslash\text{enspace}$	0.5 em	$\rightarrow\! \! \leftarrow$
$\backslash\text{nobreakspace}$ 或 \sim	空格	$\rightarrow\! \! \leftarrow$

可以使用下面的命令表示可以换行的水平间距：

$\backslash\text{quad}$	1 em	$\rightarrow\! \! \leftarrow$
$\backslash\text{qqquad}$	2 em	$\rightarrow\! \! \leftarrow$
$\backslash\text{enskip}$	0.5 em	$\rightarrow\! \! \leftarrow$
$\backslash_$	空格	$\rightarrow\! \! \leftarrow$

使用水平间距的命令要注意适用，例如 $\backslash,$ 是不可断行的，因而就不适用于分隔很长的内容，但用来代替逗号给长数字分段就很合适^①：

1 $\backslash,$ 234 $\backslash,$ 567 $\backslash,$ 890

1 234 567 890

2-1-61

负距离 $\backslash!$ 则可以用来细调符号距离，或拼接两个符号，如把两个“剑号”拼起来：

^① 当然，对于输出数字如果要更多的效果，还可以使用 `fancynum`, `numprint` 等宏包。使用这类专门的宏包通常总会更方便，效果也更好。

第 2 章 组织你的文本

87

```
\newcommand\dbldag{\dag\!\dag}
\dbldag\ versus \dag\dag
```

†† versus ††

2-1-62

而正如前面已经多次见到的，分隔词组经常使用可断行的而距离也比较大的 `\quad` 和 `\qquad`。

当上面的命令中没有合适的距离时，可以用 `\hspace{距离}` 命令来产生指定的水平间距（这里 `\`，则用来分隔数字和单位）：

```
Space\hspace{1cm}1\,cm
```

Space 1 cm


2-1-63

`\hspace` 命令产生的距离是可断行的。`\hspace` 的作用是分隔左右的内容，在某些只有一边有内容的地方（如强制断行的行首）， \LaTeX 会忽略 `\hspace` 产生的距离，此时可以用带星号的命令 `\hspace*{距离}` 阻止距离被忽略：

```
text\\
\hspace{1cm}text\\
\hspace*{1cm}text
```

text
text
text

2-1-64

 `\hspace` 可以产生随内容可伸缩的距离，即所谓胶（`glue`）或者橡皮长度（`rubber length`，又称弹性长度）。橡皮长度的语法是：


（普通长度）**plus**（可伸长长度）**minus**（可缩短长度）

单词间的空格、标点后的距离都是橡皮长度，这样才能保证在分行时行末的对齐，因此在定义经常出现的距离时应该使用橡皮长度。如：

```
\newcommand\test{longggggggg%
\hspace{2em plus 1em minus 0.25em}}
\test\test\test\test\test\test\test\test
```

longggggggg longggggggg
longggggggg longggggggg
longggggggg longggggggg
longggggggg longggggggg


2-1-65

 有一种特殊的橡皮长度 `\fill`，它可以从零开始无限延伸，此时橡皮长度就真的像是一个弹簧，可以用它来把几个内容均匀排列在一行之中：

```
left\hspace{\fill}middle%
\hfill right
```

left middle right

2-1-66


 `\hfill` 命令是 `\hspace{\fill}` 的简写，还可以使用 `\stretch{⟨倍数⟩}` 产生具有指定“弹力”的橡皮长度，如 `\stretch{2}` 就相当于两倍的 `\fill`：

```
left\hspace{\stretch{2}}$2/3$%  
\hspace{\fill}right
```

left 2/3 right

2-1-67


2

 `\hrulefill` 和 `\dotfill` 与 `\hfill` 功能类似，只是中间的内容使用横线或圆点填充：

```
left\hrulefill middle\dotfill right
```

left_____middle right

2-1-68

 \LaTeX 预定义了一些长度变量控制排版的参数，可以通过 `\setlength` 命令来设置。例如段前的缩进：

```
\setlength{\parindent}{8em}  
Paragraph indent can be very wide in \LaTeX.
```

Paragraph
indent can be very wide in
 \LaTeX .

2-1-69


我们在后面的章节会陆续见到这类长度变量。还可以用 `\addtolength` 命令在长度变量上做累加，如：

```
Para\par  
\addtolength{\parindent}{2em}Para\par  
\addtolength{\parindent}{2em}Para\par
```

Para
Para
Para

2-1-70

长度变量的改变在当前分组或环境内起效，因此不必担心在一个环境内的设置会影响到外面的内容，也可以使用分组使变量的改变局部化。

 可以用 `\newlength` 命令定义自己的长度变量，这样就可以在不同的地方反复使用：

```
\newlength\mylen \setlength{\mylen}{2em}
```

这在自定义的一些环境中是十分有用的。

2.1.5.2 盒子

盒子 (box) 是 \TeX 中的基本处理单位，一个字符、一行文字、一个页面、一张表格在 \TeX 中都是一个盒子。回到活字印刷时代或许有助于理解盒子的概念：一个活字

第2章 组织你的文本

89

就表示一个字符，一行活字排好就用钢条分隔固定成为一行，一整页排完也固定在金属框内。 \TeX 也是这样，组字成行，组行为页，小盒子用胶粘连成为大盒子，逐步构成完整的篇章。

\TeX 可以处在不同的工作模式下，在不同的工作模式下产生不同的盒子。最基本的模式的水平模式（如在组字成行的时候）和垂直模式（如在组行成页的时候），水平模式下把小盒子水平排列组成大盒子，垂直模式下把小盒子垂直排列组成大盒子；此外还有更为复杂的数学模式，此时小盒子会构成复杂的数学结构。通常 \TeX 根据内容自动切换不同的模式，完成这些复杂的工作；不过也可以使用命令进入指定的模式生成盒子，我们现在只看最简单的水平模式下的盒子。

最简单的命令是 `\mbox{内容}`，它产生一个盒子，内容以左右模式排列。可以用它表示不允许断行的内容，如果不在行末看不出它与其他内容的区别：

```
\mbox{cannot be broken}
```

cannot be broken

2-1-71

`\makebox` 与 `\mbox` 类似，但可以带两个可选参数，指定盒子的宽度和对齐方式：

```
\makebox[宽度][位置]{内容}
```

对齐参数可取 `c`（中）、`l`（左）、`r`（右）、`s`（分散），默认居中。

例如：

```
\makebox[1em]{\textbullet}text \\  
\makebox[5cm][s]{some stretched text}
```

• text
some stretched text

2-1-72

甚至可以使用 `\makebox` 产生宽度为 0 的盒子，产生重叠（`overlap`）的效果：

```
% word 左侧与盒子基点对齐  
\makebox[0pt][l]{word}文字
```

word

不过， \LaTeX 已经提供了两个命令来专门生成重叠的效果，即 `\llap` 和 `\rlap`。这两个命令分别把参数中的内容向当前位置的左侧和右侧重叠：

```
语言文字\llap{word}\  
\rlap{word}语言文字
```

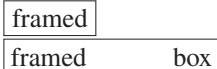
语言word
word文字

2-1-73

命令 `\fbox` 和 `\framebox` 产生带边框的盒子，语法与 `\mbox` 和 `\makebox` 类似：

2-1-74

```
\fbox{framed} \\  
\framebox[3cm][s]{framed box}
```



边框与内容的距离由长度变量 `\fboxsep` 控制（默认为 3 pt）：

2-1-75

```
\setlength{\fboxsep}{0pt} \fbox{tight}  
\setlength{\fboxsep}{1em} \fbox{loose}
```



边框线的粗细则由长度变量 `\fboxrule` 控制（默认为 0.4 pt）。



可以用 `\newsavebox{命令}` 声明盒子变量，用

```
\sbox{命令}{(内容)}  
\savebox{命令}[(宽度)][(对齐)]{(内容)}  
\begin{lrbox}{(命令)}(内容)\end{lrbox}
```

给盒子变量赋值，在文章中使用 `\usebox{内容}` 反复使用：

2-1-76

```
\newsavebox{\mybox} % 通常在导言区定义  
\sbox{\mybox}{test text}  
\usebox{\mybox} \fbox{\usebox{\mybox}}
```



`\savebox` 与 `\sbox` 的区别类似 `\makebox` 与 `\mbox` 的区别，基本功能相同，只是增加了宽度和对齐的选项。



盒子变量中一般保存比较复杂的内容。特别是可以使用 `lrbox` 环境保存抄录环境（参见 2.2.5 节）等难以放在其他命令参数中的内容作进一步的处理^①（见 124 页例 2-2-73）：

2-1-77

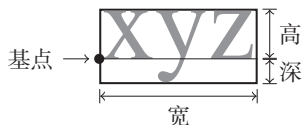
```
\newsavebox{\verbbox} % 通常在导言区定义  
\begin{lrbox}{\verbbox}  
\verb|#$%^&{|  
\end{lrbox}  
\usebox{\verbbox} \fbox{\usebox{\verbbox}}
```



^① 不过，这个功能可以直接使用 `fancyvrb` 宏包的 `\SaveVerb` 和 `\UseVerb` 命令。



\TeX 中的每个盒子都有其宽度 (width)、高度 (height) 和深度 (depth)，高度和深度以基点 (base) 为界：



可以用

```
\settowidth{(长度变量)}{(内容)}  
\settoheight{(长度变量)}{(内容)}  
\settodepth{(长度变量)}{(内容)}
```

把内容的宽度、高度或深度赋值给长度变量。也可以用

```
\wd{(盒子变量)}          \ht{(盒子变量)}          \dp{(盒子变量)}
```

分别得到盒子的宽度、高度和深度。



除此之外，在 \makebox 、 \framebox 等盒子命令的参数中，也可以使用 \width 、 \height 、 \depth 、 \totalheight 来分别表示盒子内容的自然宽度、高度、深度，以及高度和深度之和。例如，下面的例子产生一个带边框的盒子，其总宽度恰好是文字自然宽度的 2 倍：

```
\framebox[2\width]{带边框}
```

带边框

2-1-78

2.2 段落与文本环境

2.2.1 正文段落

我们已经知道， \LaTeX 使用空行表示分段，在自定义命令中，也常用 \par 命令分段。自然段是 \LaTeX 最基本的正文分划方式。

在 2.1.5 节已经看到，每个自然段在第一行有一个固定的缩进，可以由长度变量 \parindent 控制。在西文标准文档类（如 article ）中，每个章节的第一段是不缩进的，而中文文档类（如 ctexart ）则每段缩进，并自动设置段落缩进为两个汉字宽。如果要在某一段开头临时禁用缩进，可以在段前使用 \noindent 命令；而如果要在本来没有缩

进的地方使用缩进，可以用 `\indent` 命令产生一个长为 `\parindent` 的缩进。在西文文档中，可以使用 `indentfirst` 宏包启用章节首段的缩进。

除了段落首行缩进，另一个关于分段的重要参数是段与段之间的垂直距离，这由变量 `\parskip` 控制。`\parskip` 的默认值是橡皮长度 `0pt plus 1pt`，在中文排版中常常使用

2

2-2-1

```
\setlength{\parskip}{0pt}
```

把段间距定义为固定长度，禁止段落间距离伸长。当然有时为了文字清晰，也常设置一个较大的 `\parskip`。

段落最明显的属性是对齐方式。`LaTeX` 的段落默认是两端均匀对齐的，也可以改为左对齐、右对齐或居中格式。`\raggedright` 命令设置段落左对齐（`ragged right` 意为右边界参差不齐），这在双栏文档一行非常窄的时候特别有用，此时强求均匀对齐会使单词间的距离过大，十分难看，如下所示：

```
English words like 'technology' stem from a  
Greek root beginning with the letters τεχ\ldots
```

English words
like 'tech-
nology' stem
from a Greek
root beginning
with the letters
τεχ...

而如果使用左对齐就可以解决这个问题：

```
\raggedright  
English words like 'technology' stem from a  
Greek root beginning with the letters τεχ\ldots
```

2-2-2

English words
like
'technology'
stem from a
Greek root
beginning
with the letters
τεχ...

类似地，右对齐使用 `\raggedleft` 命令，居中使用 `\centering` 命令。右对齐常用于排版签名、日期、格言警句等；居中的段落则有强调的意味。

第2章 组织你的文本

93

\LaTeX 提供了三个环境来排版不同对齐方式的文字：`flushleft` 环境左对齐、`flushright` 环境右对齐和 `center` 环境居中。这几个环境会在段落前后增加一小段垂直间距以示强调，对于少量的段落，它们不会影响前后的文字，因此比 `\raggedright` 等命令更常用一些。不过如果不想额外的垂直间距，则不应该使用它们，如：

```
\begin{center}
居中
\end{center}
```

居中

2-2-3

2

在默认的段落设置下， \LaTeX 可能会在单词的两个音节中间断行，并在前一行末尾加上连字符，即所谓断词（用连字号连接，`hyphenation`）。例如：

This is a hyphen-
ation test.

\TeX 的断词算法通常工作得很好，不需要人为干预，不过仍然可能会有一些特殊的单词是 \TeX 不能正确处理的，此时可以在单词中使用 `\-` 命令告诉 \LaTeX 可能的断点，如 `man\-\u\-\script`。还可以使用 `\hyphenation` 命令在导言区全局地设置断点列表，如：

```
\hyphenation{man-u-script com-pu-ter gym-na-sium}
```

2-2-4

断词只在均匀对齐的段落中起作用，左对齐的段落就没有断词。使用 `\sloppy` 命令可以允许段落中更大的空格，从而禁用断词功能；与之相对的命令是 `\fussy`，让段落恢复默认的较严格的间距。更多地是使用等效的 `sloppypar` 环境，把允许更宽松间距的文本段放在环境中。以 `none` 选项使用 `hyphenat` 宏包^[284] 可以更好地禁用断词，宏包也提供了有关打字机字体的断词功能。`ragged2e` 宏包^[230] 则可以在左对齐（`\RaggedRight`）、右对齐（`\RaggedLeft`）或居中（`\Centering`）的段落中使用断词，这样可以得到更合理的段落：

```
% 导言区 \usepackage{ragged2e}
\RaggedRight
English words like 'technology' stem from a
Greek root beginning with the letters τεχ\ldots
```

English words
like 'tech-
nology' stem
from a Greek
root beginning
with the letters
τεχ...

2-2-5

ragged2e 宏包还提供了 `\justifying` 命令回到均匀对齐的段落，以及对应的 `Center`, `FlushLeft`, `FlushRight`, `justify` 环境，作为标准 \LaTeX 命令和环境的补充。

对付西文文本参差不齐的小短行，一个有力的工具是 `microtype` 宏包^[227]。这个宏包利用 `pdf \TeX` 的功能，可以通过调整单词内的字母间距改善 \TeX 的分行效果。当排版西文文档时可以总打开此宏包的功能，不过当前版本暂不支持 `X \TeX` ^①。

可以使用长度变量 `\leftskip` 和 `\rightskip` 来控制段落的宽度，如果是局部修改，注意把整个段落放在一个分组里面，或定义为单独的环境。例如：

2-2-6

```
\setlength{\leftskip}{4em}
\setlength{\rightskip}{1em}
These parameters tell  $\text{\TeX}$  how much glue
to place at the left and at the right end
of each line of the current paragraph.
```

These parameters tell \TeX how much glue to place at the left and at the right end of each line of the current paragraph.

上面介绍的只是基本的段落形状。使用原始 \TeX 命令 `\parshape`（参见 Abrams et al. [1]、Knuth [126]）， \LaTeX 还可以排版出更为特殊形状的段落，但 `\parshape` 命令使用起来比较复杂，这里不作介绍。较为常用的一种特殊段落是“悬挂缩进”，它可以由命令 `\hangafter` 和 `\hangindent` 控制，例如：

2-2-7

```
\hangindent=5pc \hangafter=-2
These two parameters jointly specify
‘‘hanging indentation’’ for a paragraph.
The hanging indentation indicates to  $\text{\TeX}$ 
that certain lines of the paragraph should
be indented and the remaining lines should
have their normal width.
```



These two parameters jointly specify “hanging indentation” for a paragraph. The hanging indentation indicates to \TeX that certain lines of the paragraph should be indented and the remaining lines should have their normal width.

① `microtype` 宏包 v2.5 以后的版本支持 `X \TeX` 引擎，但目前没有正式发布，需要用户从 <http://t1contrib.metatex.org/> 等网站单独安装测试版本。

第2章 组织你的文本

95



正的 `\hangindent` 作用于段落左侧, 负的 `\hangindent` 作用于段落右侧; 正的 `\hangafter` 作用于段落的 n 行之后, 负的 `\hangafter` 作用于段落的前 n 行。`\hangindent` 和 `\hangafter` 的设置只对当前段起作用。

  `lettrine` 宏包^[79] 利用特殊的段落形状产生首字下沉的效果, 例如:

```
% 导言区 \usepackage{lettrine}
\lettrine{T}{he} \TeX{} in \LaTeX{} refers
to Donald Knuth's \TeX{} typesetting system.
The \LaTeX{} program is a special version of
\TeX{} that understands \LaTeX{} commands.
```

THE \TeX in \LaTeX refers to Donald Knuth's \TeX typesetting system. The \LaTeX program is a special version of \TeX that understands \LaTeX commands.

2-2-8

  `shapepar` 宏包^[15] 提供了 `\parshape` 命令的一个较为方便的语法接口, 特别是定义了一些预定义的形状, 可以方便地排出一些有趣的效果:

```
% 导言区 \usepackage{shapepar}
\heartpar{%
  绿草苍苍, 白雾茫茫, 有位佳人, 在水一方。
  绿草萋萋, 白雾迷离, 有位佳人, 靠水而居。
  我愿逆流而上, 依偎在她身旁。无奈前有险滩, 道路又远又长。
  我愿顺流而下, 找寻她的方向。却见依稀仿佛, 她在水的中央。
  我愿逆流而上, 与她轻言细语。无奈前有险滩, 道路曲折无已。
  我愿顺流而下, 找寻她的足迹。却见仿佛依稀, 她在水中伫立。}
```

2-2-9

绿草苍苍, 白雾茫茫, 有位佳人, 在水一方。绿草萋萋, 白雾迷离, 有位佳人, 靠水而居。我愿逆流而上, 依偎在她身旁。无奈前有险滩, 道路又远又长。我愿顺流而下, 找寻她的方向。却见依稀仿佛, 她在水的中央。我愿逆流而上, 与她轻言细语。无奈前有险滩, 道路曲折无已。我愿顺流而下, 找寻她的足迹。却见仿佛依稀, 她在水中伫立。



2.2.2 文本环境

有几种常用的特殊文本段落类型，在 \LaTeX 中以文本环境的形式给出，它们是引用环境、诗歌环境和摘要环境。

引用环境有两种，分别是 `quote` 环境和 `quotation` 环境。`quote` 环境在段前没有首行的缩进，每段话的左右边距比正文大一些，通常用于小段内容的引用：

2-2-10

```
前文……  
\begin{quote}  
学而时习之，不亦说乎？  
有朋自远方来，不亦乐乎？  
\end{quote}  
后文……
```

前文……

学而时习之，不亦说乎？有朋自远方来，不亦乐乎？

后文……

而 `quotation` 环境则在每段前有首行缩进，因而适用于多段的文字引用：

2-2-11

```
前文……  
\begin{quotation}  
学而时习之，不亦说乎？  
有朋自远方来，不亦乐乎？  
  
默而识之，学而不厌，诲人不倦，何有于我哉？  
\end{quotation}  
后文……
```

前文……

学而时习之，不亦说乎？有朋自远方来，不亦乐乎？

默而识之，学而不厌，诲人不倦，何有于我哉？

后文……

`verse` 环境用来排版诗歌韵文：

2-2-12

```
\begin{verse}  
在一段内使用 \verb=\!= 换行，\\  
分段仍用空行。  
  
过长的长会在折行时悬挂缩进，  
就像现在这一句。  
\end{verse}
```

在一段内使用 `\\` 换行，分段仍用空行。

过长的长会在折行时悬挂缩进，就像现在这一句。

摘要环境 `abstract` 是 `article` 和 `report` 文档类（包括中文的 `ctexart` 和 `ctexrep`）定义的，它产生一个类似 `quotation` 的小号字环境，并增加标题：

```
\begin{abstract}
本书讲解 \LaTeX{} 的使用。
\end{abstract}
```

摘要

本书讲解 \LaTeX
的使用。

2-2-13

2

摘要的标题由 `\abstractname` 定义，英文默认是“Abstract”，中文是“摘要”。可以通过重定义 `\abstractname` 来设置，在 `ctex` 文档类中也可以使用 `\CTEXoptions` 设置^[58]，如：

```
\CTEXoptions[abstractname={摘\quad 要}]
```

2-2-14

2.2.3 列表环境

2.2.3.1 基本列表环境

列表是常用的本文格式。 \LaTeX 标准文档类提供了三种列表环境：编号的 `enumerate` 环境、不编号的 `itemize` 环境和使用关键字的 `description` 环境。在列表环境内部使用 `\item` 命令开始一个列表项，它可以带一个可选参数表示手动编号或关键字。

`enumerate` 环境使用数字自动编号：

```
\begin{enumerate}
\item 中文
\item English
\item Français
\end{enumerate}
```

1. 中文
2. English
3. Français

2-2-15

`itemize` 环境不编号，但会在每个条目前面加一个符号以示标记：

```
\begin{itemize}
\item 中文
\item English
\item Français
\end{itemize}
```

- 中文
- English
- Français

2-2-16

description 环境总是使用 \item 命令的可选参数，把它作为条目的关键字加粗显示：

2-2-17

```
\begin{description}
  \item[中文] 中国的语言文字
  \item[English] The language of England
  \item[Français] La langue de France
\end{description}
```

中文 中国的语言文字

English The language of Eng-
land

Français La langue de France

上面三种列表环境可以嵌套使用（至多四层）， \LaTeX 会自动处理不同层次的缩进和编号，如 enumerate 环境：

2-2-18

```
\begin{enumerate}
  \item 中文
  \begin{enumerate}
    \item 古代汉语
    \item 现代汉语
    \begin{enumerate}
      \item 口语
      \begin{enumerate}
        \item 普通话
        \item 方言
      \end{enumerate}
    \end{enumerate}
    \item 书面语
  \end{enumerate}
  \item English
  \item Français
\end{enumerate}
```

1. 中文

(a) 古代汉语

(b) 现代汉语

i. 口语

A. 普通话

B. 方言

ii. 书面语

2. English

3. Français

所有条目都以 \item 命令开头。使用 \item 命令的可选参数可以为 enumerate 环境或 itemize 环境临时手工设置编号或标志符号，如：

```
\begin{enumerate}
  \item 中文
  \item[1a.] 汉语
  \item English
\end{enumerate}
```

1. 中文
- 1a. 汉语
2. English

2-2-19

```
\begin{itemize}
  \item[\dag] 中文
  \item English
  \item Français
\end{itemize}
```

- † 中文
- English
- Français

2-2-20

2

2.2.3.2 计数器与编号

`enumerate` 环境的编号是由一组计数器 (counter) 控制的。当 \LaTeX 进入一个 `enumerate` 环境时, 就把计数器清零; 每遇到一个没有可选参数的 `\item` 命令, 就让计数器的值加 1, 然后把计数器的值作为编号输出, 达到自动编号的目的。4 个不同嵌套层次的 `enumerate` 环境使用不同的计数器, 分别是 `enumi`, `enumii`, `enumiii` 和 `enumiv`。 \LaTeX 的计数器都有一个对应的命令 `\the` 计数器名, 用来输出计数器的值, 如第一级 `enumerate` 环境使用 `\theenumi`:

```
\begin{enumerate}
  \item 这是编号 \theenumi
  \item 这是编号 \theenumi
\end{enumerate}
```

1. 这是编号 1
2. 这是编号 2

2-2-21

而 `enumerate` 环境又定义了命令 `\labelenumi`、`\labelenumii`、`\labelenumiii`、`\labelenumiv` 输出条目实际的标签, 一般就是在编号后面增加一个标点。有关 `enumerate` 编号命令的汇总见表 2.9。

\LaTeX 计数器的值可以使用命令 `\arabic`、`\roman`、`\Roman`、`\alph`、`\Alph` 或 `\fnsymbol` 带上计数器参数输出, 它们分别表示阿拉伯数字、小大写罗马数字、小大写字母、特殊符号^①:

^① `ctex` 宏包及文档类还提供了 `\chinese` 命令, 生成汉字的数字。

表 2.9 enumerate 环境的编号和标签

嵌套层次	计数器	计数器输出		条目标签	
		命令	默认值	命令	默认值
1	enumi	\theenumi	\arabic{enumi}	\labelenumi	\theenumi.
2	enumii	\theenumii	\alph{enumi}	\labelenumii	(\theenumii)
3	enumiii	\theenumiii	\roman{enumi}	\labelenumiii	\theenumiii.
4	enumiv	\theenumiv	\Alph{enumi}	\labelenumiv	\theenumiv.

2-2-22

```
\begin{enumerate}
\item 编号
\arabic{enumi}, \roman{enumi}, \Roman{enumi},
\alph{enumi}, \Alph{enumi}, \fnsymbol{enumi}
\item 编号
\arabic{enumi}, \roman{enumi}, \Roman{enumi},
\alph{enumi}, \Alph{enumi}, \fnsymbol{enumi}
\item 编号
\arabic{enumi}, \roman{enumi}, \Roman{enumi},
\alph{enumi}, \Alph{enumi}, \fnsymbol{enumi}
\end{enumerate}
```

1. 编号 1, i, I, a, A, *
2. 编号 2, ii, II, b, B, †
3. 编号 3, iii, III, c, ‡

因此，可以通过重定义 \theenumi (默认定义是 \arabic{enumi}) 和 \labelenumi (默认定义是 \theenumi.) 等命令，控制 enumerate 环境的编号：

2-2-23

```
\renewcommand\theenumi{\roman{enumi}}
\renewcommand\labelenumi{(\theenumi)}
\begin{enumerate}
\item 使用中文
\item Using English
\end{enumerate}
```

- (i) 使用中文
- (ii) Using English

计数器在 \LaTeX 中非常常用，除了列表环境的编号以外，页码、章节和图表的编号等也是由计数器控制的。如页码的计数器是 page，因此现在这句话在第 \thepage 页，即在第 100 页。

可以自定义计数器完成一些工作。新定义计数器用 `\newcounter` 命令，给计数器赋值用 `\setcounter` 命令，计数器自增用 `\stepcounter` 命令，给计数器的值加上一个数用 `\addtocounter` 命令。例如我们仿照 `enumerate` 环境的编号过程：

```
% 计数器设置，通常在导言区
\newcounter{mycnt}
\setcounter{mycnt}{0} % 默认值就是 0
\renewcommand\themycnt{\arabic{mycnt}} % 默认值就是阿拉伯数字
% 计数器使用，通常做成自定义命令的一部分
\stepcounter{mycnt}\themycnt 输出计数器值为 1；
\stepcounter{mycnt}\themycnt 输出计数器值为 2；
\addtocounter{mycnt}{1}\themycnt 输出计数器值为 3；
\addtocounter{mycnt}{-1}\themycnt 输出计数器值为 2；
\addtocounter{mycnt}{-1}\themycnt 输出计数器值为 1。
```

2-2-24

`\refstepcounter` 命令与 `\stepcounter` 功能类似，并且会将当前 `\label` 命令设置为对应的计数器（参见 3.2 节），因而在自动编号的命令或环境的定义中更为常用。在 `\addtocounter` 的参数等需要用到数字的地方，可以使用 `\value{计数器}` 使用计数器的数值^①。

`\newcounter` 命令还可以在后面增加一个可选参数，内容是一个已有的计数器，表示新计数器会随着旧计数器的自增而自动清零。这特别适合定义每个章节独立的编号，例如：

```
\newcounter{quiz}[section]
\renewcommand\thequiz{\thesection-\arabic{quiz}}
```

2-2-25

`amsmath` 提供了 `\numberwithin{计数器1}{计数器2}` 命令，扩展了 `\newcounter` 自动清零的功能，用来让已有的计数器（计数器₁）随（计数器₂）的增加而清零重编号，同时定义其编号格式^[7]，如让数学方程按节编号：

```
\usepackage{amsmath}
\numberwithin{equation}{section}
```

2-2-26

`chngcntr` 宏包^[289] 提供了类似功能的命令 `\counterwithin`，以及相反功能的命令 `\counterwithout`，用来取消计数器间的关联。

^① 事实上，`\value` 命令得到的是 L^AT_EX 计数器对应的原始 T_EX 数字寄存器。

计数器不仅可以用于编号，也能用于复杂的条件控制和循环，`ifthen` 宏包就提供了有关条件判断和循环的功能，而 `calc` 宏包则提供了有关长度（参见 2.1.5 节）和计数器的一些简单运算功能。可参见文档 Carlisle [47]、Thorup et al. [262]。

2.2.3.3 定制列表环境



与 `enumerate` 环境（见表 2.9）类似，`itemize` 环境也使用一组命令来控制前面的标签，见表 2.10。不过 `itemize` 环境比较简单，没有编号。

表 2.10 `itemize` 环境的标签

嵌套层次	命令	默认值	效果
1	<code>\labelitemi</code>	<code>\textbullet</code>	•
2	<code>\labelitemii</code>	<code>\normalfont\bfseries \textendash</code>	—
3	<code>\labelitemiii</code>	<code>\textasteriskcentered</code>	*
4	<code>\labelitemiv</code>	<code>\textperiodcentered</code>	.

`description` 环境相对比较简单，它使用 `\descriptionlabel` 命令来控制标签的输出格式。`\descriptionlabel` 在标准文件类中的原始定义如下^[139]：

```
\newcommand*\descriptionlabel[1]{\hspace\labelsep
\normalfont\bfseries #1}
```

这是一个带有一个参数的命令，参数是标签的文本。标签前面的间距 `\labelsep` 默认为 0.5em。可以修改 `\descriptionlabel` 命令得到不同效果的 `description` 环境：

```
{% 使用分组让 \descriptionlabel 的修改局部化
\renewcommand\descriptionlabel[1]{\normalfont\Large\itshape
\textbullet\ #1}
\begin{description}
\item[标签] 可以修改 \verb=\descriptionlabel= 改变标签的格式。
\item[其他] 其他格式的也可以参考后面的列表环境修改。
\end{description}
}
```

2-2-27

*本节内容初次阅读可略过。

- **标签** 可以修改 `\descriptionlabel` 改变标签的格式。
- **其他** 其他格式的也可以参考后面的列表环境修改。

基本列表环境 `enumerate`, `itemize`, `description` 都是由广义列表环境 `list` 生成的, `list` 环境语法为:

```
\begin{list}{(标签)}{(设置命令)}
  (条目)
\end{list}
```

其中 (标签) 中给出标签的内容, 如果是编号列表可以使用计数器; (设置命令) 则可以设置列表使用的计数器和一些长度参数。`\usecounter{(计数器)}` 表示使用指定的计数器编号。下面是一个简单的编号列表的例子:

```
\newcounter{mylist}
\begin{list}{\#\themylist}%
  {\usecounter{mylist}}
  \item 中文
  \item English
\end{list}
```

#1 中文
#2 English

2-2-28

与 `list` 环境相关的参数有很多, 见图 2.3。可以在 (设置命令) 项中使用 `\setlength` 等命令设置, 可以这样产生一个紧凑的类 `itemize` 环境:

```
\begin{list}{\textbullet}{%
  \setlength{\topsep}{0pt} \setlength{\partopsep}{0pt}
  \setlength{\parsep}{0pt} \setlength{\itemsep}{0pt}}
  \item 中文

  又一段中文

  \item English
  \item Français
\end{list}
```

- 中文
又一段中文

2-2-29

- English
- Français

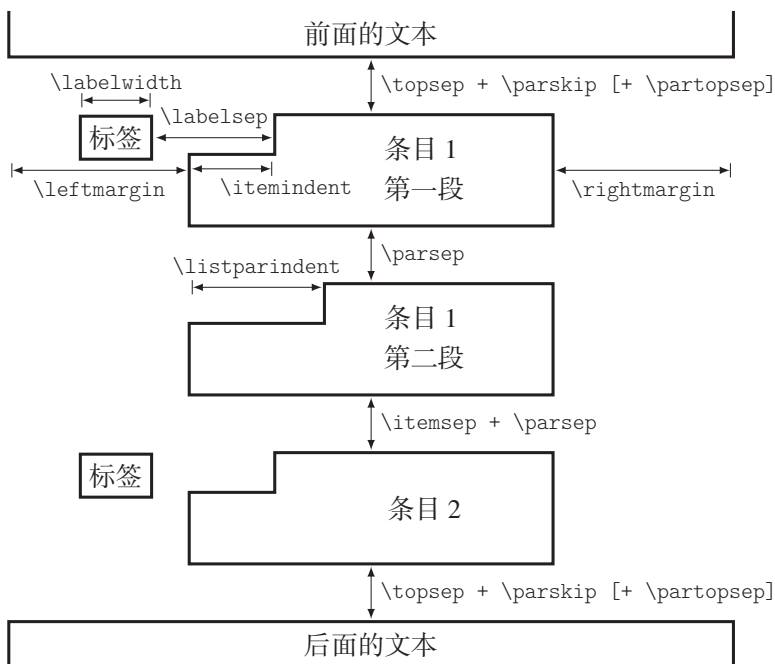


图 2.3 list 列表环境的长度参数。其中 $\backslash\text{partopsep}$ 在列表环境开始新的一段（即前面有空行）时生效

list 环境语法比较烦琐，一般并不直接使用，而是用它来定义新的环境，例如前面的紧凑列表可以定义为 myitemize 环境：

```
\newenvironment{myitemize}{%
  \begin{list}{\textbullet}{%
    \setlength{\topsep}{0pt} \setlength{\partopsep}{0pt}
    \setlength{\parsep}{0pt} \setlength{\itemsep}{0pt}}
{\end{list}}
```

2-2-30

还有一种平凡列表环境 trivlist，可以生成空标签的列表。trivlist 一般不单用，而是用来生成一些与列表看起来没什么关系的环境，例如 center 环境就是由 trivlist 加上 $\backslash\text{centering}$ 命令生成的^[33]，等价定义为：

```
\newenvironment{mycenter}
```

```
{\begin{trivlist}
  \centering
  \item[]
{\end{trivlist}}
```

2-2-31

使用 `list` 环境定制列表比较复杂，特别是对 `enumerate` 等环境更是难以修改其格式。使用 `enumitem` 宏包^[28] 可以方便地对各种列表环境的标签、尺寸进行定制，也可以用它来定义新的列表^①。使用 `enumitem` 宏包，可以直接在 `enumerate` 等环境后加可选参数来定制参数，如：

```
% \usepackage{enumitem}
\begin{enumerate}[itemsep=0pt,parsep=0pt,
  label=(\arabic*)]
  \item 中文
  \item English
  \item Français
\end{enumerate}
```

- (1) 中文
- (2) English
- (3) Français

2-2-32

也可以使用 `\setlist` 命令整体设置参数，用 `\newlist` 命令定义新列表，如：

```
\usepackage{enumitem}
% 仿照 enumerate 环境定义可二级嵌套的 mylist
\newlist{mylist}{enumerate}{2}
% 分别定义每级的格式
\setlist[mylist,1]{itemsep=0pt,parsep=0pt,label=(\arabic*)}
\setlist[mylist,2]{itemsep=0pt,parsep=0pt,label=(\alph*)}
```

2-2-33

`enumitem` 宏包接受的参数大多与图 2.3 中对应，其中标签的参数 `label` 用星号 * 表示计数器，对非标准计数器输出（如 `\chinese`）还要单独进行设置，如：

```
% \usepackage{enumitem}
\AddEnumerateCounter{\chinese}{\chinese}{}
\begin{enumerate}[label={\chinese*、},labelsep=0pt]
  \item 内容清晰
  \item 格式美观
\end{enumerate}
```

2-2-34

^① `enumitem` 宏包继承并扩展了在此之前 `enumerate`、`mdwlist`，特别是 `paralist`^[224] 宏包的功能，提供了更为强大的功能。

一、内容清晰

二、格式美观

有关 `enumitem` 更多功能的详细内容可参见宏包文档 `Bezos` [28]。

2

2.2.4 定理类环境

定理类环境是 \LaTeX 中的一类重要的文本环境，它可以产生一个标题、编号和特定格式的文本，我们在 1.2.4 节中已经看到了定理类环境的定义使用：

2-2-35

```
\newtheorem{thm}{定理} % 一般在导言区
\begin{thm}
直角三角形斜边的平方等于两腰的平方和。
\end{thm}
```

定理 1 直角三角形斜边的平方等于两腰的平方和。

在上面的例子中，命令 `\newtheorem` 用来声明一个新的定理类环境，两个参数分别是定理类环境名和定理输出的标题名。因而，`\newtheorem{thm}{定理}` 就定义了一个 `thm` 环境，效果是输出标题为“定理”的一段文字。新定义的 `thm` 环境允许带有可选参数表示定理的小标题：

2-2-36

```
\begin{thm}[勾股定理]
直角三角形斜边的平方等于两腰的平方和。
\end{thm}
```

定理 2 (勾股定理) 直角三角形斜边的平方等于两腰的平方和。

`\newtheorem` 命令可以在最后使用一个可选的计数器参数 (*Ctr*)，用来表示定理编号是 (*Ctr*) 的下一级编号，并会随 (*Ctr*) 的变化而清零。这通常用于让定理按章节编号，如：

2-2-37

```
\newtheorem{lemma}{引理}[chapter]% 按章
\begin{lemma}偏序集可良序化。 \end{lemma}
\begin{lemma}实数集不可数。 \end{lemma}
```

引理 2.1 偏序集可良序化。


引理 2.2 实数集不可数。

也可以在两个参数之间使用一个可选的计数器参数 (*Ctr*)，表示定理使用 (*Ctr*) 进行编号。这通常用于让几个不同的定理类环境使用相同的编号（定理类环境的计数器就是环境名），如：


```
\newtheorem{prop}[thm]{命题}
\begin{prop}
直角三角形的斜边大于直角边。
\end{prop}
```

命题 3 直角三角形的斜边大于直角边。

2-2-38

 默认的定理类环境的格式是固定的，但这往往并不符合我们的期望。`theorem` 宏包^[159]扩展了定理类环境的格式，可以方便地修改定理类环境的格式，它提供了 `\theoremstyle{格式}` 命令来选择定理类环境的格式，可用的预定义格式有：

plain	默认格式；
break	定理头换行；
marginbreak	编号在页边，定理头换行；
changebreak	定理头编号在前文字在后，换行；
change	定理头编号在前文字在后，不换行；
margin	编号在页边，定理头不换行。

 定理类环境的默认字体是 `\slshape`，定理头默认是 `\bfseries`，可以通过 `\theorembodyfont{字体}` 和 `\theoremheaderfont{字体}` 分别设置定理内容和定理头的字体，并可以设置定理前后的垂直间距变量 `\theorempreskipamount` 和 `\theorempostskipamount`，例如：

```
% 导言区
\usepackage{theorem}
\theoremstyle{changebreak}
\theoremheaderfont{\sffamily\bfseries}
\theorembodyfont{\normalfont}
\newtheorem{definition}{定义}[chapter]
```


2-2-39

```
\begin{definition}
有一个角是直角的三角形是\emph{直角三角形}。
\end{definition}
```

2.1 定义

有一个角是直角的三角形是直角三角形。


2-2-40

 `ntheorem` 宏包^[153]进一步扩充了 `theorem` 宏包的功能，它增加了下面几种 `\theoremstyle`：

nonumberplain 类似 `plain` 格式，但没有编号；

nonumberbreak 类似 **break** 格式，但没有编号；
empty 没有编号和定理名，只输出可选参数。

可以输出无编号的定理。也可以用 `\newtheorem*` 来定义无编号的定理。

 **ntheorem** 增加了更多的设置命令和大量辅助的功能，如给宏包增加 `[thmmarks]` 选项后可以使用 `\theoremsymbol` 命令设置定理类环境末尾自动添加的符号，这对定义证明环境表示证毕符号特别有用：

```
% 引言区
\usepackage[thmmarks]{ntheorem}
{ % 利用分组，格式设置只作用于证明环境
  \theoremstyle{nonumberplain}
  \theoremheaderfont{\bfseries}
  \theorembodyfont{\normalfont}
  \theoremsymbol{\mbox{$\Box$}} % 放进盒子，或用 \ensuremath
  \newtheorem{proof}{证明}
}
```

2-2-41

```
\begin{proof}
证明是显然的。
\end{proof}
```

2-2-42

证明 证明是显然的。 □

ntheorem 宏包的功能很多，除了详细定制定理格式，还可以重定义已有的定理类环境、分类管理定理格式、生成定理目录等；具体的使用中也有很多注意事项（如与 **amsmath** 宏包使用时要加 `[amsmath]` 选项），读者可查阅宏包的文档 May and Schedler [153] 获得进一步的信息。

除了 **theorem** 和 **ntheorem** 宏包，美国数学会发布的 **amsthm** 宏包^[10] 也常用于定理类环境的定制。在使用 **X_YLaTeX** 时，**amsthm** 必须在 **fontspec** 宏包之前载入，而由于 **ctex** 文档类中的 **xeCJK** 宏包会自动调用 **fontspec**，因此不能在 **ctexart** 等文档类中直接使用 **amsthm**，而只能在 **article** 等标准文档类中使用 **amsthm** 宏包再用 **ctexcap** 宏包，比较不便。

amsthm 提供了预定义的 **proof** 环境用来表示证明并自动添加证毕符号，但这个环境的可定制性较差，只能通过重定义 `\proofname` 宏修改证明的“Proof”字样，或重定义 `\qedsymbol` 宏修改证毕符号，例如：


```
\usepackage{amsthm}
\renewcommand\proofname{证明}
\renewcommand\qedsymbol{\ensuremath{\Box}}
```

2-2-43

amsthm 的证毕符号没有 ntheorem 的智能，在显示公式中无法正确判断位置，此时需要手工使用 \qedhere 命令添加证毕符号，如：

```
% \usepackage{amsthm}
\begin{proof}
最后我们有
\[
f(x) = 0. \quad \text{\qedhere}
\]
\end{proof}
```

2-2-44

 amsthm 预定义了一些格式，可以使用 \theoremstyle 命令选择，但如果用 \newtheoremstyle 命令定义新的格式则语法比较复杂。amsthm 与 ntheorem 宏包的功能大致相当，但 ntheorem 宏包的语法容易理解一些，也没有 X_YTeX 下与 fontspec 的冲突问题，因此最好使用 ntheorem 包。有关 amsthm 定制定理格式的进一步用法可以详细内容参见宏包的文档 American Mathematical Society [10]，这里不做过多介绍了。

练习

2.4 Schwarz 的 thmtools 宏包为 amsthm 或 ntheorem 提供了一个更为友好的基于键值语法的用户界面。试参阅文档 [231]，使用 thmtools 宏包的功能重写例 2-2-39。

2.2.5 抄录和代码环境

2.2.5.1 抄录命令与环境

在 2.1.2 节中我们已经看到，L^AT_EX 输入特殊符号相当复杂。然而我们有时候必须经常性地使用特殊符号，例如在排版计算机程序源代码时（特别是排版有关 T_EX 的文章时）就不可避免地使用大量在 L^AT_EX 中有特殊意义的符号，此时就需要使用抄录（verbatim，即逐字）功能。

\verb 命令可用来表示行文中的抄录，其语法格式如下：

`\verb(符号)<抄录内容>(符号)`

在 `\verb` 后, 起始的符号和末尾的符号相同, 两个符号之间的部分将使用打字机字体逐字原样输出:

2

2-2-45

```
\verb"LaTeX \& \TeX" \quad
\verb!\/{\#\$%&~!
```

```
\LaTeX \& \TeX \quad \/{\#\$%&~
```

使用带星号的命令 `\verb*` 则可以使输出的空格为可见的 `␣`:

2-2-46

显示空格 `\verb*!1 2 3 4!`

显示空格 `1␣2␣␣3␣␣␣4`

大段的抄录则可以使用 `verbatim` 环境:

2-2-47

```
\begin{verbatim}
#!usr/bin/env perl
$name = "guy";
print "Hello, $name!\n";
\end{verbatim}
```

```
#!usr/bin/env perl
$name = "guy";
print "Hello, $name!\n";
```

同样, 可以使用带星号的 `verbatim*` 环境输出可见空格:

2-2-48

```
\begin{verbatim*}
#include <stdio.h>
main() {
    printf("Hello, world.\n");
}
\end{verbatim*}
```

```
#include␣<stdio.h>
main()␣{
    ␣␣␣␣printf("Hello,␣world.\n");
}
```

抄录命令和环境都属于特殊命令, 一般不能作为其他命令的参数出现, 例如使用 `\fbox{\verb!abc!}` 将会产生错误。可以使用 `lrbox` 环境把它们保存在自定义盒子中再进行使用, 参见 2.1.5 节, 这也可以由 `fancyvrb` 宏包^[301] 提供的命令实现, 例如:

2-2-49

```
% \usepackage{fancyvrb}
\SaveVerb{myverb}|#\$%^&|
\fbox{套中 \UseVerb{myverb}}
```

套中 `\#\$%^&`

值得一提的是 `cprotect` 宏包^[80]，它定义了 `\cprotect` 等命令，可以方便地在其他命令参数中使用 `\verb` 命令或 `verbatim` 环境。`\cprotect` 的用法与 `\protect` 有些类似，把它用在带参数的命令前面，来“保护”参数中有抄录的命令：

```
% \usepackage{cprotect}
\cprotect\fbbox{套中 \verb|#$%^&|}
```

套中 #\$\$~&

2-2-50

尽管使用方便，`cprotect` 宏包的使用限制会多一些，在一些命令中（如 `\parbox`）仍然需要用先保存再使用的方式。

`verbatim` 宏包^[233] 提供了 `verbatim` 环境的一些扩展。特别是定义了 `\verbatiminput` (文件名) 命令用于逐字抄录整个文件的内容。

`shortvrb` 宏包^[161] 提供了 `\verb` 命令的简写形式，可以使用 `\MakeShortVerb`(符号) 来定义这种简写，或用 `\DeleteShortVerb`(符号) 取消定义，如在导言区定义：

```
\usepackage{shortvrb}
\MakeShortVerb|
```

那么就可以使用竖线 `|` 作为简写方式了：

```
verbatim |\LaTeX|
```

verbatim \LaTeX

2-2-51

`fancyvrb` 宏包提供了一系列 `verbatim` 环境的扩展，它提供的 `Verbatim` 环境可以修改字体、边框、填充颜色、行号等多种格式，甚至在抄录环境中插入任意 `LaTeX` 代码，功能强大。读者可参考宏包的文档 Zandt [301]，这里不作详细介绍了。

2.2.5.2 程序代码与 listings

可以使用 `verbatim` 环境排版程序代码。不过，如果还想在程序代码中增加语法高亮功能，那么 `verbatim` 环境就捉襟见肘了，比如要排版下面的效果：

```
1 /* hello.c
2  *   A 'hello world' program. */
3 #include <stdio.h>
4 int main()
5 {
6     printf("Hello, \_world.\n");
7     return 0;
8 }
```


这种带语法高亮的程序代码可以使用 listings 宏包^[174] 排版。

listings 宏包提供的基本功能是 lstlisting 环境，可以把它看做是加强的 verbatim 环境。不过在未做任何设置时，lstlisting 环境并没有语法高亮的效果，而且看起来比直接使用 verbatim 还难看些，如下所示：

2

```
% 导言区使用 \usepackage{listings}
\begin{lstlisting}
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
\end{lstlisting}
```

```
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
```

要想得到漂亮的排版效果，必须仔细设置 lstlisting 环境的格式。可以使用 lstlisting 环境的可选参数设置格式，也可以使用 \lstset{{设置}} 进行全局设置。最基本的参数是 language，设置代码使用的语言，如：

2-2-52

```
\begin{lstlisting}[language=C]
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
\end{lstlisting}
```

```
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
```

此时可以看到 C 语言的注释用斜体排出，关键字用粗体排出，字符串中的空格也排版为可见的。listings 宏包支持的语言非常多，几乎包括了各种常见的语言（见表 2.11）。

前面的例子中字体并不是很好看，可以用 basicstyle 选项设置 lstlisting 环境的整体格式，或用 keywordstyle 设置关键字的格式，用 identifierstyle 设置标识符格式，用 stringstyle 设置字符串的格式，用 commentstyle 设置注释的格式，等等。以上这些都是影响 lstlisting 环境输出效果最明显的一些参数，例如：

表 2.11 listings 宏包预定义的语言。这里一些语言的定义只是初步的，如 HTML 和 XML，带下画线的方言是默认的方言

ABAP (R/2 4.3, R/2 5.0, R/3 3.1, R/3 4.6C, R/3 6.10)	Ada (<u>2005</u> , 83, 95)
ACSL	Ant
Algol (60, <u>68</u>)	Awk (<u>gnu</u> , POSIX)
Assembler (Motorola68k, x86masm)	Basic (<u>Visual</u>)
bash	C++ (ANSI, GNU, <u>ISO</u> , Visual)
C (<u>ANSI</u> , Handel, Objective, Sharp)	CIL
Caml (<u>light</u> , Objective)	Cobol (1974, <u>1985</u> , ibm)
Clean	command.com (<u>WinXP</u>)
Comal 80	csh
Comsol	Eiffel
Delphi	erlang
Elan	Fortran (77, 90, <u>95</u>)
Euphoria	Gnuplot
GCL	HTML
Haskell	inform
IDL (empty, CORBA)	JVMIS
Java (empty, AspectJ)	Lingo
ksh	Logo
Lisp (empty, Auto)	Mathematica (1.0, 3.0, <u>5.2</u>)
make (empty, gnu)	Mercury
Matlab	Miranda
MetaPost	ML
Mizar	MuPAD
Modula-2	Oberon-2
NASTRAN	Octave
OCL (decorative, <u>OMG</u>)	Pascal (Borland6, <u>Standard</u> , XSC)
Oz	PHP
Perl	Plasm
PL/I	POV
PostScript	Promela
Prolog	Python
PSTricks	Reduce
R	RSL
Rexx	S (empty, PLUS)
Ruby	Scilab
SAS	SHELXL
sh	SPARQL
Simula (<u>67</u> , CII, DEC, IBM)	tcl (empty, tk)
SQL	TeX (AllLaTeX, common, LaTeX, <u>plain</u> , primitive)
TeX (AllLaTeX, common, LaTeX, <u>plain</u> , primitive)	Verilog
VBScript	VRML (<u>97</u>)
VHDL (empty, AMS)	XSLT
XML	

```
\lstset{ % 整体设置
  basicstyle=\sffamily,
  keywordstyle=\bfseries,
  commentstyle=\rmfamily\itshape,
  stringstyle=\ttfamily}
\begin{lstlisting}[language=C]
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
\end{lstlisting}
```

```
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
```

`listings` 宏包默认将字符等宽显示，这样可以使文字的不同列容易对齐，但也会使得一些字体看上去非常怪异。可以通过设置 `column` 选项为 `flexible`（默认为 `fixd`），或使用 `flexiblecolumns` 选项来把字符列设置为非等宽的。采用非等宽的列可以让默认的 `Roman` 字体看上去也比较顺眼：

```
\lstset{flexiblecolumns}% column=flexible
\begin{lstlisting}[language=C]
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
\end{lstlisting}
```

```
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
```

可以设置 `numbers` 选项为 `left` 或 `right` 在左右增加行号（默认为 `none`），使用 `numberstyle` 选项设置行号格式：

```
\lstset{columns=flexible,
  numbers=left,numberstyle=\footnotesize}
\begin{lstlisting}[language=C]
/* hello.c */
#include <stdio.h>
main() {
    printf("Hello.\n");
}
\end{lstlisting}
```

```
1  /* hello.c */
2  #include <stdio.h>
3  main() {
4      printf("Hello.\n");
5  }
```

2-2-55

2

`listings` 宏包还提供了 `\verb` 命令的对应物 `\lstinline`，这样可以直接在行文段落中使用带语法高亮的代码：

```
\lstset{language=C,flexiblecolumns}
语句 \lstinline!typedef char byte!
```

语句 **typedef char byte**

2-2-56

`listings` 宏包对汉字等非 ASCII 字符的支持不是很好。使用 $\text{Xe}_{\text{L}}\text{TeX}$ 时，一般需要对汉字用逃逸字符处理，这样才能得到正确的结果^①：

```
\lstset{language=C,flexiblecolumns,
  escapechar='} % 设置 ' 为逃逸字符
\begin{lstlisting}
int n; // '一个整数'
\end{lstlisting}
```

int n; // 一个整数

2-2-57

`listings` 将逃逸字符之间的内容当做普通的 $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 代码处理，因而汉字可以得到正确处理。事实上这种方法也可以用来输出一些特殊效果的代码，比如在代码中使用数学公式：

```
\lstset{language=C,flexiblecolumns,
  escapechar='}
\begin{lstlisting}
double x = 1/sin(x); // '$\frac{1}{\sin x}$'
\end{lstlisting}
```

double x = 1/sin(x); // $\frac{1}{\sin x}$

2-2-58

^① 不使用 $\text{Xe}_{\text{L}}\text{TeX}$ 而使用 CJK 方式处理汉字时，还需要设置 `extendedchars=false` 选项。

上面介绍的选项只是 listings 宏包各种功能中的很少一部分，事实上，使用 listings 宏包还可以设置背景颜色、强调内容、标题、目录等。请读者自行查阅宏包文档 Moses [174]。

2.2.6 tabbing 环境

tabbing 环境用来排版制表位，即让不同的行在指定的地方对齐。在 tabbing 环境中，行与行之间用 \\ 分隔，使用 \= 命令来设置制表位，用 \> 命令则可以跳到下一个前面已设置的制表位，因而可以用 tabbing 环境制作比较简单的无线表格（真正的表格参见 5.1 节）：

2-2-59

```
\begin{tabbing}
格式\hspace{3em} \= 作者 \\
Plain \TeX \> 高德纳 \\
\LaTeX \> Leslie Lamport
\end{tabbing}
```

格式	作者
Plain T _E X	高德纳
L ^A T _E X	Leslie Lamport

\kill 命令与 \\ 类似，它会忽略这一行的内容，只保留制表位的设置。使用 \kill 可以做出 tabbing 环境的样本行：

2-2-60

```
\begin{tabbing}
格式\hspace{3em} \= 作者 \kill
Plain \TeX \> 高德纳 \\
\LaTeX \> Leslie Lamport
\end{tabbing}
```

Plain T _E X	高德纳
L ^A T _E X	Leslie Lamport

关于制表位的其他命令如下：

\'	使它前后的文字以当前制表位为中心对齐
\'	使后面的文字右对齐
\<	与 \> 相反，跳到前一个制表位
\+	后面的行开始都右跳一个制表位
\-	后面的行开始都左跳一个制表位
\pushtabs	保存当前制表位
\poptabs	恢复由 \pushtabs 保存的制表位

第2章 组织你的文本

117

由于在 tabbing 环境中原来表示字母重音的命令 $\backslash=$ 、 \backslash' 、 $\backslash\prime$ 被重定义为制表位的操作，所以在 tabbing 环境中重音命令改为 $\backslash a$ 后加 $=$ 、 $'$ 、 \prime ，如在 tabbing 中的 $\backslash a=o$ 就得到 \bar{o} 。

下面给出一个相对复杂的例子，使用上面的命令排版一个算法，并说明这些命令的意义：

```
\newcommand\kw{\textbf} % 表示描述算法的关键字
\begin{tabbing}
\pushtabs
算法：在序列  $A$  中对  $x$  做二分检索 \\\
输入： $A$ ， $x$  及下标上下界  $L$ ， $H$  \\\
\qqquad\backslash=\backslash\kw{integer}  $L$ ， $H$ ， $M$ ， $j$  \\\
\kw{while} \backslash=\backslash  $L \leq H$  \kw{do} \backslash'  $L$  与  $H$  是左右分点 \\\
 $M$  \gets  $\lfloor(L+H)/2\rfloor$  \backslash'  $M$  是中间分点 \\\
\kw{case} \backslash=\backslash\backslash
    condition \backslash= foo \backslash+\backslashkill
     $x > A[M]$  \backslash'  $H$  \gets  $M-1$  \\\
     $x < A[M]$  \backslash'  $H$  \gets  $M+1$  \\\
    \kw{else} \backslash' \backslash=  $j$  \gets  $M$  \backslash' 找到  $x$ ，返回位置 \\\
    \> \kw{return}  $(j)$  \\\
\<\< \kw{endcase} \backslash-\backslash-\backslash\backslash
 $j$  \gets  $0$  \\\
\kw{return}  $(j)$  \backslash-\backslash\backslash
\poptabs
算法示例：\\
 $A = \{2, 3, 5, 7, 11\}$ ， $x=3$ \\
\qqquad\backslash=\backslash  $M$  \qqquad \backslash=  $L$  \qqquad \backslash=  $H$  \qqquad \backslash= \\\
    无 \> 1 \> 5 \> 初始值，进入循环 \\\
    3 \> 1 \> 2 \>  $H$  变化 \\\
    2 \> 无 \> 无 \> 找到  $x$ ，输出位置 2
\end{tabbing}
```

2

算法：在序列 A 中对 x 做二分检索

输入： A ， x 及下标上下界 L ， H

integer L ， H ， M ， j

2-2-61

118

2.2 段落与文本环境

```

while  $L \leq H$  do
     $M \leftarrow \lfloor (L + H)/2 \rfloor$ 
    case
         $x > A[M]$ :  $H \leftarrow M - 1$ 
         $x < A[M]$ :  $H \leftarrow M + 1$ 
        else:  $j \leftarrow M$ 
        return( $j$ )
    endcase
 $j \leftarrow 0$ 
return( $j$ )
    
```

L 与 H 是左右分点
 M 是中间分点

找到 x , 返回位置

算法示例:

$A = \{2, 3, 5, 7, 11\}, x = 3$

M	L	H	
无	1	5	初始值, 进入循环
3	1	2	H 变化
2	无	无	找到 x , 输出位置 2

尽管使用 `tabbing` 环境可以很自由地排版复杂的算法, 但时刻考虑制表位来调整算法结构有时实在太考验人的耐心了。排版算法这类结构化的伪代码可以使用专门的算法宏包。`clrscode`^[57] 是许多算法宏包中最为简单的一种, 它可以按著名教材《算法导论》^[56] 中的格式进行算法排版, 事实上它就是用 `tabbing` 环境实现的; 另一个使用广泛的算法宏包是 `algorithm2e`^[78], 它提供丰富的命令和复杂的定制功能; `algorithmicx`^[117] 也提供了类似的易定制的算法排版功能, 有需要的读者可以参见这些算法宏包的说明文档来做进一步的选择。



练习

2.5 参考 `clrscode` 或 `algorithm2e` 宏包的文档, 重新排版例 2-2-61 的算法。

2.2.7 脚注与边注

\LaTeX 使用 `\footnote{脚注内容}` 产生脚注, 例如¹。这是脚注的默认格式, 使用一个阿拉伯数字的上标作为编号, 脚注内容出现在页面底部, 以 `\footnotesize` 的字

¹这是一个脚注。

号输出。脚注的内容可以是大量的文字，但注意前后应该都和正文紧接着，避免出现多余的空格，特别要注意标点前后的位置，像前面的例子：

例如`\footnote{这是一个脚注。}`。

2-2-62


`\footnote` 是自动编号的，例如¹。`\footnote` 可以带一个可选的参数，表示手工的数字编号，例如用 `\footnote[1]{又是一？}` 的效果¹，它不改变原来的脚注编号²。

脚注的使用有一些限制，它通常只能用在一般的正文段落中，不能用于表格、左右盒子（如一个 `\mbox` 的参数中）等受限的场合。此外，脚注也不能直接用在 `\section` 等章节命令的参数中，也不能用在 `\parbox` 中。不过可以把脚注用在 `minipage` 环境（参见 2.2.8 节）里面，此时脚注出现在 `minipage` 环境产生的盒子的底部，并使用局部的编号（默认按字母编号）：

```
\begin{minipage}{8em}
这是小页环境\footnote{脚注。}中的脚注。
\end{minipage}
```

这是小页环境^a中
的脚注。
^a脚注。

2-2-63

脚注使用的计数器是 `footnote`，在 `minipage` 环境中则使用 `mpfootnote` 计数器， 可以修改 `\thefootnote` 和 `\thempfootnote` 改变编号的格式。一种常见的修改是使用符号编号，即定义：

```
\renewcommand\thefootnote{\fnsymbol{footnote}}
```

2-2-64

得到的效果是[‡]。另一种常见修改是带圈的数字，这可以利用 `\textcircled` 命令生成的带圈文字来完成：

```
\renewcommand\thefootnote{\textcircled{\arabic{footnote}}}
```

得到的效果是^④。使用 `pifont` 宏包^[229] 提供的带圈数字符号效果更好些，使用 `pifont` 宏包的 `\ding` 命令可以输出符号表中的符号，查表找到阳文带圈数字从 172 号符号开始，此时需要 ϵ -TeX 的 `\numexpr` 命令支持数字的计算：

```
\usepackage{pifont}
\renewcommand\thefootnote{\ding{\numexpr171+\value{footnote}}}
```

2-2-65

¹另一个脚注。

¹又是一？

²继续前面编号。

[‡]符号编号的脚注。

^④带圈数字编号脚注。

得到的效果是^①，这也是本书使用的编号符号。

在不能使用脚注的位置，例如盒子、表格中，可以使用命令 `\footnotemark` 和 `\footnotetext` 分开输入脚注的编号和内容。`\footnotemark[⟨数字⟩]` 命令产生正文中的脚注编号，如果没有可选参数，脚注计数器自动自增；`\footnotetext[⟨数字⟩]{⟨内容⟩}` 命令产生脚注的内容，脚注计数器不自增，如果有可选参数就使用手工编号。这里在表格中使用脚注的方法是典型的（也可以把表格放在 `minipage` 环境中，直接使用 `\footnote` 命令即可）：

```
\begin{tabular}{r|r}
  自变量 & 因变量\footnotemark \\\hline
  $x$ & $y$
\end{tabular}
\footnotetext{$y=x^2$。}
```

2-2-66

自变量	因变量 ²
x	y

◆ 如果要在章节或图表标题中使用脚注，则要用 `\protect\footnote` 代替 `\footnote`，因为这里 `\footnote` 是脆弱命令（`fragile command`），当脆弱命令用在所谓活动的命令参数（`moving arguments`）中时，就必须使用 `\protect` 命令保护起来^[136, C.1.3]（参见 8.1.2 节）。此外还要注意使用 `\section` 等命令的可选参数避免把脚注符号装进页眉和目录中：

```
\section[节标题]{节标题\protect\footnote{标题中的脚注}}
```

2-2-67

实际中主要遇到的活动参数就是会生成目录项的命令，即章节命令和 `\caption` 等。

◆ 几条脚注之间的距离用长度变量 `\footnotesep` 设置。脚注线由命令 `\footnoterule` 定义，默认的定义是长为 2in 的一条线，可以重定义 `\footnoterule` 改变脚注线，如可以使用命令 `\renewcommand\footnoterule{}` 定义脚注线为空。

◆ 更多脚注的格式的定制可以使用 `footmisc` 宏包^[71] 完成。例如， \LaTeX 的脚注默认每章重新清零编号（如果没有 `\chapter` 一级，则全文统一编号），可以使用

```
\usepackage[perpage]{footmisc}
```

2-2-68

让脚注每页清零编号，这也是中文排版更常见的样式。`footmisc` 还提供了控制脚注段落形状的许多选项，以及一些杂项命令，可参考文档 `Fairbairns` [71] 获得更多信息。

^①更好的带圈脚注。

² $y = x^2$ 。

L^AT_EX 还提供了边注的命令 `\marginpar{内容}`，用来给文档添加在页边的旁注。边注具
边注的使用方法和脚注差不多，只是边注不编号，内容出现的位置与正文更接近。页边
距通常很窄，所以不要在边注中长篇大论，最好把它留给个别需要强调指出的地方。

在单面模式 `onecolumn` 下（参见 2.4.2 节），边注在页面的右侧；在双面模式
`twocolumn` 下，边注在页面的外侧（即奇数页在右，偶数页在左）。`\marginpar` 命令可
以带一个可选参数，设置出现在偶数页左侧的边注，而原来的参数表示在右侧的边注，
例如：

```
有边注的文字\marginpar[\hfill 左 $\rightarrow$]{ $\leftarrow$ 右}
```

有边注的文字

2-2-69

← 右

可以使用命令 `\reversemarginpar` 改变边注的左右（或内外）位置，或者用命令
`\normalmarginpar` 复原边注的左右位置。

边注主要由三个长度变量控制：`\marginparwidth` 是边注的长宽，`\marginparsep`
是边注与正文的距离，`\marginparpush` 是边注之间的最小距离。这些长度都可以
用 `\setlength` 设置，不过前面两个变量也可以使用 `geometry` 宏包设置，这样更为方
便（参见 2.4.2 节）。

`\marginpar` 产生的边注是浮动的，这样当边注内容较多不能直接放下时，实际内
容会与插入命令的位置略有变化，如果不想要浮动的边注，可以改用 `marginnote`
宏包^[132] 提供的 `\marginnote` 命令，禁止不需要的浮动。有时候，浮动的边注在双面模
式下会被放在错误的一侧，可以使用 `mparhack` 宏包^[234] 进行修正。

除了边注和脚注，`endnotes` 宏包还提供了尾注功能，一般用在注释特别多或长，不
适于用脚注的情形，详细说明可参见宏包的文档 Lavagnino [144]。

2.2.8 垂直间距与垂直盒子

垂直间距的命令与水平间距的命令是对应的，类似用 `\hspace` 和 `\hspace*` 生成水
平间距，可以用 `\vspace{长度}` 和 `\vspace*{长度}` 生成垂直间距。垂直间距也是
弹性距离，可以用 `\vfill` 表示 `\vspace{\fill}`。


`\vspace` 的参数可以是长度的值，也可以是像 `\parskip`、`\itemsep` 这样的长度变
量。L^AT_EX 也有一些预定义的垂直间距命令，`\smallskip`、`\medskip`、`\bigskip` 分别
表示较小的、中间的和较大的垂直间距：


<u><code>\smallskip</code></u>	<u><code>\medskip</code></u>	<u><code>\bigskip</code></u>
--------------------------------	------------------------------	------------------------------


这三个间距的大小由长度变量 `\smallskipamount`、`\medskipamount`、`\bigskipamount` 定义；其具体的值在文档类中定义，默认 `\smallskipamount` 的值是 `3pt plus 1pt minus 1pt`，`\medskipamount` 和 `\bigskipamount` 分别是它的 2 倍和 4 倍。

`\addvspace{<长度>}` 与 `\vspace{<长度>}` 的功能类似，只是它在重复使用时只起一个的作用，即 `\addvspace{<s1>}\addvspace{<s2>}` 相当于 `\addvspace{max{<s1>,<s2>}}` 的功能。


为了保证正确的间距效果，这些间距命令一般放在后面一段的开头，而不是前面一段的末尾^①。

 **LaTeX** 使用两种机制处理断页问题，可以使用命令 `\raggedbottom` 告诉 **LaTeX** 让页面中的内容保持它的自然高度，把每一页的页面底部用空白填满。相反，`\flushbottom` 则让 **LaTeX** 将页面高度均匀地填满，使每一页的底部直接对齐。在标准文档类中，**LaTeX** 会为单面输出的文档（`oneside` 选项）设置 `\raggedbottom`，而为双面输出的文档（`twoside` 选项）设置 `\flushbottom`。当排满一页后，页面剩余空间比较大的时候，如果还要排版一个很高的内容（如多行的公式或表格），就会造成难看的断页，通常这是由浮动环境（参见 5.3.1 节）解决的，但在无可避免的时候就需要在两种断页机制下选择一种：双面印刷的书籍使用 `\flushbottom` 可以保证摊开时左右两页对称，但如果有多于松散的页面就不如使用 `\raggedbottom` 了。

 `\pagebreak` 可以指定页面断页的位置，它可以带一个 0 到 4 的可选参数，表示建议断页的程度，0 不允许分页，默认值 4 表示必须断页。`\nopagebreak` 与 `\pagebreak` 功能和参数的意义相反。例如可以使用 `\pagebreak[3]` 标示一个特别适合断页的地方，**LaTeX** 会优先考虑在此处断页。

 在遇到一页最后只剩孤立的一行没有排完时，还可以临时用 `\enlargethispage{<长度>}` 增加当前页版心的高度，把剩下的一点内容装到当前页，避免难看的断页（尤其是在书籍一章的末尾）。还可以用带星的命令 `\enlargethispage*`，此时不仅增加页面版心高度，还会适当缩小行距。

可以使用 `\newpage` 直接对文字手工分页。分页不同于简单的断页，这也是通常我们习惯上所说的换一页的意义。`\newpage` 实际相当于首先强制分段，然后使用 `\vfill` 把页面填满，最后用 `\pagebreak` 换页。连续使用多个 `\newpage` 并不会多次分页产生空白页，如果需要多次手工分页，可以在空白页使用一个空的盒子 `\mbox{}` 占位。

 `\clearpage` 与 `\newpage` 功能类似，也完成填充空白并分页的工作，不同的是它还会清理浮动体（参见 5.3.1 节）；`\cleardoublepage` 与 `\clearpage` 类似，只是在双面文档（`twoside`，参见 2.4.1 节）的奇数页会多分一页，使新的一页也在奇数页。

^① 即在分段或 `\par` 之后。在盒子构造的场合，有时也可以不分段。

第2章 组织你的文本

123

在 2.1.5 节我们已经知道了盒子的基本概念。不过，2.1.5 节介绍的盒子都是水平盒子，文字不能在其中分行分段（除非嵌套其他内容）。使用 `\parbox` 命令或 `minipage` 环境生成的子段盒子就没有这种限制，垂直盒子在 \LaTeX 中也称为子段盒子（`parbox`），其语法格式如下：

```
\parbox[<位置>][<高度>][<内容位置>]{<宽度>}{<盒子内容>}  
\begin{minipage}[<位置>][<高度>][<内容位置>]{<宽度>}  
<盒子内容>  
\end{minipage}
```

`\parbox` 和 `minipage` 环境必须带有一个宽度参数，表示内容的宽度，超出宽度的内容会自动换行：

```
前言\parbox{2em}{不搭后语}。
```

前言
不搭
后语。

2-2-70

在垂直盒子中会自动设置一些间距，如段落缩进 `\parindent` 会被设置为 `0pt`，段间距会被设置为 `0pt`（正文默认为 `0pt plus 1pt`）。

`\parbox` 和 `minipage` 环境还可以带三个可选参数，分别表示盒子的基线位置、盒子的高度以及（指定高度后）盒子内容在盒子内的位置。位置参数可以使用 `c`（居中）、`t`（顶部）、`b`（底部），默认为居中；内容位置参数可以使用 `c`、`t`、`b`、`s`（垂直分散对齐）。其中 `s` 参数只在有弹性间距时生效。而 `t` 选项指按第一行的基线对齐，而不是盒子顶端。例如：

```
前言\parbox[t]{2em}{不搭后语}。  
后语\parbox[b]{2em}{不搭前言}。
```

不搭
前言不搭。后语前言。
后语

2-2-71

```
\begin{minipage}[c][2.5cm][t]{2em} 两个 \end{minipage}\quad  
\begin{minipage}[c][2.5cm][c]{3em} 黄鹂鸣翠柳， \end{minipage}\quad  
\begin{minipage}[c][2.5cm][b]{3em} 一行白鹭上青天。 \end{minipage}\quad  
\begin{minipage}[c][2.5cm][s]{4em}  
\setlength{\parskip}{0pt plus 1pt}% 恢复正文默认段间距  
窗含西岭千秋雪，\par  
门泊东吴万里船。  
\end{minipage}
```

2-2-72

两个	窗含西岭
	千秋雪，
黄鹂鸣	一行白
翠柳，	鹭上青
	门泊东吴
	天。 万里船。

在例 2-1-77 中我们看到了如何利用 `lrbox` 环境把 `\verb` 命令产生的抄录内容放在盒子里面使用，但如果把里面的 `\verb` 换成 `verbatim` 就会失败，因为 `lrbox` 里面只能放置水平模式的内容，`verbatim` 环境则是在垂直模式数行内容。此时，可以在 `verbatim` 环境外再套一层 `minipage`，将整个抄录环境的内容看做一个整体^①：

```
\newsavebox{\verbatimbox} % 通常在导言区定义
\begin{lrbox}{\verbatimbox}
\begin{minipage}{10em}
\begin{verbatim}
#!/bin/sh
cat ~/${file}
\end{verbatim}
\end{minipage}
\end{lrbox}
\fbbox{\usebox{\verbatimbox}}\quad\fbbox{\usebox{\verbatimbox}}
```

2-2-73

```
#!/bin/sh
cat ~/${file}
```

```
#!/bin/sh
cat ~/${file}
```

除了水平盒子和垂直盒子，还有一种重要的盒子，称为标尺（`rule`）盒子。标尺盒子用 `\rule` 命令产生：

`\rule[升高距离]{宽度}{高度}`

一个标尺盒子就是一个实心的矩形盒子，不过通常只是使用一个细长的标尺盒子画线，这正是“标尺”一词的由来，例如：

```
\rule{1pt}{1em}Middle\rule{1pt}{1em} \\
Left\rule[0.5ex]{2cm}{0.6pt}Right \\
\rule[-0.1em]{1em}{1em} 也可以用作证毕符号
```

2-2-74

■ Middle
Left——Right
也可以用作证毕符号

^① 不过，这个功能可以直接使用 `fancyvrb` 的 `SaveVerbatim` 环境和 `\UseVerbatim` 命令。

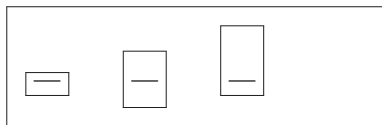
第2章 组织你的文本

125

这里参数〈升高距离〉默认为 0pt，它实际上是设置盒子深度的相反数，效果就像把盒子做了移位一样。

类似幻影 `\phantom`， \LaTeX 有一种宽或高为零的盒子，专门用来占位，称为“支架”（`\strut`）。 \LaTeX 有预定义的垂直支架 `\strut`，占有当前字号大小的高度和深度；有时也可以使用长或宽为零的标尺盒子来表示任意大小的支架，例如：

```
\fbox{---}\quad\quad
\fbox{\strut---}\quad\quad
\fbox{\rule{0pt}{2em}}---
```



2-2-75

2

可以用 `\raisebox` 造成升降的（水平）盒子：



```
\raisebox{〈距离〉}[〈高度〉][〈深度〉]{〈内容〉}
```

其中距离为正时盒子里面的内容上升，距离为负时下降。例如我们可以使用 `\raisebox` 和 `\hspace` 自己定义 \TeX 的标志：

```
% 这与实际 \TeX 的定义基本等价
\mbox{T\hspace{-0.1667em}}%
\raisebox{-0.5ex}{E}%
\hspace{-0.125em}X}
```

 \TeX

2-2-76

`\parbox` 命令和 `minipage` 环境产生的垂直盒子必须事先确定宽度，这对于自动折行是十分必要的。然而在个别的场合，我们只是需要盒子的内容能手工分成几行，却又希望它保持按手工分行的“自然”宽度，这时可以改用 `varwidth` 宏包^[18] 提供的 `varwidth` 环境代替 `minipage` 环境。`varwidth` 对应 `minipage` 的宽度参数，表示盒子的最大宽度，例如：

```
\fbox{\begin{varwidth}{10cm}
自然\宽度
\end{varwidth}}
```

自然
宽度

2-2-77

不过在 5.1.2 节我们将看到，在一些简单的情况下这可以使用一个单列的表格完成。

 \TeX 标志

高德纳教授为他的 \TeX 软件设计了一个高低起伏的标志：“字母‘E’”是不平的，错位的‘E’提醒我们 \TeX 是关于排版的，并且也把 \TeX 与其他系

统的名字区分开来。”^[126]与此同时，与 $\text{T}_{\text{E}}\text{X}$ 同时设计的字体描述语言 METAFONT^[125] 也被赋予了独特的写法——使用以 METAFONT 语言描述并绘制的专用字体排印。

这一独特的命名趣味随即被 $\text{T}_{\text{E}}\text{X}$ 的各种相关工具所效仿。 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的标志是在 $\text{T}_{\text{E}}\text{X}$ 的标志前面加上错排的 L, A 两个字母，而其当前版本 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}2_{\epsilon}$ 在后面加上 2_{ϵ} 来表示一个超过 2 而又接近 3 的数字。受 METAFONT 影响而产生的绘图语言 METAPOST 则使用了与前者如出一辙的标志；美国数学会（AMS）编写的各种宏包、字体被冠以 $\mathcal{A}\mathcal{M}\mathcal{S}$ 的称号，这实际是用与 $\text{T}_{\text{E}}\text{X}$ 同样方法错位排布的三个数学花体字； $\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$ 与 $\text{X}_{\text{E}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的标志中有一个反写的字母 E； $\text{BibT}_{\text{E}}\text{X}$ 中 BIB 三个字母（bibliography 的缩写）则用了小型大写字母，等等。甚至连绘图包 $\text{X}_{\text{Y}}\text{pic}$ 、基于 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的文档处理系统 $\text{L}_{\text{Y}}\text{X}$ 、 $\text{T}_{\text{E}}\text{X}_{\text{MACS}}$ 等，也有自己独特的名字和标志。 $\text{T}_{\text{E}}\text{X}$ 相关软件的作者们喜欢以此彰显自己的不同。

当然，所有这些标志都不难用 $\text{T}_{\text{E}}\text{X}$ 本身排版出来，例 2-2-76 就给出了 $\text{T}_{\text{E}}\text{X}$ 标志的排版方式，使用类似的方法不难实现各种相关软件的标志。尽管本书充斥着这类软件标志，但它们在实际的书籍文章中其实很少会用到。表 2.12 给出了一些常见标志在 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 中的命令。

表 2.12 $\text{T}_{\text{E}}\text{X}$ 及部分相关软件所使用的标志

标志	命令	额外的宏包
$\text{T}_{\text{E}}\text{X}$	<code>\TeX</code>	
$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$	<code>\LaTeX</code>	
$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}2_{\epsilon}$	<code>\LaTeXe</code>	
METAFONT	<code>\MF</code>	mflogo
METAPOST	<code>\MP</code>	mflogo
$\mathcal{A}\mathcal{M}\mathcal{S}$	<code>\AmS</code>	amsmath
$\text{X}_{\text{E}}\text{T}_{\text{E}}\text{X}$	<code>\XeTeX</code>	metalogo 或 hologo
$\text{X}_{\text{E}}\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$	<code>\XeLaTeX</code>	metalogo 或 hologo
$\text{BibT}_{\text{E}}\text{X}$	<code>\BibTeX</code>	doc 或 hologo

2.3 文档的结构层次

纲举而目张，科技文档首重结构条理。 \LaTeX 鼓励结构化的文档编写，并提供了许多设置文档结构的手段。这一节我们将看到如何正确使用 \LaTeX 提供的命令将文档的内容与格式分离，使文档不仅有良好的输出效果，也有清晰整齐易于维护的源文件。

2.3.1 标题和标题页

\LaTeX 的标题文档类提供专门的命令输入文章或书籍的标题。在 \LaTeX 中，使用标题通常分为两个部分：声明标题内容和实际输出标题。每个标题则由标题、作者、日期等部分组成。

声明标题、作者和日期分别使用 `\title`、`\author` 和 `\date` 命令。它们都带有一个参数，里面可以使用 `\\` 进行换行。标题的声明通常放在导言区，也可以放在标题输出之前的任何位置，例如：

```
\title{杂谈勾股定理\\——勾股定理的历史与现状}  
\author{张三\\九章学堂}  
\date{庚寅盛夏}
```

2-3-1

`\author` 定义的参数可以分行，一般第一行是作者姓名，后面是作者的单位、联系方式等。如果文档有多个作者，则多个作者之间用 `\and` 分隔，例如：

```
\author{张三\\九章学堂 \and 李四\\天元研究所}
```

2-3-2

张三	李四
九章学堂	天元研究所

`\date` 命令可以省略，如果省略，就相当于定义了 `\date{\today}`，即设置为当天的日期。`\today` 输出编译文档当天的日期，在英文文档类中默认使用英文（如 April 28, 2013），在中文 `ctexart` 等文档类中默认使用中文（如 2013 年 4 月 28 日）。使用 `ctex` 宏包可以用 `\CTEXoptions` 来设置 `\today` 的输出格式^[58]：

<code>\CTEXoptions[today=small]</code>	2013 年 4 月 28 日
<code>\CTEXoptions[today=big]</code>	二〇一三年四月二十八日
<code>\CTEXoptions[today=old]</code>	April 28, 2013

在声明标题和作者时，可以使用 `\thanks` 命令产生一种特殊的脚注，它默认使用特殊符号编号，通常用来表示文章的致谢、文档的版本、作者的详细信息等，例如：

2-3-3

```
\title{杂谈勾股定理\thanks{本文由九章基金会赞助。}}  
\author{张三\thanks{九章学堂讲师。}}\九章学堂
```

使用 `\maketitle` 命令可以输出前面声明的标题，在 1.2.2 节中我们已经看到它的使用，通常 `\maketitle` 是文档中 `document` 环境后面的第一个命令。整个标题的格式是预设好的，在 `article` 或 `ctexart` 文档类中，标题不单独成页；在 `report`, `book` 或 `ctexrep`, `ctexbook` 文档类中，标题单独占用一页。也可以使用文档类的选项 `titlepage` 和 `notitlepage` 来设置标题是否单独成页。

对于标准文档类，标题的格式是固定好的， \LaTeX 并没有提供更多的修改标题格式的命令和选项，如果只是修改字体，可以直接把字体命令放进 `\title`、`\author` 等命令中（参见 1.2.8 节）。对于更复杂的标题格式，由于标题在文档中只出现一次，而且不影响文档其他部分的内容，所以文档标题可以通过直接手工设置文字和段落格式排版得到，可以不使用 `\title`、`\maketitle` 等命令提供的功能。

单独成页的标题格式通常形式多变，可以在 `titlepage` 环境中排版。`titlepage` 环境提供没有页码的单独一页，并使后面的内容页码从 1 开始计数，例如：

2-3-4

```
% 手工排版的标题页  
\begin{titlepage}  
  \vspace*{\fill}  
  \begin{center}  
    \normalfont  
    {\Huge\bfseries 杂谈勾股定理}  
  
    \bigskip  
    {\Large\itshape 张三}  
  
    \medskip  
    \today  
  \end{center}  
  \vspace{\stretch{3}}  
\end{titlepage}
```

除了直接手工排版标题，或在声明标题时就加上了字体、字号等命令，也可以使用 `titling` 宏包详细设置标题、作者、日期在使用 `\maketitle` 时的输出方式。这个宏包适合模板作者使用，可参见宏包文档 Wilson [291]。

2.3.2 划分章节

\LaTeX 的标准文档类可以划分多层章节。在 1.2.2 节中, 我们只看到了 \section (节) 的层次, 事实上, 在 \LaTeX 中可以使用 6 到 7 个层次的章节, 如表 2.13 所示。

表 2.13 章节层次

层次	名称	命令	说明
-1	part (部分)	\part	可选的最高层
0	chapter (章)	\chapter	report, book 或 ctexrep, ctexbook 文档类的最高层
1	section (节)	\section	article 或 ctexart 类最高层
2	subsection (小节)	\subsection	
3	subsubsection (小小节)	\subsubsection	report, book 或 ctexrep, ctexbook 类 默认不编号、不编目录
4	paragraph (段)	\paragraph	默认不编号、不编目录
5	subparagraph (小段)	\subparagraph	默认不编号、不编目录

一个文档的最高层章节可以是 \part , 也可以不用 \part 直接使用 \chapter (对 book 和 report 等) 或 \section (对 article 等); 除 \part 外, 只有在上一层章节存在时才能使用下一层章节, 否则编号会出现错误。在 \part 下面, \chapter 或 \section 是连续编号的; 在其他情况下, 下一级的章节随上一节的编号增加会清零重新编号。

例如, 在 book 类中, 可以使用如下的提纲:

```

1 \documentclass{book}
2 \title{Languages}\author{someone}
3 \begin{document}
4 \maketitle
5 \tableofcontents
6 % 这里用缩进显示层次
7 \part{Introduction} % Part I
8   \chapter{Background} % Chapter 1
9 \part{Classification} % Part II
10  \chapter{Natural Language} % Chapter 2
11  \chapter{Computer Languages} % Chapter 3
12    \section{Machine Languages} % 3.1
    
```



```
13 \section{High Level Languages}           % 3.2
14 \subsection{Compiled Language}          % 3.2.1
15 \subsection{Interpretative Language}    % 3.2.2
16 \subsubsection{Lisp}
17 \paragraph{Common Lisp}
18 \paragraph{Scheme}
19 \subsubsection{Perl}
20 \end{document}
```

2-3-5

可以使用带星号的章节命令（如 `\chapter*`）表示不编号、不编目的章节。例如教材每章后面的习题往往不编号，也不必放在目录中，就可以用 `\section*{习题}` 开始习题这一节。

有时，我们希望在正文中的章节题目与在目录、页眉中的不同——正文中使用完整的长标题，目录和页眉使用短标题。可以给命令添加可选参数来做到这一点，如：

```
\chapter[展望与未来]{展望与未来：畅想新时代的计算机排版软件}
```

计数器 `secnumdepth` 控制除 `\part` 外，对章节进行编号的层次数，它的默认值为 3，也就是对 **book**、**report** 类编号到 `\subsection`；对 **article** 类编号到 `\subsubsection`（见表 2.13）。可以在导言区修改这个计数器的值来修改编号的层次数。

计数器 `tocdepth` 控制除 `\part` 外，对章节编入目录的层次数，它的默认值为 3（见表 2.13）。可以在导言区修改此计数器的值。

章节的计数器与其命令同名，如 `\chapter` 的计数器就是 `chapter`、`\section` 的计数器就是 `section`。直接重定义这些计数器的输出格式可以一定程度上的修改章节格式，详细的设置参见 2.3.4 节。

`\appendix` 命令用来表示附录部分的开始。命令 `\appendix` 后面的所有章（对于 **book**、**report** 等）或节（对于 **article** 等）都将改用字母进行编号：如编号的“Chapter 1”（中文文档类为“第一章”）改为“Appendix A”（中文文档类为“附录 A”）。例如某教材的习题解答作为附录的一章，可以用

```
% ...
\appendix
\chapter{习题解答}
% ...
```

标准文档类对附录部分章节格式是固定的，即适当修改了相应的章节格式。可以自己可以重定义 `\appendix` 命令来设置自己的附录格式（参见 2.3.4 节）。

第 2 章 组织你的文本

131

也可以使用 `appendix` 宏包^[286] 设置附录格式，它提供了更多的命令和选项来控制 `\appendix` 及相关命令的行为，这里不再赘述。

对于 `book` 或 `ctexbook` 类，还可以把全书划分为正文前的资料（`front matter`）、正文的主要部分（`main matter`）、后面的附加材料（`back matter`）。这是由 `\frontmatter`、`\mainmatter` 和 `\backmatter` 控制的，例如：

```
1 \documentclass{ctexbook}
2 \title{语言}
3 \author{张三 \and 李四}
4 \begin{document}
5
6 \frontmatter
7 \maketitle
8 \tableofcontents
9 \chapter{序} % 不编号
10 % ...
11
12 \mainmatter % 页码重新计数
13 \chapter{自然语言}
14 % ...
15 \chapter{计算机语言}
16 % ...
17
18 \backmatter
19 \chapter{进一步的参考资料} % 不编号
20 % ...
21 \end{document}
```

这三个命令都会使用 `\clearpage` 或 `\cleardoublepage` 另起新页（参见 2.2.8 节、5.3.1 节），输出以前未处理的浮动图表。`\frontmatter` 会令页码按小写罗马数字编号，并关闭 `\chapter` 的编号；`\mainmatter` 会令页码按阿拉伯数字编号；`\backmatter` 会关闭 `\chapter` 的编号。

2.3.3 多文件编译

对一篇只有几页纸的文章，把所有的内容都放进一个 $\text{T}_{\text{E}}\text{X}$ 源文件就足够了。但如果要排版更长的内容，例如与本书篇幅相当的文档，单一文件的编译方式就不那么方便了。更好的方式是按文档的逻辑层次，把整个文档分成多个 $\text{T}_{\text{E}}\text{X}$ 源文件，这样文档的内容更便于检索和管理，也适合大型文档的多人协同编写。

$\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ 提供的 $\backslash\text{include}\{\text{文件名}\}$ 命令可用来导入另一个文件的内容作为一个章节，文件名不用带 $.tex$ 扩展名。 $\backslash\text{include}$ 命令会在之前和之后使用 $\backslash\text{clearpage}$ 或 $\backslash\text{cleardoublepage}$ 另起新页（参见 2.2.8 节），同时将这个文件的内容贴到 $\backslash\text{include}$ 命令所在的位置^①。于是，可以按下面的方式组织一本书：

```
1 % languages.tex
2 %   整个文档的主文件
3 \documentclass{ctexbook}
4 \title{语言}
5 \author{张三 \and 李四}
6 \begin{document}
7 \maketitle
8 \tableofcontents
9 \include{lang-natural}
10 \include{lang-computer}
11 \end{document}
```

```
1 % lang-natural.tex
2 %   “自然语言”一章，不能单独编译
3 \chapter{自然语言}
4 .....
```

```
1 % lang-computer.tex
2 %   “计算机语言”一章，不能单独编译
3 \chapter{计算机语言}
4 .....
```

^① 这个命令实际完成的工作还要多一些，如写不同的辅助文件、判断个别例外不读入等。


第2章 组织你的文本

133

这样，就把一本书籍划分成了三个文件：一个主文件 `languages.tex`，它是文档的中心，编译文档时只要对主文件编译即可；两章内容的文件 `lang-natural.tex` 和 `lang-computer.tex`，每个文件只有一章的内容。如果这本书由张三和李四分工编写，在他们共同写好 `languages.tex` 这个提纲以后，就可以分头编写两章的内容了，二人的工作可以互不影响。

使用 `\include` 划分文档以后有一个特别的便利，就是可以通过修改主文件的几行来选择编译整个文档的某一章或某几章。当然可以把不要的章节注释掉来达到这个目的，不过更好的办法是使用 `\includeonly{<文件列表>}` 命令，其中 `<文件列表>` 是用英文逗号隔开的若干文件名。在导言区使用 `\includeonly` 命令以后，只有文件列表中的文件才会被实际地引入主文件。更好的是，如果以前曾经完整地编译过整个文档，那么在使用 `\includeonly` 选择编译时，原来的章节编号、页码、交叉引用等仍然会保留为前一次编译的效果，例如：

```
1 % languages.tex
2 % 整个文档的主文件
3 \documentclass{ctexbook}
4 \title{语言}
5 \author{张三 \and 李四}
6 \includeonly{lang-natural}      % 只编译“自然语言”一章
7 \begin{document}
8 \maketitle
9 \tableofcontents
10 \include{lang-natural}
11 \include{lang-computer}
12 \end{document}
```

 使用 `\include` 命令需要注意的是，最好不要在子文件中重新定义计数器、声明新字体，否则在使用 `\includeonly` 时，会因为找不到出现在辅助文件中而在源文件中缺失的计数器而出错。

比 `\include` 命令更一般的是 `\input` 命令，它直接把文件的内容复制到 `\input` 命令所在的位置，不做其他多余的操作。`\input` 命令接受一个文件名参数，文件名可以带扩展名，也可以不带扩展名（此时认为扩展名是 `.tex`）。例如，如果一个文档的导言区设置非常多，可以把导言区的设置都放在一个 `preamble.tex` 文件中，然后在主文件中就可以这样写：

```
1 % main.tex
```

```
2 % 主文档
3 \documentclass{ctexart}
4 \input{preamble} % 复杂的导言区设置
5 \begin{document}
6 ..... (文档的内容)
7 \end{document}
```

除了导言区，经常也把复杂图表代码放在一个单独的文件中，然后在主文件中使用 `\input` 命令插入，这样可以使文档的正文部分看起来比较清爽，图表的代码也可以被另外的专用主文档引入单独进行测试。另外，如果需要引入的文件只是 `article` 中的一节，不需要多余的换页时，也可以用 `\input` 命令代替 `\include` 命令引入不换页的小的章节文件。

在被引入的文件末尾，可以使用 `\endinput` 命令显式地结束文件的读入。在 `\endinput` 命令的后面，就可以直接写一些注释性的文字，而不必再加注释符号，例如：

```
1 % lang-natural.tex
2 \chapter{自然语言}
3 .....
4 \endinput
5 这是“自然语言”一章，不能单独编译。要编译文档，直接编译主文件：
6 xelatex languages.tex
```

主文件中的 `\end{document}` 命令也有类似的功能。

 对于大型文档，在编写中有时只需要 \LaTeX 编译程序检查语法，并不需要实际输出。此时可以使用 `syntonly` 宏包的 `\syntaxonly` 命令^[168]：

```
1 % languages.tex
2 % 整个文档的主文件
3 \documentclass{ctexbook}
4 \usepackage{syntonly}
5 \syntaxonly % 只检查语法，不输出 DVI/PDF 文件
6 \begin{document}
7 .....
8 \end{document}
```

第2章 组织你的文本

135

使用 `syntonly` 宏包编译文档时不输出 DVI/PDF 文件，速度比直接编译输出要快一些，可以节约文档编写的时间。

2.3.4 定制章节格式



L^AT_EX 标准文档类 `book`, `report`, `article` 的章节格式是固定的，章节标题用什么字体、字号，前后的间距如何，用怎样的编号方式等，都是预定好的，并没有提供直接进行控制的命令，一般都要通过其他的宏包完成相关的定制。

`ctex` 宏包的三个文档类 `ctexbook`, `ctexrep`, `ctexart` 使用的默认章节格式与英文标准文档类的格式略有区别。`ctex` 宏包还提供了 `\CTEXsetup` 命令来设置章节标题的格式^[58, § 2.5]，其语法格式如下：

```
\CTEXsetup[(选项1)=(值1), (选项2)=(值2), ...]{对象类型}
```

其中 (对象类型) 可以是 `part`, `chapter`, `section`, `subsection`, `subsubsection`, `paragraph`, `subparagraph`; (选项1), (选项2) 等是设置的选项，包括 `name`, `number`, `format`, `nameformat`, `numberformat`, `titleformat`, `aftername`, `beforeskip`, `afterskip`, `indent`；而 (值1), (值2) 等是对应选项的设置内容。

下面逐条解释 `\CTEXsetup` 的选项：

☞ **name**={前名}, {后名}

用来设置章节的名字，包括章节编号前面和后面的词语，前后两个名字之间用西文逗号分开，例如：

```
\CTEXsetup[name={第, 节}]{section}
```

将设置 `\section` 的标题的名字类似“第 1 节”。

☞ **number**={编号格式}

相当于设置章节的计数器的 `\the{counter}` 命令，例如：

```
\CTEXsetup[number={\chinese{section}}]{section}
```

如果配合前面章节名的设置，将产生类似“第一节”的名字。

☞ **format**={格式}

用于控制章节标题的全局格式，作用域为章节名字和随后的标题内容。常用于控制章节标题的对齐方式，例如设置节标题左对齐，整体粗体：

*本节内容初次阅读可略过。

```
\CTEXsetup[format={\raggedright\bfseries}]{section}
```

☞ **nameformat={格式}**

类似 format，控制章节名和编号的格式，不包括后面的章节标题。

☞ **numberformat={格式}**

类似 format，仅控制编号的格式，不包括前后章节名和章节标题。

☞ **titleformat={格式}**

类似 format，仅控制章节标题的格式，不包括前面的章节名和编号。

☞ **aftername={代码}**

控制章节名和编号（如“第一章”）与后面标题之间的内容。通常设置为一定间距的空格，或换行（换行仅对 chapter 和 part 有效），例如：

```
\CTEXsetup[aftername={\\vspace{2ex}}]{part}
```

☞ **beforeskip={长度}**

控制章节标题前的垂直距离。

☞ **afterskip={长度}**

控制章节标题后的垂直距离。

☞ **indent={长度}**

控制章节标题的缩进长度。

ctex 的文档类中，\CTEXsetup 的选项的默认值可进一步参见文档 ctex.org [58]。

在 \CTEXsetup 命令中，可以使用 += 代替 =，表示在默认选项的基础上增加格式设置。例如，如果要让每小节 \subsection 的标题使用仿宋体，就可以用命令：

```
\CTEXsetup[titleformat+={\fansi}] {subsection}
```

ctex 宏包提供的 \CTEXsetup 只能用于中文文档，而且支持的格式还不够多变自由，难以生成更复杂的章节标题格式。在西文文档中，或是要设置更复杂的章节标题（如本书），可以使用 titlesec 宏包。这里只对一些初级命令做简单介绍，详细的设置请参见 Bezos [27]。

titlesec 宏包使用时可以带几个可选参数，用来全局地修改标题的字体、对齐方式。选项

rm sf tt bf up it sl sc

用来设置标题使用的字体族、字体系列和字体形状，默认是 bf，即普通字体的粗体系列。选项

第2章 组织你的文本

137

big medium small tiny

用来设置标题的字号，默认是最大的 big，最小号的 tiny 则使所有标题字体与正文大小相同，medium 和 small 介于最大和最小之间。选项

raggedright center raggedleft

用来设置标题的对齐方式，例如：

```
\usepackage[sf,bf,it,centering]{titlesec}
```

2-3-7

将设置章节标题的字体为 \sffamily\bfseries\itshape，居中。

宏包选项 compact 会使章节标题前后的垂直间距比较紧。

\titlelabel{<代码>} 命令用来设置编号标签的格式，<代码> 中可以使用 \thetitle 指代 \thesection、\thesubsection 等命令。对标准文档类，默认设置是：

```
\titlelabel{\thetitle\quad}
```

可以修改为需要的格式，例如：

```
\titlelabel{\S~\thetitle\quad}
```

2-3-8

将在所有的章节编号前加符号 §。

带星号的 \titleformat* 命令则可以作为宏包选项的补充，用来设置某一级标题的格式，例如：

```
\titleformat*{\section}{\Large\itshape\centering}
```

2-3-9

将使节标题使用 \Large 字号的意大利体并居中。这里的设置将会覆盖默认的和在宏包选项中出现的全局设置，因此一般对字体、字号、对齐方式等都要进行设置。

titlesec 的其他功能包括使用不带星号的 \titleformat 命令对章节格式做更详细的设置，使用 \titlespace 命令设置章节标题的间距，增添横线，控制页面版式（可代替 fancyhdr，参见 2.4.3 节），控制章节分页，增加新的章节层次等功能，篇幅所限，这里就不再一一介绍了。



练习

2.6 试试看，下面代码的作用是什么？

```
\CTEXsetup[name={\S,}, number={\arabic{section}}, aftername={---},  
format={\raggedright}, nameformat={\Large\bfseries},  
titleformat={\LARGE\sffamily}, indent={1pc}]{section}
```

使用 `titlesec` 宏包的功能重新实现上面的设置（可以只考虑 `\section`）。

2.7 本书的章节格式就是用 `titlesec` 宏包设置的。查阅文档，试试你能否仿做出本书的章节格式。

2

2.4 文档类与整体格式设计

这一节，我们主要将注意力集中在导言区的代码，讨论文档的一些全局设置，特别是版面设计的相关内容。

2.4.1 基本文档类和 `ctex` 文档类

文档类（document class）是 $\text{\LaTeX} 2_{\epsilon}$ 中基本的格式组织方式。文件类通常都是针对某一类格式相近的文档设计的，从适用范围广泛的“文章”、“书籍”到非常专门的“某大学博士论文模板”，不一而足。选定了一个文档类，就相当于选定了一集 \LaTeX 格式，可以在一个规范的框架下进行文档编写。

本书的大部分内容都是基于 $\text{\LaTeX} 2_{\epsilon}$ 的基本文档类和 `ctex` 文档类叙述的。`ctex` 文档类是由 \CTeX 中文社区^①组织编写的 $\text{\LaTeX} 2_{\epsilon}$ 基本文档类的中文对应物，它是在 $\text{\LaTeX} 2_{\epsilon}$ 基本文档类的基础上编写的。 $\text{\LaTeX} 2_{\epsilon}$ 基本文档类和 `ctex` 文档类主要提供了文档的整体框架（如 `\maketitle`、`\section` 等命令）、基本设置（如默认字号、默认的中文字体）、基本工具（如 `enumerate` 等环境）等，它们提供了最基本的通用文档格式，但并不对文档的适用性做过多限制。下面就对这两组适用最广的文档类做一简单介绍。

$\text{\LaTeX} 2_{\epsilon}$ 基本文档类主要有三个：`article`、`report` 和 `book`。这三个基本文档类分别设计用来编写小篇幅的文章、中篇幅的报告和长篇幅的书籍。三个文档类的格式都很简单，提供的命令也相差不多，只有少数区别，如 `article` 没有 `\chapter`、`\mainmatter` 只有 `book` 类才有等。基本文档类的许多格式是可以通过选项调整的，例如：

2-4-1

```
\documentclass[a4paper,titlepage]{article}
```

将设置文档为 A4 纸大小，标题单独占一页。

基本文档类预定义的所有选项总结见表 2.14。

单面文档的奇偶数页面是相同的，双面文档则不同。对双面印制的文档，通常在页面左侧装订，翻开后奇数页一般在右边，偶数页在左边，因而在页面边距、页眉页脚、边注位置等方面都与单面文档有所区别，大多取左右对称的设置。如果选定 `twoside`

^① <http://bbs.ctex.org/>

表 2.14 \LaTeX 2_ε 标准文档类的选项

类型	选项	说明
纸张大小	a4paper	21.0 cm × 29.7 cm
	a5paper	14.8 cm × 21.0 cm
	b5paper	17.6 cm × 25.0 cm
	letterpaper	8.5 in × 11 in （默认值）
	legalpaper	8.5 in × 14 in
	executivepaper	7.25 in × 10.5 in
纸张方向	landscape	横向，即长宽交换 （默认无，即为纵向页面）
单双面	oneside	单面 （article, report 默认值）
	twoside	双面，奇偶页面版式不同，左右对称 （book 默认值）
字号大小	10pt	正文字号为 10 pt，\large 等命令与之相衬（默认值）
	11pt	正文字号为 11 pt
	12pt	正文字号为 12 pt
分栏	onecolumn	单栏 （默认值）
	twocolumn	双栏
标题格式	titlepage	标题独自成页 （report, book 默认值）
	notitlepage	标题不独自成页 （article 默认值）
章格式	openright	每章只从奇数页开始 （book 默认值）
	openany	每章可从任意页开始 （article, report 默认值）
公式编号	leqno	公式编号在左边 （默认无，即公式编号在右边）
公式位置	fleqn	公式左对齐，固定缩进 （默认无，即公式居中）
草稿设置	draft	草稿，会把行溢出的盒子着重显示为黑块
	final	终稿 （默认值）
参考文献	openbib	每条文献分多段输出 （默认无）

模式，可以使用 `openright` 使每个 `\part` 和 `\chapter` 都只出现在右边的页面（奇数页），之前不足的页面用空白补足。

纸张大小、方向的设置可进一步参见 2.4.2 节，字号参见 2.1.4 节，分栏参见 2.4.4 节，标题参见 2.3.1 节 `titlepage` 环境的说明，`titlepage` 选项也会使摘要独自成页，公式设置参见 4.5 节，参考文献参见 3.3.5 节。

基本文档类的默认选项总结如下：

<code>article</code>	<code>letterpaper,10pt,oneside,onecolumn,notitlepage,final</code>
<code>report</code>	<code>letterpaper,10pt,oneside,onecolumn,titlepage,openany,final</code>
<code>book</code>	<code>letterpaper,10pt,twoside,onecolumn,titlepage,openright,final</code>

\LaTeX 在文档类中的选项是全局设定的，不仅会影响文档类的代码，也会影响文档所使用的宏包。例如文档类中的纸张选项也会影响 `geometry` 宏包，草稿设置选项也会影响插图的 `graphicx` 宏包（参见 5.2.1 节）等。

`ctex` 宏包^[58] 提供了三个文档类：`ctexart`、`ctexrep` 和 `ctexbook`，分别与三个标准文档类对应，用来编写中文短文、中文报告和中文书籍。除了三个文档类，`ctex` 宏包还包括 `ctex.sty` 和 `ctexcap.sty` 两个格式文件，以及上述文档类和格式文件的 UTF8 编码变体。

`ctex.sty` 提供基本的中文输出支持，`ctexcap.sty` 则提供 `ctex.sty` 的全部功能和英文标题的汉化，几个文档类则在 `ctexcap` 的基础上进一步设置默认的字号大小为中文字号。在大多数情况下，我们都直接使用 `ctex` 提供的文档类，但在一些非标准的文档类中，则可以按需要使用 `ctexcap` 或 `ctex` 格式。例如，在编写宏包文档的 `ltxdoc` 文档类（基于标准文档类）中，可以使用 `ctexcap` 支持中文^①：

```
\documentclass{ltxdoc}
\usepackage{ctexcap}
\zihao{5}
\begin{document}
.....
\end{document}
```

而在已经失去标准文档类原有结构的个人简历文档类 `moderncv` 中，就可以使用 `ctex` 支持中文：

^① 在更新版本中的 `ctex` 宏包将简化不同格式的使用，`ctex` 格式将默认带有 `ctexcap` 及文档类的汉化、字号设置等功能，此时 `ctexcap` 就不再需要了。

```
\documentclass{moderncv}
\usepackage{ctex}
\zihao{-4}
\begin{document}
.....
\end{document}
```

2-4-2

2

表 2.14 中列出的标准文档类的选项也可在 `ctex` 文档类中使用。`ctex` 宏包及文档类的其他选项（这里去掉了专用于旧的中文处理方式 `CJK` 和 `CCT` 的选项）见表 2.15。

表 2.15 `ctex` 宏包及文档类的选项

类型	选项	说明
字号大小	<code>c5size</code>	正文五号字 （仅用于文档类，默认值）
	<code>cs4size</code>	正文四号字 （仅用于文档类）
章节标题	<code>sub3section</code>	使 <code>\paragraph</code> 标题单独占一行 （仅用于 <code>ctexcap</code> 及文档类，默认无）
	<code>sub4section</code>	使 <code>\paragraph</code> 和 <code>\subparagraph</code> 标题都单独占一行 （仅用于 <code>ctexcap</code> 及文档类，默认无）
中文编码	<code>GBK</code>	使用 GBK 编码，但对 $\text{X}\text{\LaTeX}$ 无效
	<code>UTF8</code>	使用 UTF8 编码
$\text{X}\text{\LaTeX}$ 中文字库	<code>nofonts</code>	不设定中文字体，需要在文中自己定义
	<code>winfonts</code>	设定 Windows 操作系统预装的中文字体 （默认值）
	<code>adobefonts</code>	设定 Adobe 中文字体
排版风格	<code>cap</code>	使用中文标题、编号、日期等 （默认值）
	<code>nocap</code>	保留英文标题、编号、日期等
	<code>punct</code>	启用标点压缩 （默认值）
	<code>nopunct</code>	关闭标点压缩
	<code>indent</code>	标题后首行缩进 （默认值）
宏包兼容	<code>noindent</code>	标题后首行不缩进
	<code>fancyhdr</code>	调用 <code>fancyhdr</code> 并与之兼容 （默认无）
	<code>hyperref</code>	调用 <code>hyperref</code> 并自动设置防止标签乱码选项 （默认无）

续表

类型	选项	说明
	fntef	调用 CJKfntef 并定义等价的 \CTEX 开头的命令 (默认无)

\paragraph 和 \subparagraph 的标题在标准文档类中与其后的正文段在同一行, 使用 sub3section 和 sub4section 选项可以使它们的标题单独占用一行。

ctex 宏包有选择编码的选项 GBK 和 UTF8, 但它们主要是为旧方案中使用 CJK 宏包处理中文所准备的; 在 Xe_{La}TeX 编译的文档中, 总是相当于使用了 UTF8 编码选项。如果需要使用 Xe_{La}TeX 处理 GBK 编码的中文文档, 可以在每个文件的开头使用 Xe_{La}TeX 原始命令 \XeTeXinputencoding 选择编码:

```
\XeTeXinputencoding "GBK"
\documentclass{ctexart}
\begin{document}
GBK 编码的中文文档。
\end{document}
```

2-4-3

ctex 宏包以前对 UTF-8 编码的支持是由另外单独的文件而不是选项实现的, 现在为了兼容也保留了 ctexutf8, ctexcaputf8 两个格式和 ctexartutf8, ctexreputf8, ctexbookutf8 三个文档类, 相当于加了 UTF8 选项。

关于中文字库的选项请参见 2.1.3.2 节, 标点压缩请参见 2.1.1.2 节, fancyhdr 选项请参见 2.4.3 节, fntef 选项请参见 2.1.3.3 节。关于 ctex 宏包提供的其他功能可进一步参考宏包的文档 ctex.org [58], 这里不再赘述。

2.4.2 页面尺寸与 geometry

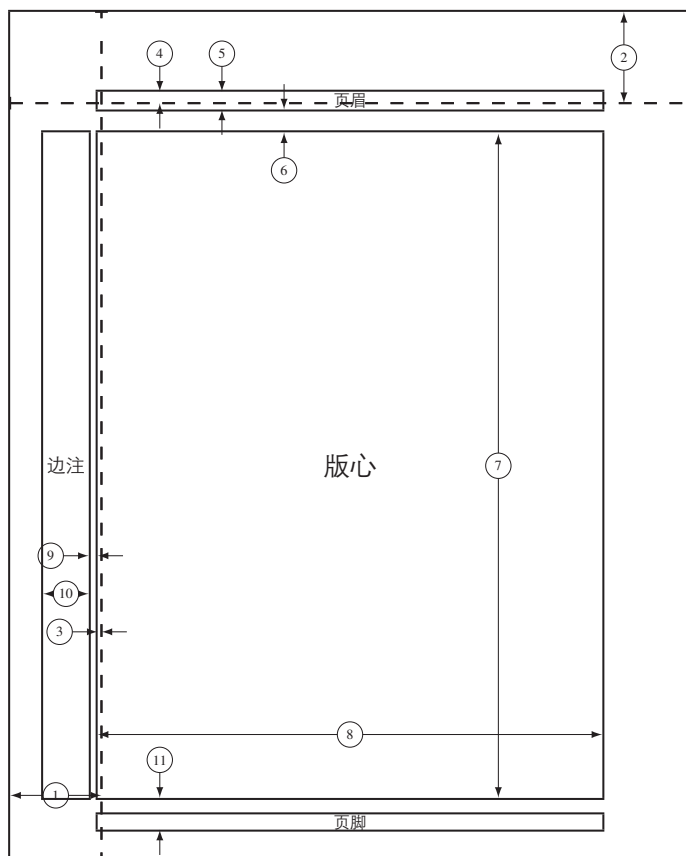


在文档类的选项中, 可以选择几种常见的排版所用的纸张页面大小。不过, 实际的排版往往要求设置更为复杂的页面尺寸参数, 下面我们就来考量有关页面尺寸的设置。

如图 2.4 所示^①, L^AT_EX 的页面尺寸布局是由一系列长度变量控制的。其中 \paperwidth 和 \paperheight 是纸张的宽和高; \hoffset 和 \voffset 是减去一英寸

*本节内容初次阅读可略过。

① 这个页面示意图是利用 layout 宏包^[157]绘制的。



- | | |
|-----------------------------|------------------------------|
| 1 一英寸 + \hoffset | 2 一英寸 + \voffset |
| 3 \evensidemargin = -3.98pt | 4 \topmargin = -9.67pt |
| 5 \headheight = 14.05pt | 6 \headsep = 18.07pt |
| 7 \textheight = 525.87pt | 8 \textwidth = 398.34pt |
| 9 \marginparsep = 7pt | 10 \marginparwidth = 36.14pt |
| 11 \footskip = 25.29pt | \marginparpush = 5pt (未显示) |
| \hoffset = 45.52pt | \voffset = 42.68pt |
| \paperwidth = 534.91pt | \paperheight = 668.64pt |

图 2.4 本书的页面尺寸设置示意图（偶数页）

的页面整体偏移量^①；`\textwidth`和`\textheight`是版心的宽和高；`\topmargin`是额外的上边距，`\oddsidemargin`和`\evensidemargin`在双面模式下分别是奇数页和偶数页的额外左边距（对单面模式，页偶数页都是`\oddsidemargin`）；`\headheight`是页眉高，`\headsep`是页眉与版心间距；`\marginparwidth`是边注宽，`\marginparsep`是边注与版心间距，`\marginparpush`是相邻边注的最小间距；`\footskip`是页脚基线与正文最后一行基线的间距。

可以直接设置图 2.4 中的长度变量来控制页面尺寸，不过设置这些参数时必须非常小心，比如当我们想要设置页面的右边距（不含边注）为 3 cm 时，就必须按下面的关系式做一番计算：

$$1\text{ in} + \text{\hoffset} + \text{\oddsidemargin} + \text{\textwidth} + \text{右边距} = \text{\paperwidth}$$

从中反解出需要调整的参数值来。在涉及双面文档时这些计算和调整的工作尤为麻烦。

`geometry` 宏包把我们从 \LaTeX 众多相互影响的页面参数中解救出来，提供了一个设置页面参数相对简单直观的用户界面。例如，设置右边距的工作就可以简化为一句 `\geometry{right=3cm}` 的命令。

`geometry` 主要提供两种设置页面的方式，一是作为宏包的选项，例如设置纸张大小和左右边距：

2-4-4

```
\usepackage[a4paper,left=3cm,right=3cm]{geometry}
```

二是使用 `\geometry` 命令，内容和宏包选项一样：

2-4-5

```
\usepackage{geometry}  
\geometry{a4paper,left=3cm,right=3cm}
```

`geometry` 宏包支持方便的 `(参数) = (值)` 的语法，并且可以根据命令给出的页面距离值和其他参数的默认值，自动计算原始 \LaTeX 的页面参数进行设置，十分方便。

图 2.5 来自 `geometry` 的文档 Umeki [272]，它展示了 `geometry` 宏包最常用的距离参数名称。这些参数与前面给出的 \LaTeX 标准的页面长度变量名字相同或更为简化。同时，`geometry` 宏包也支持 `a4paper`，`landscape` 这样的纸张参数，并且可以在输出 PS/PDF 时也对页面进行剪裁。

使用 `geometry` 宏包不需要给出完全的参数设置，有时甚至可以相当模糊，比如可以用 `centering` 选项设置版心居中，使用 `scale = (比例)` 选项设置版心占页面长度的比例，使用 `ratio = (比例)` 选项设置版面边距占页面长度的比例，使用 `lines = (行数)` 设置版心高度在默认字体和行距下能容纳的文本行等。例如，可以不使用任何长度就设置好文档的页面参数：

^① 这里的一英寸是通常打印机驱动默认设置的页面与纸张边距。

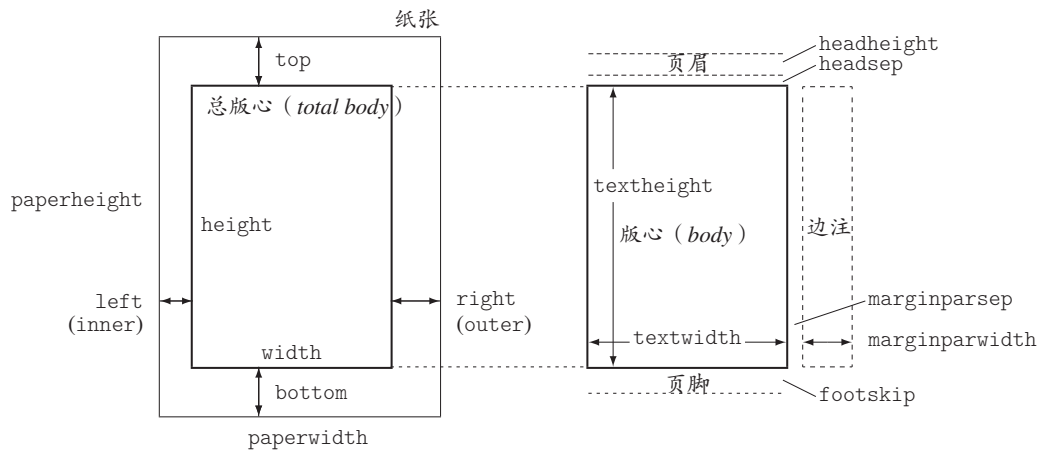


图 2.5 `geometry` 宏包的距离参数名称。默认有 `width = textwidth` 及 `height = textheight`。`left`, `right`, `top` 和 `bottom` 是左、右、上、下 4 个边距。使用 `twoside` (双面) 选项后, `left` 和 `right` 的方向交换, 即分别表示 (靠近装订线的) 内部和外部, 此时可以分别用 `inner` 和 `outer` 表示

```
\geometry{b5paper,scale=0.8,centering}
```

除了上面所介绍的, `geometry` 提供的参数选项还有很多, 如用来调试的 `showframe` 选项、设置编译引擎的 `pdftex`, `xetex` 选项等, 读者可进一步参考宏包文档 `Umeki` [272], 这里不再赘述。

2.4.3 页面格式与 `fancyhdr`



2.4.2 节中已经对页面的大小尺寸有了一个整体的设置, 本节来进入页面, 看看页面的格式设置——也就是页眉、页眉、页脚等地方的设计。

页码的计数器是 `page`, 它会随着文档自动计数。`LaTeX` 提供了一个简单的命令 `\pagenumbering{<格式>}` 来设置页码的编号方式, 这个命令会令页码重新从 1 开始按 `<格式>` 参数进行计数, 例如:

```
\pagenumbering{roman}
```

2-4-6

*本节内容初次阅读可略过。

相当于设置 `\thepage` 为 `\roman{page}`, 并设置计数器值为 1。book 类中的 `\frontmatter` 等命令就有控制页码编号的作用, 而在 report 和 article 类中就必须用 `\pagenumbering` 了。

LaTeX 提供了多种预定义的页面风格 (page style), 它们控制页眉页脚的整体风格设置:

empty	没有页眉页脚;
plain	没有页眉, 页脚是居中的页码;
headings	没有页脚, 页眉是章节名称和页码;
myheadings	没有页脚, 页眉是页码和用户自定义的内容。

可以用 `\pagestyle{风格}` 整体设置页面风格, 也可以用 `\thispagestyle{风格}` 单独设置当前页的风格。标准文档类中, book 类默认使用 headings 风格, report 和 article 默认使用 plain 风格; 中文的几个 ctex 文档类则都默认使用 headings 风格^①, 例如, 可以在 article 类中设置带章节名称的页眉:

2-4-7

```
\documentclass{article}
\pagestyle{headings}
```

又如, 可以在插入大幅图片的页面使用 plain 风格, 取消复杂的页眉:

2-4-8

```
\begin{figure}[p]
  \thispagestyle{plain}
  .....
\end{figure}
```

LaTeX 已经对一些必要的地方自动设置好了页面风格。例如在标题页 (包括手工或自动由 `\maketitle` 生成的 titlepage 环境), 会使用 empty 风格禁用所有页眉页脚; 而在不单独成页的 `\maketitle`, 单独成页的 `\part`, 以及 `\chapter` 命令所在的一页, 则使用 plain 风格只显示页码: 这些都是排版中的一些定式。

headings 和 myheadings 两种风格是由标准文档类定义的, 它们的表现其实基本相同, 都是在页眉显示页码和一些文字。不同的是, headings 风格的页眉内容不能改变, 它是由 `\chapter`、`\section` 等命令自动生成的 (如图 2.6 所示), 而 myheadings 风格的页眉可以由用户自己使用 `\markright` 和 `\markboth` 命令设置:

单面文档 (oneside)	<code>\markright{页眉文字}</code>
双面文档 (twoside)	<code>\markboth{左面页眉}{右面页眉}</code>

^① 如果使用了 fancyhdr 选项, 则为 fancy 风格。

章（节）标题	页码
单面版心	

(a) 单面文档

页码	章标题	节标题	页码
双面偶数页 （左页）版心		双面奇数页 （右页）版心	

(b) 双面文档

图 2.6 标准文档类的 headings 页面风格

这里“右面”指奇数页，“左面”指偶数页，因为当双面文档在侧面装订好后，翻开的页面正好是右奇左偶，单面文档的所有页面都认为是右面。`\markright` 和 `\markboth` 命令实际会修改 `\leftmark` 和 `\rightmark` 两个宏的内容，并在页眉处输出。例如，如果想让文章在每页的页眉显示作者的名字，就可以使用：

```
\documentclass{ctexart}
\pagestyle{headings}
\markright{张三}
```

2-4-9

\LaTeX 和标准文档类提供的页面风格非常朴素，而且不能做进一步的格式修改。为此，`fancyhdr` 宏包提供了新的页面风格 `fancy`，以及一系列设置的命令，用作标准 \LaTeX 的 `myheadings` 及 `\markboth`, `\markright` 机制的扩展。

`fancyhdr` 的 `fancy` 页面风格把页面的页眉和页脚都分成左、中、右 3 个部分，因而一个页面就有 6 个部分。对于双面文档，则还分奇数页和偶数页，即有 12 个部分（见图 2.7）。

页眉左	页眉中	页眉右
版心内容		
页眉左	页脚中	页脚右

图 2.7 `fancyhdr` 宏包的 `fancy` 页面风格

图 2.7 中的各个部分可以用下列命令进行设置修改：

<code>\lhead{内容}</code>	设置页眉左
<code>\chead{内容}</code>	设置页眉中
<code>\rhead{内容}</code>	设置页眉右
<code>\lfoot{内容}</code>	设置页脚左
<code>\cfoot{内容}</code>	设置页脚中
<code>\rfoot{内容}</code>	设置页脚右
<code>\fancyhead[位置]{内容}</code>	设置页眉，位置可以是 E、O 与 L、C、R 的组合
<code>\fancyfoot[位置]{内容}</code>	设置页脚，位置可以是 E、O 与 L、C、R 的组合
<code>\fancyhf[位置]{内容}</code>	设置页眉及页脚，位置可以是 H、F 与 E、O 与 L、C、R 的组合

这里，`\fancyhead`、`\fancyfoot` 和 `\fancyhf` 命令可以带表示位置的可选参数，其中 H、F 分别表示页眉（header）和页脚（footer）；E、O 分别表示双面文档的偶数页（even page）和奇数页（odd page），单面文档仅奇数页有效；L、C、R 分别表示左（left）、中（center）、右（right）。位置参数可以任意组合，多个参数用逗号分隔。如果省略位置参数，则表示所有的页眉、页脚。例如：

```
\documentclass[twoside]{ctexrep}
\usepackage{fancyhdr}
\pagestyle{fancy}      % 使用 fancy 风格
\fancyhf{}             % 清除所有页眉页脚
\cfoot{\thepage}       % 页脚居中页码
\fancyhead[CO]{张三}   % 奇数页居中页眉作者名
\fancyhead[CE]{论语言} % 偶数页居中页眉文章题目
\fancyfoot[RO,LE]{\heartsuit}
                        % 奇数页脚右，偶数页脚左（即外侧）装饰符号
```

2-4-10

在 fancy 页面风格的设置中，可以在页眉页脚的内容中使用 `\leftmark` 和 `\rightmark` 命令，它们的意义与 headings 风格中的页眉相同，即为文档的章节标题内容：article 只有 `\rightmark` 是节标题；report 和 book 的 `\leftmark` 是章标题，`\rightmark` 是节标题。事实上，fancy 风格的默认设置就是：

```
\fancyhead[LE,RO]{\slshape \rightmark}
\fancyhead[LO,RE]{\slshape \leftmark}
\fancyfoot[C]{\thepage}
```

在 `ctex` 宏包提供的文档类中, 可以使用 `fancyhdr` 选项, 表示使用 `fancyhdr` 宏包及 `fancy` 页面风格, 因而例 2-4-10 中的前几行也可以简化为:

```
\documentclass[twoside,fancyhdr]{ctexrep}
\fancyhf{}
% .....
```

2-4-11

2

除了页眉页脚的内容, `fancy` 页面风格还会给页眉和页脚加一条横线。可以重定义宏 `\headrulewidth` 和 `\footrulewidth` 来修改页眉线和页脚线的宽度, 如果宽度为零就是没有页眉页脚线, 注意它们只是文本宏而不是长度变量, 如:

```
\renewcommand\headrulewidth{0.6pt} % 默认为 0.4pt
\renewcommand\footrulewidth{0.6pt} % 默认为 0pt
```


2-4-12

使用 `fancyhdr` 还可以使用 `\fancypagestyle` 命令重定义原有的页面风格, 通常可以用它来重定义 `plain` 风格, 这样在每章的第 1 页等位置也可以使用特殊的页面风格, 如:

```
\fancypagestyle{plain}{%
  \fancyhf{}
  \cfoot{--\textit{\thepage}--} % 改变页码形状
  \renewcommand\headrulewidth{0pt} % 无页眉线
  \renewcommand\footrulewidth{0pt} % 无页脚线
}
```

2-4-13

`fancyhdr` 宏包的功能大致就是这么多, 有关页面设置的更多命令和技巧, 可参见 `fancyhdr` 的文档 van Oostrum [190]。

 `titlesec` 宏包使用 `pagestyles` 选项时, 也提供了与 `fancyhdr` 类似的命令, 如 `\sethead`, `\setfoot`, `\renewpagestyle` 等, 可以完全代替 `fancyhdr` 的功能, 可参见文档 Bezos [27, § 5], 这里不再赘述。

2.4.4 分栏控制与 `multicol`

给文档类加 `twocolumn` 选项就可以使文档双栏排版。双栏排版的文档通常比较节省纸张, 较短的行在阅读时也比较省力。书籍的索引默认就是双栏排版的, 许

多期刊都使用双栏排版, 参考文献等也常常分栏。例如:

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


分栏也可以在正文中使用命令切换。`\twocolumn` 进入双栏模式, `\onecolumn` 进入单栏模式, 两个命令都会先使用 `\clearpage` 换页, 并不产生一页之内单双栏混合的效果。例如, 可以在书籍的某一章:

```
\twocolumn
\chapter{双栏的一章}
\onecolumn
```

`\twocolumn` 命令可以带一个可选参数, 在新开始的双栏页面顶部插入一部分单栏的内容。这个功能特别适合有通栏标题的双栏文章, 例如:

```
% 文档类选项并不使用双栏
\documentclass{article}
\title{Languages}
\author{someone}
\begin{document}
% 通栏标题
\twocolumn[\maketitle]
% 双栏的正文
blah blah blah...
\end{document}
```

在双栏模式下, `\newpage` 和 `\pagebreak` 只表示分栏, 不表示分页, 此时可以使用 `\clearpage` 和 `\cleardoublepage` 完成分页或进入双面奇数页的功能。

栏与栏的间距是由长度变量 `\columnsep` 控制的, 可以使用 `\setlength` 等命令自行修改。栏宽的变量是 `\columnwidth`, 对双栏文档, 它的

大小等于 $(\text{\textwidth} - \text{\columnsep})/2$, 可以把它用在设置其他长度的地方, 但不要手工修改这个值。

栏与栏之间有一条竖线分隔, 竖线的宽度由长度变量 `\columnseprule` 控制。`\columnseprule` 默认值为 0pt, 也就是没有竖线, 可以设置一个大于 0 的宽度增加分栏线, 通常的线宽是 0.4pt:

```
\setlength{\columnseprule}{0.4pt}
```

双栏文档的一页内容如果没有填满, 默认是左右不平衡的, 内容会先填满左边一栏, 再填充右边一栏 (见图 2.8)。使用 **balance** 宏包^[61] 提供的 `\balance` 命令可以让页面左右平衡, `\nobalance` 命令则恢复不平衡的双栏页面, 如:

```
\documentclass[twocolumn]{article}
\usepackage{balance}
\balance
% ...
```

multicol 宏包^[162] 提供了更为强大的分栏功能, 它可以在不另起一页的情况下把页面的后半部分分成多栏, 也可以随时停止分栏或改变栏数。**multicol** 宏包提供的分栏也是平衡的。宏包提供的 `multicols` 环境完成所有的工作, 注意在 `multicols` 宏包中不能使用浮动体和边注。宏包不再使用 `\newpage` 强制分栏, 而使用 `\columnbreak`。例如:

```
% 引言区
% \usepackage{multicol}
% 正文
\begin{multicols}{3}
```


分成三栏的内容……

`\end{multicols}`



flowfram 宏包^[247]在这方面走得更远，它可以把文档的文本分成好几个文本流，每个文本流是一个固定位置的块，让正文按顺序通过各个块。报纸和时尚杂志常常把版面拆分成许多“豆腐块”，然后把文章排列在各个“豆腐块”中，这种复杂的版面就可以用 **flowfram** 来

实现，有兴趣的读者可以参考宏包的文档，这里不再详细介绍。



grid 宏包^[208]提供了一组选项和环境，可以用来设置页面的高度和行数，控制各类环境的高度，以保证在双栏排版中让左右两栏的文字能逐行对齐，这种对于某些双栏期刊是非常有用的功能。本节没有使用这种机制，所以左右两列的文字行可能是参差不齐的。

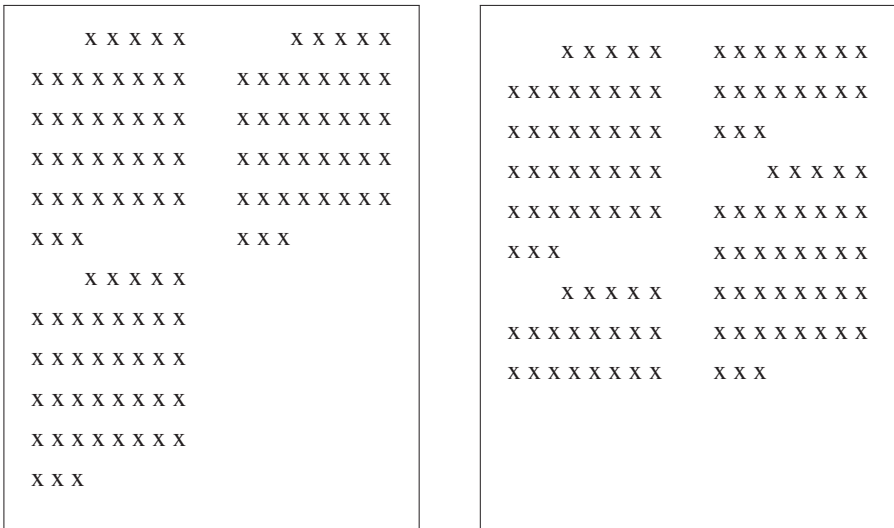


图 2.8 不平衡的双栏页面和平衡的双栏页面

2.4.5 定义命令与环境



在 1.2.4 节中我们初识了命令与环境，经过本章前面的内容，已经见识了许多有用的 **LaTeX** 命令和环境，1.2.8 节甚至还自己定义了一些命令与环境，命令的重定义也在

*本节内容初次阅读可略过。

前面的格式设置中屡屡出现。现在，我们来总结一下这些零散的内容，进一步讨论在 \LaTeX 中的宏定义功能。

一个 \TeX 宏 (macro) 是以反斜线 \backslash 开头，后面紧接着一串字母的字符串，它在 \TeX 中通常用来代替另外一个字符串，也有时表示其他一些特殊的含义。传统上将这种字符串代替的机制称为宏， \TeX 就是一种复杂的宏语言。

在 \LaTeX 中，宏通常被分为两类：一般的宏被形象地叫做“命令” (command)，而以命令 $\backslash\text{begin}\{\langle\text{环境名}\rangle\}$ 和 $\backslash\text{end}\{\langle\text{环境名}\rangle\}$ 包围的结构则称为“环境” (environment)。正同 1.2.4 节所说的，命令和环境都可以带有若干可选的和必须的参数，表示不同的含义，命令和环境是 \LaTeX 中使用格式相对固定两类宏。

可以使用 $\backslash\text{newcommand}$ 来新定义一个命令，其语法格式如下：

$\backslash\text{newcommand}\langle\text{命令}\rangle[\langle\text{参数个数}\rangle][\langle\text{首参数默认值}\rangle]\{\langle\text{具体定义}\rangle\}$

命令名只能由字母组成，并且不能以 $\backslash\text{end}$ 开头^①。

$\backslash\text{newcommand}$ 最简单的用法就是定义没有参数的命令（此时不需要 $\langle\text{参数个数}\rangle$ 和 $\langle\text{首参数默认值}\rangle$ ），其功能就是简单的字符串替换，例如：

2-4-14

```
\newcommand\PRC{People's Republic of \emph{China}}
```

定义了一个 $\backslash\text{PRC}$ 命令，在文中任何地方使用 $\backslash\text{PRC}$ 就相当于使用了“People's Republic of \emph{China} ”这么一串内容。这正是宏的本来含义。注意在定义宏时， $\langle\text{具体定义}\rangle$ 的外面必须要有花括号界定，即使定义的内容只有一个字符。在使用宏时，命令产生的效果则没有这对花括号。

如果指定了参数的个数（从 1 到 9），则可以在宏的定义中使用参数。在 $\langle\text{具体定义}\rangle$ 中，宏的参数依次编号，用 $\#1, \#2, \dots, \#9$ 表示，使用宏时它们会替换为参数的字符串，例如：

2-4-15

```
\newcommand\loves[2]{\#1喜欢\#2}  
\newcommand\hatedby[2]{\#2不受\#1喜欢}
```

那么使用 $\backslash\text{loves}\{\text{猫儿}\}\{\text{鱼}\}$ 就相当于写 猫儿喜欢鱼，而 $\backslash\text{hatedby}\{\text{猫儿}\}\{\text{萝卜}\}$ 则相当于 萝卜不受猫儿喜欢。

如果在指定参数个数的同时还指定了首个参数的默认值，那么这个命令的第一个参数就成为可选的参数（要使用中括号指定）。例如，我们可以加强前面的例子：

2-4-16

```
\newcommand\loves[3][喜欢]{\#2\#1\#3}
```

^① $\backslash\text{end}$ 开头的命令名保留用于系统实现生成环境，因而不能作为普通命令的开头。

第2章 组织你的文本

153

此时 `\loves{猫儿}{鱼}` 的作用与前面无异，还表示 猫儿喜欢鱼，同时可以用可选参数覆盖默认值，如使用 `\loves[最爱]{猫儿}{鱼}` 表示 猫儿 最爱鱼。

重定义一个 \LaTeX 命令使用：

```
\renewcommand(命令)[<参数个数>][<首参数默认值>]{<具体定义>}
```

它的意义和用法与 `\newcommand` 完全相同，只是 `\renewcommand` 用于改变已有命令的定义，而 `\newcommand` 只能用于定义新命令，例如：

```
\renewcommand\abstractname{内容简介}
```

这与 2.2.2 节中使用 `\CTEXoptions` 命令的效果相同。

如果用 `\newcommand` 重定义已有命令或用 `\renewcommand` 定义新命令，都会产生一个编译时的错误。



命令可以嵌套定义。如果要在一个命令的定义中定义或重定义命令，则里面一层的命令参数就要多加一个 `#`，例如：

```
\newcommand\Emph[1]{\emph{#1}}  
\newcommand\setEmph[1]{%  
  \renewcommand\Emph[1]{%  
    #1{##1}}}
```

2-4-17

这里首先把自定义一个表示强调的命令 `\Emph`，初始与 `\emph` 相同；`\setEmph` 则提供了对 `\Emph` 的修改。如 `\setEmph\textbf` 会将 `\Emph` 重定义为 `\textbf`，而 `\setEmph\textsc` 则将 `\Emph` 重定义为 `\textsc`。



`\providecommand` 的语法与 `\newcommand`、`\renewcommand` 相同，它也可以用 `\providecommand` 来定义命令。不同的是，它检查命令是否已经定义后，对已定义的命令不会报错，也不重定义，而是保留旧定义忽略新定义。这种方式可以用来保证一个命令的存在性，特别是可以用它来准备一个“备用”的定义，作为标准定义的补充。例如，`url` 宏包和 `hyperref` 宏包都提供有排版 URL 网址的 `\url` 命令，如果在文档中（特别是在设计通用的宏包和模板时）并不知道是否已经定义了 `\url` 命令，就可以提供一个质量比较差的版本：

```
\providecommand\url[1]{\texttt{#1}}
```

这样就可以在后面放心使用 `\url` 命令，保证这个命令可用。`\providecommand` 也可以与 `\renewcommand` 连用，表示无论之前是否定义过这个命令，都改用 `\renewcommand` 的新定义。

\LaTeX 环境与命令类似, 事实上, 一个 \LaTeX 环境就相当于一个分组, 在分组内部的最前面和最后面各有一条命令。例如 `quote` 环境 `\begin{quote}` (环境内容) `\end{quote}` 就大致等于 `{\quote (环境内容) \endquote}`。环境可以有参数, 就相当于环境前面命令的参数。

定义新环境和重定义环境分别使用下面的命令:

```
\newenvironment{<环境名>}[<参数个数>][<首参数默认值>]
    {<环境前定义>}{<环境后定义>}
\renewenvironment{<环境名>}[<参数个数>][<首参数默认值>]
    {<环境前定义>}{<环境后定义>}
```

例如, `book` 类中没有显示摘要的 `abstract` 环境, 我们可以仿照 `article` 类中的格式利用 `quotation` 环境定义一个 (同时增加了改变标题的可选参数):

```
\newenvironment{myabstract}[1][摘要]%
{
  \small
  \begin{center}\bfseries #1\end{center}%
  \begin{quotation}%
  \end{quotation}}
```

2-4-18

需要注意的是, 在定义有参数的环境时, 只有 (环境前定义) 中可以使用参数, 在 (环境后定义) 中不能再使用环境参数。如果有这种需要, 可以先把前面得到的参数保存在一个命令中, 在后面使用。例如可以定义一个带出处的引用环境:

```
1 \newenvironment{Quotation}[1]%
2   {\newcommand\quotesource{#1}%
3    \begin{quotation}%
4    {\par\hfill —— 《\textit{\quotesource}》}%
5    \end{quotation}}
6
7 \begin{Quotation}{易·乾}
8 初九, 潜龙勿用。
9 \end{Quotation}
```

2-4-19

初九, 潜龙勿用。

——《易·乾》

定义命令和环境是进行 \LaTeX 格式定制、达成内容与格式分离目标的利器。使用自定义的命令和环境把字体、字号、缩进、对齐、间距等各种琐细的内容包装起来, 赋以

第2章 组织你的文本

155

一个有意义的名字，可以使文档结构清晰、代码整洁、易于维护。在使用宏定义的功能时，要综合利用各种已有的命令、环境、变量等功能，事实上，前面所介绍的长度变量与盒子（参见 2.1.5、2.2.8 节），字体字号（参见 2.1.3、2.1.4 节）等内容，大多并不直接出现在文档正文中，而主要都是用在实现各种结构化的宏定义里。

有关 \LaTeX 宏的内容暂且介绍这些，还有一些更深入的内容，可参见 8.1 节和其他文献。



练习

2.8 输出标志 “ \LaTeX 2_ϵ ” 的命令是 `\LaTeXe`，为什么不是看起来更合理的 `\LaTeX2e` 呢？

2.9 使用 2.2.3.3 节中的 `list` 环境和 `ctex` 宏包提供的 `\chinese` 命令（参见 2.2.3.2 节），自定义计数器，定义一个 `clist` 环境，产生一个用中文数字编号的列表。

本章注记

符号和字体是 \LaTeX 以至各种排版软件的中心问题之一。Pakin [192] 是一份全面的符号列表，可用的字体目录见 Hartke [100]、Jørgensen [118]。 \LaTeX 字体命令和 NFSS 机制的详解可见 \LaTeX 3 Project Team [142]、陈志杰等 [317]，在 Mittelbach and Goossens [166, Chapter 7] 中则有更为详尽的论述。 \TeX 的原始字体机制可参见 Knuth [126]。关于 PostScript、TrueType 和 OpenType 字体的背景知识与安装使用，可以参考 Goossens [83]。

有关过去旧的中文处理方式，如 CCT、CJK、天元系统等，现在已逐渐让步于以 \XeTeX 和 `ctex` 宏包为核心的方式，故本书没有涉及旧的中文处理方式。如果使用的 \TeX 系统较为陈旧，需要处理历史文档，可参考其他文档。CCT 参见 吴凌云 [308]、张林波 [311, 312]；CJK 参见 Lemberg [147]、吴凌云 [308]。CCT 和 CJK 也可以使用 `ctex` 宏包及文档类^[58]。

基于 CJK 的中文支持方式目前仍然使用广泛，较新的进展是 `zhmetrics` 包和 `zhmCJK` 包^[307]。`zhmetrics` 把汉字都看成是相同的方块尺寸，从而简化了中文字体安装配置。而本书作者编写的 `zhmCJK` 宏包则在 `zhmetrics` 机制的基础上，动态配置中文字体，免除了 CJK 中文字体的安装工作。`zhmCJK` 同时提供了类似 `xeCJK` 的字体设置语法。

\XeTeX 的使用属于较新的内容，在较早出版的书籍中都少有论述。除了 `fontspec` 和 `xeCJK` 的文档 [212、310] 外，相关使用还可以参见文档 Goossens and Rahtz [86]、kmc [123]。

LuaTeX 的东亚语言支持目前还没有完全成熟。由 LuaTeX-j_a 团队开发的 luatexj_a 包^[150] 是目前基于 LuaTeX 和 fontspec 机制的一种相对完备的日文支持方案。LuaTeX-j_a 的机制部分源自日本原有的 pTeX 系统，对日文排版有很多针对性的设计，马起园为 LuaTeX-j_a 完成了一些关于中文排版的配置，使之也可以用来处理中文文档。

本书描述的 ctex 宏包是在 T_EX Live 2012 与 C_T_EX 套装 2.9 中默认安装的版本，在更新版本的 ctex 宏包中，会采用更现代的 $\langle key \rangle = \langle value \rangle$ 格式的宏包选项，增强 ctex.sty 格式的功能，并对 zhmcjk、LuaTeX-j_a 等新的中文处理方式也有支持，不过大部分功能和用法是一致的（或向后兼容的），使用新版本的用户可以参考相应的宏包手册使用。

L_ATeX 在段落方面提供的新命令较少，T_EX 的高级段落功能仍参见 Abrahams et al. [1]、Knuth [126] 等书籍。

L_ATeX 的全部基本功能都可以在源代码 Braams et al. [33] 中找到，基本文档类的全部功能则可以参见源代码 Lamport et al. [139]。

memoir 是一个面向专业排版的文档类，它的文档 Wilson [294] 也是很好的排版实践参考书，对标准文档类的设计也有借鉴意义；KOMAScript 是另一套类似的面向专业排版的文档类^[133]，它与 memoir 一样提供方便定制的排版环境。排版理论的书籍如 Bringhurst [35] 也可作为参考。