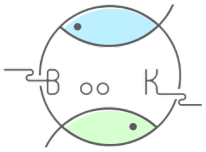
LaTeX 入门

作者：刘海洋

项目：GitHub lihui588211/BookRewriting

时间：2023/10/2

书籍模板：ElegantBook



目录

第一章 熟悉 LaTeX

LaTeX 是一种基于 TeX 的文档排版系统。TeX 只这么交错起伏的几个字母，便道出了“排版”二字的几分意味：精确、复杂、注重细节和品味。而 LaTeX 则为了减轻这种写作、排版一肩挑的负担，把大片排版的格式隐藏在若干样式之后，以内容的逻辑结构统帅纷繁的格式，使之成为现在最流行的科技写作——尤其是数学写作的工具之一。

无论你是因为心慕 LaTeX 漂亮的输出结果，还是因为要写论文投稿被逼上梁山，都不得不面对一个事实：LaTeX 是一种并不简单的计算机语言，不能只点点鼠标就弄好一篇漂亮的文章，也不是一两个小时的泛泛了解就尽能对付得过去。[1]

LaTeX 的读音和写法

TeX 一名源自 technology 的希腊词根 $\tau\varepsilon\chi$，TeX 之父高德纳教授近乎固执地要求它的发音必须是（按国际音标）[tɛx]，尽管英语中它常被读作 [tɛk]。（同样，高德纳教授也近乎固执地要求别人说他的姓 Knuth 时不要丢掉 K，叫他 Ka-NOOTH，尽管在英语环境他时常会变成 Nooth 教授。）对比汉语，TeX 的发音近似于“泰赫”。

LaTeX 这个名字则是把 LaTeX 之父 Lamport 博士的姓和 TeX 混合得到的。所以 LaTeX 大约应该读成“拉泰赫”。不过人们仍然按照自己的理解和拼写发音习惯去读它：['lɑːtɛk]、['leɪtɛk] 或是 [lɑː'tɛk]，甚至不怎么合理的 ['leɪtɛks] [2]。好在 Lamport 并不介意 LaTeX 到底被读做什么。“读音最好由习惯决定，而不是法令。”——Lamport 如是说。

两个创始人对于名称和读音的不同态度或许多少说明了这样一个事实：LaTeX 相对原始的 TeX 更少关注排版的细节，因此 LaTeX 在很多时候并不充当专业排版软件的角色，而只是一个文档编写工具。而当人们在 LaTeX 中也抱以追求完美的态度并用到一些平时不大使用的命令时，通常总说这是在 TeX 层面排版——尽管 LaTeX 本身正是运行于 TeX 之上的。

类似地，TeX 和 LaTeX 字母错位的排版也体现出一种面向排版的专业态度，即使在字符难以错位的场合，也应该按大小写交错写成 TeX 和 LaTeX。

现在我们使用的 LaTeX 格式版本为 2ε，意思是超出了第 2 版，接近却没有达到第 3 版，因此写成LaTeX 2ε。在只能使用普通字符的场合，一般写成 LaTeX2e。

1.1 让 LaTeX 跑起来

学习 LaTeX 的第一步就是上手试一试，让 LaTeX 跑起来。首先安装 TeX 系统及其他一些必要的软件，然后跑一个测试的例子。下面的几节包含了一大堆具体软件安装和使用的内容，虽然比较烦琐，但这是使用 LaTeX 进行写作的必要前提。如果你早已做好这些准备，或者在读本书之前就已经迫不及待地做了不少尝试的话，可以直接跳到开始一个实际的例子。

1.1.1 LaTeX 的发行版及其安装

TeX/LaTeX 并不是单独的程序，现在的 TeX 系统都是复杂的软件包，里面包含各种排版的引擎、编译脚本、格式转换工具、管理界面、配置文件、支持工具、字体及数以千计的宏包和文档。一个 TeX 发行版就是把这样的部件都集合起来，打包发布的软件。

[1] 不过现代人好像都追求快，沉不下心来。这也是我通过此项目强迫自己学习的理由

[2] 额。我的读法...

尽管内容复杂，但现在的 TeX 发行版的安装还是非常方便的。下面将介绍两个最为流行的发行版，一是1.1.1.1的 CTeX 套装，二是1.1.1.2 的 TeX Live。前者是 Windows 系统下的软件，后者则可以用在各种常用的桌面操作系统上。对 Windows 用户来说，两个发行版并没有显著的优劣之分，你可以任选一个安装使用[3]。

1.1.1.1 C TeX 套装

已过时，跳过。

1.1.1.2 TeX Live

TeX Live 是由 TUG （TeXUser Group ， TeX 用户组）发布的一个发行版。TeX Live 可以在类 Unix/Linux 、macOS 和 Windows 系统等不同操作系统下安装使用，并且提供相当可靠的工作环境。TeX Live 可以安装到硬盘上运行，也可以经过便携（portable ）安装刻录在光盘上直接运行（故有“Live ”之称）。

不同操作系统下安装设置 TeX Live 的方式基本一样，这里以 Windows 操作系统为例进行演示。[4]

TeX Live 一般以安装镜像的方式在互联网上发布。光盘镜像文件可以从官网上下载[5]。载入镜像后，执行 `install-tl-windows.bat` 进行安装。只要选好安装的位置，不断单击“下一步”就可以安装 TeX Live 了，如图所示。

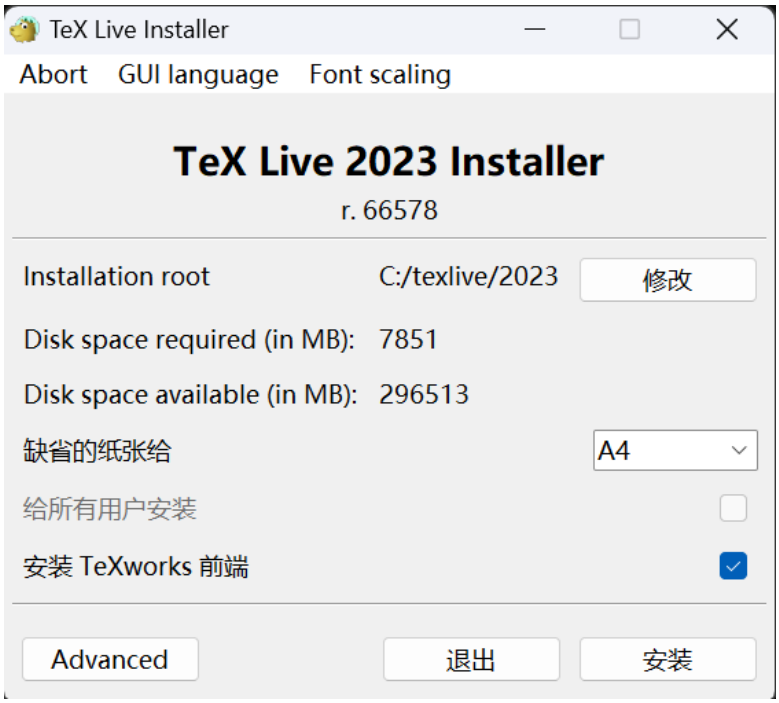


图 1.1: 在 Windows 11 上安装 TeX Live2023

程序安装好了之后，会在开始菜单栏增加 TeX Live 文件夹图标。其中包含的内容比较简单，它包含以下项目：

- **TeXworks editor ：** 这是 TeX Live 预装的一个 TeX 文件编辑器，简单方便。大部分工作都可以在这个编辑器中完成。
- **DVIOUT DVI viewer ：** 这是一个 DVI 文件预览器。我们很少用到它。

[3]并不是。由于 CTeX 早已停止维护及历史原因，现在 TeX Live 是更好的选择。于是我选择跳过1.1.1.1

[4]原书中还包含部分 Linux 系统上的操作讲解，这里删去。

[5]https://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/Images/

- **TeX Live command-line**：它打开 Windows 的命令提示符，并设置好必要的环境变量，可以在其中使用命令行编译处理 TeX 文档。
- **TeX Live documentation**：这是一个 HTML 页面的链接，里面是 TeX Live 系统中所有 PDF 或 HTML 格式的文档列表。在首页你可以找到几种语言（包括简体中文）的 TeX Live 发行版文档，以及到近 4000 份各种文档的列表的链接——这份有一公里长的列表多少说明了 TeX Live 是一个多么复杂的系统，以及它在安装时为什么占用了这么大的空间。当然，你不需要读完里面的所有文档才能学会 LaTeX，不过你会发现在工作中总需要时不时地查看里面的东西。
- **TLShell TeX Live Manager**：这是 TeX Live 管理工具的图形界面，简称 `tlmgr`。管理工具也可以在命令行下用 `tlmgr` 命令运行，用 `tlmgr gui` 可以在命令行下打开图形界面。

1.1.2 编辑器与周边工具

1.1.2.1 编辑器举例——TeXworks

像其他计算机语言一样，LaTeX 使用纯文本描述、因而任何能编辑纯文本的编辑器都能编辑 LaTeX 文档，如 Windows 系统的记事本、写字板，Linux 下的 VI、GEdit。不过，使用专门为 LaTeX 设计或配置的编辑器，进行语法高亮、命令补全、信息提示、文档排版等工作，会使工作方便很多。

LaTeX 代码编辑器有很多，大致可以分为两类：一种主要为 LaTeX/TeX 代码编辑而专门设计的编辑器，二是可以为 LaTeX/TeX 代码编辑配置或安装插件的通用代码编辑器。前者如 WinEdt、TeXworks、TexMaker、Kile，后者如 Emacs、VIM、Eclipse、SciTE 等。通常前一种编辑器配置和使用更简单一些，下面主要以 TeXworks 为例说明编辑器的一些简单配置。其他大部分编辑器在基本功能和设置上都大同小异，不难举一反三。

TeXworks 的界面非常简洁（见图1.2）：它分为两个部分，左侧是 TeX 源文件的编辑窗口，右侧是生成的 PDF 文件的预览窗口。左边的编辑器窗口最上面是标题栏和标准菜单栏，接着是工具栏，中间最大的编辑区，最下面是显示行列号的状态栏。右边的预览窗口把编辑区换成了 PDF 预览区。

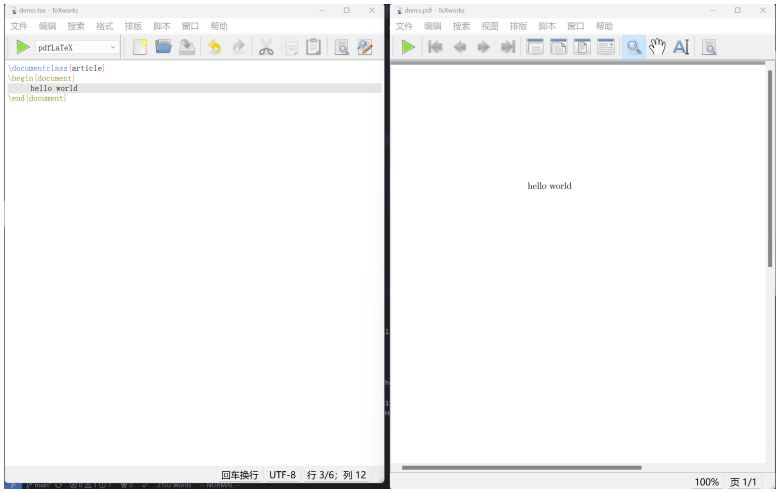


图 1.2: TeXworks 界面

除了文本编辑区，编辑器窗口中最常用的是工具栏。工具栏的最左边的按钮是整个编辑器最为重要的“排版”按钮，它调用具体的命令把输入的 TeX 源文件**编译**为对应的 PDF 结果，刷新右边 PDF 文件的显示。紧靠排版按钮右边的下拉菜单中用来选择排版时所使用的命令，通常对应一条单一的命令，但也可以配置为好几条命令的复合。通常我们使用最多的排版命令是“XeLaTeX”或“PDFLaTeX”，视具体情况而定。使用排版按钮时，未保存的文档会自动保存。工具栏剩下的按钮则是一系列常见的标准按钮：新建、打开、保存；撤销、重做；剪切、复制、粘贴；查找和替换，不必多说。

PDF 预览窗口的工具栏也是一排按钮。最前面的排版按钮与编辑区的功能一样。右面是 4 个向前后翻页的按钮；而后是显示比例的按钮；再后面是放大工具、滚屏工具；最后是 PDF 文本查找工具。

使用 TeXworks 也很简单：

1. 在编辑区输入 TEX 源文件
2. 单击“保存”按钮，给源文件起名并保存在指定位置
3. 在排版按钮旁的下拉菜单中选择“XeLaTeX”，单击排版按钮，查看结果。

编译时在文本编辑区下方的“控制台输出”面板中会显示编译进度和信息。如果编译过程有错误或提示输入、程序会停下来等待处理。如果编译结束无误，控制台输出面板会自动关闭，而在预览窗口会显示新的 PDF 结果。

在文本编辑区或 PDF 预览区用鼠标左键单击可以从源文件跳转到 PDF 文件中的对应位置；或反过来从 PDF 跳转到源文件中的对应位置。这个功能称为 TEX 文档的**反向查找**，对编写长文档特别有用。正反向查找是由 SyncTeX 机制实现的，需要源代码编辑器、PDF 阅读器和 TEX 输出程序的共同参与，一些旧的发行版或程序可能并不支持。

TeXworks 支持**自动补全**功能。输入一个助记词或命令的一部分，再按 `Tab` 键，则 TeXworks 会根据配置补全整个命令或是环境；连续按 `Tab` 键可以切换补全的不同形式。例如，输入 `\doc` 再按 `Tab` 键，会补全命令 `\documentclass{}`；使用 `beq` 补全则可以得到公式环境：

```
\begin{equation}
    |
\end{equation} *
```

光标在环境中央等待输入，再次按下 `Ctrl + Tab` 组合键可以跳转到后面的圆点处继续下面内容的输入，而不需要使用方向键。

下面来看看 TeXworks 中一些常见的配置。

刚刚安装的 TeXworks 通常会使用很小的字体，而且可能没有语法高亮等功能，给编辑工作带来诸多不便。在 TeXworks 的“格式”菜单中，“字体”项可以用来临时更改显示的字体，而“语法高亮显示”项可以临时控制如何进行语法高亮。要使字体和语法高亮的设置对所有文档生效，则应该修改 TeXworks 的默认选项。单击 TeXworks “编辑”菜单的最后一项“首选项”，将弹出 TeXworks 首选项窗口（见图1.3）。在“编辑器”选项卡中，可以设置编辑器默认的字体和字号，下面则有语法高亮、自动缩进等格式。

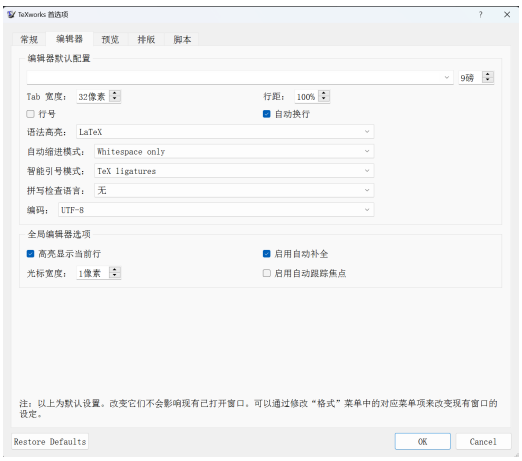


图 1.3: TeXworks 编辑器首选项设置

TeXworks 支持多种语言界面和多种文字编码。TeXworks 默认的界面会与操作系统的默认语言（Locale 设置）一致，可以在首选项设置窗口的“一般”（General ）选项卡中设置程序的界面语言为中文。在“编辑器”选项卡中则有“编码”选项，一般应选择 TeXworks 的默认值，即 UTF-8 编码，编辑器保存和打开文件将默认使用此编码。

TeXworks 首选项窗口的“排版”选项卡可以用来设置 TeXworks 的“排版”按钮所执行的命令。选择对应的处理工具，单击“编辑...”按钮，就可以在弹出的窗口中设置对应的命令及参数。参数中使用的变量，可参见

TeXworks 的帮助文档。

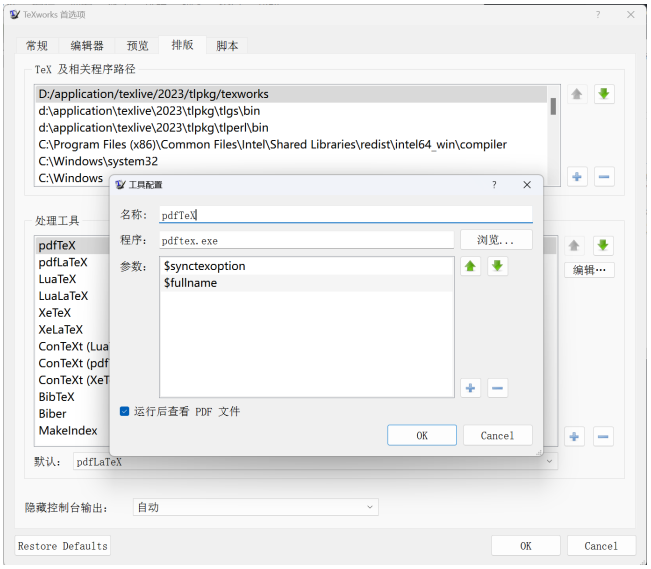


图 1.4: TeXworks 排版首选项设置

文本编码与 Unicode

在使用 TeX 编辑器时，必须要注意文档保存的文字编码。如果编码使用错误，轻则遇到“乱码”，重则导致程序运行错误。我们提到的“UTF-8”编码，就是现在最为常用的编码之一。

文字在计算机内部都是以数字的形式表示、储存和运输的，人们圈定一些在计算机中使用的字符，称为字符集（character set ），一个字符通常就用它在字符集中的序号来表示。不过由于在计算机中数字的二进制表示也有不同的格式，因而相同的字符集也可能有不同的二进制表示方式，也就是字符编码（character encoding ）。IBM 公司以前给自己系统中每一种编码编一个号，即所谓代码页（code page ），后来其他计算机厂商如微软、Oracle 都把自己的字符编码用代码页的方式给出，不过使用的代码页编号都不一样。我们通常见到的代码页，都是微软公司的编号。字符集、字符编码、代码页这些概念，在很多时候都不加区分，可以混用。

ISO 于 1990 年推出了通用字符集（Universal Character Set, UCS ）标准 ISO 10646 ，包括 UCS-2 和 UCS-4 两种长度的编码；1991 年一个叫做通用码协会（Unicode Consortium ）的组织发布了 Unicode 1.0 标准。两个组织都打算把全世界所有文字的符号都用一套字符集和编码统一起来。后来，两个组织协作起来，从 Unicode 2.0 起，Unicode 就符合 ISO 10646 了。2012 年 1 月发布的 Unicode 6.1 已经定义了 110181 个字符，包括世界上 100 种文字，而且还在不断修订拓充之中。[6]Unicode 已逐渐成为字符编码的新方向，包括 GB 18030 也可以看成是 Unicode 字符集的一种编码格式。除了 UCS-2 和 UCS-4 ，Unicode 标准还提出了多种编码形式，称为 Unicode 交换格式（Unicode Transformation Format, UTF ）：主要包括变长的 UTF-8 、UTF-16 和定长的 UTF-32 编码。UTF-8 编码与 ASCII 编码向下兼容，因而最为常用。

TeX 系统原本只支持 ASCII 编码。但只要设置好超过 127 的数字对应的符号，所有拓展 ASCII 编码都能正确排版，如 ISO 8859 的各种标准。汉字编码 GB 2312 、GBK 和 UTF-8 都是兼容 ASCII 的多字节编码，因而在 LaTeX 中通过 CJK 宏包也可以通过特殊的方式，把多个字符对应到一个汉字上，支持中文的排版。

CJK 宏包这种支持多字节编码的方法其实是一种黑客手段，后来 TeX 的新实现 XeTeX 和 LuaTeX 都直接支持 UTF-8 编码，新的中文排版方式也自 2007 年起随着这两种新排版引擎应运而生。LuaLaTeX 的本地化支持目前还暂处于起步阶段，本书将着重介绍基于 XeTeX 的方式。

[6] 目前最新版本的 Unicode 13.0 中，包含的字符总数已达 143859 个。

1.1.2.2 PDF 阅读器

TeX Live 已经预装了 DVI 文件和 PostScript 文件的阅读器。然而却没有安装最重要的 PDF 格式阅读器。现在使用的 TeX 系统基本上最终都输出 PDF 格式的结果，而且 TeX 系统中的大部分文档也都是 PDF 格式的，因此一个 PDF 阅读器是不可或缺的。

PostScript 阅读器 GSview 和 PS_View 也都可以当做 PDF 阅读器来使用，不过效果不是很好。我们使用 PDF 阅读器有两个目的，一是在编辑文档过程中随时查看编译的效果，这对于编辑复杂公式、插图以及幻灯片来说都非常重要；二是为了阅读 PDF 格式的文档资料，或查看自己编写文档的最后效果。这两个目的各有不同的要求，前者要求快捷方便，最好还能在 PDF 的效果和 TeX 源文件之间方便地切换检索；后者则要求显示准确美观、功能全面。

要满足第一个要求，编辑器 TeXworks 内置的显示功能最为方便。TeXworks 把源代码和 PDF 结果左右排开，对照显示，TeX 代码编译后右边的 PDF 文件就会更新。而且可以用 `Ctrl`加鼠标左键单击进行从源文件到 PDF 文件或 PDF 文件到源文件的正反向索引。

要满足第二个要求，我们建议使用 Adobe Reader 最新的版本[7]（Linux 系统中通常称为 arcoread ）。Adobe Reader 是官方免费提供的 PDF 格式阅读器，通常它的显示结果最好，支持全面的 PDF 特性（如 JavaScript 脚本、动画、3D 对象等，而且这些很可能在 LaTeX 制作的幻灯片中使用到。）其他一些常见的 PDF 阅读器，如 Foxit Reader ，则可能在一些功能上有欠缺。

如果你还没有安装 Adobe Reader ，可以直接从 Adobe 的官方，或几乎任何的下载站点中得到并安装。不必抱怨它占用上百兆的安装空间：除了一些高级功能的插件，Adobe Reader 还提供了许多种高质量的 OpenType 西文字体，这些都可以在你未来的排版中用到。

PS 格式和 PDF 格式

PS 是 PostScript 的简称。PostScript 是 Adobe 公司于 1984 年发布的一种页面描述语言，自 1985 年苹果公司的 LaserWriter 打印机开始，此后的很多高档激光打印机都带有 PostScript 。于是，PostScript 逐渐成为电子与桌面出版的标准格式，并一直延伸到整个出版业，风靡一时。就连国内最大的北大方正公司的排版系统也是以变形的 PostScript 格式输出，并沿用至今。

“PostScript”这个名字多少体现了这门语言的特点：它是一种基于后缀表达式和栈操作的解释型计算机语言。例如，表达式 1+2 就被写成 `1 2 add`。而 `0 0 moveto ; 100 100 lineto` 则是在描述从坐标（0，0）到坐标（100，100）的直线路径。使用这种后缀语法原本是为了方便计算机芯片高效解释 PostScript 这种复杂的语言，大部分 PostScript 代码也都是由其他计算机程序自动生成的。不过，富有经验的老手可以就凭借着这种看起来有些怪异的语法直接画出图来，这种技艺也一直延伸到将要讲到的`PSTricks`宏包中。

PostScript 拥有强大的图形能力，可以用一段 PostScript 语言的代码表示很复杂的图形。然而，作为一门完整的计算机语言，PostScript 过于复杂，因而出现了所谓封装的 PostScript（Encapsulated PostScript ）格式，即 EPS 格式。EPS 格式的文件也是一段 PostScript 代码，但只能表示一页，而且加上了诸多限制，成为一种专门用来存储可以嵌入其他应用的图形格式。TeX 的许多输出引擎都支持这种图形格式。

由于在电子出版领域的地位，PostScript 一度成为 TeX 最重要的输出格式，至今也能在网络上见到大量 TeX 系统产生的 PostScript 格式的书籍和文章，一些期刊也一直要求以能生成 PostScript 格式的 TeX 文档投稿。然而随着新一代廉价的喷墨打印机的出现，需要复杂解释芯片的 PostScript 打印机逐渐式微；而网络技术的发展进一步催生了电子文档交换的需求，PDF ——Portable Document Format （可移植文档格式）便应运而生。PDF 由 Adobe 公司于 1993 年发布，它是 Adobe Acrobat 系列产品的原生文件格式，并随着文件格式的公开和阅读器 Adobe Reader 免费的发放，迅速风靡起来。

PDF 和 PostScript 使用相同的 Adobe 图形模型，可以得到与 PostScript 相同的输出效果，而在程序语言

[7]全名为 Adobe Acrobat Reader

方面则比 PostScript 大为削减，并增强文档格式结构化，可以迅速地由计算机处理。尽管 PDF 最初只是 PostScript 削减功能适应电子文档处理的结果，但 PDF 转而在电子文档交互式表单、多媒体嵌入等方面大下功夫，并不断进行各方面的扩充，最终成为一种比 PostScript 还复杂的格式。PDF 也继 PostScript 之后成为现在新一代的电子出版业的事实标准。

现代的 TeX 输出引擎几乎都以 PDF 为输出格式。同时 PDF 格式也可以像 EPS 格式一样作为图形格式被 TeX 和其他软件使用。现在能够输出 PDF 图形的软件和支持嵌入 PDF 图形的 TeX 引擎比 EPS 格式的还要多些，PDF 也成为现在 TeX 系统中最重要的图形格式。

1.1.2.3 命令行工具

一、命令行

尽管大多数常用编译操作可以在编辑器中完成，TeX Live 还给出了图形界面的配置工具，但 TeX 发行版的主体仍然是命令行下的程序。不了解命令行，就难以了解 TeX 的处理流程，也不能很好地使用诸如 Makeindex 这样的基本 LaTeX 工具。因此，有必要对命令行和一些命令行工具的使用做一个了解。

命令行是以文字形式与计算机交互的方式，与图形方式相对。在 Windows 系统中，命令行通常由命令行解释程序 `cmd.exe` 处理；在 Linux 及其他类 UNIX 操作系统中，命令行解释程序通常称为 Shell ，最常见的 Shell 是 Bash 。在命令行下可以执行一些基本的文件操作，也可以运行其他程序，批处理脚本也是由命令行解释程序执行的。Linux 中 shell 的使用一般远比在 Windows 中频繁，因此这里仍以 Windows 为例。

Windows 中默认的命令行解释程序 `cmd.exe` 可以在“开始”菜单搜索框中输入 `cmd` 查找到“命令提示符”进入。如果使用频繁，可以在桌面或任务栏中创建快捷方式，或设定快捷键随时使用。

使用 TeX 经常需要在特定文件所在的目录进行命令行操作，这时只需要在资源管理器的地址栏内输入 `cmd` 回车。

打开命令行窗口之后，会显示命令提示符，默认的命令提示符由当前盘符、目录和一个大于号 > 组成，如

```
C:\>
```

表示当前目录是 C 盘的根目录 > 。后面的光标等待输入命令，Windows 命令行命令和文件名不区分大小写，输入一行命令按下回车键即开始执行。

使用最频繁的命令是文件列表命令 `dir` （directory 的缩写），直接输入 `dir` 后按下回车就会显示当前目录下所有文件的详细列表。`dir` 命令后可以指定要列出的盘符、目录和文件名，如

```
dir C:\WINDOWS
```

将列出 C 盘 WINDOWS 目录下的所有文件。

目录和文件名可以使用 `?` 和 `*` 作为通配符。`?` 可以替代任意一个字符，`*` 可以代替任意多个字符。例如，命令

```
dir book*.tex
```

将列出所有以 `book` 开头，后缀为 `.tex` 的文件。目录和文件名可以用 `Tab` 键自动补全，如输入 `book` 后，连按 `Tab` 将交替地补全当前目录所有以 `book` 开头的文件。有两个特殊的目录名 `.` 和 `..` 分别用来表示当前目录（可省略不写）和当前目录的上一级目录。

`cd` 命令（或 `chdir` ，change directory 的缩写）用来改变当前所在的目录。如

```
cd pictures
```

将进入当前目录下的 `pictures` 目录（如果有的话），而从 C 盘用命令

```
cd \WINDOWS\Fonts
```

则进入 Windows 的字体目录。注意更换盘符不能用 cd 命令，而要单独使用盘符后加冒号：进入，如输入

```
D:
cd \test
```

将进入 D 盘根目录下的 test 目录。

把多个命令行写到一个文件中，保存为后缀为 .bat 或 .cmd 的文件，就得到一个批处理文件（又称为批处理脚本）。在命令行下可以像运行其他程序一样调用批处理文件，也可以在图形界面鼠标点击批处理文件执行。批处理可以一次完成多项任务，如完成多道工序的 TeX 源文件编译工作。批处理还提供命令行参数、变量定义、文件判断等简单的编程功能，详细内容可参见微软的联机帮助。

GhostScript

GhostScript 是一种 PostScript 的解释器，它的主体也是命令行工具。Windows 版本的 TeX Live 附带安装了一份简化版本的 GhostScript，程序名为 rungs。一般还是最好单独下载安装完全版本的 GhostScript[8]，因为一些 LaTeX 输出引擎有时仍会调用它。

可以用 GhostScript 查看 PostScript 或 PDF 格式的文件，PostScript 文件查看器 GSview 和 PS_View 都是调用 GhostScript 工作的。GhostScript 更常用的功能是进行文档格式转换，做 PS、PDF 格式的相互转换，或把它们转换为点阵图片格式，如 PDF 输出引擎 DVIPDFMx 就会在处理 ERS 图片时自动调用 GhostScript。

GhostScript 为一些常用的转换提供简单的命令，最常见的是从 .ps 到 .pdf 文件的转换，可以用 ps2pdf 命令完成，如

```
ps2pdf foo.ps
```

命令会将 foo.ps 文件转换为 foo.pdf。类似的命令还包括 pdf2ps 和 ps2eps等。

所有显示和转换的工作都可以通过 GhostScript 的主程序完成。GhostScript 的主程序是一个命令行程序，在 Windows 下名叫 gswin64.exe，在 Linux 等系统下通常叫做 gs，也可以使用 TeX Live 的 rungs，这里统一用 GS 表示。一个调用 GS 的命令通常带有许多命令行参数，以完成各种复杂的操作，例如，

```
GS -dBATCH foo.ps
```

将使 GhostScript 在屏幕上显示 foo.eps 的内容并退出；下面的命令（第一行末的| 并不存在，只表示延续到下一行）：

```
GS -q -sDEVICE=png256 -dEPSCrop -r128 -dGraphicsAlphaBits=4 \
   -dTextAlphaBits=4 -o bar.png foo.eps
```

则把 foo.eps 转换为 256 色 PNG 图像 bar.png，使用 128dpi 的分辨率，裁剪到适当大小，并对文字和图像做边缘抗锯齿处理。关于 GhostScript 的详细命令行参数可以参考 GhostScript 的联机文档。

三、ImageMagick

ImageMagick 是一款优秀的基于命令行的位图处理软件，可以在超过 100 种不同的图像格式之间转换，或对图像进行各种变换和处理。熟悉平面设计的人可以把它看做是 Adobe Photoshop 这类软件的一些图像滤镜的命令行版本。ImageMagick 并不直接与 TeX 相关，但 TeX 用户经常用它来做一些有关图形转换的工作，个别与 TeX 相关的软件（如 Asymptote）也会调用 ImageMagick。

ImageMagick 是自由软件，可以在 https://imagemagick.org/script/download.php进行下载安装。

在 Windows 下安装 ImageMagick 可以在开始菜单栏中找到它的帮助文档和 ImageMagick 中唯一的图形界面程序 IMDisplay。但注意 IMDisplay 只是一个图片查看器，并不具备任何 ImageMagick 的图像处理功能，我们主要还是在命令行下使用 ImageMagick。

ImageMagick 是个很复杂的软件，包括 10 多个不同的命令行工具，具有 200 多种不同的命令行参数。这里只介绍 ImageMagick 最基本的图像类型转换功能，也是最常用的功能，更详细的功能可以参见 ImageMagick 的联机帮助文档。

[8] https://www.ghostscript.com/releases/gsdnld.html

命令 convert 用于图像的转换，即把一幅图像转换为另一幅图像，尽管功能复杂，但基本的使用方法是十分简明的，如：

```
convert foo.bmp bar.png
```

是将 BMP 格式的图像 foo.bmp 转换为 PNG 格式的图像 bar.png ，类似地，

```
convert foo.eps bar.jpg
```

则是把 EPS 格式的图片 foo.eps 转换为 JPG 格式的图片 bar.jpg 。不过 ImageMagick 在处理涉及 PostScript 和 PDF 格式的图片时，内部实际还是调用 GhostScript 来完成的，这时可以把它看做是 GhostScript 命令的一种方便的变形。

获取命令行帮助

熟练的用户使用命令行完成一些工作比使用图形界面的软件更高效快捷。不过对于刚接触命令行不久的人来说，命令行的最大问题就是记不住命令的用法，因此应该了解如何在命令行下获取帮助信息。

专门的联机文档或在线文档是比较通用的帮助形式，如在 Windows 下，由“开始”菜单栏进入联机帮助，以“命令行”、“cmd”等关键字搜索，很容易就能得到详尽的命令行帮助。有时帮助文档则以专门的文件储存，如 GhostScript 和 ImageMagick 在 Windows 下都提供网页形式的帮助文档，可以在“开始”菜单栏中找到。也有很多程序提供 CHM 、PDF 格式的文档。

另一种方式是直接在命令行下得到帮助，这通常是通过特殊的命令行参数得到的。通常，Windows 命令行的基本命令后加 /? 参数获得帮助。如输入 dir /?可以在屏幕上获取到 dir 命令的帮助信息。来自类 UNIX 系统的程序命令行选项以 -开头，命令行帮助通常可以在命令后加上 --help 获得。如输入convert --help 将会在屏幕上得到关于 ImageMagick 的所有命令行选项的说明。

类 UNIX 系统在命令行下还有一个 man 命令，可以用来调出文档。如用 man ls 将在命令行中直接调出 ls命令（相当于 Windows 系统下的 dir 命令）的详细帮助，类似的文档程序还有 info 。在 Windows 中，用 help 可以达到类似的效果，不过效果与使用 /? 选项差不多。TeX 系统继承了 UNIX 中 man 的用法，也提供了一个 texdoc 程序，可以在命令行下调出 TeX 宏包、工具和字体等的文档。

1.1.3 “Happy TeXing 与“特可爱排版”

在做完所有的准备工作后，我们一起来运行一个简单的例子，测试整个系统。

首先，打开你的 TeX 编辑器，如 TeXworks ，新建一个文件，输入下面的内容：

```
1 \documentclass{article}
2
3 \begin{document}
4 This is my first document.
5
6 Happy \TeX ing !
7 \end{document}
```

新建一个测试用的目录，将刚刚输入的文件保存到这个目录里面，选择 PDFLaTeX 或者 XeLaTeX 命令，点击编辑器上对应的排版按钮。如果一切顺利，将在 PDF 预览窗口看到编译的结果，内容类似下面的样子：

This is my first document.
Happy TeXing !

这个文件中有一些以反斜杠\ 开头的语句，大多没有出现在最终的 PDF 文档中。虽然我们之前没有接触过这些语句，不过不难猜测其涵义：\documentclass{article} 声明了文档的类型是一篇文章；\begin{document} 和 \end{document} 语句标识出正文的范围；至于正文中的 TeX，看结果就知道它表示“TeX”这个高低不平的符号。这是我们的第一个例子，看起来很简单。

可是，如果你马上兴致勃勃地把里面的内容换成汉字，再点击按钮看结果时，就会发现汉字并没有出现在 PDF 中，只有英文字符出现，这是因为 TeX 原本是面向西文写作的，默认并没有加载中文字体。

通过更换文档类型，下面这个稍稍复杂的例子可以正确显示出中文：

```
\documentclass[UTF8]{ctexart}

\begin{document}
\section{汉字}
特可爱排版
\section{数学}
\[
   a^2 + b^2 = c^2
\]
\end{document}
```

1 汉字

特可爱排版

2 数学

$$a^2 + b^2 = c^2$$

这段代码其实也不难看懂：文档类换成了 ctexart，即中文 TeX 的文章（article ）类型，这个文档类使得中文可以正确地显示；在ctexart 前面的 UTF8 是使用这个文档类的选项，表明了中文所使用的编码；两个 \section 命令各自生成了一节的标题；唯一不大直观的是由 \[和 \] 包裹起来的数学公式，不过 LaTeX 数学公式的能力太出名了，你一定早就听说过它了。

上面两个简单的例子给了我们一个 LaTeX 的直观印象，而且正确运行它们也许能够增强你学习 LaTeX 的信心。粗略地看，LaTeX 是一种标记式排版语言，有相关背景的人大概会觉得 LaTeX 的代码与 HTML 代码有很多相似之处，整个文档通过一些标记（命令）分成结构化的部分。LaTeX 的命令以反斜线 \ 开头，命令一般以英文单词命名，有的可以带参数。通过一个程序的处理，我们称为 **编译**过程，LaTeX 源代码就能生成对应的输出结果，通常就是一个 PDF 文档。

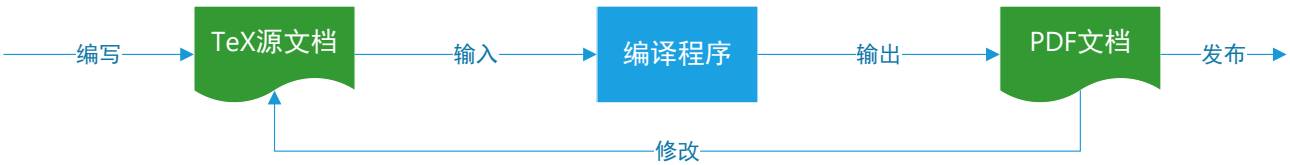


图 1.5: LaTeX 文档的写作流程

LaTeX 的写作流程见图1.5。通常这个流程都是自动化完成的，编写 TeX 源文件通常是在专门的 TeX 编辑器中进行，例如 TeXworks，而后按下一个按钮，源文件就会被送给 TeX 的编译程序进行处理，输出 PDF 文件，此时编辑器调用 PDF 阅读器查看结果。如果出了问题，需要根据输出的结果或程序的错误信息修改源文件或编译方式。

编译程序

TeXworks 等编辑器里面给出了许多编译程序的按钮，往往让人有种应接不暇的感觉，如果你留心来自各种书籍、文档和网络的资料，上面介绍的编译方法五花八门。如果是使用命令行编译，则输入起来更觉头疼，那么，这些不同的编译程序做了什么？该如何选择和使用呢？

高纳德设计的 TEX 原本只是一个相对简单的程序，命令 `tex` 可以调用最基本的 TEX 程序。它使用高纳德定义的一个相对简单的格式 Plain TEX 进行排版。`tex` 读入 TEX 源文件，输出一种“与设备无关”（DeVive Independent ）的格式，即 DVI 文件，DVI 文件曾经是 TEX 的标准输出格式，但功能比较受限，不能嵌入字体和图形等，但在 PostScript 和 PDF 流行之后，DVI 格式就主要成为一种到 PS 或 PDF 的中间格式了。

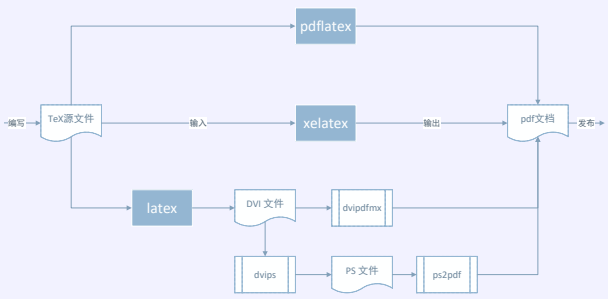
程序 Dvips 将 DVI 文件转换为 PostScript 文件，可以直接拿到支持 PostScript 的 ps2pdf 或 Adobe Acrobat 提供的 Distiller 等程序再从 PostScript 转换到 PDF 文件。PDF 流行之后，又有了能直接把 DVI 文件直接转换成 PDF 的 dvipdf 程序，后来又出现了更为先进的 dvipdfm 和 dvipdfmx ，可以支持更丰富的 PDF 功能和东亚字体等，现在新的发行版中主要还在使用的是 dvipdfmx （常常写作 DVIPDFMx ）。这类把 DVI 文件转换为其他实用格式的程序常被称为 TEX 输出的驱动（driver ）。

除了最初的 TEX 程序，后来有很多人对 TEX 进行了拓展。先是有了 ε– TEX，后来在 ε– TEX 的基础上，Hàn Thê Thành 设计了能够直接输出 PDF 格式的 PDFTEX。不过 PDFTEX 程序也保留了输出 DVI 格式的能力，因而现在很多输出 DVI 格式的命令内部也是使用的 PDFTEX 程序。PDFTEX 的后续是 LuaTEX，这是一种把脚本语言 Lua 和 TEX 结合起来的程序。ε– TEX 的另一发展则是 XeTeX ，它将中间层 DVI 格式扩充为更强大的 xdv 格式，一般会直接调用 dvipdfmx 的后继 xdvipdfmx ，直接输出 PDF 格式。LuaTEX 和 XeTeX 都将原来 TEX 支持的 ASCII 编码改为 UTF-8 编码，并且可以更方便地使用各种字体。TEX 程序连同这些扩展被称为不同的 TEX 引擎（engine ）。

不同的引擎都可以编译 Plain TEX、LATEX 或是 ConTEXt 等不同格式的文档，不同的组合就使用不同的命令，如下表所示，我们主要关注 PDFTeX 和 XeTeX 引擎使用 LATEX 格式的命令。

命令 / 格式 / 引擎	Plain TEX	LATEX	ConTEXt	
TEX/ε-TEX	tex / etex			输出 DVI
pdfTEX	tex	**latex**		输出 DVI
pdfTEX	pdftex	**pdflatex**	texexec	输出 PDF
XƎTEX	xetex	**xelatex**	特殊参数	输出 PDF
LuaTEX	luatex	lualatex	context	输出 PDF

使用 LATEX 格式的排版得到 PDF 文件的方式也有好几种。其中使用 `latex + dvips` 的方式最为古老，不便于中文文档的排版。用 `latex` 和 `pdflatex` 命令排版在处理中文时都使用 CJK 宏包的机制，而 `xelatex` 则使用新的 xeCJK 宏包的机制。功能上 `xelatex` 最为方便，尤其是在处理中文时；而用 `pdflatex` 编译，一些宏包的兼容性会更好一些。不过本书的大部分内容并不限于任何一种模式，只是在处理中文时，将主要讨论 `xelatex`。



1.2 从一个例子说起

这一节将研究一个相对实际的例子。在这个简单的例子中，我们将看到在真正的写作排版工作中常遇到的一些模式、问题的解决思路。有一些代码或许一时难以理解，不用担心，我们将在后面的章节里面详细讨论。

1.2.1 确定目标

现在来把话题限定在初等平面几何，假定我们要写一篇关于勾股定理的短文，短文是一般的科技论文的模式，结构上包括标题、摘要、目录、几节的正文和最后的参考文献；内容包括文字、公式、图形、表格等。短文的格式很平凡，没有什么特别的地方，但也足够实际，可以代表大多数使用 LaTeX 的人日常接触最多的文档类型，只不过现实中的例子在内容上比这里的例子更丰富、更深刻。

为了能在书中方便地显示这个例子，我们把短文的页面设置的很小，四页拼成一页，完成的样子如图1.6所示。如果你以前已经对 LaTeX 有一些基础，不妨自己动手排一下这个小例子（不偷看本章后面的说明），看看你能否准确高效地完成；即使你对 LaTeX 的实际了解有限，也不妨考虑一下，在这个极其简单的例子中，有哪些内容需要表现，它们对应的形式是什么，需要注意哪些问题。

1.2.2 从提纲开始

无论是对已经写好的文章进行排版，还是从零开始直接写文章，从提纲开始都是一个好主意。写出 LaTeX 文档的框架，进行必要的基本设置，然后再填入内容就方便了。

我们的例子《杂谈勾股定理》的提纲如下：

```
1  % -*- coding: UTF-8 -*-
2  % gougu.tex
3  % 勾股定理
4  \documentclass[UTF8]{ctexart}
5
6  \title{杂谈勾股定理}
7  \author{张三}
8  \date{\today}
9
10 \bibliographystyle{plain}
11
12 \begin{document}
13
14 \maketitle
15 \tableofcontents
16 \section{勾股定理在古代}
17 \section{勾股定理的近代形式}
18 \bibliography{math}
19
20 \end{document}
```

源文件的一些东西我们已经见过了，也有一些是没见过的。但可以看出整个文章的框架，现逐条进行说明：

- 前面以百分号% 开头的是**注释**。在 TeX 中，源文件一行中% 后的内容都会被忽略。这里有三行注释，第 1 行表明了这个文档的编码是 UTF-8，这对中文文档往往非常有用；第 2 行是源文件的文件名`gougu.tex`；第三行则说明了源文件的内容。注释并不是 TeX 源文件必需的，这里对文件内容的注释似乎与文档标题重复，不过对于比较大的文档，源文件往往分成几个文件，这类说明性文字就十分重要了。

杂谈勾股定理

张三

2023 年 10 月 4 日

摘要

这是一篇关于勾股定理的小短文。

目录

1 勾股定理在古代

西方称勾股定理为毕达哥拉斯定理，将勾股定理的发现归功于公元前 6 世纪的毕达哥拉斯学派 [1]。该学派得到了一个法则，可以求出可排成直角三角形三边的三元组。毕达哥拉斯学派没有书面著作，该定理的严格表述和证明则见于欧几里得[1]《几何原本》的命题 47：“直角三角形斜边上的正方形等于两直角边上的正方形之和。”证明是用面积做的。

我国《周髀算经》载商高（约公元前 12 世纪）答周工问：

> 勾广三，股修四，径隅五。

又载陈子（约公元前 7–6 世纪）答荣方问：

> 若求邪至日者，以日下为勾，日高为股，勾股各自乘，并而开方除之，得邪至日。

都较希腊较早。后者已经明确道出勾股定理的一般形式。图 1是我国古代对勾股定理的一种证明 [2]。

[1]欧几里得，约公元前 330–275 年。

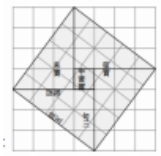


图 1: 宋赵爽在《周髀算经》注中作的弦图（仿制），该图给出了勾股定理的一个极具对称美的证明。

2 勾股定理的近代形式

勾股定理可以用现代语言表述如下：

定理 1 (勾股定理) 直角三角形斜边的平方等于两腰的平方和。

可以用符号语言表述为：设直角三角形 ABC，其中 $C = 90°$，则有

$$AB^2 = BC^2 + AC^2 \tag{1}$$

满足式(1)的整数称为勾股数。第 1 节所说毕达哥拉斯学派得到的三元数组就是勾股数。下面列出一些较小的勾股数：

直角边 a	直角边 b	斜边 c
3	4	5
5	12	13

$(a^2 + b^2 = c^2)$

参考文献

[1] 克莱因. 古今数学思想. 上海科学技术出版社, 2002.

[2] 曲安京. 商高、赵爽与刘徽关于勾股定理的证明. 数学传播, 20(3), 1998.

[3] 矢野健太郎. 几何的有名定理. 上海科学技术出版社, 1986.

图 1.6: 完整排版的小例子

- 第 4 行是**文档类**，因为是中文的短文，所以使用 ctexart ，并用[UTF8]选项说明编码。
- 第 6 行至第 8 行，声明了整个文章的**标题**、**作者**和**写作日期**，其中\today 当然是“今天”的日期。这些信息并不马上出现在编译的结果中，而要通过第 14 行的\maketitle排版。
- 第 10 行的\bibliographystyle声明**参考文献**的格式。

以上在\begin{document}之前的部分称为**导言区**（preamble ），导言区通常用来对文档的性质做一些设置，或者自定义一些命令。

- 第 12 行和第 20 行以\begin{document}和\end{document} 声明了一个 document 环境，里面是论文的正文部分，也就是直接输出的部分。
- 第 14 行的\maketitle命令实际输出论文标题。
- 第 15 行的\tableofcontents命令输出目录。
- 第 16 至 17 行的两个\section开始新的一节。
- 最后第 18 行的\bibliography{math} 则是提示 TEX 从文献数据库 math中提取文献信息，打印参考文献列表。

为了格式上的清晰，源文件中适当加入了一些空行作为分隔。在正文外的部分，空行不代表任何意义。

这里的提纲非常简单，整个文档也没有什么复杂的层次结构。编译提纲将得到只有一些标题的文件。我们并没有写任何编号或数字，所有编号、包括目录和页码都是自动生成的。注意这里生成目录至少要编译两次，让 LATEX 有机会读完整个论文来计算目录结构。

1.2.3 填写正文

```
1 西方称勾股定理为毕达哥拉斯定理，将勾股定理的发现归功于公元前␣6␣世纪的毕达哥拉斯学派。该学派得到了一个法则，
2 可以求出可排成直角三角形三边的三元数组。毕达哥拉斯学派没有书面著作，该定理的严格表述和证明则见于欧几里得《
3 几何原本》的命题␣47：“直角三角形斜边上的正方形等于两直角边上的正方形之和。”证明是用面积做的。
4
5 我国《周髀算经》载商高（约公元前␣12␣世纪）答周工问……
```

填写正文的部分看起来比较简单，就是直接填写大段的文字，不过仔细查看代码，也有如下一些要注意的地方（这里用␣表示空格。

- **使用空行分段**。单个换行并不会使文字另起一段，而只是起到使源代码更易读的作用。空白行，也就是至多有空白的行，会使文字另起一段。空行只是起分段的作用，使用很多空行并不起增大段间距的作用。
- **段前不用打空格**。LATEX 会自动完成文字的缩进。即使手工在前面打了空格，LATEX 也会将其忽略，事实上它会忽略每行开始的所有空格。也不要使用全角的汉字空格，这通常会使排版的效果变得糟糕。
- **通常汉字后面的空格会被忽略，而其它符号后面的空格则会保留**，因而用左␣右| 就得到连续的“左右”，但left␣right就会输出有空格的“left right ”。单个的换行就相当于一个空格，因此源代码中大段文字可以安全地分成短行。空格只起分隔单词或符号的作用，使用很多空格并不起任何增大字词间距的作用。使用xelatex 编译文档，ctexart 文档类会调用 xeCJK 宏包，自动处理汉字与其他符号之间的距离，无论你有没有在它们之间加上正确的空格，这是十分方便的。不过，在源代码中仍然可以给汉字与其它符号之间加上一个空格，这会使代码更加清晰。

换行与空格的使用，正是在 LATEX 中文字排版中最基本的部分，却也是最容易被忽略的。现在你的心思可能早已经飘到脚注和《周髀算经》的引用这些显眼的地方了，但在进行下一步之前最好还是巩固一下前面的内容。

1.2.4 命令与环境

继续排版短文的第 1 节，我们来处理脚注和引用内容。

脚注是在正文“欧几里得”的后面用脚注命令\footnote得到的：

```
……见于欧几里得\footnote{欧几里得，约公元前330--275年。}《几何原本》的……
```

在这里，\footnote 后面的花括号内的部分是命令的参数，也就是脚注的内容。

文中还使用\emph命令改变字体形状，表示强调（emphasis ）的内容：

```
……满足式的整数称为\emph{勾股数}。……
```

一个 LaTeX 命令（宏）的格式为：

```
无参数：      \command
有n个参数     \command{arg1}{arg2}……{argn}
有可选参数    \command[opt args]{arg1}{arg2}……{argn}
```

命令都以\ 开头，后接命令名，命令名或者是一串字母，或是单个符号。命令可以带一些参数，如果命令的参数不止一个字符（不包括空格），就必须用花括号括起来。可选参数如果出现，则用方括号括起来。这里的脚注命令\footnote 就是带有一个参数的命令，前面看到的\documentclass 就是一个能带可选参数的命令。

引用的内容则是在正文中使用quote 环境得到的。

```
……答周工问：
\begin{quote}
   勾广三，股修四，径隅五。
\end{quote}
又载陈子（约公元前 7--6 世纪答荣方问：
\begin{quote}
   若求邪至日者，以日下为勾，日高为股，勾股各自乘，并而开方除之，得邪至日。
\end{quote}
都较希腊较早。……
```

quote环境即以\begin{quote} 和 \end{quote} 为起止位置的部分。它将环境中的内容单单独分行，增加缩进和上下间距排版，以突出引用的部分。

不过，如果只使用quote 环境，并不能达到预期的效果：\quote 环境并不改变引用内容的字体，因此还需要再使用改变字体的命令，即：

```
\begin{quote}
   \zihao{-5} \kaishu
   勾广三，股修四，径隅五。
\end{quote}
```

这里，\zihao 是一个带有参数的命令，选择字号（-5 就是小五号）；而 \kaishu 则是一个没有参数的命令，把字体切换为楷书，注意用空格把命令和后面的文字分开。

类似地，文章的摘要也是在\maketitle 之后用 abstract 环境生成的：

```
\begin{abstract}
   这是一篇关于勾股定理的小短文。
\end{abstract}
```

摘要环境的预设已经满足我们的要求了，不必再修改了。

上面使用的选择字体字号的命令与之前的脚注命令不同。\footnote{内容} 只在原地发生效果，即生成脚注；但\zihao{字号} 与 \kaishu 命令则会影响后面的所有文字，直到整个分组结束。这种命令又称为声明（declaration ）。

分组限定了声明的作用范围。一个 LaTeX 环境天生就是一个分组（group ），因此前面的字号、字体命令会影响整个 quote 环境。最大的分组是表示正文的 document 环境，也可以用成对的花括号 { } 产生一个分组。

LaTeX 环境（environment ）的一般格式是：

```
\begin{环境名}
   环境内容
\end{环境名}
```

有的环境也有参数或可选参数，格式为：

```
\begin{环境名}[可选参数]{其它参数}
   环境内容
\end{环境名}
```

quote 环境是无参数的，后面我们很快会在制作表格时遇到有参数的环境。

文章第二节的定理，是用一类定理环境输出的。定理环境是一类环境，在使用前需要在导言区做定义：

```
\newtheorem{thm}{定理}
```

这就定义了一个叫thm 的环境。定理环境可以有一个可选参数，就是定理的名字。于是前面的勾股定理就可以由新定义的thm 环境得到：

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

   可以用符号语言表述为……
\end{thm}
```

最后来注意一个小细节，前面在表示起迄年份时，用了两个减号-- ，这在 LaTeX 中将输入一个“en dash ”，即宽度与字母“n ”相当的短线，通常用来表示数字的范围。

1.2.5 遭遇数学公式

现在来看我们最关心的问题——输入数学公式，这大概是多数使用 LaTeX 的人花费精力最多的地方了。

最简单的输入公式的办法是把公式用一对美元符号$ $ 括起来，如使用$a+b$ 就得到漂亮的 $a+b$ ，而不是直接输入 a+b 得到干巴巴的 a+b 。这种夹在行文之中的公式称为“正文公式”（in-text formula ）或“行内公式”（inline formula ）。

对比较长或比较重要的公式，一般单独居中写在一行；为了方便引用，经常还给公式编号。这种公式称为“显示公式”或“列表公式”（displayed formula ），使用equation 环境就可以方便地输入这种公式：

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

$$a(b+c) = ab + bc \tag{1.1}$$

键盘上没有的符号，就需要一个命令来输入。例如表示“角”的符号 ∠ 就可以用\angle 来输入。命令的名字通常就是符号的名字，“角”的符号是\angle，希腊字母 π 也就用其拉丁拼写\pi。用命令表示的数学符号在 LaTeX 中使用起来与用键盘输入的数学符号用起来没有什么差别：

```
$\angle ACB = \pi / 2$
```

$$\angle ACB = \pi/2$$

数学公式不只是符号的堆砌，还具有一定的数学结构，如上下标、分式、根式等。在勾股定理的表述中，就用到了上标结构表示乘方：

```
\begin{equation}
   AB^2 = BC^2 + AC^2
\end{equation}
```

$$AB^2 = BC^2 + AC^2 \tag{1.2}$$

符号^用来引入一个上标，而_则引入一个下标，它们用起来差不多等同于一个带有一个参数的命令，因此多个字符的上下标需要用花括号分组，如`$2^{10}=1024$`得到 $2^{10} = 1024$。

怎么输入 90°？如果去查数学符号表，你可能一无所获，由于 LaTeX 默认的数学字体中，并没有一个专门用于表示角度的符号，自然也就没有这个命令。角度的符号°是通过上标输入的：`$^{\circ}$`。这里的`\circ`其实是一个通常用来表示函数复合的二元运算符“∘”，我们把它的上标形式借用来表示角度，90° 可以用`$90^{\circ}$`输入。

这篇小短文用到的数学公式暂且就只有这么多，我们将在第 4 章再来深入探讨这个话题。

1.2.6 使用图表

准备图表比起输入文字和公式就要麻烦一些了，很多人能够驾驭十分复杂的数学公式，却往往在图表问题上一筹莫展。这篇关于勾股定理的短文使用的图表形式都比较简单，但也是典型的。

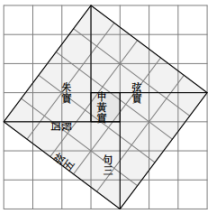
首先来看插图。在 LaTeX 中使用插图有两种途径，一是插入事先准备好的图片，二是使用 LaTeX 代码直接在文档中画图。大部分情况下都是使用插入外部图片的形式，只有在一些特别的情况下才大量用代码作图（如数学的交换图）。

插图功能不是由 LaTeX 的内核直接提供的，而是由 graphicx 宏包提供。要使用 graphicx 宏包的插图功能，需要在源文件的导言区使用`\usepackage`命令引入宏包：

```
\documentclass[UTF8]{ctexart}
\usepackage{graphicx}
% ……导言区其他内容
```

引入 graphicx 宏包之后，就可以使用`\includegraphics`命令插图了：

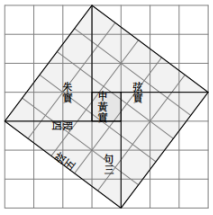
```
\includegraphics[width=3cm]{xiantu.pdf}
```

： 

这里`\includegraphics`有两个参数，方括号中的可选参数`width=3cm`设置图形在文档中显示的宽度为 3 cm，而第二个参数`xiantu.pdf`则是图形的文件名（放在源文件目录）。有最常见的情况，图形使用其他画图工具做好，但在制作的时候尺寸不符合文章的要求，需要在插图时设置参数缩放到指定的大小。还有一些类似的参数，如`scale=`缩放因子、`height=`高度 等，我们在这篇小短文中实际使用的是`scale=0.8`。插图命令支持的图形文件格式与所使用的编译程序有关，这篇中文文章使用 xelatex 命令编译，支持的图形格式包括 PDF、PNG、JPG、EPS 等，这里的图形实际是利用 Asymptote 语言制作的。

插入的图形就是一个有内容的矩形盒子，在正文中和一个很大的字符没有区别。因此如果把插图和文字混在一起，就会出现这样的情况：

```
  文字文字
\includegraphics[width=3cm]{xiantu.pdf}
  text text
```

文字文字：  text

text

除了一些很小的图标，我们很少把插图直接夹在文字之间，而是使用单独的环境列出。而且很大的图形如果固定位置，会给分页造成困难。因此，通常都把图形放在一个可以变动相对位置的环境中，称为“浮动体”（float）。在浮动体中还可以给图形加入说明性的标题，因此，在《杂谈勾股定理》中实际是使用下面的代码插图的：

```
\begin{figure}[ht]
   \centering
   \includegraphics[scale=0.6]{xiantu.pdf}
   \caption{宋赵爽在《周髀算经》注中作的弦图（仿制），该图给出了勾股定理的一个极具对称美的证明。}
   \label{fig:xiantu}
\end{figure}
```

在上面的代码中，第 1 行和第 6 行使用了figure 环境，就是插图使用的浮动体环境。figure 环境有可选参数ht，表示浮动体可以出现在环境周围的文本所在处（here ）和一页的顶部（top ）。figure 环境内部相当于普通的段落（默认没有缩进）；第 2 行用声明\centering 表示后面的内容居中；第 3 行插入图形；第 4 行和第 5 行使用\caption 命令给插图加上自动编号和标题；第 6 行的\label 命令则给图形定义一个标签，使用这个标签就可以在文章的其他地方引用\caption 产生的编号（编号引用我们会在后面讲到）。这段插图的代码非常格式化，在绝大多数情况下，文章中的插图都是用与这里几乎完全一样的代码插入的。

下面再来看表格。插图可以用其他软件做好插入，但表格一般都是在 LaTeX 里面完成的。制作表格，需要确定的是表格的行、列对齐模式和表格线，这是由tabular 环境完成的：

```
\begin{tabular}{|rrr|}
   \hline
   直角边 $a$ & 直角边 $b$ & 斜边 $c
       $ \\
   \hline
   3 & 4 & 5 \\
   5 & 12 & 13 \\
   \hline
\end{tabular}
```

直角边 a	直角边 b	斜边 c
3	4	5
5	12	13

tabular 环境有一个参数，里面声明了表格中列的模式。在前面的表格中，rrr|| 表示表格有三列，都是右对齐，在第一列前面和第三列后面各有一条垂直的表格线。在tabular 环境内部，行与行之间用命令\\ 隔开，每行内部的表项用符号& 隔开。表格中的横线则是用命令 \hline 产生的。

表格与\includegraphics 命令得到的插图一样,都是一个比较大的盒子。一般也放在浮动环境中,即table 环境，参数与大体的使用格式也与figure 环境差不多，只是\caption 命令得到的标题是“表”而不是“图”。在《杂谈勾股定理》中，我们稍稍改变了一下figure 环境通常的内容：

```
\begin{table}[H]
   \begin{tabular}{|rrr|}
      \hline
      直角边 $a$ & 直角边 $b$ & 斜边
          $c$ \\
      \hline
      3 & 4 & 5 \\
      5 & 12 & 13 \\
      \hline
   \end{tabular}%
   \qquad
   ($a^2 + b^2 = c^2$)
\end{table}
```

直角边 a	直角边 b	斜边 c
3	4	5
5	12	13

$(a^2 + b^2 = c^2)$

这里并没有给表格加标题，也没有把内容居中，而是把表格与一个公式并排排开，中间使用一个 \qquad 分隔。命令 \qquad 产生长为 2em （大约两个“M ”的宽度）的空白。因为我们已经使用\qquad 生成足够长度

的空格了，所以再用\end{tabular}后面的注释符取消换行产生的一个多余的空格，这正好达到我们的预期效果。

之所以使用这种方式防止表格，是因为在正文中表格前面写道：

……下表列出一些较小的勾股数：

也就是说表格和正文是直接连在一起的，而且后面的公式也说明了表格的意义，自然也就不需要多余的标题了，这样一来表格就于正文连在一起，不允许再浮动了，因而这里本来是不应该使用浮动的table环境的，但我们仍然使用了table环境，在表示位置的参数处使用了[H]，表示“就放在这里，不浮动”。[H]选项并不是标准 LaTeX 的table的参数，而是由 float 宏包提供的特殊功能。因此要让上面的代码正确运行，还要在导言区使用\usepackage{float}。在这种表格很小（不影响分页），行文又要求连贯的场合，float 宏包的这种不浮动的图表环境是很有用的。

1.2.7 自动化工具

到目前为止，《杂谈勾股定理》这篇小文的大部分内容已经排完了，如果把前面提到的所有代码结合在一起，装进一个文件，差不多就能得到一篇完整的文章了——这里说“差不多”，其实还缺少一些重要的东西，最明显的就是参考文献列表。

你一定已经注意到了，在前面文档的提纲中，我们已经用\bibliographystyle命令声明了参考文献的格式，又用\bibliography命令要求打印出参考文献列表。不过，这只是使用 BibTeX 处理文献的一个空架子，我们尚没有定义“参考文献数据库”，自然也不会产生任何文献列表。

BibTeX 使用的参考文献数据库其实就是一个后缀为.bib的文件。我们的《杂谈勾股定理》使用了一个包含 3 条文献的数据库文件math.bib，内容如下：

```
% This file was created with JabRef 2.6
% Encoding: UTF8

@BOOK{Kline,
   title = {古今数学思想},
   publisher = {上海科学技术出版社},
   year = {2002},
   author = {克莱因}
}

@ARTICLE{quanjing,
   author = {曲安京},
   title = {商高、赵爽与刘徽关于勾股定理的证明},
   journal = {数学传播},
   year = {1998},
   volume = {20},
   number = {3}
}

@BOOK{Shiye,
   title = {几何的有名定理},
   publisher = {上海科学技术出版社},
   year = {1986},
   author = {矢野健太郎}
}
```

正如上面所看到的，一个文献数据库文件的格式并不复杂，每则文献包括类型、引用标签、标题、作者、出版年、出版社等信息，可以直接手工输入。不过正像前面数据库文件的注释（以% 开头的两行）所显示的那样，这个数据库文件并不是直接输入上面的文件内容得到的，而是使用文献管理工具 JabRef 制作的。

在现实中，BibTeX 数据库经常并不需要我们自己录入，而可以从相关学科的网站直接下载或是从其他类型的文献数据库转换得到。即使是在需要我们自己录入的情况下，使用 JabRef 这种软件来管理也更方便，不易出错。

BibTeX 是一个专门用于处理 LaTeX 文档文献列表的程序吗，使用 BibTeX 处理文献时，编译 `gougu.tex` 这一个文档的步骤就增加为四次运行程序（或点击四次按钮）：

```
xelatex gougu.tex
bibtex gougu.tex
xelatex gougu.tex
xelatex gougu.tex
```

第一次运行 `xelatex` 为 BibTeX 准备好辅助文件，确定数据库中的哪些文献将被列出来。然后`\bibtex` 处理辅助文件 `gougu.tex`，从文献数据库中选取文献，按照指定的格式生成文献列表的 LaTeX 代码。后面两次 `xelatex` 再读入文献列表代码并生成正确的引用信息。这种利用多躺编译处理辅助文件的方式看起来有些复杂，但这是使用自动文献生成的代价之一，不过好处也是明显的，文献的管理、文献列表的排序和排版格式等都能高效漂亮地完成。

如果现在就使用上面的步骤编译，你仍然会一无所获，因为还没有选择要列出的文献。LaTeX 只选择被引用的文献。引用文献的方法是在正文中使用`\cite` 命令，如：

```
西方称勾股定理为毕达哥拉斯定理，将勾股定理的发现归功于公元前 6 世纪的毕达哥拉斯学派\cite{Kline}。

……是我国古代对勾股定理的一种证明\cite{quanjing}。
```

`\cite` 命令的参数 Kline 和 quanjing 分别是其中两篇文献的引用标签，也就是在 `math.bib` 中每个条目第一行出现的东西。使用 `\cite` 命令会在引用的位置显示文献在列表中的编号（它在第 3 次 `xelatex` 编译之后才能确定），同时在辅助文件中说明某文献将被引用。如果要在列表中显示并不直接引用的文献，可以使用 `\nocite` 命令，一般是把它放在`\bibliography` 之前，像我们这篇文章一样：

```
\nocite{Shiye}
\bibliography{math}
```

有了上面的引用代码，加上完整的数据库文件，通过多步编译，最终就能得到文献列表。

BibTeX 是这篇文章用到的最复杂的自动化工具，最简单的自动化工具则是页码、定理和公式的自动编号，其余的还包括生成目录与图表公式的交叉引用。

目录也是自动从章节命令中提取并写入目录文件的，我们在提纲中就使用了`\tableofcontents` 命令，它将在第二次 `xelatex` 编译时生效。

引用不仅限于参考文献。图表、公式的编号，只要事先设定了标签，同样可以通过辅助文件为中介引用。基本的交叉引用命令是`\ref`，它以标签为参数，得到被引用的编号。例如，在插图时已经用`\label` 命令为弦图定义了标签`fig:xiantu`，于是，在正文中就可以使用

```
图\ref{fig:xiantu} 是我国古代对勾股定理的一种证明 \cite{quanjing}。
```

来对弦图的编号进行引用。

公式编号的应用也可照此办理，不过需要在公式中定义标签：

```
\begin{equation}\label{eq:gougu}
   AB^2 = BC^2 + AC^2
\end{equation}
```

而后在正文中以\ref{eq:gougu} 引用。实际中引用公式非常常用，数学宏包 amsmath 就定义了 \eqref 命令，专门用于公式的引用，并能产生括号：

```
% 导言区使用 \usepackage{amsmath}
满足式 \eqref{eq:gougu} 的整数称为 \emph{勾股数}。
```

1.2.8 设计文章的格式

写到这里，原先的提纲骨架已经变成一篇完整的文章，似乎已经没有什么可说的了。然而，TEX 的精神是精益求精、追求完美，如果我们对比预定的目标来审视我们现在排版的结果，还是会发现有些不同，如标题的字体还需要修正，目录中少了“参考文献”一项，插图标题的字体、字号和对齐都不正确等。更重要的还有，目标预设的文章页面很小，页边距也非常紧凑，与现在宽大的页面大相径庭。这些都属于文章的整体格式，需要进一步的设计完善。

绝大部分设计工作是在文章的导言区通过一些命令定义和参数设定来完成的，但往往相当复杂，好在其中的大多数工作可以通过使用一些宏包来简化，前面已经用到过 graphicx 、float 、amsmath 几种宏包完成一些工作，这里也要用到几种。

设计页面尺寸可以使用 geometry 宏包：

```
\usepackage{geometry}
\geometry{a6paper,centering,scale=0.8}
```

这是最简单的设定方式，定义页面使用 A6 纸大小，版心居中，长宽占页面的 0.8 倍。

改变图表标题格式可以使用 caption 宏包：

```
\usepackage[format=hang, font=small, textfont=it]{caption}
```

设定图表所有标题使用悬挂对齐方式（即编号向左突出），整体用小字号，而标题文本使用斜体（对汉字来说就是楷书）。

增加目录的项目则可以用 tocbibind 宏包：

```
\usepackage[nottoc]{tocbibind}
```

宏包默认会在目录中加入目录项本身、参考文献、索引等项目。这里使用 nottoc 选项取消了在目录中显示目录本身。

标题和作者的字体可以直接在\title、\author 命令中设定，因为标题本身就是用这些命令在导言区定义的：

```
\title{\heiti 杂谈勾股定理}
\author{\kaishu 张三}
\date{\today}
```

其中\heiti 是和 \kaishu 类似的中文字体命令，把字体切换成黑体。

这篇短文到这里就全部排完了，不过正文中表示引用的quote 环境里面还夹杂着字体命令，这种散落在各处的格式设置很难看清，而且不方便修改。为了解决这个问题，可以利用 \newenvironment 命令定义一个新的环境，在原来quote 的基础上再增加格式控制：

```
\newenvironment{myquote}
{ \begin{quote} \kaishu \zihao{-5} }
{ \end{quote} }
```

这里，\newenvironment 有三个参数，第一个参数是环境的名字，后两个参数分别是在环境开始和末尾处的代码，因此，就可以使用新环境

```
\begin{myquote}
    勾广三，股修四，径隅五。
\end{myquote}
```

来替换原来的quote 环境了。如果此时需要修改引用的格式，那么只需要在导言区修改 myquote 的定义，而不必在全文中搜索所有的quote 环境的使用了。

类似地，原来数学公式中角度的单位^\circ 也不直观，可以用 \newcommand 命令定义一个新的命令\degree：

```
\newcommand{\degree}{^\circ}
```

其中，\newcommand 命令的两个参数分别是新命令和新命令的定义，于是我们就可以用 $90\degree$ 来代替原来不直观的 90° 了。

在整篇文章编排结束之际，我们还是使用自定义的环境myquote 和自定义的命令\degree 代替了文中出现的特殊格式控制。类似地，在设定插图标题的字体时，并没有把字体、字号的命令塞进\caption 命令的参数中，而是使用 \caption 宏包统一管理。这样看起来比最“直接”的做法要多绕一道弯子，但好处是更清晰和更容易修改格式。这篇短文排在普通 A4 大小的纸张上只有一两页，还看不出什么特别之处，但当你开始编写和维护几十上百页的长文档时，在设计阶段所付出的精力就会得到回报了。

LaTeX 是一种结构化的排版语言，在填写标准格式的模板时（就像我们前面写的提纲）可以忽略编号、格式等许多具体细节。在文档排版中应该主动追求内容与格式的分离，在document 环境之内避免直接使用诸如字体字号、对齐缩进的格式控制命令，而取而代之的应该是有具体意义的环境和命令，让文档变得清晰。这种模式化的操作能够提高工作效率，许多 LaTeX 的拥护者把这种工作方式称为“所想即所得”。可是不要忘记，机器还远没有智能化到想人之所想的程度，LaTeX 也不能阻止我们编排出效果糟糕、代码混乱的文章，要想得到好的文章，无论是在内容上还是在排版形式上，都得靠我们自己。

第二章 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，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

在 LaTeX 中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：

χαϊδεύηζ Крюкова

上面的例子中，前一组词是来自拉丁字母，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 XeTeX 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

χαϊδεύηζ \quad Крюкова
```

café Gödel Antonín Dvořák
χαϊδεύηζ Крюкова

不过，要想正确输入、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 7 8 9 0 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，TeX 系统输出时所使用的字体，也必须要能直接显示这些字母。LaTeX 默认的 Computer Modern 在一个字体中只覆盖很小的字符集，XeLaTeX 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、西里尔这些字体需要更换其他字体。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 TEX 中使用覆盖更大字符集的字体。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 支持的重音标记及其命令见表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	ȯ	\.o	ŏ	\u{o}
ǒ	\v{o}	ő	\H{o}	o͡o	\t{oo}	ů	\r{o}
ǫ	\c{o}	ọ	\d{o}	o̲	\b{o}		

使用命令可以输入的 LATEX 字母，见表2.2。其中，ı，ȷ 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	Ł	\L	ł	\l
Ø	\O	ø	\o	ı	\i	ȷ	\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，可以方便地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

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

将使用英语和希腊语，其中作为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 αβζδε 。需要少量俄文的西里尔字母，可以换用 OT2 编码的字体德奥。

例如：

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

АБЦабц

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。由于这些语言很少用，这里不做更多说明。

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：

```
% coding: utf-8
% pdflatex 命令编译
\documentclass{article}
\usepackage[OT2,OT1]{fontenc}
\usepackage[utf8]{inputenc}
```

```
\begin{document}
café \quad Gödel
{\fontencoding{OT2} \selectfont КрЮкова}
\end{document}
```

LaTeX 在排版中会将单词中的一些字母连写成为一个符号，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 和 Latin Modern 字体中，小写字母组合 `ff,fi,fl,ffi,ffl` 都有连字：

```
differ find flight difficult ruffle
```

differ find flight difficlut ruffle

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用的更多：

ﬀ ﬁ ﬂ ﬃ ﬄ ﬅ ﬆ

偶尔出于意义或美观的考虑，需要取消连字。此时可以借用`\/`命令。

```
fleet f\/leet
```

fleet fleet

使用 XeLaTeX 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。

2.1.1.2 正确使用标点

在键盘上，可以直接使用的符号有 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，这些标点有空格还有一些更微妙的关系。

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。如果遇到单引号和双引号连续出现的情形，则在中间用 `\,` 命令分开：

```
``\,`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割单词。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。

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：

```
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.

不过，按中文写作习惯，表示数字范围也常使用符号 ~（`$\sim$`），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，它所拉开的距离要合理些：

```
Good: One, two, three \ldots

Bad: One, two, three...
```

Good: One, two, three …
Bad: One, two, three...

\ldots 和 \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、叹号等标号，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，通常在句中使用时，前后都要加空格，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 ”（后一个间距比前面大）。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….

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大排的一大半：

~ # $ % ^ & { } _ \

他们都有特殊作用，其中的许多我们已经熟知：数学模式符号 $ 、注释符号% 、上标 ^ 、分组 {} 、宏命令 \，只有个别例外：

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

~ # $ % ^ & { } _ \

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，而出现其他地方：可以是重音符号；可以出现在程序代码中；此外还有一个数学符号 ~。

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，一般不直接使用它们。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	•
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+- 和 Shift+6 得到。

在科技文章中，为与数字、字母区分，中文的句号一般也用一个圆点表示，此时应该使用全角的“．”而非混用西文句点。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 XeTeX 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，默认是全角式，即所有标点占一个汉字宽度，只有在行末或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：

\punctstyle{quanjiao} 全角式，所有标点全角，有挤压。例如，“标点挤压”。又如《标点符号用法》。
\punctstyle{banjiao} 半角式,所有的标点半角,有挤压。例如,“标点挤压”。又如《标点符号用法》。
\punctstyle{kaiming} 开明式,部分的标点半角,有挤压。例如,“标点挤压”。又如《标点符号用法》。
\punctstyle{hangmobanjiao} 行末半角式，仅行末挤压。例如，“标点挤压”。又如《标点符号用法》。
\punctstyle{plain} 无格式，只有禁则，无挤压。例如，“标点挤压”。又如《标点符号用法》。

2.1.1.3 看不见的字符——空格与换行

文本中的空格起分隔单词的作用，任意多个空格的功能与一个空格相同；只有字符后面的空格是有效的，每行最前面的空格则被忽略，这样有利于复杂代码的对齐；单个换行也被看成一个空格。例如（我们仍然用 ␣ 表示空格：

```
This␣is␣␣␣␣␣a␣short
sentence.␣␣This␣is
␣␣␣␣␣␣␣␣␣␣␣␣another.
```

This is a short sentence. This is another.

以字母命名的宏，后面空格会被忽略。如果需要再命令后面使用空格，可以使用\␣，它表示两个普通字母间的空格距离；也可以在命令后加一个空白的分组{}，有时也可以把命令用一个分组包裹起来：

```
Happy␣\TeX␣ing.␣Happy␣\TeX\␣ing.
Happy␣␣\TeX{}␣ing.␣Happy␣{\TeX}␣ing.
```

Happy TEXing. Happy TEX ing.
Happy TEX ing. Happy TEX ing.

有一种不可打断的空格，在 TEX 中称为带子（ties ），用~ 表示。TEX 禁止在这种空格之间分行，因而可以用来表示一些不宜分开的情况，例如：

```
Question~1          % 名称与编号间
Donald~E. Knuth     % 教名之间，但姓可以断行
Mr.~Knuth           % 称谓缩写与名字间
function~$f(x)$     % 名字后面的短公式
1,~2,and~3          % 序列的部分符号间
```

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 会把大写字母后的点看作是缩写标记，把小写后面的点看作是句子结束，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，这就必须明确地告诉 LATEX 使用普通单词间的空格 \␣，或用\@，指明. 是大写字母后的句末。例如：

```
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。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：

```
中文和English的混排效果并不依赖␣space␣的有无
```

中文和 English 的混排效果并不依赖 space 的有无

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

```
\mbox{条目}a␣不同于条目b
```

条目a 不同于条目 b

还有时需要完全禁用汉字与其他内容之间的空格，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[1]（默认是一个空格）：

```
\CKJsetecglue{}
汉字word
```

汉字word

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

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适合的缩进。任意多个空行有一个空行的效果相同。

分段也可用用\par 命令生成，这种方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 LaTeX 直接另起一行，并不分段。优良中相关的命令：\\ 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，上一行仍按照一行散开对齐：

```
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

\\ 一般用于特殊的环境中，如排版诗歌的 verse 环境，特别是在对齐、表格和数学公式中使用广泛，但很少用在普通正文的行文中；\linebreak 命令则用于对个别不适合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：

[1]好像现在这个方法已经失效

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

$$[2-(3+5)]\times 7 = 42$$
$$[2+(3-5)]\times 7 = 0$$

2.1.2 特殊符号

除了一般的文字，有时还需要一些特殊的符号，其中在正文中最为常用的如表2.3所示。

表 2.3: 正文中常用的部分特殊符号

§	\S	†	\dag	‡	\ddag	¶	\P
©	\copyright	®	\textregistered	™	\texttrademark	£	\pounds
•	\textbullet						

上面几种符号中，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。

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，也定义了其他大量的符号，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 ['lɑtɛk] (\textipa{["lAtEk]})）；dingbat 、bbding 、pifont 等宏包提供了许多指示、装饰性的小符号，如 ☞[2]，等等。在这里一一枚举所有这些宏包和命令冗长无味，也没有必要，Pakin 编写的“LaTeX 符号大全”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 ），可以随意调用；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”对这些情况并没有仔细区分，可能需要查看相应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，而像\'e 这样的符合重音标记也会自动转为字体中带有重音的字母。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\' 这样的复合形式，你也可能需要更换其他字体或切换编码来显示这些符号。

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：

```
® © £ § ¶ † ‡ • ™ € ‰
```

® © £ § ¶ † ‡ • ™ € ‰

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，可以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 编码也只支持西文的拼音文字，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使用键盘输入，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，

[2]这里用的是 dingbat 的 \leftpointright 命令

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表示，其语法形式如表所示。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	数字	90
十六进制	"数字	"5A
八进制	'数字	'132
字符形式	`数字	`Z

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 ）在文档都列出了很长的字体号码表，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：

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

Þþ

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：

```
\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”的时候，指的是什么意思呢？可能是想换掉文字的整体感觉，如“TEXT”和“TEXT”；可能是想把直立的的文字改成倾斜的，如“TEXT”和“*TEXT*”；可能是想把细的文字加粗，如“TEXT”和“**TEXT**”；可能是想把包含一些符号的字体改为包含另一种符号的；还可能只是想改变字体的大小，如“文字”和“TEXT”。尽管所使用的文字内容都是一样的（“TEXT ”）。

上面说的五种性质，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，看作不同于字体的单独设置；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（font series ）。

LaTeX 提供了带参数的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}

{\bfseries Bold font test}
```

Italic font test

Bold font test

预定义命令的字体族由 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	\textrm{文字}	\rmfamily	Roman font family
无衬线	\textsf{文字}	\sffamily	Sans serif font family
打字机	\texttt{文字}	\ttfamily	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 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	\textup{(文字)}	\upshape	Upright shape
意大利	\textit{(文字)}	\itshape	*Italic shape*
倾斜	\textsl{(文字)}	\slshape	*Slanted shape*
小型大写	\textsc{(文字)}	\scshape	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”往往指意大利形状，它类似于更加圆滑的手写体。因为数学公式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 LaTeX 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的一种；很多字体也可能缺少小型大写字符的符号。另一方面，一些其他字体可能也会提供更多的字体形状，如 Venturis ADF 系列字体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	\textmd{文字}	\mdseries	Medium series
加宽加粗	\textbf{文字}	\bfseries	**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，同一维度的性质不能叠加，但不同类的性质则可以。三种性质的组合效果见下表：

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 XeLaTeX 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	
形状 \ 系列	\mdseries	\bfseries	\mdseries	\bfseries	\mdseries	\bfseries
\upshape	Text	**Text**	Text	**Text**	Text	**Text**
\itshape	*Text*	***Text***	同 *slanted*	同 ***slanted***	*Text*	同 ***slanted***
\slshape	*Text*	***Text***	*Text*	***Text***	*Text*	***Text***
\scshape	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：

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

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

使用斜体声明（\itshape 、\slshape 时，最后一个倾斜的字母会超过右边界，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
```

*M*M 倾斜
*M*M
*M*M

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。倾斜校正一般只用在声明形式的斜体命令中，但偶尔也使用它取消连字，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命令的参数形式）：

```
Bold `{\bfseries leaf}'

Bold `{\bfseries leaf\/}'

Bold `\textbf{leaf}'
```

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

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
```

*M*M
*M*M

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字体，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，选择中文字体族通常用 CJKfamily 命令[3]，如：

```
{\CJKfamily{hei} 这是黑体}

{\CJKfamily{kai} 这是楷体}
```

这是黑体
这是楷体

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）；如果使用了其他选项，则可能会有不同字体，为了方便实用，ctex 宏包提供了简化的命令：

[3]还是不知道怎么用

```
{\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 ，粗体是黑体，意大利体是楷体，如：

```
% 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（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 LaTeX2.09 中直接制定具体字体旧方案而说的。在旧的字体选择方案中，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 和旧版本的 LaTeX2.09 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 LaTeX2ϵ 在标准文档类中也保留了 \rm，\bf，\it，\sc，\sl，\sf，\tt等命令（以及数学模式底下的\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命令来表示。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、Adobe PostScript 字体文档。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

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

2.1.3.2 使用更多字体

使用 \rmfamily,\bfseries,\itshape,\kaishu 这类命令时，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，但如果想改变文章的整体风格，或者缺少预装的字体文件（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。

选择非默认的字体的方法有两类：当不使用 XeTeX 引擎时，可以通过字体宏包或 LaTeX 底层字体命令调用在 TeX 系统中安装的中西文字体；当使用 XeTeX 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 TeX 系统中只安装了很少几种质量较差的中文字体，一般总要调用操作系统中的字体来使用 XeTeX 引擎的功能。

一个 TeX 发行版通常都同时安装了大量的西文字体，直接使用 LaTeX 底层命令指定字体通常很难用，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，非常繁琐，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很好地配合。有几个宏包可以达到这个目的，最简单的是 times 宏包，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，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，通常只需要导入该宏包即可：

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

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时最为常用的，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），就需要综合使用字体包 ccfonts 和 euler:

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

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

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 字体和 Euler 字体效果（T1 编码，ccfonts，euler）

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，同一组的字体可能有不同编码的版本。除了 XeLaTeX 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 、拓展正文字体 T1 、数学字母 OML 、数学符号 OMS 、数学符号拓展 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 ，而拓展编码 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，才能获得更好的效果，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，与输入文档使用的文档编码不是一回事。

初拥 LaTeX 的最大的问题往往是并不知道 LaTeX 中哪些字体可用，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文档是 Stephen 的 HartkeFree_Math_Font_Survey[4]，里面介绍了 20 多种带有数学支持的免费 TeX 字体。该文档在 TeX Live 和 CTeX 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 LaTeX 字体目录是 FontCatalogue [5]，它展示了 Linux 源中的 TeX 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 XeTeX 来选择字体。在 XeLaTeX 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 [6] ）

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}
```

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}
```

[4]https://www.ctan.org/tex-archive/info/Free_Math_Font_Survey/en

[5]https://tug.org/FontCatalogue/

[6]https://ctan.org/tex-archive/macros/unicodetex/latex/fontspec

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.}
```

This is Lucida Sans.

XeLaTeX 下中文字体的设置使用 xeCJK 宏包（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己定义字体，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 PL 报宋 GBK”
% 并设置中文的 kai 字体族为 “文鼎 PL 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-1st 命令来列出，通常列出的字体信息会非多，不能在一屏内完全显示。Windows 下还会因编码不统一品出现乱码，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）：

```
D:/application/texlive/2023/texmf-dist/fonts/truetype/public/dejavu/DejaVuSerifCondensed-
    Italic.ttf: DejaVu Serif,DejaVu Serif Condensed:style=Condensed Italic,Italic
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/domitian/Domitian-Bold.otf:
    Domitian:style=Bold
D:/application/texlive/2023/texmf-dist/fonts/opentype/google/roboto/RobotoSerif-
    ExtraLight.otf: Roboto Serif,Roboto Serif ExtraLight:style=ExtraLight,Regular
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/drm/drmdozittc9.otf:
    drmdozittc9:style=Regular
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/drm/drmdozl7.otf: drmdozl7:
    style=Regular
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/cm-unicode/cmunoti.otf: CMU
    Concrete:style=Italic
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/archivo/Archiv0-MediumItalic
    .otf: Archiv0 Medium:style=Medium Italic,Italic
D:/application/texlive/2023/texmf-dist/fonts/opentype/public/kerkis/Kerkis-BoldSmallCaps.
    otf: Kerkis:style=BoldSmallCaps
C:/Windows/fonts/LHANDW.TTF: Lucida Handwriting:style=Italic,kursiv,Cursiva,Kursivoitu,
    Italique,Corsivo,Cursief,Itálico
C:/Windows/fonts/vgasysr.fon: System: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 LaTeX 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。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 The XeTeX Companion-TeX meets OpenType and Unicode [7] 、fontspec 宏包文档 [8]。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。如可以设置正文学体为宋体，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=SmHei,ItalicFont=KaiTi,BoldItalicFont=LiSu]{SimSun}
```

ctex 宏包默认设置 Windows 字体时正是采用类似以的方法。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`\setmahtrm`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
```

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
```

[7] https://xml.web.cern.ch/XML/lgc2/

[8] https://ctan.org/pkg/fontspec

```
\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 XeLaTeX 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学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 Unicode 字体编码设置为默认的编码，即最后一个选项。这种方法不影响使用 fontspec 中`\newfontfamily`声明的字体和 XeTeX 声明的汉字字体。但 `\setmainfont`，`setsansfont` 和 `setmonofont` 这三人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 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.
汉字楷书与 {\times Times New Roman} 字体。
\texttt{Consolas 0123}
\end{document}
```

XeTeX 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 XeTeX 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[9]会把国际音标字体包 tipa 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。因此默认的正文体 Latin Modern 因为符号不全就不能用于国际音标输出，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}
% \usepackage{xunicode} 我这里加这行才能编译
% 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}
```

[9]现在好像不会自动加载了

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 XeTeX 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}
```

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。如使用 Computer Modern 的 Fibonacci 字体，就可以直接设置字体族：

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

Computer Modern Fibonacci Roman

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}
```

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学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}
```

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 XeTeX 也提供 \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}
```

最后介绍一个有用的宏包 fonttable，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 T1 编码下 Latin Modern Roman 的字体符号表：

```
\xfonttable{T1}{lmr}{m}{n}
```

	′0	′1	′2	′3	′4	′5	′6	′7	
′00x	` 0	´ 1	ˆ 2	˜ 3	¨ 4	″ 5	° 6	ˇ 7	″0x
′01x	˘ 8	¯ 9	˙ 10	¸ 11	˛ 12	, 13	‹ 14	› 15	
′02x	“ 16	” 17	„ 18	« 19	» 20	– 21	— 22	23	″1x
′03x	0 24	1 25	J 26	ff 27	fi 28	fl 29	ffi 30	ffl 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	< 60	= 61	> 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	Ă 128	Ą 129	Ć 130	Č 131	Ď 132	Ě 133	Ę 134	Ğ 135	″8x
′21x	Ĺ 136	Ľ 137	Ł 138	Ń 139	Ň 140	Ŋ 141	Ő 142	Ŕ 143	
′22x	Ř 144	Ś 145	Š 146	Ş 147	Ť 148	Ţ 149	Ű 150	Ů 151	″9x
′23x	Ÿ 152	Ź 153	Ž 154	Ż 155	IJ 156	İ 157	đ 158	§ 159	
′24x	ă 160	ą 161	ć 162	č 163	ď 164	ě 165	ę 166	ğ 167	″Ax
′25x	í 168	ľ 169	ł 170	ń 171	ň 172	ŋ 173	ő 174	ŕ 175	
′26x	ř 176	ś 177	š 178	ş 179	ť 180	ţ 181	ű 182	• 183	″Bx
′27x	ÿ 184	ź 185	ž 186	ż 187	ij 188	¡ 189	¿ 190	£ 191	
′30x	À 192	Á 193	Â 194	Ã 195	Ä 196	Å 197	Æ 198	Ç 199	″Cx
′31x	È 200	É 201	Ê 202	Ë 203	Ì 204	Í 205	Î 206	Ï 207	
′32x	Đ 208	Ñ 209	Ò 210	Ó 211	Ô 212	Õ 213	Ö 214	Œ 215	″Dx
′33x	Ø 216	Ù 217	Ú 218	Û 219	Ü 220	Ý 221	Þ 222	SS 223	
′34x	à 224	á 225	â 226	ã 227	ä 228	å 229	æ 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	œ 247	″Fx
′37x	ø 248	ù 249	ú 250	û 251	ü 252	ý 253	þ 254	ß 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利体，把意大利体改为直立体：

```
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：

```
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，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：

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

This is **emphasized** text.

下划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划线：

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

Emphasized text and another.

不过\underline 的一个很大的缺点是下划线的部分不能行，如果仔细看上面的例子还会发现下划线与文字的距离不整齐。ulem 宏包的\uline 命令解决了这些问题，并且默认情况下还会把 \emph命令也改为使用下划线：

```
% 导言区使用 \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.

如果不希望用下划线代替标准的\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} \qquad
\uwave{boat} \qquad
\sout{wrong} \qquad
\xout{removed} \qquad
\dashuline{dashing} \qquad
\dotuline{dotty}
```

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

CJKfntef 宏包对汉字也提供了类似的功能，同时也进行了一些扩充：

```
\CJKunderdot{汉字，下加点} \\
\CJKunderline{汉字，下画线} \\
\CJKunderdblline{汉字，下画线} \\
\CJKunderwave{汉字，下画线} \\
\CJKsout{汉字，删除线} \\
\CJKxout{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*~~

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

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。

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 TEX 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 TEX 系统时，也同时为 TEX 设计了配套的字体格式和字体描述语言，这就是 METAFONT。TEX 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，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 → 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 z10dir -28..dir -15z5..z5dir -165..dir -152z6 --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 TEX 字体度量文件（TeXfontmetric）.tfm。TEX 引擎从 .tfm文件中读入字符尺寸信息，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 .gk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。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 Type1 格式的使用后，TEX 中原来用 METAFONT 描述的字体也都纷纷转换为 TyPe1 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 Type3 格式。TyPe1 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 TEX 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 TEX 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，pdfTeX，XeTeX 和 LuaTeX 都支持 TrueType 和 OpenType 字体格式，不过 dvipdfmx 和 pdfTeX 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 XeTeX 和 LuaTeX 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。

```
{\tiny tiny}\\
{\scriptsize script size} \\
{\footnotesize footnote size} \\
{\small small} \\
{\normalsize normal size} \\
{\large large} \\
{\Large larger} \\
{\LARGE even larger} \\
{\huge huge} \\
{\Huge largest}
```

tiny
script size
footnote size
small
normal size
large
larger
even larger
huge
largest

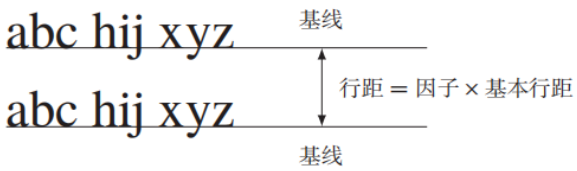
字号命令表示的具体尺寸随着所使用的文档类和大小选项的不同司而不同。在标准 LaTeX 文档类 article, report 和 book 中，可以设置文档类选项 10 pt,11 pt 和 12 pt, 全局地设置文档的字号，默认为 10 pt，即\normalsize的大小为 10 pt，这个数值表示字体中一个\quad 的长度，通常也就是整个符号所占盒子的高度（对汉字来说，一般也相当于汉字宽度）。

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、显示数学公式的垂直间距。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 2.5。

类似地，ctexart,ctexrep 和 ctexbook 文档类也提供了两个选项 c5size 和 cs4size 影响全文的字体大小和 \normalsize 等命令的意义（见表2.6）。c5size 和 cs4size分别表示 \normalsize为五号字和小四号字，默认为 c5size。

LaTeX 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。



可以使用命令\linespread{因子}来设置实际行距为基本行距的倍数。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2.7。

表 2.5: 中文字号

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

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，如设置为:

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

即要求两行间至少有 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，可以使用的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（1pc = 12pt，相当于四号字大小）
- in　inch 英寸（1in = 72.27pt）
- bp　big point 大点（在 PostScript 等其他电子排版领域的 point 都指大点，1in = 72bp ）
- cm　centimeter 厘米（2.54 cm =1in）
- mm　millimeter 毫米（10mm = 1 cm ）

表 2.6: 不同文档类选项下的字体命令。这里给出不同字号命令对应的字号尺寸。正文的默认学号是 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
\tiny	5	6	6	七	小六
\scriptsize	7	8	8	小六	六
\footnotesize	8	9	10	六	小五
\small	9	10	10.95	小五	五
\normalsize	10	10.95	12	五	小四
\large	12	12	14.4	小四	小三
\Large	14.4	14.4	17.28	小三	小二
\LARGE	17.28	17.28	20.74	小二	二
\huge	20.74	20.74	24.88	二	小一
\Huge	24.88	24.88	24.88	一	一

表 2.7: setspace 提供的命令和环境

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

- dd　didot point（欧洲大陆使用，1157 dd =1238pt）
- cc　cicero（欧洲大陆使用，pica 的对应物，1cc =12dd
- sp　scaled point（TEX 中最小的长度单位，所有长度都是它的倍数 65536 sp = 1 pt ）
- em　全身（字号对应的长度，等于一个\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：

\thinspace 或 \,	0.1667 em	⟶|
\negthinspace 或 \!	-0.1667 em	⟶‖
\enspace	0.5 em	⟶| |
\nobreakspace 或~	空格	⟶|

可以使用下面的命令表示可以换行的水平间距：

\quad	1 em	⟶| |
\qquad	2 em	⟶| |
\enskip	0.5 em	⟶| |
\␣	空格	⟶|

使用水平间距的命令要注意适用，例如\,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\,234\,567\,890
```

1 234 567 890

负距离\!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：

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

†† versus ††

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

\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

\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{longggggggggggggggggg
\hspace{2em plus 1em minus 0.25em}}
\test\test\test\test\test\test\test\test
```

longgggggggggggggggg longgggggggggggggggg
longgggggggggggggggg longgggggggggggggggg
longgggggggggggggggg longgggggggggggggggg
longgggggggggggggggg longgggggggggggggggg

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

\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

\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

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.

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

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。最基本的模式是水平模式（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，如果不在行未看不出它与其他内容的区别：

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

cannot be broken

\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][对齐方式]{内容}
```

对齐方式参数可取 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

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
```

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
```

语言文字word
word语言文字

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

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

framed
framed box

边框与内容的距离由长度变量\fboxsep控制（默认为 3pt）：

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

tight loose

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

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

test text test text

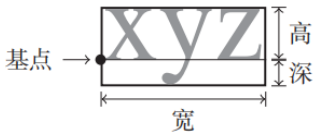
\savebox 与 \sbox 的区别类似 \makebox与 \mbox 的区别，基本功能相同，只是增加了宽度和对齐的选项。

盒子变量中一般保存比较复杂的内容。特别是可以使用 lrbox 环境保存抄录环境等难以放在其他命令参数中的内容作进一步的处理。

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

#$%^&{} #$%^&{}

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

2.2 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适定的缩进，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 命令产生一人长为 \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

```
\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）意谓右边界参差不齐），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 tεx \dots
```

English words like ‘technol- ogy’ stem from a Greek root beginning with the letters tεx …

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 tεx \dots
```

English words like ‘technology’ stem from a Greek root beginning with the letters tεx …

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

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。不过如果不想要额外的垂直距离，则不应该使用它们，如：

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

居中

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}
```

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 \dots
```

English words like ‘tech-nology’ stem from a Greek root beginning with the letters τεχ…

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。这个宏包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。

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：

```
\setlength{\leftskip}{4em}
\setlength{\rightskip}{1em}
These parameters tell \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，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：

```
\hangindent=5pc \hangafter=-2
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.
```

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

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 \hangindent 的设置只对当前段起作用。

letrine 宏包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 LATEXrefers to Donald Knuth’s TEX typesetting system. The LATEXprogram is a special version of TEX that understands LATEX commands.

shapepar 宏包提供了 \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。
绿草姜姜，白雾迷离，有位佳人，靠水而居。
我愿逆流而上，依在她身旁。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。}
```

绿 草 苍 苍，　　白 雾 茫 茫，
有位佳人，在水一　方。绿草姜姜，白雾迷离，有位佳人，靠水而居。我愿逆流而上，依在她身旁。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：

```
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：

```
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，海人不倦，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，海人不倦，何有于我哉？

后文……

verse 环境用来排版诗歌韵文：

```
\begin{verse}
在一段内使用 \verb=\\= 换行，\\
分段仍用空行。

过长的长句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。

过长的长句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。

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，如：

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

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

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

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：

```
\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 England

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次和编号，如 enumerate 环境：

```
\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}
         \item 书面语
      \end{enumerate}
   \end{enumerate}
   \item English
   \item Francais
\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

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

† 中文

- English
- FranÇais

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

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 2.8。

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：

```
\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,C,‡

表 2.8: 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{enumii}	\labelenumii	(\theenumii)
3	enumiii	\theenumiii	\roman{enumiii}	\labelenumiii	\theenumiii.
4	enumiv	\theenumiv	\Alph{enumiv}	\labelenumiv	\theenumiv.

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：

```
\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 TEX 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 55 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。
```

\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，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。

\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\thequi{\thesection-\arabic{quiz}}
```

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 1 随计数器 2 的增加而清零重编号，同时定义其编号格式，如让数学方程按节编号：

```
\usepackage{amsmath}
```

```
\numberwithin{equation}{section}
```

chngcntr 宏包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。

计数器不仅可以用编号，也能用于复杂的条性控制和循环，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和计数器的一些简单运算功能。

2.2.3.3 定制列表环境

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

表 2.9: itemize 环境的标签

嵌套层次	命令	默认值	效果
1	\labelitemi	\textbullet	•
2	\labelitemii	\normalfont\bfseries \textendash	–
3	\labelitemiii	\textasteriskcentered	*
4	\labelitemiv	\textperiodcentered	·

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

```
\newcommand*\descriptionlabel[i]{\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}
}
```

•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

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 Francais
\end{list}
```

- 中文
  又一段中文
- English
- Francais

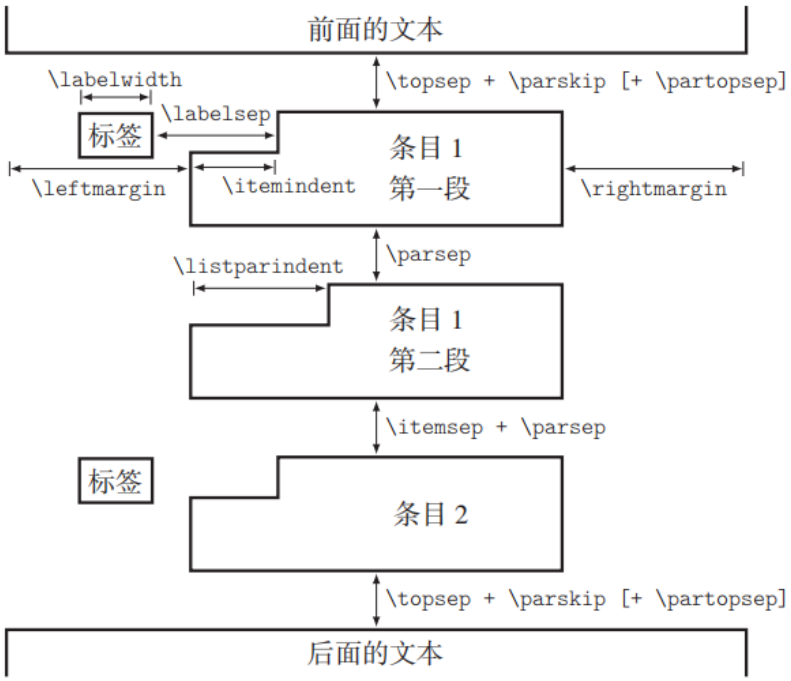


图 2.3: list 列表环境的长度参数。其中\partopsep 在列表环境开始新的一段（即前面有空行）时生效

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}}
```

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\centering命令生成的，等价定义为：

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

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。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 Francais
\end{enumerate}
```

(1) 中文
(2) English
(3) Francais

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*)}
```

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}
```

一、内容清晰
二、格式美观

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如何使用：

```
\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：

```
\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：

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

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租同的编号（定理类环境的计数器就是环境名），如：

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

命题 2.3 直角三角形的斜边大于直角边。

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\theorempresk 和 \theorempostskipamount，例如:

```
% 导言区
\usepackage{theorem}
\theoremstyle{changebreak}
\theoremheaderfont{\sffamily\bfseries}
```

```
\theorembodyfont{\normalfont}
\newtheorem{def}{定义}[chapter]
```

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

2.1 定义
有一个角是直角的三角形是直角三角形

ntheorem 宏包进一步扩充了 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：

```
% 导言区
\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}
}
```

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

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、生成定理自录等；具体的使用中也看很多注意事项（如与 amsmath 宏包同时使用时要加 `[amsmath]` 选项），读者可查阅宏包的文档获得进一步的信息。

除了 theorem 和 ntheorem 宏包，美国数学会发布的 amsthm 宏包也常用于定理类环境的定制。在使用 XeTeX 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 等文档类中首接使用 amsthm，而只能在 article 等标准文档类中使用 amsthm 宏包用再使用 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}}
```

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
\[
f(x) = 0. \qedhere
\]
\end{proof}
```

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 XeTeX 下与 fontspec 的冲突问题，因此最好使用 ntheorem 包。

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，LaTeX 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 TeX 的文章时）就不可避免地使用大量在 LaTeX 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 命令可以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：

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

\LaTeX \& \TeX \/}{#$%&~

使用带星号的命令 \verb* 则可以使输出的空格为可见的 ␣。

大段的抄录则可以使用 verbatim 环境：

```
\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：

```
\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 \fboxverb!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 宏包提供的命令实现，例如：

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

套中#$%^&

值得一提的是 cprotect 宏包，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\fbox{套中 \verb|#$%^&|}
```

套中 #$%^&

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

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，这里不作详细介绍了。

2.2.5.2 程序代码与 listings

可以使用 verbatim 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 #include <stdio.h>
3 main() {
4     printf("Hello.\n");
5 }
```

这种带语法高亮的程序代码可以使用 listings 宏包排版。

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：

```
% 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:

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

```c
/* 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。

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。以上这些者都是影响 lstlisting 环境输出效果最明显的一些参数，例如：

```
\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}
```

```c
/* 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：

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

```c
/* 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  }
```

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

listings 宏包对汉字等非 ASCII 字符的支持不是很好。使用 XeLaTeX 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[10]：

```
\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

listings 将逃逸字符之间的内容当做普通的 TEX 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) // $\frac{1}{\sin x}$

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。

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，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：

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

格式	作者
Plain TEX	高德纳
LATEX	Leslie Lamport

[10]现在好像不需要了

\kill 命令与 \\ 类似，不过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：

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

Plain TEX 高德纳
LATEX 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

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

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 $L$，$H$ \\
\qquad \= \+ \kw{integer} $L，H，M，j$ \\
\kw{while} \= \+ $L \leq H$ \kw{do} \` $L$ 与 $H$ 是左右分点 \\
   $M \gets \lfloor(L+H)/2\rfloor$ \` $M$ 是中间分点\\
   \kw{case} \= \+ \\
      condition \= foo \+ \kill
      $x > A[M]$ : \' $H \gets M-1$ \\
      $x < A[M]$ : \' $H \gets M+i$ \\
         \kw{else}:\' \= $j \gets M$ \`'找到 $x$，返回位置 \\
            \>\kw{return}$(j)$ \\
      \<\< \kw{endcase} \-\-\-\\
$j \gets 0$ \\
\kw{return} $(j)$ \- \\
\poptabs
算法示例：\\
$A = \{2，3，5，  7，  11\}$，  $x=3$ \\
\qquad \= \+ $M$ \qquad \= $L$ \qquad \= $H$ \qquad \= \\
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}
```

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 $LHMj$
while $L \le H$ **do** L 与 H 是左右分点
$M \leftarrow \lfloor (L+H)/2 \rfloor$ M 是中间分点
case
$x > A[M]$: $H \leftarrow M-1$
$x < A[M]$: $H \leftarrow M+i$
else: $j \leftarrow M$ ’找到 x，返回位置
return(j)
endcase
$j \leftarrow 0$
return (j)

算法示例：
$A = \{235711\}$，$x = 3$

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 是许多算法宏包中最为简单的一种，它可以按照著名教材《算法导论》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`，它提供丰富的命令和复杂的定制功能；algorithmicx 也提供了类以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。