ConTEXt 蹊径

李延瑞

2026.04.08

ConTEXt 蹊径

李延瑞（lyr.m2@live.cn）

项目所在：**https://github.com/liyanrui/ConTeXt-notes**

自序

我最早听闻并接触 ConTEXt，大概是 2008 年在王垠[1]的个人主页上看到他对 ConTEXt的谥美之文。当时我猎奇心甚为严重，又觉得大家都在用的 LATEX 无法体现我的气质，便开始自学 ConTEXt。或许这是年青人的通病。

现在，我已不再年青，却依然喜欢 ConTEXt。曾经沧海难为水，或许这是人老了之后的通病，而我觉得更可能是因为 ConTEXt 依然年青。2008 年，ConTEXt 正处于从 MkII 版本向 MkIV 版本跃迁期间。待 2019 年 MKIV 版本尘埃落定时，ConTEXt 的开发者大刀阔斧，如火如荼，开启了下一个版本 LMTX[1]的开发工作，至今方兴未艾。

任何工具，只要有人长时间维持和改进，便会趋向于复杂而难以被他人驾驭，但是它能胜任复杂的任务。TEX 是复杂的，以它为基础的 Plain TEX，LATEX 和 ConTEXt 则更为复杂。事实上，并非工具趋向于变得复杂，而是任务趋向于变得复杂，更本质一些，是人心趋向于变得复杂。例如，使用 Markdown 之类的标记语言写散文类的文章，轻松愉快，这是近年来 Markdown 广泛用于网文创作的主要原因。倘若用 Markdown 写一本含有许多插图、数学公式、表格、参考文献等内容的书籍，便需要为它增加许多功能，最终的结果相当于又重新发明了一次 TEX。

Plain TEX，LATEX 和 ConTEXt 虽然皆为构建在 TEX 系统上的宏包，但是国内熟悉前两者的人数远多于 ConTEXt，究其原因，我觉得是 ConTEXt 入门文档甚少且中文文档更为罕见。ConTEXt 创始人兼主要开发者 Hans Hagen 为 ConTEXt 撰写了一份内容全面、排版精美的英文版入门文档[2]。有英文阅读能力的人，原本可从该文档入门，但遗憾的是，除非读者知道如何在 ConTEXt 中使用汉字（也包括日、韩文字）字体，否则所学知识仅能用于英文排版。

我曾于 2009 年写过一份 ConTEXt学习笔记，介绍了在 ConTEXt MkIV 中如何加载汉字字体以及基本的 ConTEXt 排版命令的用法，内容颇为粗陋，由 CTeX 论坛里的朋友整合至 ctex-doc 项目并打包呈交 https://ctan.org 网站。2011 年夏天曾许诺将我发布于网络上的一些相关文章合并至该笔记，后因兴趣漂移，不了了之。

光阴荏苒，时过境迁，当年曾在 CTeX 论坛一起折腾 TEX 的朋友大多已不知所踪——在他们看来，我亦如是。2018 年，CTeX 论坛因国内日益严厉的互联网监管政策被迫无限期关闭，导致国内 TEX 学习、研究和使用热情似乎遭受了毁灭性打击，爱好者们离散江湖，白头宫女在，闲坐说玄宗。现在，若学习 LATEX 尚能找到一些讨论区，而 ConTEXt 似乎再也无人问津了。爱好终归属于自己。即使现在国内只有我一个人还在喜欢 ConTEXt，依然应当为自己写一份新的 ConTEXt 学习笔记，以偿旧诺。

值此情境，需篡改古词一阙，歌以咏志：芦叶满汀洲，寒沙带浅流，二十年重过南楼。欲买桂花同载酒，终不负，少年游。

2023 年 3 月写于山东郯城乡下老家

[1] 一个敢于自我否定的人，曾致力于在国内推广 GNU/Linux 和 TEX，后来以其对计算机科学和编程语言设计的洞察而闻名，详见其个人主页 https://www.yinwang.org 。

自序之二

两年过去了，这份文档该更新了。漫长的时间里，偶尔还能继续以前的工作，会显得人颇为执着。实际上并非如此，这两年里，我几乎已经将这份文档忘记了。

那一年，我已忘记是哪一年了，奶海寄给我一本《LaTeX 入门》，他写的，很厚，也很全面，在我看来，是中文 LaTeX 书里迄今为止最好的一本。我当时正在百度贴吧里玩刀，心里想的都是蝴蝶 581，蜘蛛 C81，Esee 3……已将 TeX 忘记许多年了，甚至连 CTeX 论坛何时倒闭的都不知情，而我还一直忝居某个版块的版主。如果我不讲这个故事，那么在熟悉我的人看来，我十七年如一日喜欢着 ConTeXt。

这一次也是如此，我可以用十多天的时间，修缮这份文档并补充一些我感兴趣的主题，然后我会继续忘记它。

我发现，为某个事物写文档，最大的好处就是写好之后可以放心忘记它。当我收到奶海寄来的书时，我是忘记了 ConTeXt 的很多知识，但这种忘记是此心难安的。放心式的忘记，第一次感受到它，是在上一次完成这份文档之后。尽管在此之前，我在自己的几个博客上写过一些笔记，但写博客终归过于随意且虚荣，并不能支撑这种放心式的忘记。

当我真正理解了自己的这个想法后，我可以正面回答 TeX 爱好者们经常要面对的一个问题，即 TeX 比 MS Word 这种大众化意义上的排版软件，比 Markdown 这样的轻量级标记语言……好在何处呢？答案是，在不预设立场的前提下，分别用三者为同一事物编写文档，之后忘记这份文档的放心程度是 TeX > MS Word > Markdown。

做一件事情，有许多工具可选，而你选择的工具，用起来最难但是能得到好的结果，这意味着你对这件事极为重视，无形之中拓展了对它的思考广度和深度。这就是 TeX 的长处，它能帮助你创造这样的机会。

Donald Knuth 是为了排版他所写的《计算机编程艺术》这套书而创造了 TeX，他从中得到的放心忘记的机会应该远大于我辈 TeX 用户。为什么要追求放心式的忘记呢？为了尽量减少杂念。之后的几年里，如果我又重新玩刀或者别的什么，忽然有一天，我收到了奶海寄来的《LaTeX 入门》第 2 版，我心里不会再收到由愧意和嫉妒交织成的复杂情感，并在它的驱动下后责问自己，你还会多少 ConTeXt 呢？

如果你觉得 TeX 太难——无论是 LaTeX 还是 ConTeXt，学习成本过高，故而觉得用它编写文档，得不偿失。我的建议是，你真正应该考虑的是，你所写的文档，你是否需要放心忘记它。事实上，大多数文档连忘记都是不配提的，更勿言放心。如果将 TeX 生成的 PDF 打印到纸上，你不仅可以放心忘记它，甚至无需担心整个计算机世界的覆灭。

2025 年 8 月写于淄博蜗居

目录

自序 i.

自序之二 ii.

第 1 章　命令行

1.1 Windows 命令行 1. 1.2 Linux 和 macOS 终端 3. 1.3 安装 ConTEXt 4. 1.4 接力式安装 6. 1.5 TEX Live 中的 ConTEXt 6.

第 2 章　新手村

2.1 卡片 7. 2.2 伪文 8. 2.3 换行 8. 2.4 分段 8. 2.5 行距 10. 2.6 对齐 10.

第 3 章　汉字

3.1 安装字体 12. 3.2 使用字体 14. 3.3 中文断行 14. 3.4 写一封真正的信 16. 3.5 字族 16. 3.6 汉字字族 18. 3.7 字形替换 19.

第 4 章　散文

4.1 标题 21. 4.2 写一篇散文 21. 4.3 正式踏入 ConTEXt 世界 24. 4.4 页码 24. 4.5 内容与样式分离 25.

第 5 章　列表

5.1 待办事项 27. 5.2 无序号列表 27. 5.3 有序号列表 27. 5.4 留白 28. 5.5 引用 28. 5.6 画符 29. 5.7 带圈的数字 31.

第 6 章　表格

6.1 基本格式 34. 6.2 边线 34. 6.3 对齐的假象 35. 6.4 三线表 35. 6.5 段落成列 36. 6.6 像插图那样 37. 6.7 横列表 39. 6.8 样式设定 40.

第 7 章　插图

7.1 外图 42. 7.2 标题 43. 7.3 阵列 45. 7.4 路径 47. 7.5 内图 47.

第 8 章　数学

8.1 TEX 风格 48. 8.2 ConTEXt 风格 48. 8.3 序号 49. 8.4 定理和证明 50.

第 9 章　一字不差

9.1 抄录 52. 9.2 着色 53. 9.3 逃逸区 53. 9.4 显示空格 54. 9.5 样式设定 55.

第 10 章　盒子

10.1 TEX 盒子 56. 10.2 ConTEXt 盒子 57. 10.3 对齐 58. 10.4 背景 58. 10.5 盒子的深度 59. 10.6 段落盒子 60. 10.7 自定义盒子 61.

第 11 章　作图

11.1 作图环境 62. 11.2 画一个盒子 62. 11.3 颜色 65. 11.4 文字 65. 11.5 方向路径 66. 11.6 画面 67. 11.7 宏 68. 11.8 简单的流程图 69. 11.9 代码简化 72. 11.10 层叠 74.

第 12 章　幻灯片

12.1 纸面尺寸 77. 12.2 页面布局 78. 12.3 字体 80. 12.4 常规样式 81. 12.5 页脚 83. 12.6 封面 84.

第 13 章　写书

13.1 结构 86. 13.2 目录 87. 13.3 引用 89. 13.4 脚注 90. 13.5 索引 91. 13.6 书签 91. 13.7 双面 92. 13.8 页眉 93. 13.9 页脚 95.

第 14 章　文献

14.1 数据格式 96. 14.2 自制样式 97. 14.3 样式微调 100. 14.4 更具一般性 101. 14.5 第三条路 101.

第 15 章　zhfonts

15.1 初印象 102. 15.2 安装 103. 15.3 用法 104. 15.4 设定 104. 15.5 数学字体 105.

第 16 章　杂技

16.1 标题 107. 16.2 特定页面 108. 16.3 奇怪的间距 109. 16.4 编组 110. 16.5 样式切换 111. 16.6 非常规序号 112. 16.7 三段论 113. 16.8 Lua 宏 114.

第 17 章　编程

17.1 宏 117. 17.2 变量 118. 17.3 条件 119. 17.4 函数 119. 17.5 Lua 120.

跋 122.

参考文献 123.

索引 125.

1 命令行

无论是安装还是使用 ConTEXt，皆需要对命令行环境有所了解。本章先分别介绍 Windows、Linux 和 macOS 系统的命令行环境的基本用法，以刚好满足安装和运行 ConTEXt 环境需求为要。

若你对命令行环境已颇为熟悉，只是想了解如何安装 ConTEXt，可直接从 **1.3** 节开始阅读，否则要有耐心从一个简单任务开始熟悉命令行的基本用法。这个任务是，在命令行环境里，创建 foo 目录，并在该目录内创建一份 Shell 脚本，令其可在命令行窗口中输出「不怕命令行」。

1.1 Windows 命令行

Windows 用户似乎天生畏惧甚至厌憎命令行环境，甚至很多人认为它是早已被淘汰的上个世纪的东西。这种认识是错误的，原因很简单，最新的 Windows 系统依然不仅依然提供它，甚至处心积虑强化它。

在 Windows 系统中打开命令行窗口，有很多种方法，其中最快的应当是使用如图 **1.1** 所示的组合键「Win + R」，打开「运行」对话框，在其中输入「cmd」，然后点击对话框上的「确定」按钮或单击回车键（Enter 键），便可打开与图 **1.2** 类似的命令行窗口，只不过你看到的应该是背景为黑色的窗口——我为了打印这一页时能省一些墨，将其改成了白色。

图 1.1　Win + R 组合键

图 1.2　Windows 命令行窗口

在命令行窗口里，所有要执行的命令皆在「>」符号后输入，故而该符号叫作**命令提示符**。需要注意，在输入命令之前，请查看输入法状态，将其切换为英文输入状态。

任何一条命令都是在某个目录下执行的，该目录称为**工作目录**。命令提示符左侧内容表示文件目录，它就是工作目录。在命令行窗口中输入命令「d:」，然后单击回车键便可执行该命令，结果是工作目录被切换为 d 盘，倘若它存在。在命令行环境里，输入命令后，必须单击回车键，命令方能得以执行。

假设当前的命令行工作目录为 d 盘，请尝试执行命令「md foo」，该命令可在 d 盘创建目录 foo。然后执行命令「cd foo」，将工作目录切换为 foo。现在工作目录就是 d:\foo 或 D:\foo——Windows 的命令行语句里，无论是命令还是目录或文件名，不区分大小写。

像 d:\foo 这样的目录形式称为**绝对路径**，反斜线 \ 称为路径分隔符。有**相对路径**吗？有。在 d:\foo 中执行命令「cd ..」，便可将工作目录切换到上一级，即 d 盘。符号「..」便是相对路径，表示工作目录的上级目录。

在 d:\foo 中，若执行命令「cd ..\foo」，则工作目录不会发生变化，因为 d:\foo 的上级目录的子目录 foo 依然是 d:\foo。像「..\foo」这样的路径也是相对路径。

在 d 盘执行命令「cd ..」，工作目录会发变化吗？不会。因为 Windows 任何一个盘，其上一级目录为空。

若能理解上述内容，凭借几个简单的命令，便可遨游 Windows 文件系统了，只是我们的任务尚未开始。该任务是在 d:\foo 中创建一份 Shell 脚本，执行该脚本，在命令行窗口中输出「不怕命令行」。在 Windows 系统中，Shell 脚本即命令行批处理文件，其扩展名通常为.bat。在实现该脚本之前，需要了解 echo 命令的用法。

echo 命令亦称回显命令，其功能是读取一些文字，然后将其原样输出。例如，

```
> echo 不怕命令行
不怕命令行
```

上述代码的第一行是执行「echo」命令，第二行是命令的输出结果。

似乎 echo 是一个什么都不会做的命令，这样的命令有什么用呢？它的一个用途是，通过命令行的输出重定向机制，将一些内容写入制定的文件。例如，

```
d:\foo> echo @echo off > foo.bat
```

上述命令通过输出重定向符「>」将原本会输出到命令行窗口的文字「@echo off」输出到文件 d:\foo\foo.bat c 。倘若该文件事先并不存在，系统会自动创建它。

现在，用 echo 命令向文件 d:\foo\foo.bat 的尾部追加一行文字：

```
> echo echo 不怕命令行 >> foo.bat
```

「>>」也是输出重定向符号，它能够向一份已存在的文件追加内容。在上述代码中，若使用「>」，则文件原有内容会被新的内容完全替换。

现在，略有纪念意义的时刻到了，这可能是你此生首个甚至也可能是最后一个批处理文件，执行它吧！

```
> .\foo.bat
不怕命令行
```

上述命令里的「.」表示工作目录本身，故而「.\foo.bat」也是一种相对路径，表示工作目录里的 foo.bat 文件。不过，在 Windows 命令行环境里，这种形式的相对路径可以忽略，亦即上述命令可写为

```
> foo.bat
不怕命令行
```

在执行某个程序或批处理文件时，倘若未给出其路径，Windows 系统默认先从工作目录中搜索文件，若未搜到，才会在系统环境变量 PATH 设定的路径中搜索。

系统环境变量 PATH 是什么呢？既然是变量，必定有值，其值是绝对路径集，以下命令可以查看：

```
> echo %PATH%
```

顺便指出，这是 echo 命令的另一种用途，即查看环境变量的值。

使用 setx 命令可以将上述示例创建的批处理文件 foo.bat 所在目录 d:\foo 追加至 PATH 变量现有路径集的尾部，如下：

```
> setx /M PATH "%PATH;%d:\foo"
```

务必注意，该命令仅在「以管理员身份」启动的命令行窗口中生效。在 Windows 开始菜单里的搜索栏，输入「cmd」并单击回车键键提交，然后鼠标右键单击搜索结果，在弹出的菜单中选择「以管理员身份运行」。

验证 d:\foo 是否被成功添加到系统 PATH 变量，只需在除 d:\foo 之外的任一目录验证能否执行 foo.bat，例如

```
> c:
> cd windows\system32
> foo.bat
不怕命令行
```

若不知如何以管理员身份运行命令行窗口，亦可通过图形界面设置 PATH 变量。我已将上述构建 d:\foo\foo.bat 以及如何通过图形界面设置系统 PATH 变量等过程录制为视频，网络链接为 https://www.bilibili.com/video/BV1vk4y1h7LE/，从而避免层峦叠障的 Windows 窗口截图占据本章太多篇幅。

1.2 Linux 和 macOS 终端

在 Linux 系统中，命令行窗口通常称作终端（Terminal），终端里运行的是某种 Shell 程序，它负责执行用户输入的各种命令。 Linux 用户大都知道如何开启终端，甚至我可以放心假设他们已经很熟悉下文所讲的一切了。下面，我以 Bash Shell 为例，写一个脚本，在终端里输出「不怕命令行」。

首先，在终端里进入 $HOME 目录，亦即 ~ 目录，在其中创建子目录 foo：

```
$ cd ~
$ mkdir foo
```

注意，Linux 的命令提示符通常是 $。

进入 foo 目录，用命令输出重定向，将 echo 命令的输出结果写入 foo.sh 脚本：

```
$ cd foo
$ echo #!/bin/bash > foo.sh
$ echo echo 不怕命令行 >> foo.sh
```

使用 chmod 命令为 foo.sh 增加可执行权限，使之能够像程序一样运行：

```
$ chmod +x foo.sh
```

以下命令将 ~/foo 添加至系统环境变量 PATH 并使之在当前终端里生效：

```
$ cd ~
$ echo 'export PATH=~/foo:$PATH' >> .bashrc
$ source .bashrc
```

在任一目录下执行 foo.sh 以验证 ~/foo 是否已被添加至 PATH 变量，例如

```
$ cd /tmp
$ foo.sh
不怕命令行
```

由于 macOS 和 Linux 皆为 Unix-like（类 Unix）系统，故而二者终端环境的用法近乎相同。如果你知道如何开启 macOS 的终端窗口，则上文所述的 Linux 命令的用法也适于 macOS。不过，macOS 从 Catalina 版开始，默认的 Shell 不再是 Bash，而是 zsh，故而在设置 PATH 变量时，命令需变更为

```
$ cd ~
$ echo 'export PATH=~/foo:$PATH' >> .zshrc
$ source .zshrc
```

类 Unix 系统里的 Shell 虽然有很多种，但它们的用法颇为相似，而且以 Bash 最为常用。若想对 Bash Shell 有更多的了解，可以阅读我写的一些文章[3 - 5]。

1.3 安装 ConTEXt

一旦掌握了你所用的操作系统中的命令行基本用法，安装 ConTEXt 就简单多了，下文将分别介绍在 Windows、Linux 以及 macOS 系统入如何安装它。

ConTEXt 的最新版本是 ConTEXt LMTX，面向 Windows 系统的安装包可从 https://wiki.contextgarden.net/Introduction/Installation 获取，例如 Intel 或 AMD 的 64 位 CPU 架构的 Windows 机器，选择下载 X86 64bits 版本即可。下文以该版本的安装包为例，讲述 ConTEXt LMTX 的安装过程。

首先，从 https://lmtx.pragma-ade.com/install-lmtx/context-win64.zip 下载面向 Windows 64 位系统的安装包 context-win64.zip，假设在 d:\ 将其解开，得目录 context-win64，其结构当如图 1.3 所示。完成解包工作，context-win64.zip 包就没用了，可以删除。

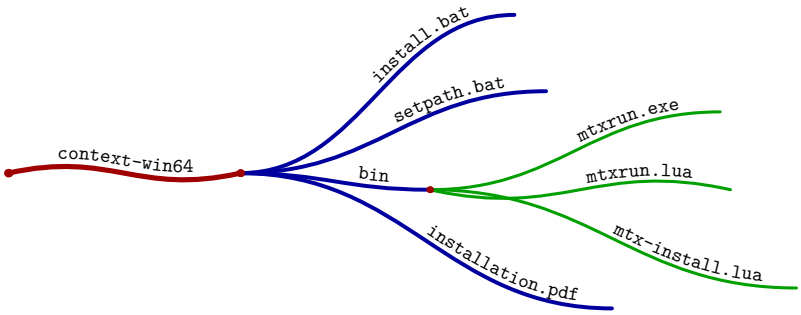


图 1.3 ConTEXt LMTX 的 windows 64 位安装包结构

将 context-win64 目录改名为 context，然后在命令行窗口依序执行以下命令：

```
> d:
> cd context
> install.bat
> setpath.bat
> mtxrun --generate
```

上述命令里，批处理文件 install.bat 可从网络上下载 ConTEXt LMTX 的所有文件，存放在工作目录。安装时长取决于网络下载速度。ConTEXt LMTX 的服务器在境外，由于国内众所周知的原因，导致文件下载速度可能会非常缓慢，而且安装过程甚至可能中断，需要你多次执行 install.bat 文件，方能完成整个安装过程。幸好 install.bat 是增量安装，即使有所中断，每次它会从中断处开始安装，而非从头重新安装。后文会给出符合国情的快捷方案。

从现在开始，称 d:\context 目录为 **ConTEXt 安装目录**。若想验证 ConTEXt LMTX 是否安装完全，可在 ConTEXt 安装目录执行以下命令：

```
> setpath.bat
> md test
> cd test
> echo \startTEXpage[frame=on,offset=1pt] > foo.tex
> echo Hello \CONTEXT! >> foo.tex
> echo \stopTEXpage >> foo.tex
> context foo.tex
```

倘若在 d:\context\test 目录下能够得到 foo.pdf 文件，且其内容为 Hello ConTEXt，则意味着已成功安装 ConTEXt LMTX。

最后一步，将 d:\context\tex\texmf-win64\bin 添加到系统 PATH 变量，便可在任一目录使用 context 命令将扩展名为.tex 的文本文件编译为同名的 PDF 文件。

上述命令创建的 foo.tex 文件，你可以用任何一款文本编辑器打开，作一些编辑，例如用 Windows 的记事本程序打开它：

```
> notepad foo.tex
```

如果你没有 PDF 阅读器，或者每次重新编译.tex 文件时，你的 PDF 阅读器没有自动更新显示 PDF 文件的内容，我建议你用 Sumatra PDF 阅读器，其下载页面位于 https://www.sumatrapdfreader.org/free-pdf-reader。

假设你的 Linux 是 Inter 或 AMD 架构的 64 位系统，并且你希望将 ConTEXt LMTX 安装在 ~/opt 目录，亦即 ~/opt，则整个安装过程可表述为以下 Bash 命令：

```
$ mkdir -p ~/opt/context
$ cd ~/opt/context
$ wget https://lmtx.pragma-ade.com/install-lmtx/context-linux-64.zip
$ unzip context-linux-64.zip
$ sh install.sh
$ echo 'export PATH=~/opt/context/tex/texmf-linux-64/bin:$PATH' >> ~/.bashrc
$ mtxrun --generate
$ rm context-linux-64.zip
```

在 macOS 系统中安装 ConTEXt LMTX 所用命令与上述 Linux 系统安装命令近乎相同，只是 macOS 环境里默认可能没有 wget 命令，不过你可以用 curl 命令替代它。以下命令可获得面向 macOS 64 位系统的 ConTEXt LMTX 安装包：

```
$ curl -O https://lmtx.pragma-ade.com/install-lmtx/context-osx-64.zip
```

在安装过程结束后，你可能需要消除一些文件的隔离标记。假设你的 ConTEXt 安装目录为 ~/opt/context，可执行以下命令完成该工作。

```
$ sudo xattr -r -d com.apple.quarantine ~/opt/context/bin/mtxrun
$ sudo xattr -r -d com.apple.quarantine ~/opt/context/tex/texmf-osx-64/bin/*
```

1.4 接力式安装

上述安装方法源于 https://wiki.contextgarden.net/Introduction/Installation，其中面向 Windows 和 Linux 的安装过程，我皆已实践，除文件下载速度太慢以及安装过程偶有中断之外，没有其他问题。面向 macOS 系统的安装过程，我没有条件予以验证，只能纸上谈兵。

为了加快安装进程，我将 Windows 和 Linux 里装好的 ConTEXt LMTX 分别打包，以时间为后缀，上传到了网盘。你可以下载它们，只需将其解包到指定目录，便可拥有一个版本略微落后的 ConTEXt LMTX 环境。之后只需执行安装目录里的 install 脚本便可追及最新版本——相当于意外中断后继续安装。由于这部分内容经常变化，而本文档更新周期通常以年为单位，故而我将其做成一个网页：

https://zhuanlan.zhihu.com/p/1943204598711054617 。

只需按照该网页的说明，便可创建一个纯净的 ConTEXt LMTX 环境。

1.5 TEX Live 中的 ConTEXt

TEX Live 是迄今为止最为权威的 TEX 系统，LATEX 用户大多使用该系统，不过该系统也收录了 ConTEXt 包。相比于 ConTEXt，TEX Live 是更为宏大的 TEX 体系，前者可谓是后者的一颗行星。由于 TEX Live 提供了图形界面的安装程序，如果你始终都不得命令行其门而入，可以考虑用该程序安装 ConTEXt LMTX。

鉴于本章的主题是在命令行环境里安装 ConTEXt，倘若在此讲述 TEX Live 系统的图形化安装过程，结果必定会喧宾夺主，仅是安装过程的一些截图所占的篇幅就已经超过了前文所有内容。不过，为了让你能多一条路，该安装方式在上一节的 ConTEXt 接力棒的在线文档中里也有记述，并且以视频的形式演示了安装过程。

结语

也许你已经学会了如何在命令行环境里安装 ConTEXt，但我更希望你能熟悉甚至习惯以命令行的方式工作。著名的开源运动先驱者 Eric Raymond 在其《Unix 编程艺术》一书的附录部分写了一些禅宗式的寓言，其中有一则与图形界面程序有关。在这则寓言里，一个程序员对 Unix 领域的一个智者说，现代的，设计得当的操作系统可以在图形用户界面里做任何事情。智者没说话，只是用手指了一下月亮，旁边有条狗冲着智者的手狂吠。程序员不解其意。智者陆续又指了佛像、窗户、程序员的脑袋和路边的石头，程序员依然不解。智者只好轻拍程序员鼻子两下，将其扔到旁边的垃圾桶中。当程序员试图从垃圾桶中爬出来时，狗跑过来，在他身上撒尿。此刻程序员终于悟出，图形界面的问题在于难以用语言与他人交流软件的用法。

2 新手村

不要急于探索如何使用 ConTEXt 排版你的论文或专著，尽管它极为擅长这类任务，但是我还是希望你在新手村里待两天，学几个简单的招式，打一些立竿见影的怪物，培养一些信心。在这个游戏里，你需要的装备，只是一个文本编辑器和一个 PDF 阅读器，前者用于编写 ConTEXt 源文件，后者用于查看 context 命令的编译结果。

你的 PDF 阅读器最好能自动响应 PDF 文件的变化，亦即 PDF 文件发生变化时，PDF 阅读器能自动承诺更新打开它并将页面定位到上一次的观看位置。Windows 里的 Sumatra PDF、Linux 里的 Evince、macOS 里的 Skim 等 PDF 阅读器皆有此功能。

2.1 卡片

在第 1 章中，为了验证 ConTEXt LMTX 是否已成功安装，我使用三条 echo 命令构造了一份简单的 ConTEXt 源文件 foo.tex。事实上，你需要一个文本编辑器，哪怕是功能最为简单的文本编辑器，也要比 echo 命令更适合做此事。

用你的文本编辑器重写一次 foo.tex，其内容如下：

```
\startTEXpage[frame=on]
 Hello \CONTEXT!
\stopTEXpage
```

然后用 context 命令，将 foo.tex 编译成 PDF 文件 foo.pdf，亦即

```
$ context foo.tex
```

或

```
$ context foo
```

从本章开始，在表达命令行语句时，将一直使用 Linux 风格的命令提示符 $。此外，将 context 命令称为 **ConTEXt 编译器**，有时我也会将后者简称为 ConTEXt。

在 ConTEXt 源文件里，前缀为反斜线 \ 的西文单词，皆为排版命令。形如 \startxxx...\stopxxx 这样的排版命令称为 xxx 环境。TEXpage 环境就是我们的新手村，用于演练今后所学的一些基本的 ConTEXt 排版命令，该环境的内容会被 ConTEXt 编译器安排在尺寸紧致的矩形排版空间里，像一张卡片。

TEXpage 环境的 frame 参数用于控制边框是否开启，若该参数不存在或其值为 off，表示无边框。offset 参数可用于扩大或缩小对排版空间，例 2.1 将排版空间从中心向四周扩大了 2.5 mm；若尺寸为负，则产生缩小效果。

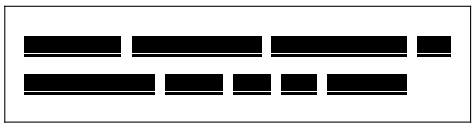
```
\startTEXpage[frame=on,offset=2.5mm]
 Hello \CONTEXT!
\stopTEXpage
```

例 2.1　新手村

2.2 伪文

ConTEXt 自带一个名为 visual 的模块，该模块提供了命令 \fakewords，可生成一些黑色的长短随机的矩形块。若将这些矩形块视为文字，在新手村里，我们举止会更为随心所欲一些，例如下例排版了两行文字，每一行由 3～5 个单词构成。

```
\usemodule[visual] % 载入 visual 模块
\startTEXpage[frame=on,offset=2.5mm]
\fakewords{3}{5}\\
\fakewords{3}{5}
\stopTEXpage
```



例 2.2　两行伪文字

在例 2.2 的源码中，% 及其后面的同一行文字，是 TEX 注释文本，在编译过程中，它们会被忽略，不会出现在排版结果中。除了用于注释源码，注释符也能用于消除其后的换行符，以后你自定义 TEX 命令时会用到这个功能。

2.3 换行

在例 2.2 的源码中，\\ 是强制换行命令，若将其删除，即使将文字分为两行，

```
This is the first line.
This is the second line.
```

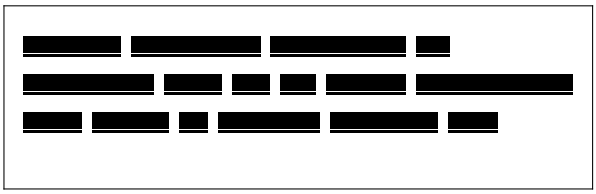
排版所得结果依然是一行，而且换行符会被 ConTEXt 编译器视为一个空格，你可以亲自动手试验一下。在使用强制换行命令时，即使两行文字在源代码中处于同一行，例如

```
This is the first line.\\ This is the second line.
```

排版所得结果依然是两行。\crlf 也能用于文字强制换行。

lines 环境可以排版多行文本，无需 \\ 或 \crlf，见下例。

```
\startlines
\fakewords{3}{5}
\fakewords{4}{7}
\fakewords{5}{9}
\stoplines
```

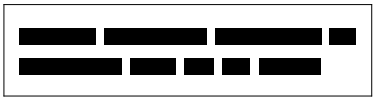


例 2.3　排版多行文本

2.4 分段

观察例 2.4，虽然排版结果依然是两行，但实际上是两段。\par 是分段命令。

```
\fakewords{3}{5}\par
\fakewords{3}{5}
```



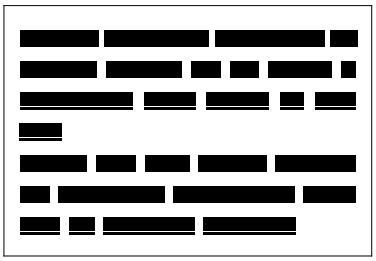
例 2.4　分段

上例也能写成以下形式：

```
\fakewords{3}{5}\par\fakewords{3}{5}
```

通常很少使用分段符对文本进行分段，因为在 ConTEXt 源文档中，只需在两段文字之间空一行便可实现分段。例 **2.5** 使用空行进行分段，并将页面宽度设定为 6cm，从而在促狭的空间里展示了多行伪文字构成的段落。

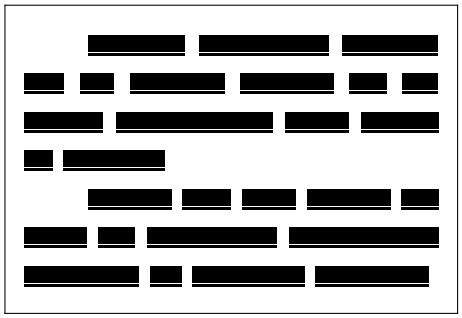
```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\fakewords{9}{15}

\fakewords{9}{15}
\stopTEXpage
```



例 2.5　多行文本构成的段落

段落可以设置首行缩进。中文排版的惯例是，段落首行需缩进 2 个汉字的宽度，例 **2.6** 将段落首行缩进距离设定为 2em，即英文字母 M 的宽度的 2 倍，刚好与两个汉字的宽度相同。ConTEXt 还有一个常用的尺寸单位 ex，它是英文字母 x 的高度。

```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\setupindenting[first,always,2em]
% 将缩进区域的颜色设为白色
\definecolor[fakeparindentcolor][white]
\fakewords{9}{15}\par
\fakewords{9}{15}
\stopTEXpage
```

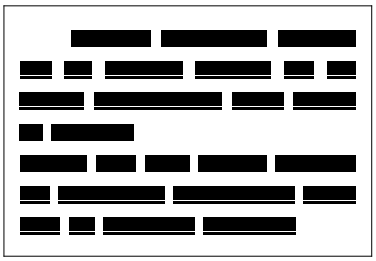


例 2.6　段落首行缩进

visual 模块默认是将段落缩进区域设定为蓝色，为了让缩进区域更为直观而非明显，上例参考了文档 [6]，将其改为白色。

在设定段落首行缩进后，若不希望某个段落的首行被缩进，可在段落开头放置命令 \noindent，参考例 **2.7**。

```
\fakewords{9}{15}

\noindent\fakewords{9}{15}
```

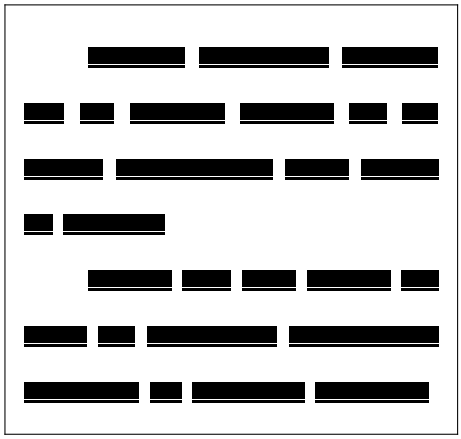


例 2.7　消除第二段的首行缩进

2.5 行距

ConTEXt 默认的段落内各行文字的间距是 2.8ex，约等于 ConTEXt 默认的正文字体大小 12pt。可使用 \setupinterlinespace 命令对行间距进行调整。使用该命令，需要确定当前正文字体所用字号。例 2.8 按 ConTEXt 默认的正文字体字号即 12pt，将行距设为该字号的 1.75 倍。

```
% 1.75 * 12pt = 21pt
\setupinterlinespace[line=21pt]
\fakewords{9}{15}\par
\fakewords{9}{15}
```



例 2.8　多行文本构成的段落

例 2.8 设定的实际上是行高。在 ConTEXt 中，行高是相邻两行文字的基线距离，可将其视为行间距。可能你并不清楚基线的概念。还记得小时候学英文时用的四线三格本子吗？那 4 条线从下往上数的第 2 条线便是基线。

ConTEXt 提供了 \bodyfontsize 命令，通过它能获得当前正文字体的字号，于是例 2.8 可改写为

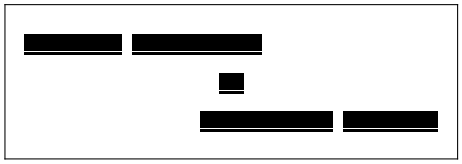
```
\setupinterlinespace[line=1.75\bodyfontsize]
\fakewords{9}{15}\par
\fakewords{9}{15}
```

ConTEXt 是 TEX 的上层建筑，上例中用一个数字直接乘以一个能获得某种尺寸的命令，这是 TEX 的语法。今后，我们会时常用这种形式构造一些尺寸。如果能够以字号确定行高，就无需单独为行高设定一个尺寸了。我认为排版作为一门艺术，其根本在于，是为不同尺度建立联系，用尽量少的尺度控制尽量的尺度，我甚至觉得，任何一门艺术皆应如此。

2.6 对齐

将一行文字居左、居中或居右放置，可分别用 \leftaligned，\midaligned 和\rightaligned 予以实现，请参考例 2.9。

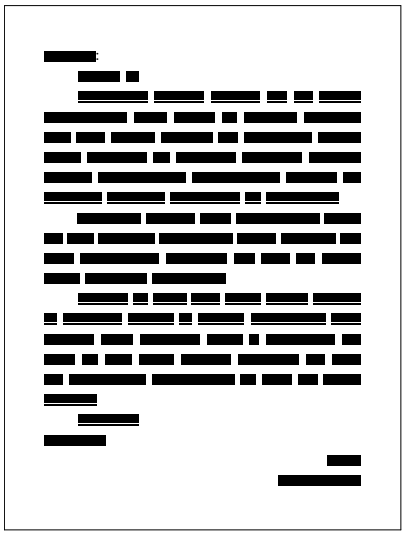
```
\leftaligned{\fakewords{1}{2}}
\midaligned{\fakewords{1}{2}}
\rightaligned{\fakewords{1}{2}}
```



例 2.9　多行文本构成的段落

现在，你已经有能力用伪文字写一封谁也看不懂内容的书信了，见例 **2.10**。注意，该例使用了 \dorecurse 命令，该命令可将其第 2 个参数复制 n 次，n 值由该命令的第 1 个参数设定。

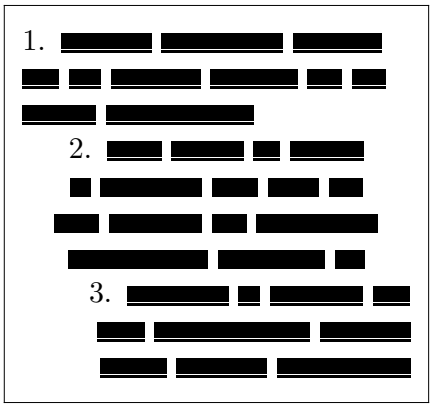
```
\usemodule[visual]
\startTEXpage[frame=on,offset=1cm,width=10cm]
\definecolor[fakeparindentcolor][white]
\setupindenting[first,always,2em]
\noindent\fakewords{1}{1}:\par
\fakewords{1}{2}\par
\dorecurse{3}{\fakewords{20}{50}\par}
\fakewords{1}{1}\par
\noindent\fakewords{1}{1}\par
\rightaligned{\fakewords{1}{1}}\par
\rightaligned{\fakewords{1}{1}}
\stopTEXpage
```



例 2.10　一封谁也看不懂的信

若一段文字需要居左、居中或居右排版，可使用 alignment 环境，通过该环境的参数控制对齐形式。例 **2.11** 展示了段落的三种对齐形式。

```
\startalignment[flushleft] % 左对齐
1. \fakewords{5}{15}
\stopalignment
\startalignment[middle] % 居中对齐
2. \fakewords{5}{15}
\stopalignment
\startalignment[flushright] % 右对齐
3. \fakewords{5}{15}
\stopalignment
```



例 2.11　段落对齐

结语

排版是一门艺术，ConTEXt 排版自然也是如此。接触艺术最好的办法是，附庸风雅，多观察一些例子，掌握基本排版命令的用法，筑好根基。理解了这一点，你就可以走出新手村了。之后，你的第一个重要任务是，在 ConTEXt 茫茫世界里寻找汉字。

3 汉字

让 TEX 系统支持汉字，这个任务曾经极易对新手学习 TEX 的热情致以毁灭性打击，后来随着 XETEX 和 LuaTEX 等现代 TEX 引擎的出现，情况终于有所好转。

ConTEXt LMTX 所用的 TEX 引擎是 luametaTEX，该引擎围绕 ConTEXt 的需求精简了 LuaTEX 并作为其继任者而继续发展。可能你现在并不理解这些术语，但也不必担心。以汽车为喻，大多数人未必知道，也无须知道他们所开的车，引擎（发动机）的型号，性能参数又是如何，只是车的制造者和维修者需要具备这些知识。字体之于 TEX，犹如燃油之于汽车，你需要知道如何为你的车注入燃油。

现在，你应该大致清楚了，ConTEXt 本质上是 TEX 引擎的上层建筑。TEX 是体，是根本，而 ConTEXt 是面，是表象。比 ConTEXt 更广为人知的 LATEX 也是 TEX 的一种表象。如果你精通 TEX，也可以自创表象。

3.1 安装字体

ConTEXt 默认不支持汉字，原因是它的体系里缺乏汉字字体。你需要找到一款汉字字体，将其安装到 ConTEXt 编译器能搜索到的目录，否则只能学仓颉，自己造字了，这是一项庞大的工程。

若你或朋友的计算机中装有 Windows 系统，可从图 **3.1** 所示的 `c:\Windows\Fonts` 目录下获得宋体（simsun.ttc）、黑体（simhei.ttf）和楷体（simkai.ttf）等字体文件，它们能胜任常规的排版工作。

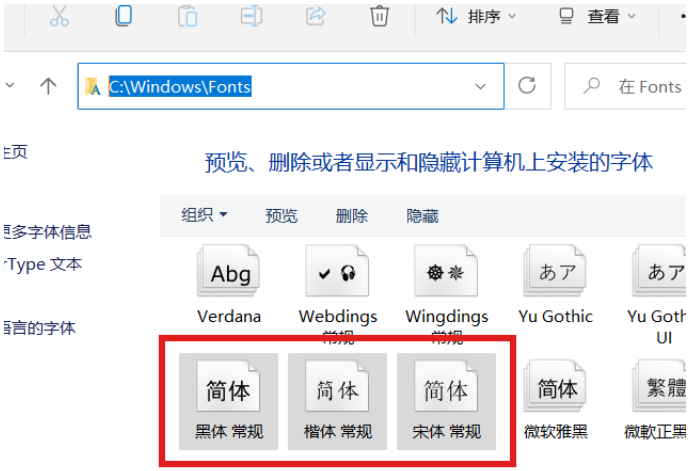


图 3.1 `c:\Windows\Fonts`

需要注意的是，我只是以这几个中文字体为例，讲解如何为 ConTEXt 安装中文字体，并不意味着你只能用这些字体。在 Linux 系统里，用这些字体，是侵权行为。

在开始安装字体之前，需要定义一个术语，即 **TEX 根目录**。以 Windows 系统为例，若 ConTEXt 的安装目录为 `d:\context` 目录，则该目录中的子目录和文件当如图 **3.2** 所示。在这种情况下，`d:\context\tex` 目录即为 TEX 根目录。对于 Linux 和 macOS 系统，若 ConTEXt 安装在 `~/context` 目录，则 `~/context/tex` 为 TEX 根目录。

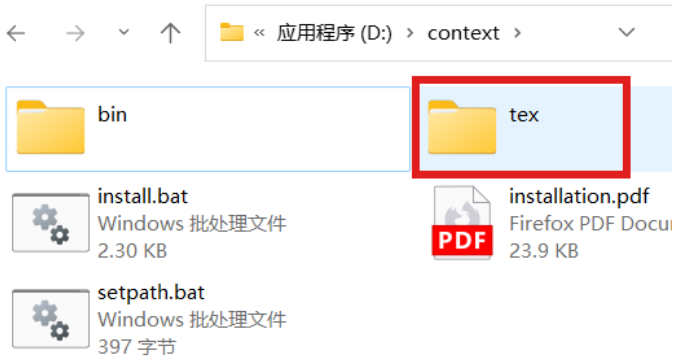


图 3.2　红框标记的目录即 TEX 根目录

以上关于 TEX 根目录的讨论，仅针对 ConTEXt 社区提供的 ConTEXt 环境。对于 TEX Live 而言，可将 `texmf-local` 目录的上级目录作为 `$TEXROOT`。为了便于描述，从现在开始，以虚构的类 Unix 环境变量风格的 `$TEXROOT` 指代 TEX 根目录，不再使用具体路径。此外，路径中的目录间隔符也统一使用类 Unix 风格进行表示，用 `/` 而非 Windows 风格的 `\`，例如 `$TEXROOT/texmf` 表示 TEX 根目录的子目录 `texmf`。

清楚何为 TEX 根目录后，便可为 ConTEXt 安装宋体、黑体和楷体等汉字字体，具体过程如下：

(1) 将字体文件复制到 `$TEXROOT/texmf-local/fonts/truetype/msfonts` 目录，若该目录不存在，可自行创建；
(2) 执行「`context --generate`」命令，刷新 ConTEXt 的文件数据库；
(3) 执行「`mtxrun --script fonts --reload --force`」命令，载入新添加的字体。

字体安装完毕后，可通过查询字体文件名确认字体是否安装成功，例如查询 simsun.ttc：

```
$ mtxrun --script font --list --file simsun.ttc
familyname  weight  style   width   variant  fontname  filename    subfont  ...
simsun      normal  normal  normal  normal   simsun    simsun.ttc        1
nsimsun     normal  normal  normal  normal   nsimsun   simsun.ttc        2
```

查询结果中的 `fontname` 栏很重要，因为在排版时，通常要使用字体名字指代某个字体。`subfont` 栏表明 simsun.ttc 文件包含了两种字体，一种是 `simsun`（宋体），另一种是 `nsimsun`（新宋体）。

如果你依稀记得某个已安装到 ConTEXt 中的字体的名字，可以用模糊形式获得字体信息。例如你只记得有个字体的名字含有 `sun`，可用以下命令查询其信息：

```
$ mtxrun --script font --list -all --pattern=sun
identifier       familyname   fontname   filename     subfont   ...
nsimsun          nsimsun      nsimsun    simsun.ttc   2         ...
...              ...          ...        ...          ...       ...
```

请留意上述命令输出信息 `identifier` 和 `familyname` 栏，因为 **3.6** 节需要这些名字。

需要注意，上述过程安装的字体都是有版权的，倘若作商业用途，需要向开发这些字体的公司支付授权费用。本文档之所以选择使用它们，主要是为了兼容国内在文档字体选

用上的积习。网络上能够找到 Google 公司开发的一系列免费的汉字字体，例如 Noto 系列，其安装方式可参考上述过程，无须赘述。此外，若安装扩展名为「.otf」的字体，即 OpenType 字体，建议将它们安装到 $TEXROOT/texmf-local/fonts/opentype 目录，若无该目录，可自行创建。

3.2 使用字体

例 3.1 演示了如何在单页环境定义字号为 12pt 的宋体，黑体和楷体等字体切换命令，并使用它们各排版一行文字。

```
\startTEXpage[frame=on,offset=4pt]
\definefont[song][name:simsun at 12pt]
\definefont[hei][name:simhei at 12pt]
\definefont[kai][name:kaiti at 12pt]
\song 潜龙勿用。\\
\hei 见龙在田，利见大人。\\
\kai 君子终日乾乾，夕惕若厉，无咎。
\stopTEXpage
```

潜龙勿用。
见龙在田，利见大人。
君子终日乾乾，夕惕若厉，无咎。

例 3.1　三种汉字字体

第一次在 ConTEXt 中使用新安装的汉字字体，源文件编译过程会较为缓慢，因为 ConTEXt 需要解析字体文件中的一些编码信息并将结果存到它的字体缓存目录。待下一次使用经过缓存后字体时，编译速度便会正常，因此不应急于宣判 ConTEXt 不适合处理中文文档。我的计算机 CPU 是 Intel i5-4460T @ 1.90GHz。这份文档写至此处，已有 23 页图文并茂的内容，其源文件单次编译时间不到 2 秒。我感觉 ConTEXt 处理中文文档，并不是很慢，但是应当承认，其引擎肯定比 pdfTEX 或 XHTEX 慢一些。

3.3 中文断行

现在尝试用宋体字排版一段中文，见例 3.2。结果表明 ConTEXt 此刻尚不知在限定宽度的版面内如何对汉字进行断行，以致文字超出版面。究其原因，是汉字之间不像西文单词以空格作为间隔，因此在 ConTEXt 看来，一段汉字文字等同于一个很长的西文单词而且不知如何断行。

```
\song
潜龙勿用。见龙在田，利见大人。%
君子终日乾乾，夕惕若厉，无咎。
```

潜龙勿用。见龙在田，利

例 3.2　中文段落无法断行

需要注意例 3.2 中的注释符的用法。虽然注释内容为空，但注释符可令 TEX 引擎忽略其后的所有空白字符（包括换行符）。倘若将该例中的注释符去掉，第一行汉字和第二行汉字之间的换行符会被 TEX 引擎视为空白字符，它会以为自己面临的是两个较长的单词而施以断行，结果见例 3.3。这种断行，是误打误撞，且结果并不堪用。

```
\song
潜龙勿用。见龙在田，利见大人。
君子终日乾乾，夕惕若厉，无咎。
```

潜龙勿用。见龙在田，利见
君子终日乾乾，夕惕若厉，

例 3.3　中文段落无法断行

现在，到了你灵机一动的时刻了。既然换行符被 TEX 引擎视为空白字符从而误打误撞完成了断行，倘若在汉字之间手工插入一些空格字符，TEX 能否实现汉字断行呢？答案是，的确如此。例 **3.4** 源码中的 ␣ 符号表示空格字符，这些空格字符的存在使得汉字发生了断行，只是它们也让汉字分布颇为蓬松。

```
\song
潜␣龙␣勿␣用。␣见␣龙␣在␣田，␣利␣见␣大␣人。
君␣子␣终␣日␣乾␣乾，␣夕␣惕␣若␣厉，␣无␣咎。
```

潜 龙 勿 用。 见 龙 在 田， 利 见 大 人。 君 子 终 日 乾 乾， 夕 惕 若 厉， 无 咎。

例 3.4　中文段落手工插入空格进行断行

如果能尽量缩小空格的宽度，便可以得到真正堪用的中文段落断行效果。TEX 引擎提供了粘连（Glue）机制，可以用它定义指定宽度且具备些许弹性的空格。例如，定义一个宽度为 0，最大可伸展 2pt 且不可收缩的粘连：

```
\def\foo{\hskip 0pt plus 2pt minus 0pt}
```

用该粘连代替空格，插入到汉字之间，便可实现中文断行，见例 **3.5**。

```
\song
潜\foo␣龙\foo␣勿\foo␣用。\foo␣见\foo␣龙\foo␣在%
\foo␣田，\foo␣利\foo␣见\foo␣大\foo␣人。%
\foo␣君\foo␣子\foo␣终\foo␣日\foo␣乾\foo␣乾，%
\foo␣夕\foo␣惕\foo␣若\foo␣厉，\foo␣无\foo␣咎。
```

潜龙勿用。见龙在田，利见大人。君子终日乾乾，夕惕若厉，无咎。

例 3.5　中文段落手工插入粘连进行断行

需要注意的是，TEX 会自动忽略排版命令后尾随的一个空格，因此 \foo 之后虽然有一个空格，但该空格不会在排版结果里出现。

由于没有人愿意像例 **3.5** 那样排版汉字，因此 ConTEXt 提供了一个可以自动在汉字之间插入粘连的命令，即

```
\setscript[hanzi]
```

只需将该命令置于需要断行的中文文本之前，便可生效。

上述过程之所以大费周章，仅仅是让你明白 \setscript[hanzi] 的原理。此外，你甚至学会了如何定义一个 TEX 宏，即 \foo，从而避免频繁输入以下排版命令：

```
潜\hskip 0pt plus 2pt minus 0pt 龙\hskip 0pt plus 2pt minus 0pt 勿……
```

倘若你擅长定义你所需要的 TEX 宏，在时间的作用下，渐渐形成可与 ConTEXt 媲美的宏包亦非难事。现在，已经隐隐知道了ConTEXt 的一些真相了吧。关于 TEX 的宏，第 17 章给出了更为深刻的探讨。

3.4 写一封真正的信

```
\startTEXpage[frame=on,width=6cm,offset=6pt]
\definefont[songti][name:simsun at 10.5pt]
\setscript[hanzi] % 中文断行支持
\songti
\setupindenting[always,first,2em]
\setupinterlinespace[1.5]
\noindent 亲爱的朋友：\par
你们好吗？\par
现在工作很忙吧，身体好吗？我现在五指山挺好的。
虽然我很少写信，其实我很怀念花果山。\par
五百年后的春节，我一定回山。
好了，先写到这吧。\par
此致\par
\noindent 敬礼\par
\rightaligned{孙悟空}
\rightaligned{2035.10.1}
\stopTEXpage
```

亲爱的朋友：

你们好吗？

现在工作很忙吧，身体好吗？我现在五指山挺好的。虽然我很少写信，其实我很怀念花果山。

五百年后的春节，我一定回山。好了，先写到这吧。

此致

敬礼

孙悟空

2035.10.1

例 3.6 孙悟空的信

3.5 字族

如同我们习惯于将汉字分为许多书体，诸如常用的宋体、楷体、仿宋、隶书、幼圆、黑体等，西方人对他们的字体也是有着一套分类体系。TEX 系统原本是针对西方文字排版而设计和开发的，因此我们需要先了解西方人对字体的分类，然后将汉字字体按自己的需要与之相应。

回忆一下，在学会安装和使用汉字字体之前，用 ConTEXt 排版英文内容，我们并未设置任何西文字体，ConTEXt 依然能完成排版。这意味着 ConTEXt 已经为用户定义了一套西文字体，且在排版环境中默认启用了。这套字体由十多种字体组成，统称为 Computer Modern Roman（简称 cmr）字体。它们可分为四族：衬线（Serif）、无衬线（Sans Serif）、等宽（Monospace）以及数学符号。每个字族又细分为正体（Regular 或 Normal）、粗体（Bold）、斜体（Italic）和粗斜体（BoldItalic）四个类别，亦即一套完整的西文字体通常至少由 16 种字体组成。

ConTEXt 默认启用的正文字体是衬线字族中的正体。\rm，\ss 和 \tt 可分别用于将字体切换为衬线、无衬线和等宽字族的正体。\tf，\bf，\it 和 \bi 可分别用于切换每个字族中的正体、粗体、斜体和粗斜体等字体。例 3.7 演示了如何切换各种字体，另外该示例用到了 TEX 编组语法。在默认情况下，TEX 引擎会将一对花括号所包含的内容视为一个整体，即编组（Group）。编组构造了局部环境，其内部的排版命令不会对编组外部产生任何影响，但编组外部的排版命令会影响编组内部。

```
% 衬线字体
Hello. {\bf Hello.}
{\it Hello.} {\bi Hello.}\\
% 无衬线字体
\ss Hello. {\bf Hello.}
{\it Hello.} {\bi Hello.}\\
% 等宽字体
\tt Hello. {\bf Hello.}
{\it Hello.} {\bi Hello.}\\
% 将字体切换为衬线字体
\rm
% 数学字体
Math in text mode: $\int_a^b f(x)dx$\\
Math in display mode:
\startformula
\int_a^b f(x)dx
\stopformula
```

Hello. **Hello.** *Hello.* ***Hello.***
Hello. **Hello.** *Hello.* ***Hello.***
`Hello. Hello. Hello. Hello.`
Math in text mode: $\int_a^b f(x)\,dx$.
Math in display mode:

$$\int_a^b f(x)\,dx$$

例 3.7　ConTEXt 默认字体

ConTEXt 默认正文字体的大小为 12 pt，并以该尺寸为基准，定义了 6 种不同级别的字体尺寸，从小到大依序为 xx，x，a，b，c，d，其中 x 级比正文字体小，a 级比正文字体大。例 **3.8** 演示了无衬线字族的正体字体 7 种大小级别的切换。

```
\ss {\tfxx A} {\tfx A} A or {\tf A}
{\tfa A} {\tfb A} {\tfc A} {\tfd A}
```

A A A or A A A A A

例 3.8　字体大小的 7 种级别

需要注意的是，尺寸单位 em 的含义与当前字体的字号相关，之前我对它给出解释是字母 M 宽度，恰好等于 1 个汉字的宽度，该说法仅对当前字体字号成立。在例 **3.9** 和 **3.10** 中，段落首行缩进设定命令出现的位置不同，导致段落缩进距离产生了差异。例 **3.9** 在设定段落首行缩进时，当前字体是 ConTEXt 默认的正文字体，其字号为 12pt，故而 1em 等于 12pt，于是缩进为 24pt，而例 **3.10** 中设置段落首行缩进时，当前字体的字号是 9pt，故而 1em 是 9pt，于是缩进为 18pt。

```
\setscript[hanzi]
\definefont[songti][name:simsun at 9pt]
\setupindenting[always,first,2em]
\songti
我现在五指山挺好的。
虽然我很少写信，其实我很怀念花果山。
```

我现在五指山挺好的。虽然我很少写信，其实我很怀念花果山。

例 3.9　段落缩进距离是 24 pt

```
\setscript[hanzi]
\definefont[songti][name:simsun at 9pt]
\songti
\setupindenting[always,first,2em]
我现在五指山挺好的。
虽然我很少写信，其实我很怀念花果山。
```

我现在五指山挺好的。虽然我很少写信，其实我很怀念花果山。

例 3.10　段落缩进距离是 18 pt

3.6 汉字字族

使用 \definefontfamily 可将一组汉字字体定义为字族，用 \setupbodyfont 命令将其设定为默认字族。例 **3.11** 将 10.5pt 的宋体作为正文默认字体——该字号对应中文排版的 5 号字。注意 \setupindenting 却只能在单页环境之内有效，这让我有些不解，权作新手村的规矩吧。

```
\definefontfamily[myfonts][rm][nsimsun]
\setupbodyfont[myfonts,10.5pt]
\setscript[hanzi]
\startTEXpage[frame=on,width=6cm,offset=6pt]
\setupindenting[always,first,2em]
我现在五指山挺好的。虽然我很少写信，其实我很怀
念花果山。
\stopTEXpage
```

我现在五指山挺好的。虽然我很少写信，其实我很怀念花果山。

例 3.11　定义正文字体并启用

例 **3.11** 仅定义了 myfonts 的衬线字族（rm）为 nsimsun，且需要注意的是，此处的 nsimsun 并非字体名，而是字族名。在 **3.1** 节中，将 simsun.ttc 安装至 ConTEXt 环境之后，曾查询过它的相关信息，其中有一栏信息是 familyname，其中罗列的便是字体所属的字族名。

ConTEXt 会根据字族名自动搜索字体的 identifier 信息，若某字体，其 identifier 的末尾是 regular 或 normal[1]，则该字体会被 ConTEXt 自动作为该字体所属字族的正体。同理，若某字体的 identifier 的末尾是 bold，italic 或 bolditalic，则该字体会被 ConTEXt 自动作为该字体所属字族的粗体、斜体或粗斜体。

由于 nsimsun 字族只有正体 nsimsunregular，没有其他字体。在例 **3.11** 中，若使用 \bf 切换字体时，即

```
\bf 我现在五指山挺好的。虽然我很少写信，其实我很怀念花果山。
```

因为 TEX 找不到相应字体，故而在给出一些错误信息后，最终排版结果为空页。要解决该问题，需使用 \definefontfamily 的第四个参数，通过字体名指定其他字体以补充字族缺失的字体。例如，可以使用黑体和楷体作为来补充 nsimsun 字族的缺失字体：

```
\definefontfamily[myfonts][rm][nsimsun][bf=simhei,it=kaiti,bi=simhei]
```

[1] identifier 名的末尾为 regular 和 normal 的字体通常是同一个字体。

同理，也可以用宋体，黑体和楷体定义非衬线和等宽字族：

```
\definefontfamily[myfonts][ss][simhei][bf=simhei,it=simhei,bi=simhei]
\definefontfamily[myfonts][tt][kaiti][bf=simhei,it=kaiti,bi=simhei]
```

为汉字定义字族还有一种传统方法，使用 ConTEXt 的 typescript 机制，但是需要写许多代码，但以前只有这一种方法，而且我也为了理解它而耗费了许多青春。现在我们可以对它说，别了，typescript!

3.7 字形替换

也许你现在觉得自己在新手村里已经脱胎换骨，能够自由地在 ConTEXt 世界里使用汉字。是的，你可以如此觉得，只要你的排版内容没有西方文字。

排版内容里有西方文字会怎样的？图 **3.3** 给了你一个选择的机会，你是觉得上面那行文字里的英文单词美观，还是下面那行呢？前者是 simsun.ttc 中的西文字形，后者是 ConTEXt 默认的 cmr 字体里的衬线正体中的西文字形。若你选择前者，则本节内容可至此完全忽略，否则请继续阅读。

图 3.3　英文字形比较

\definefallbackfamily 可以用一种字体中的部分字形替换另一种字体的对应字形。例 **3.12** 使用 latinmodernroman 字族里的每种字体的 Unicode 码位区间 [0x0000, 0x0400] 中的所有字形强行替换了 nsimsun 字族中每种字体（包括替补字体）的相应字形。

```
\definefallbackfamily
  [myfonts][rm][latinmodernroman]
  [range={0x0000-0x0400},force=yes]
\definefontfamily
  [myfonts][rm][nsimsun]
  [bf=simhei,it=kaiti,bi=simhei]
\setupbodyfont[myfonts,16pt]
\setscript[hanzi]
\startTEXpage[frame=on,offset=4pt]
爱因斯坦 Einstein\\ \bf 爱因斯坦 Einstein\\
\it 爱因斯坦 Einstein\\ \bi 爱因斯坦 Einstein
\stopTEXpage
```

爱因斯坦 Einstein
爱因斯坦 Einstein
爱因斯坦 Einstein
爱因斯坦 Einstein

例 3.12　字形替换

也可以使用 ConTEXt 已经定义了名字的 Unicode 码位区间代替 16 进制数字形式的区间，例如，

```
\definefallbackfamily[myfonts][rm][latinmodernroman]
                     [range={basiclatin,latinsupplement},force=yes]
```

指定用 Unicode 码位区间 [0x0000, 0x00FF] 中的字形作为替换字形。更多的 Unicode 码位区间的名字见文档 [7]。

结语

ConTEXt 里最难的知识，你已基本掌握了。这部分知识有多难呢，难到 Hans Hagen 需要为之撰写一本长达 228 页的手册，很诚实地说，该手册我只能看懂寥寥数页，你若有兴趣，不妨一读。这份手册就在你的 ConTEXt 环境里，执行以下命令可以找到它：

```
$ mtxrun --find-file fonts-mkiv.pdf
```

此外，现在也许你有必要阅读本文档的第 **15** 章，它提供了更为简单的汉字使用方法。

4 散文

一旦解决了 ConTEXt 对汉字的支持问题，便可以尝试让一篇文章有它该有的样子，然后会觉得 ConTEXt 越来越有用处。

一篇文章，它应该是什么样子呢？至少要有标题，有作者信息，还可能有次标题，次次标题……还要有段落，有页码。有了这些，足以用于记事。至于科技工作者通常所需要的列表、表格、数学公式、插图等排版元素，需要在文章这些该有的样子的基础上进一步构建，现在不必急于探求。

4.1 标题

在 ConTEXt 中，标题分为两种，无编号的和有编号的。每种标题又分为诸多级别。无编号的标题，级别从高到低，排版命令依次为

```
\title{...} % 一级标题
\subject{...} % 次级标题
\subsubject{...} % 次次级标题
\subsubsubject{...} % 次次次级标题
... ... ...
```

有编号的标题，级别从高到低，排版命令依次为

```
\chapter{...} % 一级标题
\section{...} % 次级标题
\subsection{...} % 次次级标题
\subsubsection{...} % 次次次级标题
... ... ...
```

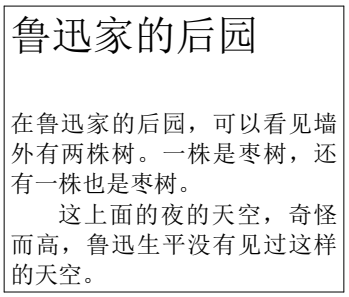
应该不难看出两种标题各自的次级标题降级规律。不建议使用级别层次太深的标题，否则会让读者觉得身陷迷宫，通常前三级标题足够使用。若是写一篇散文，标题只需要用 `\title`。若是写一本小说，只需用 `\title` 制作书名，用 `\chapter` 制作章名。我所写的这份文档，标题的层级也只是到次级标题。

4.2 写一篇散文

例 **4.1** 设定了段落首行缩进距离，用 `\title` 创建了文章标题。

```
\definefallbackfamily[myfonts][rm][latinmodernroman]
                      [range={0x0000-0x0400},force=yes]
\definefontfamily[myfonts][rm][nsimsun]
                  [bf=simhei,it=kaiti,bi=simhei]
\setupbodyfont[myfonts,16pt]
\setscript[hanzi]
```

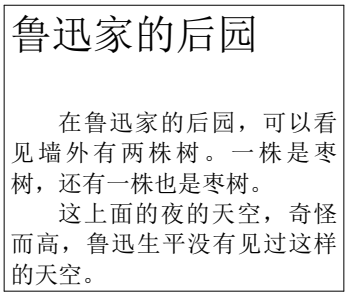
```
\startTEXpage[frame=on,offset=4pt,width=8cm]
\setupindenting[first,always,2em]

\title{鲁迅家的后园}

在鲁迅家的后园，可以看见墙外有两株树。一株是
枣树，还有一株也是枣树。\par
这上面的夜的天空，奇怪而高，鲁迅生平没有见过
这样的天空。

\stopTEXpage
```



例 4.1　散文示例 1

上例未写出作者的名字，以便你能够观察到 ConTEXt 标题之后第一段的首行是不缩进的，这是西文的排版习惯。在使用标题命令前，需要用 \setupheads 为所有标题设定其后第一段的首行必须缩进，见下例。

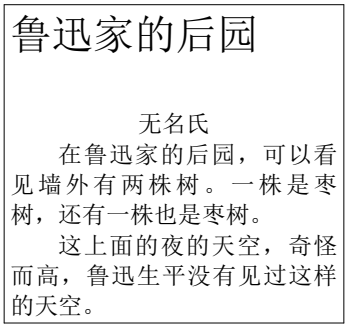
```
\setupindenting[first,always,2em]
\setupheads[indentnext=yes]

\title{鲁迅家的后园}
% ... 省略正文内容 ...
```



例 4.2　散文示例 2

现在可以为文章增加作者信息了，他叫无名氏，见下例。

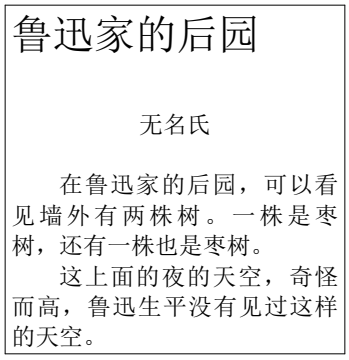
```
\setupheads[indentnext=yes]
\setupindenting[first,always,2em]

\title{鲁迅家的后园}
\midaligned{无名氏}
% ... 省略正文内容 ...
```



例 4.3　散文示例 2

上例存在的问题是，作者名字距正文过近，而难以凸显。不要尝试在作者名字之后增加一些空行来解决这个问题。TEX 引擎在遇到多个空行时，它也只是把它们当成一个空行，并将其视为 \par。在版面的竖直方向，段落之间，或标题与段落之间，或标题与标题之间……增加空白距离，可使用 \blank 命令。下例在作者和正文之间增加一个空行的距离，只需 \blank[line]；要增加 n 个空行的距离，只需 \blank[n*line]。

```
\title{鲁迅家的后园}
\midaligned{无名氏}
\blank[line]
% 省略了正文内容
```



例 4.4　散文示例 3

若需要将标题居中，只需使用 \setuphead 单独为 \title 设定样式：

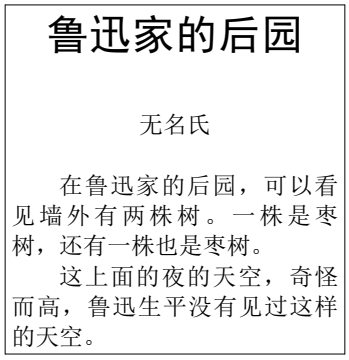
```
\setuphead[title][align=middle]
```

若汉字字族已设定粗体，则可将标题的样式设为粗体，并指定字号级别：

```
\setuphead[title][style=\bfc,align=middle]
```

例 4.5 的排版结果已经基本合规了，只是标题里的汉字的分布有些疏松，原因是汉字之间粘连的伸长特性被激活了，大概是 ConTEXt 过于追求文字居中对齐精度所致。

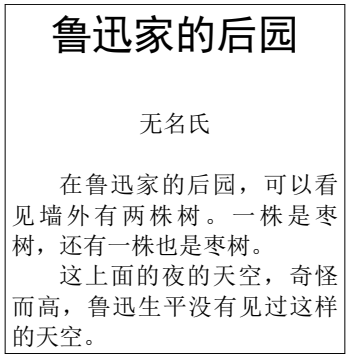
```
\setupheads[indentnext=yes]
\setuphead[title][style=\bfc,align=middle]
\setupindenting[first,always,2em]

\title{鲁迅家的后园}
\midaligned{无名氏}
% ... 省略了正文内容 ...
```



例 4.5　散文示例 4

对于上例存在的问题，只需将 \setuphead 的参数 align 的值设定为{middle,broad} 便可适当放松 ConTEXt 过于严格的对齐规则，见例 4.6。

```
\setuphead[title][style=\bfc,align={middle,broad}]
```



例 4.6　散文示例 5

不知 ConTEXt 从哪个版本开始，提供了新的参数 center，它与 {middle,broad} 等效，故而上述设定亦可写为

```
\setuphead[title][style=\bfc,align=center]
```

请记住此事，因为以后会经常需要设定其他排版元素的居中对齐，所用参数是相似的，亦即今后在设定某些排版元素的 align 参数时，至少在中文排版时，建议忘记 middle，只用 center 即可。

4.3 正式踏入 ConTEXt 世界

新手村终究太小了，小到已经不太容易让你尝试越来越多的版命令了。事实上，真正的 ConTEXt 世界用起来要比新手村更为简单，只需用正文环境亦即 text 环境代替单页环境即可。此外，建议将一切设置排版样式的命令放在正文环境之前，从而在正文环境里，只需要关心文章或书籍的内容。

以下代码应当有助于你看到 ConTEXt 世界大致面目。它是完整的，亦即可将其保存为 ConTEXt 源文件并予以编译。

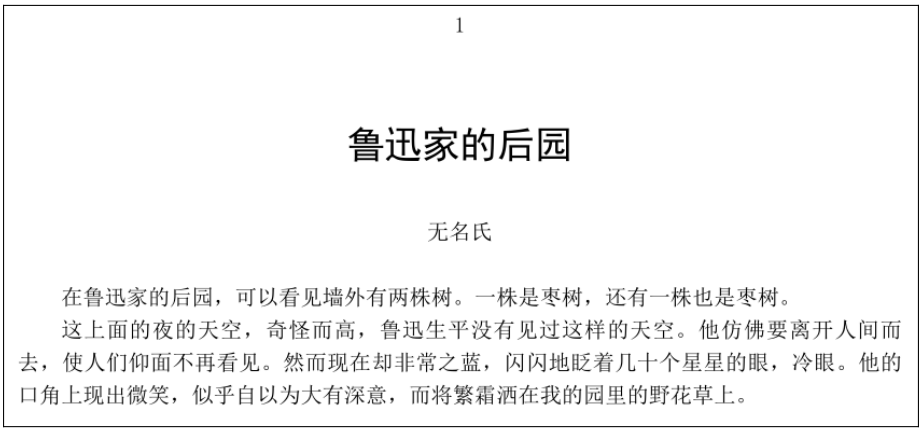
```
% 排版样式
\definefontfamily[myfonts][rm][nsimsun][bf=simhei]
\setupbodyfont[myfonts,10.5pt]
\setscript[hanzi]
\setupheads[indentnext=yes]
\setuphead[title][style=\bfc,align=center]
\setupindenting[first,always,2em]
\setupinterlinespace[line=1.5\bodyfontsize]
% 正文环境
\starttext
\title{鲁迅家的后园}
\midaligned{无名氏}
\blank[line]
在我的后园，可以看见墙外有两株树，一株是枣树，还有一株也是枣树。
... ... ...
\stoptext
```

4.4 页码

如果你亲自动手编译了 4.3 节的 ConTEXt 源文件，应当能看到，排版结果的页眉是有页码的，如图 4.1 所示。这是 ConTEXt 默认的页码样式，即页码出现在每一页，且居中位于页眉，这通常并不合乎多数中文文档的排版习惯，需要设定页码样式。

文章标题所在页面，通常不需要页码，因此需将标题样式将页眉和页脚置空：

```
\setuphead[title][header=empty,footer=empty]
```


1

鲁迅家的后园

无名氏

在鲁迅家的后园，可以看见墙外有两株树。一株是枣树，还有一株也是枣树。

这上面的夜的天空，奇怪而高，鲁迅生平没有见过这样的天空。他仿佛要离开人间而去，使人们仰面不再看见。然而现在却非常之蓝，闪闪地眨着几十个星星的眼，冷眼。他的口角上现出微笑，似乎自以为大有深意，而将繁霜洒在我的园里的野花草上。

图 4.1　ConTEXt 默认页码位置

然后，修改页码投放位置，例如将其放在页脚右侧：

```
\setuppagenumbering[location={footer,right}]
```

4.5 内容与样式分离

用 ConTEXt 或者任何一种 TEX，保持排版样式与内容的分离永远都是值得鼓励的行为。这种分离极为简单。例如，新建一份文件 foo-env.tex，令其内容为

```
\definefallbackfamily
    [myfonts][rm][latinmodernroman][range={0x0000-0x0400},force=yes]
\definefontfamily
    [myfonts][rm][nsimsun][bf=simhei,it=kaiti,bi=simhei]
\setupbodyfont[myfonts,16pt]
\setscript[hanzi]

\setupindenting[first,always,2em]
\setupinterlinespace[line=1.5\bodyfontsize]
\setupheads[indentnext=yes]
\setuphead[title][style=\bfc,align=center]
\setuphead[title][header=empty,footer=empty]
\setuppagenumbering[location={footer,right}]
```

foo-env.tex 即为样式文件，它可以重复使用，也可以分享给他人使用。

假设我们在排版一份文档时使用上述 foo-env.tex 文件中的样式，只需用 \environment 命令将该文件的内容载入即可。例如

```
\environment foo-env % 文件扩展名可以省略
\starttext
% ... 正文 ...
\stoptext
```

样式与内容分离的意义是，当我们在编写文档内容时，不会受到排版样式的任何干扰，甚至完全无需考虑任何与排版有关的事。当内容彻底定稿后，再考虑排版问题。所用

的排版样式，可以自行设计，可以复用之前的样式，也可以是从他人那里获得。这是很基本的工程学思想，即在设计上将可以复用的事物与不可复用的事物先隔离开，再以简单的形式构造建立二者的联系。

结语

现在，你已经可以用 ConTEXt 写信件、日记、随笔甚至一些读书笔记了。倘若动手尝试了 `\chapter` 命令，你甚至能用 ConTEXt 写一本小说，只是风格过于朴素而已。

若想让排版结果更为精致，ConTEXt 博大精深，总有途径能够实现你的想法，前提是你要用心。TEX 之父 Donald Knuth 曾有一言，「我从来也不期盼 TEX 会成为某种万能的排版工具，用于制作一些快速而脏的东西；我只是将其视为一种只要你足够用心就能得到最好结果的东西。」

也许，很多人觉得 TEX 太难了，实际上并非如此。TEX 应该是简单的，而用心……这件事对于大多数人而言，是件难事。

5 列表

若是帮领导起草并排版一份会议讲话稿，须知天下没有领导不偏爱含有列表的文章。大可以相信，ConTEXt 列表绝对不会让领导失望。

5.1 待办事项

在准备用 ChatGPT 或 DeepSeek 给领导起草讲稿之前，先用 ConTEXt 列表安排一下今日待办事项：

```
2023 年 3 月 22 日
\startitemize
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

2023 年 3 月 22 日

- 中午，晒十五分钟太阳
- 晚上，看流浪地球 II

例 5.1　待办事项

现在你已经学会了列表的用法了，余下的事，仅仅是根据需要定义它的外观。此外，你也学到了如何制作大写罗马数字。若需要制作小写罗马数字，只需将 \Romannumerals 换成 \romannumerals。

5.2 无序号列表

例 5.1 已经展示了样式最为简单的无序号列表，列表项符号是实心圆点。该例中的列表实际上省略了列表项符号的设定，其完整形式为

```
\startitemize[1]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

将上述代码中的数字 1 换成 2～9 的任何一个数字，可更换序号样式。例如

```
\startitemize[8]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

- □ 中午，晒十五分钟太阳
- □ 晚上，看流浪地球 II

例 5.2　改变列表项符号

5.3 有序号列表

将无序号列表的符号参数换为 n，便可得到有序号列表。例如

```
\startitemize[n]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

1. 中午，晒十五分钟太阳
2. 晚上，看流浪地球 II

例 5.3　有序号列表

有时，需要序号形式是带括号的数字，可像下面这样设定：

```
\startitemize[n][left=(,right=),stopper=]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

(1) 中午，晒十五分钟太阳

(2) 晚上，看流浪地球 II

例 5.4　数字带括号的有序号列表

其中「stopper=」是将参数 stopper 置空，从而消除列表项序号之后的西文句点「.」。

若将列表项序号参数设定为 a，A，r，R，对应的序号形式分别为小写英文字母、大写英文字母、小写罗马数字、大写罗马数字。

5.4 留白

消除列表项之间的空白，只需

```
2023 年 3 月 22 日
\startitemize[1,packed]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

2023 年 3 月 22 日

- 中午，晒十五分钟太阳
- 晚上，看流浪地球 II

例 5.5　消除列表项之间的空白

消除列表前后以及列表项之间的空白，只需

```
2023 年 3 月 22 日
\startitemize[1,nowhite]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

2023 年 3 月 22 日
- 中午，晒十五分钟太阳
- 晚上，看流浪地球 II

例 5.6　消除列表项之间的空白

5.5 引用

在写论文时，通常要引用一些文献。这些文献通常是以列表的形式附在论文之后，在正文里以引用的方式自动呈现文献在列表中的序号。如此，当文献在列表中的次序有所调整时，正文中的引用所呈现的序号也会相应变化，而无需手工维护。基于列表的引用机制可以完成这类任务，见例 **5.7**。

```
\title{论文}

正文引用了文献[\in[胡说]]和文献[\in[八道]]……

\subject{参考文献}
\startitemize[n,joinedup]
              [left={[},right={]},
               distance=.5em, stopper=]
\item[胡说] 这是胡说的论文。
\item[八道] 这是八道的论文。
\stopitemize
```

论文

正文引用了文献[1]和文献[2]……

参考文献

[1] 这是胡说的论文。
[2] 这是八道的论文。

例 5.7　参考文献列表及引用

例 **5.7** 中，列表的 distance 参数用于设定列表项的序号与内容的间距，我将其设定为半个字宽。\item 命令后的方括号中的内容是引用名。在正文里，使用 \in 命令可以通过列表项的引用名获得其序号。

不过，这种基于列表的参考文献机制只适用于文献数量不多的情况。倘若有大量的参考文献数据，而文章仅引用其中一部分，对于此类任务，ConTEXt 提供了更为专业的参考文献数据管理和引用机制，该机制的基本用法，见 **14** 章。

5.6 画符

上文所述的列表样式用于常规文档的排版应该是足够用的，但现实中还有许多非常规文档，例如幻灯片、海报和杂志等，这类文档需要排版元素具有美学特征，而非朴素的科技文档特征。ConTEXt 允许我们使用 MetaPost 语言画出我们想要的列表符号。

MetaPost 是一种语法简洁但功能丰富的矢量绘图编程语言。ConTEXt 早已内嵌了该语言的解释器，我们可以直接在 ConTEXt 源文件里编写 MetaPost 代码，然后交由 ConTEXt 编译器生成嵌入在排版结果中的图形。本文档的第 **11** 章讲述了 MetaPost 的基本语法，故而若下文代码超出你的理解范围，可先行阅读该章内容，也先保留不解，待后续读至该章时自然获解。此外，文档 [8 – 10] 亦可供参考。

由于 ConTEXt 提供的列表项符号皆为黑色图案，可以用 MetaPost 画一些彩色图案作为列表项符号。下面先用 MetaPost 语言在 uniqueMPgraphic 环境里画一个带阴影且用复合色填充的正方形：

```
\startuniqueMPgraphic{带阴影的正方形}
numeric u; path p;
u := BodyFontSize;
p := fullsquare scaled .8u;
fill p shifted (.2u, -.2u) withcolor lightgray;
fill p withcolor .5[blue, white];
\stopuniqueMPgraphic
```

uniqueMPgraphic 环境中的 MetaPost 代码可以作为插图使用，见例 **5.8**。

```
这个\,\uniqueMPgraphic{带阴影的正方形}\,是
MetaPost 图形。
```

这个 是 MetaPost 图形。

例 5.8 带阴影的正方形图案

例 **5.8** 中的「\,」是 TEX 命令，用于构造 $\frac{1}{6}$ 字宽的间距。顺便记着，「\;」可构造 $\frac{5}{18}$ 字宽的间距，而一个字宽的空格可以用 \quad 命令构造。有时在遇到普通空格过宽，或者空格被 ConTEXt 编译器吞噬时，可以用这些 TEX 间距命令构造空格。关于 ConTEXt 里的空格，你需要知道，汉字之间的空格会被 ConTEXt 编译器吞噬掉，而汉字与西文字符之间如果存在多个空格，ConTEXt 编译器只保留 1 个。

现在，请认真观察上例中的正方形图案，应该能发现它的位置与同一行的其他文字相比有些上浮，这是因为图形对象缺乏字符基线所致。字符基线是西文字体特有的概念。西文字符的基线连接起来，便构成一行文字的基线。大多数字符包括汉字都在基线之上，只是有一些西文字符，如 g、p、y 等，它们的有一部分沉于基线之下。于是，一行文字里，字符的最大高度是由基线之上的部分加上基线之下的部分，这两个部分的尺寸比例是 0.72 : 0.28。上述示例创建的正方形图形，它的高度是最大字符高度，但是在排版时，它像汉字那样位于基线之上了，故而显得有些上浮。要解决这个问题，需要将图形置于 \hbox 命令创建的水平盒子里，再用 \lower 命令让这个盒子下沉 0.2 倍的 \bodyfontsize，见例 **5.9**。

```
这个\,\lower.2\bodyfontsize\hbox{\uniqueMPgraphic{带阴影的正方形}}\,是 MetaPost
图形。
```

这个 是 MetaPost 图形。

例 5.9 带阴影的正方形图案

为什么要下沉 0.2 倍的正文字号呢？原因是，作为阴影的正方形相对于淡蓝色的正方形向下偏移了 0.25 的正文字号。如果让包含了这个 MetaPost 图形的盒子下沉这个距离，刚好能让淡蓝色的正方形底线位于基线上。

用 \definesymbol 命令可将这个正方形图案定义为列表项符号，由于 ConTEXt 的 9 种列表项符号的序号是从 1 至 9，故而我们自己画的符号可从序号 10 开始，即

```
\definesymbol[10][{\lower.2\bodyfontsize\hbox{\uniqueMPgraphic{带阴影的正方形}}}]
```

之后便可在列表环境里使用这个符号，见例 **5.10**。

```
\startitemize[10,nowhite]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

- 中午，晒十五分钟太阳
- 晚上，看流浪地球 II

例 5.10 列表项的画符

5.7 带圈的数字

对于有序列表，有些文档格式要求用带圆圈的数字，ConTEXt 没有提供这种风格的列表项序号。不过，即使你并未明白上一节的列表项画符的一些技术细节，应该也有一些这样的信心——ConTEXt 未提供的，我们总有办法创造出来。

ConTEXt 支持 Unicode 编码的文字，而汉字字体通常提供了带圈的数字，从 1 至 9，可以用 \char 命令获得它们，例如

```
\char"2460, \char"2461, \cdots, \char"2468
```

结果为 ①，②，…，⑨。

在列表环境里，可以用 \sym 代替 \item，直接指定我们想要的序号。例如

```
\startitemize
\sym{\char"2460} 中午，晒十五分钟太阳
\sym{\char"2461} 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

① 中午，晒十五分钟太阳
② 晚上，看流浪地球 II

例 5.11　带圈的序号

这样虽然能解决问题，但颇为不雅，亦即 Donald Knuth 所说的，快速而脏的东西。不过，要想得到高雅的结果，需要耗费一些心力，甚至需要一些编程。

实际上当你开始用 ConTEXt 排版第一个示例时，你已经在编程了。TEX 是一门排版编程语言，你见过的任何一个 ConTEXt 命令，本质上都是一小段甚至上千行的 TEX 程序。

TEX 编程对于大多数人而言，难以掌握，但是从 2000 年代末开始，ConTEXt 开发者将更易掌握的 Lua 语言引入 TEX 世界，使得 TEX 编程变得容易许多，当然这需要你对 Lua 语言有所了解。若你对这一主题颇感兴趣，可以先行阅读第 **17** 章，然后再阅读下文。

有序号列表的序号可用 \defineconversion 命令定义。例如

```
\defineconversion[带圈数字][\CircledNumber]

\startitemize[带圈数字]
\item ... ... ...
\item ... ... ...
\stopitemize
```

关键在于 \CircledNumber 命令如何定义，该命令的形式如下

```
\def\CircledNumber#1{... 定义...}
```

其中 #1 表示 \CircleNumber 接受 1 个参数，即列表项的序号。由于 \CircledNumber 是由 itemize 环境触发的，且后者每次触发它时，就将当前列表项的序号传给它。于是，我们在实现 \CircledNumber 的定义时，所考虑的问题仅仅是如何将列表项的序号映射为带圆圈的数字，这个过程过去只能用 TEX 语言实现，现在可以用 Lua 语言实现。不过，我们最好事先做一下热身运动，实现一种 ConTEXt 已经实现了的序号，见例 **5.12**。

```
\def\FooNumber#1{【#1】}
\defineconversion[Foo][\FooNumber]

\startitemize[Foo][distance=1em, stopper=]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

【1】中午，晒十五分钟太阳
【2】晚上，看流浪地球 II

例 5.12　汉字方括号序号列表

例 **5.12** 中的列表实际上等效于

```
\startitemize[n, nowhite][left=【, right=】, distance=1em, stopper=]
... ... ...
\stopitemize
```

不过基于该示例，你应该能够清晰地观察到 \FooNumber 所接受的参数是什么以及我们如何使用它的，只要将这个参数传递给一个 Lua 语言编写的函数，由后者将其映射为带圆圈的数字，则我们的问题便得以解决。现在，先写出这个 Lua 函数。ConTEXt 允许我们在 luacode 环境里编写 Lua 代码，见以下代码：

```
\startluacode
function to_circled_number(x)
  context('\\char"' .. tostring(2460 + x - 1))
end
\stopluacode
```

ConTEXt 编译器并非在同一个空间里执行 ConTEXt 排版代码和嵌入的 Lua 代码，这两种代码各自有一个运行空间，毕竟它们是不同的编程语言。Lua 空间里的数据需要以字符串对象的形式通过 context 函数传回 ConTEXt 空间。

上述函数 to_circled_number 所作的工作是，将外界传入的数值 x 减 1，再与 2460 相加，然后将所得结果用 tostring 转换为字符串，在其前面加上 \char" 前缀，最后由 context 函数将结果传回 ConTEXt 空间。例如，假设 x 是 3，经过 to_circled_number 的一番处理，它变成了 Lua 字符串 '\\char"2462'，该结果由 context 函数传回 ConTEXt 空间。

同理，在 ConTEXt 空间调用 Lua 空间里的一个函数，也需要一个通道，该通道由 \ctxlua 命令实现，基于这个命令便可定义 \CircledNumber 命令，即

```
\def\CircledNumber#1{\ctxlua{to_circled_number(#1)}}
```

于是，用带圆圈数字作为列表序号的任务便如此优雅地完成了，见例 **5.13**。现在不妨登上高处，俯瞰整个方案。实际上并不难，对吗？

```
\startluacode
function to_circled_number(x)
  context('\\char"' .. tostring(2460 + x - 1))
end
\stopluacode
```

```
\def\CircledNumber#1{\ctxlua{to_circled_number(#1)}}
\defineconversion[CircledNum][\CircledNumber]

\startitemize[CircledNum][stopper=]
\item 中午，晒十五分钟太阳
\item 晚上，看流浪地球 \Romannumerals{2}
\stopitemize
```

① 中午，晒十五分钟太阳
② 晚上，看流浪地球 II

例 5.13　Lua 与 TEX 的结合

结语

本章的最后两节，可能会让你有一些压迫感。也许你原以为 ConTEXt 已经为你提供好了一切命令，无论你想要什么样的排版效果，只需要找到相应的命令便可达成目的。这当然是不可能的，否则 ConTEXt 开发者就成了上帝，知晓你的一切心意。上帝是不存在的，但这个世界是可编程的。

6 表格

表格通常用于呈现一些成行成列的数据，但是不要对此有刻板印象，有时一些非常规的页面布局，若基于表格实现，也许颇具奇效。ConTEXt 有三种表格，从易到难，依次是 tabulate，table 和 xtable，详见文档 [11]，本章仅讲述 tabulate，而且我也建议你应该先掌握 tabulate，待其难以满足需求时，再考虑学习另外两种。

6.1 基本格式

以下代码构造了一个 2 行 3 列的表格，第 1 列居左（l），第 2 列居中（c），第 3 列居右（r）。

```
\starttabulate[|l|c|r|]
\NC 1 \NC 2 \NC 3 \NC\NR
\NC 4 \NC 5 \NC 6 \NC\NR
\stoptabulate
```

1	2	3
4	5	6

「|l|c|r|」是表格各列的格式声明，你可以不指定各列的对齐方式，例如「||||」，但是格式声明不能省略，否则表格的排版结果可能会变得非常诡异。\NC 用于构造新的单元格，\NR 用于构造一个新行。

6.2 边线

将 \NC 替换为 \VL，用 \HL 可以画出单元格的左右边线。例如

```
\starttabulate[|l|c|r|]
\HL
\VL 1 \VL 2 \VL 3\VL\NR
\HL
\VL 4 \VL 5 \VL 6\VL\NR
\HL
\stoptabulate
```

1	2	3
4	5	6

上例表格的竖线像是被横线打断了，实际上是因为 tabulate 环境构造的表格，各行之间默认存在一定间距导致的。若想消除该现象，只需将单元格之间的纵向间距参数 distance 设为 0 尺度，例如 0mm，也可以设成 none。例如

```
\starttabulate[|l|c|r|][distance=0mm]
\HL
\VL 1 \VL 2 \VL 3 \VL\NR
\HL
\VL 4 \VL 5 \VL 6 \VL\NR
\HL
\stoptabulate
```

1	2	3
4	5	6

6.3 对齐的假象

在上文的所有示例里，第 1 列和第 3 列的对齐实际上存在假象。即使将它们也设为居中对齐，它们的位置似乎依然不变，见下例，第 1 列依然是居左的，第 3 列依然是居右的。即使将第 1 列设为 r，将第 3 列设为 l，结果依然一样。甚至，即使你将第 2 列设为 l 或 r，结果它依然是居中的。

```
\starttabulate[|c|l|c|]
\VL 1 \VL 2 \VL 3 \VL\NR
\VL 4 \VL 5 \VL 6 \VL\NR
\stoptabulate
```

1	2	3
4	5	6

事实上，tabulate 单元格内容的对齐，并不以单元格空间亦即一个矩形框为基准，而是以这列内容本身为基准。例如，如果某列设为 l，仅仅是意味着这列内容是左对齐的。上述例子之所以看不到这种效果，是因为单元格的内容过于简单，下例让单元格的内容变得长短不一，便可呈现该效果。

```
\starttabulate[|r|c|l|]
\VL 1 \VL 2 \VL 3.14159 \VL\NR
\VL 3.14159 \VL 5 \VL 6 \VL\NR
\stoptabulate
```

1	2	3.14159
3.14159	5	6

造成假象的另一个原因是，tabulate 各列内容之间默认存在间距，而该间距并不属于单元格。你应该将表格的竖线理解成有一些潜在的宽度的线，该宽度值可通过 unit 参数予以设定，下例将其设为 0 长度，结果虽然丑陋，但如果你真的希望单元格的内容真正位于单元格的左侧、中央还是右侧，则必须如此设定。

```
\starttabulate[|r|c|l|][unit=0mm]
\VL 1 \VL 2 \VL 3.14159 \VL\NR
\VL 3.14159 \VL 5 \VL 6 \VL\NR
\stoptabulate
```

1	2	3.14159
3.14159	5	6

在上例的基础上，只要设定各列宽度，便可以得到预想的对齐效果，见下例。设定列宽的格式声明是 w(尺寸)，可与对齐声明联合使用。

```
\starttabulate[|rw(2cm)|cw(1cm)|lw(1.5cm)|][unit=0mm]
\VL 1 \VL 2 \VL 3.14159 \VL\NR
\VL 3.14159 \VL 5 \VL 6 \VL\NR
\stoptabulate
```

1	2	3.14159
3.14159	5	6

6.4 三线表

tabulate 环境主要是为科技文献排版中常用的三线表设计的，这是造成 **6.3** 节对齐假象的最根本的原因。在三线表中，是不需要表格竖线的，故而默认的 unit 参数值，不会给你造成任何不适，见下例。

```
\starttabulate[|c|c|c|]
\HL
\NC 甲 \NC 乙 \NC 丙 \NC\NR
\HL
\NC 一 \NC 二 \NC 三 \NC\NR
\NC one \NC two \NC three \NC\NR
\HL
\stoptabulate
```

甲	乙	丙
一	二	三
one	two	three

不过，三线表的顶线和底线，粗度要比表格内的线要粗一些。虽然 \HL 不能设定粗细，但 tabulate 环境提供了 \TL 和 \BL，可分别用于绘制指定粗细以及颜色的表格顶线和底线，T 表示 Top，B 表示 Bottom，这两种横线命令的用法如下：

```
\starttabulate[|c|c|c|]
\TL[3]
\NC 甲 \NC 乙 \NC 丙 \NC\NR
\HL
\NC 一 \NC 二 \NC 三 \NC\NR
\NC one \NC two \NC three \NC\NR
\BL[3,darkred]
\stoptabulate
```

甲	乙	丙
一	二	三
one	two	three

如果表格还需要定制更多横线的粗细，tabulate 环境还提供了 \FL、\ML 和 \LL，分别用于构造表格顶线之下的第一条横线，中间的横线以及最后一条横线。因而除了默认粗度的 \HL 之外，共有 5 种可设定粗细的线型备用，对于三线表而言，这些线足够用了，实际上我们只需要两种线，一种是 \HL，另一种是可设定粗细的，T、F、M、L、B 只是希望横线具有语义。

可设定横线粗细的命令，设定线的粗细的数字参数，其值是表格线默认粗度的倍数。实际上，还有一个数字参数，其值也是表格线默认粗度的倍数，用于构造虚线，即虚线的空白间隔是表格线默认粗度的倍数，见下例。

```
\starttabulate[|c|c|c|]
\TL[3]
\NC 甲 \NC 乙 \NC 丙 \NC\NR
\FL[1, 2, blue]
\NC 一 \NC 二 \NC 三 \NC\NR
\NC one \NC two \NC three \NC\NR
\BL[3,darkred]
\stoptabulate
```

甲	乙	丙
一	二	三
one	two	three

6.5 段落成列

单元格里，可以放置含有段落的内容，格式声明为 p 或 p(尺寸)，后者可限定段落的宽度。段落本身以及段落所能包含的任何内容，皆能嵌入在单元格内。下例在单元格内嵌入了一个列表。

```
\starttabulate[|c|p(4cm)|][align=normal]
\TL[3]
\NC 列表 \NC
        \startitemize
        \item 中午，晒十五分钟太阳
        \item 晚上，看流浪地球\,\Romannumerals{2}
        \stopitemize \NC\NR
\BL[3]
\stoptabulate
```

列表	• 中午，晒十五分钟太阳 • 晚上，看流浪地球 II

若段落的格式声明未提供宽度，单元格的宽度会尽可能地占满页面空间。例如，如果表格只有 1 列，则该列的宽度会占满正文宽度，如果表格有 2 列，则这两列平分正文宽度，见下例。

```
\starttabulate[|p|p|]
\TL[3]
\NC 天下皆知美之为美，斯恶已；皆知善之为善，斯不善已。
\VL 道冲而用之或不盈。渊兮似万物之宗。
\NC\NR
\BL[3]
\stoptabulate
```

天下皆知美之为美，斯恶已；皆知善之为善，斯不善已。	道冲而用之或不盈。渊兮似万物之宗。

6.6 像插图那样

虽然我还没有介绍如何在文档中插入图片，但是表格可以像插图那样出现于文档，通常居中放置，并配备标题，\placetable 命令可做此事，见下例。不过，该例也有诡异之处，即使右侧列格式是段落，\NR 之前没有放置 \NC，但是在 \placetable 的作用下，右侧单元内容的位置变得正常了。

```
\placetable[here][]{道德经里的句子}{
  \starttabulate[|p|p|]
  \TL[3]
  \NC 天下皆知美之为美，斯恶已；皆知善之为善，斯不善已。
  \VL 道冲而用之或不盈。渊兮似万物之宗。
  \NC\NR
  \BL[3]
  \stoptabulate
}
```

天下皆知美之为美，斯恶已；皆知善之为善，斯不善已。	道冲而用之或不盈。渊兮似万物之宗。

Table 6.1　道德经里的句子

ConTEXt 默认的表格标题的位置以及序号前缀皆不合中文文档的常见格式，存在以下问题：

- 序号的前缀应该「表」，而非「Table」；
- 序号部分不应该是粗体，且字号应当比正文小一号；
- 标题中的文字过于蓬松，且应当位于表格上方。

以下代码能够让表格标题的样式符合中文排版惯例。希望你还没有忘记 **4.5** 节所讲的内容与样式分离，你应当将这些设定放置于文档的样式文件。

```
% 开启中文界面
% 将「Table」和「Figure」等序号前缀切换为「表」和「图」等。
\mainlanguage[cn]

% 将表格标题改为顶部放置
\setupcaption
  [table][headstyle=normal, style=small, align=center, location=top]
% 上述设定亦可写成
% \setupcaption
%   [table]
%   [headstyle=\rm, style=\tfx, align=center, location=top]
```

`\setupcaption` 的 `headstyle` 参数用于设定表格标题的序号部分（包括前缀）的样式，其值为 `normal` 时，表示使用字族中的正体，等效于 `\rm`；该参数的默认值是 bold，即粗体。`style` 参数用于设定标题的正体风格，其值 `small` 表示用小号字，等效于 `\tfx`。`align` 参数，在 **4.2** 节已经见过了，应当将其设为 `center` 方能避免标题汉字之间的粘连被意外激活而导致文字蓬松。`location` 参数用于设定标题在表格上方（top）还是在底部（bottom），还是在其他位置。上述样式设定产生的效果见表 **6.2**。

表 6.2　道德经里的句子

天下皆知美之为美，斯恶已；皆知善之为善，斯不善已。	道冲而用之或不盈。渊兮似万物之宗。

`\placetable` 命令需要 4 个参数，其形式可抽象为

```
\placefigure[位置][引用]{标题}{\starttabulate ... \stoptabulate}
```

上例将「位置」设定为 `here`，意思是就在 `\placetable` 出现的位置插入表格，但并不是强制的，若需要强制，应将 `here` 换成 `force`。

也许你会有所不解，表格出现的位置还需要设定吗？它难道不是理所应当出现在 `\placetable` 所在的位置吗？答案是，ConTEXt 的表格和插图是浮动的，排版引擎会尽量为它们寻找合适的安放空间，而 `here` 和 `force` 可以禁止排版引擎的自作聪明。「位置」参数也可以设定为 `left` 和 `right`，分别将表格放置于页面的左侧和右侧。

至于 \placetable 的「引用」，其幕后的机制和用法与 5.5 节介绍的列表项引用相似。你可以为表格设定一个引用名，例如 foo，然后在正文里可以用 \in[foo] 以序号的形式引用该表格，例如

```
如表 \in[foo] 所示，... ... ...\par
\placefigure[here][foo]{标题}{\starttabulate ... \stoptabulate}
```

如果一个表格不需要任何引用，则可像上例，将此项空置即可。

如果你不需要表格带有标题，但是又想利用 \placetable 的功能，可在该命令的第 1 个参数添加 none，然后将表格标题置空，例如

```
\placetable[right,none][]{}{
  \starttabulate[|c|c||c]
    \NC 2 \NC 9 \NC 4 \NC\NR
    \NC 3 \NC 5 \NC 7 \NC\NR
    \NC 6 \NC 1 \NC 8 \NC\NR
  \stoptabulate
}
```

2	9	4
3	5	7
6	1	8

6.7 横列表

也许你是一位教师，经常用 ConTEXt 排版试卷。试卷的选择题，选项较短时，通常是水平排列的四项，序号是 A、B、C、D。你可以用 itemize 的分列功能来完成该任务，例如

```
\startitemize[A, columns, four]
\item 花非花 \item 雾非雾 \item 夜半来 \item 天明去
\stopitemize
```

A. 花非花　　B. 雾非雾　　C. 夜半来　　D. 天明去

若你不知 itemize 环境具有上述用法，也可以用表格解决这个问题。例如

```
\starttabulate[|p|p|p|p|]
\NC A. 花非花 \NC B. 雾非雾 \NC C. 夜半来 \NC D. 天明去 \NC\NR
\stoptabulate
```

A. 花非花	B. 雾非雾	C. 夜半来	D. 天明去

虽然表格方案并不比列表更好，但是在略微复杂一些的情况里，例如当你需要将 A 和 B 项放在一行，C 和 D 项放在下一行时，列表方案就会失效，它会将 A 和 B 放在一列，C 和 D 放在一列，见下例。

```
\startitemize[A, nowhite, columns, two]
\item 花非花 \item 雾非雾 \item 夜半来 \item 天明去
\stopitemize
```

A. 花非花
B. 雾非雾
C. 夜半来
D. 天明去

之所以如此，是因为 itemize 环境是纵向优先。要解决这个问题，列表方案必须拆解为

两个单行的列表，让第 2 个列表的序号续接第 1 个，见下例。

```
\startitemize[A, nowhite, columns, two]
\item 花非花 \item 雾非雾
\stopitemize
\startitemize[A, nowhite, columns, two, continue]
\item 夜半来 \item 天明去
\stopitemize
```

A. 花非花　　B. 雾非雾
C. 夜半来　　D. 天明去

对于上述问题，在表格方案里，就很简单，只需要构造两行两列的表格即可，如下

```
\starttabulate[|p|p|]
\NC A. 花非花 \NC B. 雾非雾 \NC\NR
\NC C. 夜半来 \NC D. 天明去 \NC\NR
\stoptabulate
```

6.8 样式设定

tabulate 环境的第 2 个参数可以设定表格的样式，例如上文设定过 distance 和 unit 的值。如果你的所有表格需要使用同一类样式，可以用 \setuptabulate 命令统一设定，无需每个表格都作设定。例如

```
\setuptabulate[distance=0mm, unit=0mm, rulethickness=4pt]
\starttabulate[|p|p|]
\HL
\NC A. 花非花 \NC B. 雾非雾 \NC\NR
\NC C. 夜半来 \NC D. 天明去 \NC\NR
\HL
\stoptabulate
```

A. 花非花	B. 雾非雾
C. 夜半来	D. 天明去

rulethickness 用于设定表格线条的粗度。

如果你不想让表格的样式设定影响所有表格，而是想让它仅仅影响某类表格，你可以用 \definetabulate 定义自己的表格类型，然后只为该类型的表格设定样式。例如

```
\definetabulate[Options][|p|p|]
\setuptabulate[Options][rulethickness=4pt]
\startOptions
\HL
\NC A. 花非花 \NC B. 雾非雾 \NC\NR
\NC C. 夜半来 \NC D. 天明去 \NC\NR
\HL
\stopOptions
```

A. 花非花	B. 雾非雾
C. 夜半来	D. 天明去

结语

本文并未介绍 tabulate 环境的一切用法，例如单元格跨列、表格分页以及前景和背景颜色设置等主题皆未涉及，如果你需要了解这部分功能，可阅读文档 [12]。tabulate 环境有很多局限性，例如你无法让单元格跨越多行，也难以让单元格的内容能在竖直方向居于单元格空间的中部。如果你遇到了它的这些局限，就意味着你应该学习 table 环境了，见文档 [13]。

7 插图

在很多情境下，一图胜千言。ConTEXt 在插图方面，除了支持常见的 JPEG，GIF 和 PNG 等位图格式，也支持 PDF 和 SVG 等矢量图格式，还支持 MetaPost 代码形式的内部图形——代码即插图。

7.1 外图

无论是位图还是矢量图，对于 ConTEXt 而言，都是外部图形，在文档中插入的方法是相同的，皆使用 \externalfigure 命令。假设源文档所在目录存在位图文件 foo.png，用以下代码便可将其插入文档，例如插入于此处：  。

```
\externalfigure[foo.png]
```

不过，这样的插图，很可能并非是你想要的插图形式。你想要的是，应该是独占一行且居中放置的插图。这个要求这并不难实现，见下例。

```
\midaligned{\externalfigure[foo.png]}
```

\externalfigure 有第 2 个参数，用于设定插图的宽度（width）和高度（hight），但通常只需设定一个，另一个可由 ConTEXt 根据图像的宽高比自动计算。下例将插图的宽度设定为正文宽度的 0.3 倍，其中 \textwidth 命令可获得正文版面宽度。

```
\midaligned{\externalfigure[foo.png][width=.3\textwidth]}
```

给插图加上标题也很容易，例如：

```
\midaligned{\externalfigure[07/foo.png][width=.3\textwidth]}
\midaligned{\tfx 这是插图的标题}
```

这是插图的标题

如果你想让插图标题能有序号，对于篇幅较小的文章，手工输入序号即可，见下例。建议在序号后，用 \quad 命令插入一个字宽的空白作为间隔，因为用普通的空格，只有半个字宽，效果不好。

```
\midaligned{\externalfigure[07/foo.png][width=.3\textwidth]}
\midaligned{\tfx 图 1\quad 这是插图的标题}
```

图 1　这是插图的标题

7.2 标题

倘若你担心插图太多，手工输入的插图序号难免会错乱，可以用 ConTEXt 的计数器功能，让序号自动递增。首先，需要用 \definenumber 命令定义一个计数器，给它取个名字，例如 myfig。

```
\definenumber[myfig]
```

一开始，计数器没有值，需要用 \incrementnumber 命令让它增 1，令其值为 1。要取得计数器的值，需要用 \getnumber 命令。例如以下代码的排版效果是在文档页面里显示数字 1 和 2。

```
\incrementnumber[myfig]\getnumber[myfig]  % 1
\incrementnumber[myfig]\getnumber[myfig]  % 2
```

知道上述计数器命令，便可为插图序号定义一个宏，每次从计数器取值，用作插图序号，然后将其增 1。在 **3.3** 节里，我们曾经定义过一个宏 \foo：

```
\def\foo{\hskip 0pt plus 2pt minus 0pt}
```

在文档里使用 \foo 时，ConTEXt 编译器会用这个宏的定义将其替换掉，亦即宏的使用，本质上是通过宏的名字引用其定义，这个过程通常称为**宏的展开**。下面用类似的办法，定义一个插图宏，以简化图片文件的插入过程。

```
\definenumber[myfignum] % 插图序号计数器

\def\myfigure#1#2{
  \midaligned{#2}
  \incrementnumber[myfignum]
  \midaligned{\tfx 图 \getnumber[myfignum]\quad #1}
}
```

\myfigure 宏和其他排版命令在形式上并无区别，后者也无非是 TEX 和 ConTEXt 开发者定义的宏而已。使用 \myfigure 需要向其提供两个参数，例如

```
\myfigure{插图的标题}{\externalfigure[foo.png]}
```

ConTEXt 编译器遇到上述语句，会自动将其替换为

```
\midaligned{\externalfigure[foo.png]}
\midaligned{\tfx 图 \getnumber[myfignum]\quad 插图的标题}
\incrementnumber[myfignum] % 插图序号增 1
```

下例演示了 \myfigure 的用法并给出了排版结果，插图和标题都是重复的，区别仅在于它们的序号。

```
\myfigure{标题}{\externalfigure[foo.png]}
\myfigure{标题}{\externalfigure[foo.png]}
```

图 7.1　标题

图 7.2　标题

不过，ConTEXt 已经为插图提供了像 \placetable 那样的命令，即 \placefigure。这个命令不仅能提供插图的序号，也能控制插图出现的位置，其用法与 \placetable 大致相似，可参考以下示例。

```
\placefigure[here][引用]{标题}
            {\externalfigure[foo.png][width=.3\textwidth]}
```

Figure 7.1　标题

类似表格，插图标题序号的前缀若要改为中文的「图」而非默认的「Figure」，需要使用 \mainlanguage[cn] 将界面切换为中文环境。插图标题的样式，也是通过 \setupcaption 设定，只是该命令的第 1 个参数是 figure。下例设定了符合中文排版惯例的插图标题样式。

```
\mainlanguage[cn]
\setupcaption[figure][headstyle=\rm, style=\tfx, align=center]
```

7.3　阵列

有时，为了节省页面空间，会将一组图片并排放置，形成一个阵列，该需求可基于 floatcombination 实现，例如以下代码可构造一行三列的插图阵列。

```
\startfloatcombination[nx=2,ny=1]
 \placefigure{}{}
 \placefigure{}{}
\stopfloatcombination
```

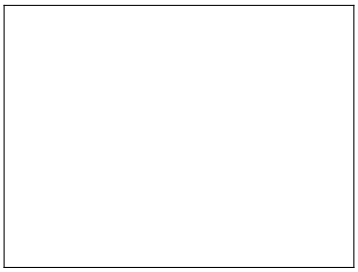


图 7.2

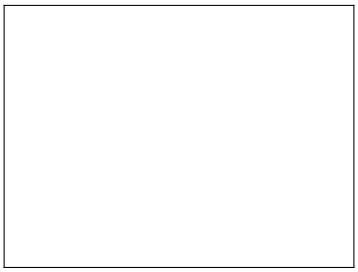


图 7.3

用 floatcombination 环境构造的阵列，类似于 \externalfigure，创造的都是行内对象，可以用 \midaligned 令其居中，也可以将其作为 \placefigure 命令中的插图对象，见下例。

```
\placefigure[force,none]{}{
  \startfloatcombination[nx=2,ny=1]
  \placefigure{}{} \placefigure{}{}
  \stopfloatcombination
}
```

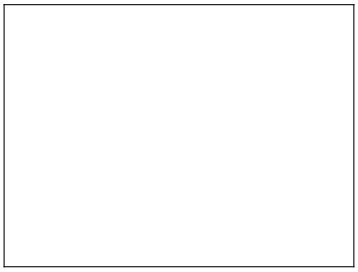


图 7.4

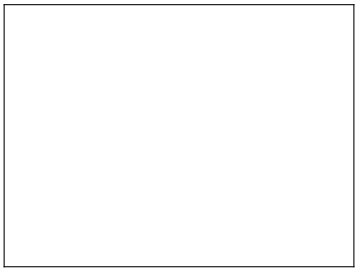


图 7.5

可以在 \placefigure 命令的第一个参数里用 nonumber 将插图标题的序号部分关闭，见下例，这个技巧可用于构造由一组子图构成的插图。

```
\placefigure[force]{}{
  \startfloatcombination[nx=2,ny=1]
  \placefigure[nonumber]{a\quad 子图}{} \placefigure[nonumber]{b\quad 子图}{}
  \stopfloatcombination
}
```

a 子图

b 子图

图 7.6

如果你从未听闻上述的 floatcombination 环境，单纯用你已掌握的 tabulate 环境和 \externalfigure 命令也能构造图片阵列，不妨一试。另外，在上述例子里，想必你已经观察到了，ConTEXt 的命令，当其参数里同时存在方括号形式和花括号形式时，对于前者而言，如果你不想设定它们，通常可以将其省略，亦即它们是**可选参数**。

7.4 路径

上文所有示例，插图所用的图片文件皆需与 ConTEXt 源文件位于同一目录。为了让目录中的文件更为整洁，可在源文件所在目录下建立专用于存放文档插图的子目录，例如 figures。为了让 ConTEXt 编译器在编译源文件时能够找到图片文件，可以在 \externalfigure 命令里设定图片文件的相对路径，例如

```
\externalfigure[figures/foo.png]
```

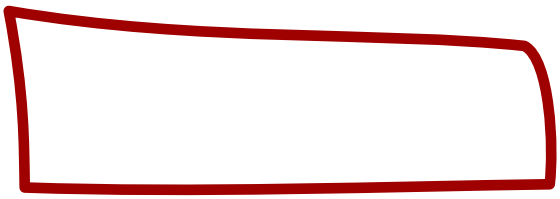
如果插图较多，不想每次都要重复输入 figures/，可以在文档的样式文件里设定图片文件所在目录，例如

```
\setupexternalfigures[directory=figures]
```

7.5 内图

还有一种插图，它是 MetaPost 绘图代码。这些代码可以嵌入在一些绘图环境里。之前在 **5.6** 节，已经见过了嵌入在 uniqueMPgraphic 环境里的 MetaPost 代码所画的正方形。下例，在另一个常用的绘图环境 useMPgraphic 里画一个扭曲的深红色矩形。

```
\startuseMPgraphic{foo}
path p; p := fullsquare randomized (.1, .3) xyscaled (7cm, 2cm);
draw p withpen pencircle scaled 4pt withcolor darkred;
\stopuseMPgraphic
\placefigure[here, nonumber]{\METAFUN\ 示例}{\useMPgraphic{foo}}
```



MetaFun 示例

此刻，也许你对 MetaPost 依然一无所知，但是，我还是希望我的介绍能够让你对它产生一些好奇。MetaPost 和 **5.7** 节用于构造带圈数字的 Lua 编程语言，二者可谓是现代 ConTEXt 飞天之双翼。

结语

所谓插图，不过是个头大一些的文字罢了，而所谓文字，不过是个头小一些的插图罢了。我忽然有些好奇，用表音文字的民族，如果他们的先祖没有发明出像插图那样的文字，他们的语言像是从天而降的……这不科学。

8 数学

TEX 原本是 Donald Knuth 为排版他的计算机算法专著而发明。他写的书属于数学类书籍，故而排版数学公式便成为 TEX 最擅长的任务——这也是我对数学的唯一兴趣了，希望你是因为喜欢数学而喜欢 TEX。近年来，ConTEXt 对 TEX 的数学公式的排版做出了许多显著的改进，俨然自成体系，限于能力，本章仅能略述一二。

8.1 TEX 风格

TEX 数学公式有两种模式，一种是正文模式（text mode），一种是显摆模式（display mode），在 ConTEXt 中，自然也是如此。当然，可能你觉得显摆模式这个称谓不够严肃，于是随了它的俗称「行间模式」，并无不可，而且后文我也如此称谓它。

正文模式里的数学公式放在一对美元符号之间，据我猜测，意思是有钱人都精通数学，或者精通数学的人都会有钱。例如 `$\int_0^{+\infty}f(x){\rm d}x$` 便是正文模式的数学公式，排版结果为 $\int_0^{+\infty}f(x)\mathrm{d}x$。TEX 数学公式的行间模式是放在一对双美元符号之间，例如

```
$$
\int_0^{+\infty}f(x) {\rm d}x
$$
```

排版结果为

$$\int_0^{+\infty}f(x)\mathrm{d}x$$

8.2 ConTEXt 风格

ConTEXt 的数学排版兼容 TEX 风格，但它另有一套功能更为丰富的语法。行内公式可以用 `\im{...}` 和 `\dm{...}` 命令，前者等效于 TEX 的行内公式，后者则是以行间公式的形式构造行内公式。这两个命令的区别，从排版一个分数便可察知，例如 `\im{\frac{1}{2}}` 的结果是 $\im{\frac{1}{2}}$，而 `\dm{\frac{1}{2}}` 的结果是 $\dm{\frac{1}{2}}$。

还有一个可带有参数的行内公式命令 `\m{...}`，例如 `\m[color=blue]{a^2 + b^2}` 可构造一个蓝色的行内公式，结果为 $\m[color=blue]{a^2 + b^2}$。

ConTEXt 的行间公式语法，其形式见下例：

```
\startformula
\int_0^{+\infty}f(x) {\rm d}x
\stopformula
```

$$\int_0^{+\infty}f(x)\mathrm{d}x$$

`formula` 环境为行间公式提供了居中对齐以及前后留白，且能在其样式设定里控制行间公式前后的空白大小。下例将公式前后的留白设为 0 长度。

```
\setupformulas[spacebefore=0pt,spaceafter=0pt]
\startformula
\int_0^{+\infty}f(x) {\rm d}x
\stopformula
```

$$\int_0^{+\infty} f(x)\,\mathrm{d}x$$

8.3 序号

\placeformula 可以让数学公式带有序号，且能使之支持引用，见下例。该命令有些像 \placetable 和 \placefigure，只是后两者皆为浮动对象，而数学公式不可浮动，也无需用花括号将其包裹起来作为 \placeformula 的参数。

```
\placeformula[formula-foo]
\startformula
\int_0^{+\infty}f(x) {\rm d}x
\stopformula
```

$$\int_0^{+\infty} f(x)\,\mathrm{d}x \tag{8.1}$$

用 \in[formula-foo] 命令可引用上例构造的数学公式的序号，结果为 **8.1**。需要注意的是，数学公式之后的第一个中文段落的首行缩进会被消除，若需缩进，可用 \indentation 命令强行缩进，见下例。

```
\placeformula
\startformula c = \pm\sqrt{a^2 + b^2} \stopformula
公式之后的段落... ... ...
\placeformula
\startformula f(x) = \sin x \stopformula
\indentation 公式之后的段落... ... ...
```

$$c = \pm\sqrt{a^2 + b^2} \tag{8.2}$$

公式之后的段落... ... ...

$$f(x) = \sin x \tag{8.3}$$

公式之后的段落... ... ...

请记住 \noindent 和 \indentation 这两个命令，在表格、插图以及数学公式之后的段落里，经常会用到它们。

由于 \placeformula 命令与 formula 环境的结合在形式上不够紧密，ConTEXt 提供了 placeformula 环境，它与 \placeformula 命令等效。此外，默认的公式序号是小括号形式，但是你可以将其设定为你需要的形式，见下例。

```
\setupformula[left={[}, right={]}]
\startplaceformula[在此放置引用]
\startformula
e = mc^2
\stopformula
\stopplaceformula
```

$$e = mc^2 \tag{8.4}$$

8.4 定理和证明

对于数学工作者而言，最重要的任务是发现数学定理，并给予证明。ConTEXt 没有为数学家们提供定理及其证明的排版命令，但是提供了实现这类命令的机制，即枚举。itemize 环境实际上可谓是枚举的特例。使用 \defineenumeration 可以定义新的枚举特例，例如

```
\defineenumeration[theorem][text=定理]
```

然后便可使用该特例，如下

```
\starttheorem
道可道也，非恒道也。
\stoptheorem
\starttheorem
名可名也，非恒名也。
\stoptheorem
```

定理 1

道可道也，非恒道也。

定理 2

名可名也，非恒名也。

例 8.1 定理环境

通过上例，可观察到，theorem 环境的序号会自动递增，这意味着，无论你发现了多少个数学定理，都可以反复使用该环境呈现它，只是其样式也许并非你想要的。

如果你希望定理的序号和内容的首行处于同一行，可将定理环境的样式设定为紧凑模式，见下例。

```
\setupenumeration[theorem][alternative=serried]
\starttheorem
道可道也，非恒道也。
\stoptheorem
```

定理 1 道可道也，非恒道也。

例 8.2 紧凑的定理环境

上例里的定理序号与内容之间的间距过大，只需对定理环境的宽度加以限制，便可消除该间距，见下例。

```
\setupenumeration[theorem]
                 [alternative=serried, width=broad]
\starttheorem 道可道也，非恒道也。 \stoptheorem
```

定理 1 道可道也，非恒道也。

例 8.3 限宽的定理环境

也可以将 width 设定为 \textwidth，与上例设定等效。至于为何会如此，以及 ConTEXt 为何不将这些设定作为默认设定呢？我不知道，也许 ConTEXt 开发者觉得，若一切都设定得太好，这是授人以鱼。

如果你希望定理的内容部分的字体也用粗体，可做以下设定：

```
\setupenumeration[theorem]
                 [alternative=serried, width=broad, style=bold]
```

同理，基于枚举环境也能为定理的证明定义一个环境，只需去掉序号部分，并在证明的结尾靠版面的右侧放置 □ 符号表示证毕，见下例。

```
\defineenumeration[proof][text=证明]
\setupenumeration[proof]
                 [alternative=serried, width=broad,
                  number=no,
                  closesymbol={\m{\square}}]
\startproof
因为名可名也，非恒名也，所以道可道也，非恒道也。
\stopproof
```

证明　因为名可名也，非恒名也，所以道可道也，非恒道也。 □

例 8.4　证明

结语

现在你已经基本学会了 ConTEXt 数学公式排版。与数学排版专家相比，你缺乏的可能主要是如何熟练地输入各种符号以及各种具体形式的数学公式。若想在这方面能有所精进，可参考文档 [14]。此外，在 ConTEXt 系统的文档目录里，有一份详尽的数学公式排版手册，使用以下命令可以找到它。

```
$ mtxrun --search mathincontext-screen.pdf
```

9 一字不差

在现在这个时代，文档出现一些计算机程序代码片段，是很常见的事。通常建议用等宽字体呈现这些代码且原样显示，像是过去打字机的输出结果，这种排版元素在西文里称为 Verbatim text，我将其译为「抄录」。

9.1 抄录

抄录有两种形式，一种是位于一行文字之内，用 \type{...} 排版，另一种是位于段落之间的 typing 环境。例如

```
\startframedtext[width=broad]
用 C 语言写一个程序，让它在屏幕上显示「\type{Hello world!}」，代码如下
\starttyping
#include <stdio.h>
int main(void) {
    printf("Hello world!\n");
    return 0;
}
\stoptyping
\stopframedtext
```

用 C 语言写一个程序，让它在屏幕上显示「Hello world!」，代码如下

```
#include <stdio.h>
int main(void) {
    printf("Hello world!\n");
    return 0;
}
```

上例不仅演示了抄录命令的基本用法，也演示了如何用 framedtext 环境构造边框。在 \setuptyping 命令的 before 和 after 参数允许你根据自己的需求，添加可被 typing 环境自动执行的排版命令。下例可以让 typing 环境自带边框。

```
\setuptyping[before={\startframedtext[width=broad]}, after={\stopframedtext}]
\starttyping
带外框的 \starttyping ... \stoptyping
\stoptyping
```

```
带外框的 \starttyping ... \stoptyping
```

\setuptyping 的 line 参数可用于设定代码行号。通过 \setuplinenumbering 可定制行号样式。下例开启代码行号，并将行号到 typing 环境的距离设为 0.5em：

```
\setuptyping
  [numbering=line,
   before={\startframedtext[width=broad]}, after={\stopframedtext}]
\setuplinenumbering[typing][distance=.5em]
\starttyping
#include <stdio.h>
int main(void) {
    printf("Hello world!\n");
    return 0;
}
\stoptyping
```

```
1  #include <stdio.h>
2  int main(void) {
3      printf("Hello world!\n");
4      return 0;
5  }
```

9.2 着色

ConTEXt 提供了代码语法着色功能，例如对 TEX 代码进行着色，

```
\starttyping[option=TEX]
\starttext
Hello \CONTEXT!
\stoptext
\stoptyping
```

```
1  \starttext
2  Hello \CONTEXT!
3  \stoptext
```

不幸的是，目前 ConTEXt 仅实现了 TEX，MetaPost，Lua，XML 等代码的着色。

9.3 逃逸区

利用 \type 和 typing 环境的逃逸（Escape）机制，也能实现代码着色。例如

```
\starttyping[escape=yes]
/BTEX\darkgreen \#include/ETEX <stdio.h>
/BTEX\darkblue int/ETEX main(/BTEX\darkblue void/ETEX) {
    printf("/BTEX\darkred Hello world!\n/ETEX");
    /BTEX\darkblue return/ETEX 0;
}
\stoptyping
```

```
#include <stdio.h>
int main(void) {
    printf("Hello world!n");
    return 0;
}
```

/BTEX.../ETEX 能够在抄录区域构造局部的 TEX 环境，从而允许你用一些 ConTEXt 排版命令制作一些特殊效果，这些命令不会被当成源码显示。

上述示例基于 TEX 逃逸区实现的代码着色胜在简单，失于繁琐，不过如果你熟悉某种编程语言，诸如 Python，Lua 或者 Awk 之类的脚本语言，你可以编写程序，自动生成这类着色代码，然后将其插入到文档里。

我应该一直都没有告诉你，如何给文字着色。ConTEXt 预定义了一些标准颜色，可直接使用这些颜色的名字对文字进行着色，例如「{\magenta 紫色}」，结果为「紫色」，也可以使用 \color 命令，例如「\color[lightmagenta]{浅紫色}」，结果为「浅紫色」。以下代码可用于查看 ConTEXt 预定义颜色，

```
\startTEXpage[offset=4pt]
\showcolor[rgb]
\stopTEXpage
```

使用 \definecolor 可以通过设定红（r）、绿（g）、蓝（b）分量定义颜色。例如

```
\definecolor[myred][r=.8,g=.2,b=.2]
\framed{\myred 给你点 color see see! }
```

给你点 color see see!

上例可以让你顺便学会另一种给文字增加外框的方法。也可以用 \colored 直接设定 rgb 颜色对文字着色，以下代码与上例等效：

```
\framed{\colored[r=.8,g=.2,b=.2]{给你点 color see see! }}
```

9.4 显示空格

ConTEXt 中文断行需要 \setscript[hanzi]，但该命令会吞噬汉字之间的空白字符，从而导致一个问题，在 \type 和 typing 环境中，汉字之间若存在空白字符，它们不会被输出到排版结果，此时，只有 space=on 可以救急。

下例是 typing 环境未开启空格显示的效果。

```
\starttyping
本 行 每 个 汉 字 之 后 都 有 空 格，但 是 你 看 不 见 它，除 非 space=on!
\stoptyping
```

```
本行每个汉字之后都有空格，但是你看不见它，除非 space=on!
```

作为比较，下例是开启了空格显示的效果。

```
\starttyping[space=on]
本 行 每 个 汉 字 之 后 都 有 空 格，你 能 看 见 它 。
\stoptyping
```

```
本␣行␣每␣个␣汉␣字␣之␣后␣都␣有␣空␣格，␣你␣能␣看␣见␣它␣。
```

9.5 样式设定

\setuptype 和 \setuptyping 可分别用于设置行内和行间抄录环境的样式。例如将行内抄录的颜色设为深红色，并打开空格显示：

```
\setuptype[color=darkred,space=on]
```

下例可将行间抄录的字号设为比正文字号小一号，并且前后留白各半行距离，且开启 TEX 逃逸。注意，\tt 是等宽字体切换命令，希望你还没有忘记 **3.5** 节所讲的那些字体切换命令。

```
\setuptyping[style=\tfx\tt,
             before={\blank[halfline]},
             after={\blank[halfline]}
             escape=yes]
```

使用 \definetype 和 \definetyping 可以定义专用的抄录环境，见下例。

```
\definetyping[foo][escape=yes,space=on,option=TEX]
\startfoo
Hello ConTEXt!
\stopfoo
```

\setuptype 和 \setuptyping 也能用于设定自定义的抄录环境，但是需要提供环境名，例如：

```
\setuptyping[foo][style=\tt\tfx]
```

结语

抄录环境还有许多功能，本文未予介绍，可以阅读文档 [15] 获得更全面的认识。面向印刷行业的抄录环境本身并不复杂，只是若是排版面向屏幕阅读的电子文档，代码着色能够更为直观地呈现代码结构，而 ConTEXt 在这方面存在很大欠缺。虽然它提供了扩展机制，但目前只能通过一些示例代码[16]去理解这该套机制。如果你要自行尝试为某种编程语言的代码编写着色器，不仅需要熟悉 Lua 语言，还需要熟悉 LPEG 库的用法。如果你对这个主题感兴趣，可以参考我所作的一些尝试[17,18]。

10 盒子

在 5.6 节中，你已经见过盒子了，只是那时可能你还不知其究竟，本章将揭开它们的一些端倪。也有可能你早已钻研过 Donald Knuth 的《The TEX Book》，对盒子的研究之深已经让我望风而拜，但是也许你未必熟悉 ConTEXt 的盒子，故而本章仍有部分内容值得一观。

10.1 TEX 盒子

之前的章节里，已多次暗示和明示，TEX 系统是 ConTEXt 的底层，二者的关系犹如引擎（发动机）和汽车的关系。对 TEX 引擎丝毫不懂，并不影响你学习和使用 ConTEXt 排版一份精致的文档。不过，懂得一些引擎层面工作原理，虽然貌似未必会有用处，但是实际上你并不能确定将来自己会不会成为一名 TEX 黑客，如同你从前也从未想过有一天会学习 ConTEXt。

在 TEX 系统中，盒子是很重要的事物。例如，在 ConTEXt 的排版的每一个段落，是一个竖向盒子，即 \vbox，该盒子之内又有一些横向盒子，即 \hbox，它们是段落的每一行。我们可以直接用这两种盒子构造一个不甚规整的段落：

```
\vbox{
  \hbox{离离原上草}\hbox{一岁一枯荣}
  \hbox{野火烧不尽}\hbox{春风吹又生}
}
```

离离原上草
一岁一枯荣
野火烧不尽
春风吹又生

例 10.1　竖向盒子和横向盒子

横向盒子可以指定它的长度，竖向盒子可以指定它的高度。例如，

```
\hbox to 5cm {赋得古原草送别}
\vbox to 3cm {
  \hbox{离离原上草}\hbox{一岁一枯荣}\hbox{野火烧不尽}\hbox{春风吹又生}
}
```

还有一种横向盒子 \line，其宽度是正文的宽度，该盒子内的文字会向两边伸展并与正文两侧边界对齐，例如

```
\line{\darkred\bf 我能吞下玻璃而不伤身体。}
```

我　能　吞　下　玻　璃　而　不　伤　身　体　。

看到上述示例，想必你想起了 4.2 节在设定文章标题的样式时，汉字之间的粘连被触发后的样子，与\line 的效果非常相似。使用 \hfill 或 \hss 可对横向盒子里的内容进行挤压。例如

```
\line{\hfill 我能吞下玻璃而不伤身体。}
\line{我能吞下玻璃而不伤身体。\hfill}
\line{\hfill 我能吞下玻璃而不伤身体。\hfill}
\line{\hss 我能吞下玻璃而不伤身体。}
\line{我能吞下玻璃而不伤身体。\hss}
\line{\hss 我能吞下玻璃而不伤身体。\hss}
```

我能吞下玻璃而不伤身体。

我能吞下玻璃而不伤身体。

我能吞下玻璃而不伤身体。

我能吞下玻璃而不伤身体。

我能吞下玻璃而不伤身体。

我能吞下玻璃而不伤身体。

`\hfill` 和 `\hss`，都是可无限伸缩的粘连，还有一个伸缩能力弱于 `\hfill` 的 `\hfil`。竖向的可无限伸缩的粘连有 `\vfil`，`\vfill` 和 `\vss`。

10.2 ConTEXt 盒子

在 ConTEXt 层面，通常很少使用 TEX 盒子，而是使用 `\inframed` 和 `\framed`——前者是后者的特例。与 TEX 盒子相比，ConTEXt 层面的盒子可以显示边框，且有非常多的参数可以定制它们的外观。

`\inframed` 用于正文，可用于给一行文字增加边框，例如

```
\inframed{\type{\inframed{...}}}
```

结果为 `\inframed{...}`。倘若使用 `\framed`，例如

```
\framed{\type{\framed{...}}}
```

结果为 `\framed{...}`。可以发现，`\inframed` 更适合在正文中使用，因为它能与文字基线对齐。事实上，`\inframed` 与 `\framed[location=low]` 等效，故而前者是后者的特例。例如

```
\framed[location=low]{\type{\framed[location=low]{...}}}
```

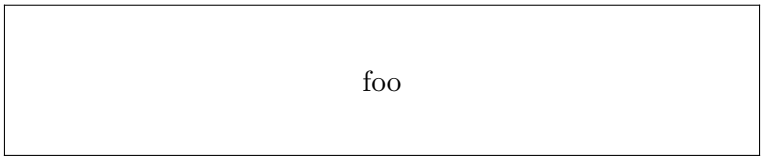
结果为 `\framed[location=low]{...}`。

如果不希望 `\framed` 显示边框，只需 `\framed[frame=off]{...}`，也可以单独显示某条边线，并设定边线粗度和颜色：

```
\line{
  \framed[frame=off,leftframe=on,rulethickness=4pt,framecolor=red]{foo}
  \framed[frame=off,topframe=on,rulethickness=4pt,framecolor=green]{foo}
  \framed[frame=off,rightframe=on,rulethickness=4pt,framecolor=blue]{foo}
  \framed[frame=off,bottomframe=on,rulethickness=4pt,framecolor=magenta]{foo}
}
```

可以设定盒子的宽度和高度，例如宽 10cm，高 2 cm 的盒子：

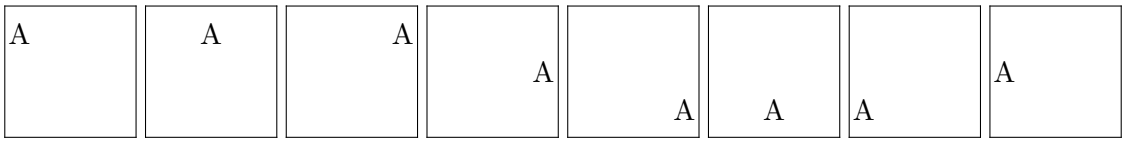
```
\hbox to \textwidth{\hfill\framed[width=10cm,height=2cm]{foo}\hfill}
```



10.3 对齐

ConTEXt 盒子的内容默认居中，即 `align=center`，此外还有 8 种对齐方式：

```
\line{
  \setupframed[width=1.75cm,height=1.75cm]
  \framed[align={flushleft,high}]{A}
  \framed[align={middle,high}]{A}
  \framed[align={flushright,high}]{A}
  \framed[align={flushright,lohi}]{A}
  \framed[align={flushright,low}]{A}
  \framed[align={middle,low}]{A}
  \framed[align={flushleft,low}]{A}
  \framed[align={flushleft,lohi}]{A}
}
```



注意，上述代码中的 `\setupframed` 命令可以设定 `\framed` 盒子的样式，所作设定会影响到该命令之后的所有 `\framed` 盒子，但是，如果是在编组内设定盒子样式，所作设定不会影响编组之外的盒子。上述代码中的 `\line` 命令之后跟随的便是编组。

10.4 背景

可将颜色作为 `\framed` 的背景。例如

```
\inframed
  [background=color,
   backgroundcolor=lightgray,
   width=2cm,
   frame=off]{\bf foo}
```

结果为 **foo** 。

通过 overlay，可将一些排版元素作为 \framed 的背景。例如

```
\defineoverlay[foo][{\framed[width=3cm,frame=off,bottomframe=on]{}}]
\midaligned{\inframed[background=foo,frame=off]{你好啊！}}
```

你好啊！

也能在 overlay 里插入 MetaPost 代码绘制的矢量图形，将其作为盒子的背景，例如

```
\startuseMPgraphic{foo}
path p;
p := fullcircle scaled OverlayWidth;
draw p withpen pencircle scaled .4pt withcolor darkred;
\stopuseMPgraphic
\defineoverlay[circle][\useMPgraphic{foo}]
\def\fooframe#1{%
  \inframed[frame=off,background=circle]{#1}%
}
```

上例定义了一个宏 \fooframe，它能为盒子里的文字套上一个圆，例如 \fooframe{123}，结果为 12345，还记得 **5.7** 节实现带圈数字的方案吗？现在又多了一个方案，而且该方案不依赖于所用字体是否提供带圈字符。

10.5 盒子的深度

如果你仔细观察，ConTEXt 盒子里的内容在水平方向是精确居中的，但是在竖直方向上却并非如此。例如

```
\inframed{\framed{我看到一棵樱桃树}}。
```

结果为 我看到一棵樱桃树 。可见 \inframed 内部的 \framed 的底部有着看似多余的空白。使用盒子的参数 depth 可以消除这些空白，但问题在于这处空白从何而来及其高度是多少。该问题与底层 TEX 的西文排版机制有关。

无论是 TEX 还是 ConTEXt 盒子，它们本身是没有深度的，但是当它们里面的文字或盒子有深度时，它们便有了深度。至于深度值具体是多大，可以借助 TEX 的盒子寄存器进行测量。例如，定义一个盒子寄存器 box0：

```
\setbox0\hbox{\inframed{\framed{我看到一棵樱桃树}}}
```

现在我们有了一个 0 号盒子，使用 \wd，\ht 和 \dp 可分别测量该盒子的宽度、高度和深度……顺便复习一下表格的用法：

```
\starttabulate[|c|c|c|]
\TL[3]
\NC 宽度 \NC 高度 \NC 深度 \NR
\HL
\NC \the\wd0 \NC \the\ht0 \NC \the\dp0 \NR
\BL[3]
\stoptabulate
```

宽度	高度	深度
94.39331pt	12.9421pt	5.03302pt

将上述所得深度信息取负作为 \inframed 的参数 depth 的值，即

```
\inframed[depth=-\dp0]{\framed{我看到一棵樱桃树}}
```

便可消除深度，结果为 

。

10.6 段落盒子

上文的示例，盒子里的内容都非常简单。事实上，\framed 能够容纳段落，只是默认情况下，它不具备段落断行功能，需要将其参数 align 的值设为 normal 方能断行，见下例。

```
\framed[width=6cm, align=normal]{%
第一段... ... ...\par
第二段... ... ...
}
```

第一段... ... ...
第二段... ... ...

注意，上述代码中，\framed 的 {后面的注释符 % 是必要的，否则花括号后的换行符会被视为一个不可忽略的空格，从而导致第一段开头是一个空格，请以下例作为对比。

```
\framed[width=6cm, align=normal]{
第一段... ... ...\par
第二段... ... ...
}
```

第一段... ... ...
第二段... ... ...

由于 \framed 用于容纳段落需要注意一些细节，故而对于此类任务，通常用语法形式封闭的 framedtext 环境来做，例如

```
\startframedtext[width=\textwidth, rulethickness=4pt, framecolor=darkgray]
第一段... ... ...\par
第二段... ... ...
\stopframedtext
```

第一段... ... ...
第二段... ... ...

10.7 自定义盒子

类似于 \type 和 typing 环境支持用户自定义一些专用的命令，\framed 盒子也可如此。下例定义了一个专用的盒子，并演示了其样式的设定方法。

```
\defineframed[mybox]
\setupframed[mybox][rulethickness=4pt,framecolor=darkred]
\mybox{自定义盒子}
```

自定义盒子

类似地，framedtext 环境也支持定义专用的段落盒子，例如

```
\defineframedtext[bluebox]
\setupframedtext[bluebox][width=\textwidth, rulethickness=4pt,framecolor=blue]
\startbluebox
自定义盒子
\stopbluebox
```

自定义盒子

结语

TEX 盒子是无形的。ConTEXt 盒子是有形的。老子曾说过，恒无欲，以观其妙；恒有欲，以观其所徼。故而，TEX 要懂一些，ConTEXt 也要懂一些。

11 作图

想必你已迫不及待想学习 MetaPost 了，这大概是来自人类上古基因的冲动。人类先学会的是绘画，而后才是文字。不要妄图通过这区区一章内容掌握 MetaPost，该期望需要有人写一本至少三百多页的书方能满足。不过，本章内容足以给你打开一扇窗户，让 MetaPost 的优雅气息拂过时常过于严肃的 ConTEXt 世界。

11.1 作图环境

MetaPost 是一种计算机作图语言，与 TEX 一样，皆为宏编程语言。ConTEXt 为 MetaPost 代码提供了五种环境：

```
\startMPcode ... \stopMPcode
\startMPpage ... \stopMPpage
\startuseMPgraphic{name} ... \stopuseMPgraphic
\startuniqueMPgraphic{name} ... \stopuniqueMPgraphic
\startreusableMPgraphic{name} ... \stopreusableMPgraphic
```

第一种环境用于临时作图，生成的图形会被插入到代码所在位置。第二种环境是生成单独的图形文件，以作其他用途。后面三种环境，生成的图形可根据环境的名称作为文章插图随处使用，区别仅在于应用场景，如下：

- `useMPgraphic`：每被使用一次，MetaPost 代码便会被重新编译一次。
- `uniqueMPgraphic`：只要图形所处环境不变，MetaPost 代码只会被编译一次。
- `reusableMPgraphic`：无论如何使用，MetaPost 代码只会被编译一次。

大多数情况下，建议选用 `uniqueMPgraphic` 环境，但若图形中存在一些需要每次使用时都要有所变化的内容，可选用 `useMPgraphic` 环境。

需要注意的是，在 ConTEXt 中使用 MetaPost 时，通常会使用 ConTEXt 定义的一些 MetaPost 宏，这些宏构成的集合，名曰 MetaFun。

11.2 画一个盒子

MetaPost 作图语句遵守基本的英文语法，理解起来颇为简单。下例，用粗度为 2 pt 的圆头笔用暗红色绘制一条经过四个点的封闭路径。

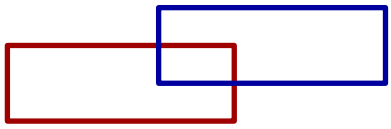
```
\startMPcode
pickup pencircle scaled 2pt;
draw (0, 0) -- (3cm, 0) -- (3cm, 1cm)
     -- (0, 1cm) -- cycle withcolor darkred;
\stopMPcode
```

上述代码中，(0, 0) -- ... -- cycle 构造的是一条封闭路径，可将其保存于路径变量：

```
path p;
p := (0, 0) -- (3cm, 0) -- (3cm, 1cm) -- (0, 1cm) -- cycle;
pickup pencircle scaled 2pt;
draw p withcolor darkred;
```

将路径保存在变量中，是为了更便于对路径进行一些运算，例如

```
\startMPcode
path p;
p := (0, 0) -- (3cm, 0)
      -- (3cm, 1cm) -- (0, 1cm) -- cycle;
pickup pencircle scaled 2pt;
draw p withcolor darkred;
draw p shifted (2cm, .5cm) withcolor darkblue;
\stopMPcode
```



路径 p 被向右平移了 2 cm，继而被向上平移动了 0.5 cm。

还有一种构造矩形路径的方法：先构造一个单位正方形，然后对其缩放。例如

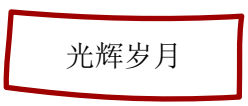
```
\startMPcode
pickup pencircle scaled 2pt;
draw fullsquare xscaled 3cm yscaled 1cm
                withcolor darkred;
\stopMPcode
```

MetaFun 宏 randomized 可用于对路径随机扰动。例如，对一个宽为 3 cm，高为 1cm 的矩形路径以幅度 2mm 的程度予以扰动：

```
\startuseMPgraphic{随机晃动的矩形}
pickup pencircle scaled 2pt;
draw (fullsquare xscaled 3cm yscaled 1cm)
                 randomized 2mm withcolor darkred;
\stopuseMPgraphic
\useMPgraphic{随机晃动的矩形}
```

还记得 overlay 吗？只要将上述 useMPgraphic 环境构造的图形制作为 overlay，便可将其作为 \framed 的背景，从而可以得到一种外观颇为别致的盒子。

```
\defineoverlay[晃晃][\useMPgraphic{随机晃动的矩形}]
\framed[frame=off,background=晃晃,width=3cm]{光辉岁月}
```



在 ConTEXt 为 MetaPost 提供的作图环境里，可分别通过 \overlaywidth 和 \overlayheight 获得 overlay 的宽度和高度。在将 overlay 作为 \framed 的背景

时，\framed 的宽度和高度便是 overlay 的宽度和高度。基于这一特性，便可实现 MetaPost 绘制的图形能够自动适应 \framed 的宽度和高度的变化。例如

```
\startuseMPgraphic{新的随机晃动的矩形}
path p;
p := fullsquare xscaled \overlaywidth yscaled \overlayheight;
pickup pencircle scaled 2pt;
draw p randomized 2mm withcolor darkred;
\stopuseMPgraphic

\defineoverlay[新的晃晃][\useMPgraphic{新的随机晃动的矩形}]
\framed[frame=off,background=新的晃晃]
       {今天只有残留的躯壳，迎接光辉岁月，风雨中抱紧自由。}
```

今天只有残留的躯壳，迎接光辉岁月，风雨中抱紧自由。

对于需要重复使用的盒子，为了避免每次重复设置其样式，可以将它定义为专用盒子。例如

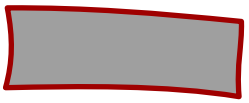
```
\defineframed[funnybox][frame=off,background=新的晃晃]
\funnybox{今天只有残留的躯壳，迎接光辉岁月，风雨中抱紧自由。}
```

MetaPost 可以为一条封闭路径填充颜色。在此需要明确，何为封闭路径。例如

```
path p, q, r;
p := (0, 0) -- (1, 0) -- (1, 1) -- (0, 0) -- (0, 0);
q := (0, 0) -- (1, 0) -- (1, 1) -- (0, 0) -- cycle;
r := fullsquare;
```

其中路径 p 的终点的坐标恰好是其起点，但它并非封闭路径，而路径 q 和 r 皆为封闭路径。下面示例，为封闭路径填充颜色：

```
\startMPcode
path p;
p := (fullsquare xscaled 3cm yscaled 1cm) randomized 2mm;
pickup pencircle scaled 2pt;
fill p withcolor darkgray;
draw p withcolor darkred;
\stopMPcode
```



注意，对于封闭路径，应当先填充颜色，再绘制路径，否则所填充的颜色会覆盖一部分路径线条。

11.3 颜色

MetaPost 以含有三个分量的向量表示颜色。向量的三个分量分别表示红色、绿色和蓝色，取值范围为 [0, 1]，例如 `(0.4, 0.5, 0.6)`。可将颜色保存到 color 类型的变量中，以备绘图中重复使用。例如定义一个值为暗红色的颜色变量：

```
color foo;
foo := (0.3, 0, 0);
```

由于 METAPOST 内部已经定义了用于表示红色的变量 `red`，因此 `foo` 的定义也可写为

```
color foo;
foo := 0.3 * red;
```

小于 1 的倍数，可以忽略前缀 0，且可以直接作用于颜色：

```
foo := .3red;
```

使用 `transparent` 宏可用于构造带有透明度的颜色值。例如

```
\startMPcode
path p; p := fullsquare scaled 1cm;
color foo; foo := .3red;
pickup pencircle scaled 4pt;
draw p withcolor transparent (1, 0.3, foo);
draw p shifted (.5cm, .5cm) withcolor transparent (1, 0.25, blue);
\stopMPcode
```

`transparent` 的第一个参数表示选用的颜色透明方法，共有 12 种方法可选：

1. normal
2. multiply
3. screen
4. overlay
5. softlight
6. hardlight
7. colordodge
8. colorburn
9. darken
10. lighten
11. difference
12. exclusion

第二个参数表示透明度，取值范围 [0, 1]，其值越大，透明程度越低。第三个参数为颜色值。需要注意的是，MetaPost 并不支持以 `color` 类型的变量保存带透明度的颜色值，而且 MetaPost 里也没有与之对应的变量类型。

11.4 文字

使用 MetaFun 宏 `textext` 可在 MetaPost 图形中插入文字，且基于 MetaPost 图形变换命令可对文字进行定位、缩放、旋转。例如

```
\startMPcode
string s; % 字符串类型变量
s = "\color[darkred]{\bf 江山如此多娇}";
draw textext(s);
draw textext(s) shifted (4cm, 0);
draw textext(s) scaled 1.5 shifted (8cm, 0) ;
draw textext(s) scaled 1.5 rotated 45 shifted (12cm, 0);
\stopMPcode
```

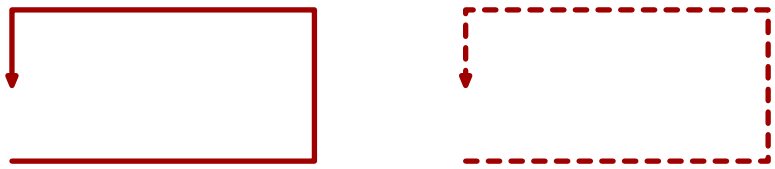
也可使用 thetextext 宏直接对文字进行定位，从而可省去 shifted 变换。例如

```
\startMPcode
string s; s = "{\bf 江山如此多娇}";
draw (0, 0) withpen pensquare scaled 11pt withcolor darkred;
draw thetextext(s, (4cm, 0)) withcolor darkred;
\stopMPcode
```

11.5 方向路径

MetaPost 宏 drawarrow 可绘制带箭头的路径。例如

```
\startMPcode
path p; p := (0, 0) -- (4cm, 0) -- (4cm, 2cm) -- (0, 2cm) -- (0, 1cm);
pickup pencircle scaled 2pt;
drawarrow p withcolor darkred;
drawarrow p shifted (6cm, 0) dashed (evenly scaled .5mm) withcolor darkred;
\stopMPcode
```



上述代码也给出了虚线路径的画法。

MetaPost 的任何一条路径，从起点到终点可基于取值范围为 [0, 1] 的参数选择该路径上的某一点。基于该功能可实现路径标注。例如选择路径参数 0.5 对应的点，在该点右侧放置 ConTEXt 旋转 90 度的文字：

```
\startMPcode
path p; p := (0, 0) -- (4cm, 0)
             -- (4cm, 2cm) -- (0, 2cm) -- (0, 1cm);
pair pos; pos := point .5 along p;
pickup pencircle scaled 2pt;
drawarrow p withcolor darkred;
draw pos withpen pensquare scaled 4pt
         withcolor darkgreen;
draw thetextext.rt("\rotate[rotation=-90]{路过}",
                   pos shifted (1mm, 0));
\stopMPcode
```

上述代码中出现了 thetextext 的后缀形式。除了默认形式，thetextext 还有 4 种后缀形式，后缀名为 .lft，.top，.rt 和 .bot，分别表示将文字放在指定位置的左侧、上方、右侧和下方。

11.6 画面

MetaPost 有一种变量类型 picture，可将其用于将一组绘图语句合并为一个图形，然后予以绘制。使用 MetaPost 宏 image 可构造 picture 实例。例如

```
\startMPcode
path a; a := fullsquare xscaled 4cm yscaled 1cm;
picture p; p := image(
  fill a withcolor darkgray;
  draw a withpen pencircle scaled 2pt withcolor darkblue;
);
draw p;
\stopMPcode
```

使用 MetaPost 宏 center 可以获得 picture 实例的中心坐标，结果可保存于一个 pair 类型的变量。例如

```
\startMPcode
picture p; p := image(draw textext("密云不雨，自我西郊"););
pair c; c := center p;
draw c withpen pensquare scaled 4pt withcolor darkred;
draw p withcolor darkblue;
\stopMPcode
```

密云不雨，自我西郊

使用 MetaFun 宏 bbwidth 和 bbheight 可以获得 picture 实例的宽度和高度。使用这两个宏，可为任何图形和文字构造边框。例如

```
\startMPcode
picture p; p := image(draw textext("归妹愆期，迟归有时"));
numeric w, h; w := bbwidth(p); h := bbheight(p);
path q; q := fullsquare xscaled w yscaled h;
fill q withcolor darkgray;
draw q withpen pencircle scaled 2pt withcolor darkred;
draw p;
\stopMPcode
```

归妹愆期，迟归有时

上述实例为文字构造的边框太紧了，利用现有所学，让它宽松一些并不困难：

```
\startMPcode
picture p; p := image(draw textext("归 妹 愆 期 ， 迟 归 有
时"));
numeric w, h; w := bbwidth(p); h := bbheight(p);
numeric offset; offset := 5mm;
path q;
q := fullsquare xscaled (w + offset)
                yscaled (h + offset)
                shifted center p;
fill q withcolor darkgray;
draw q withpen pencircle scaled 2pt withcolor darkred;
draw p;
\stopMPcode
```

归妹愆期，迟归有时

11.7 宏

定义一个宏，令其接受一个字符串类型的参数，返回一个矩形框，并令文字居于矩形框中心：

```
\startMPcode
vardef framed (expr text, offset) =
  picture p; p := image(draw textext(text));
  numeric w, h; w := bbwidth(p); h := bbheight(p);
  path q;
  q := fullsquare xscaled (w + offset) yscaled (h + offset) shifted
center p;
  image(fill q withcolor lightgray;
        draw q withpen pencircle scaled 2pt withcolor darkred;
        draw p;)
enddef;
draw framed("{\bf 亢龙有悔}", 5mm);
\stopMPcode
```

亢龙有悔

MetaPost 的 vardef 用于定义一个有返回值的宏，宏定义的最后一条语句即返回值，该条语句不可以分号作为结尾。MetaPost 还有其他几种宏定义形式，但是对于大多数作图任务而言，vardef 已足够应付。

11.8 简单的流程图

现在，请跟随我敲击键盘的手指，逐步画一幅描述数字求和过程的流程图，希望这次旅程能让你对 MetaPost 的基本语法有一些全面的认识。首先，构造一个结点，表示数据输入。

```
string f; f := "\framed[frame=off,align=center]";
picture a;
a := image(
  % 符号 & 用于拼接两个字符串
  draw textext(f & "{$i\leftarrow 1$\\$s\leftarrow 0$}");
);
```

然后，用 **11.7** 节定义的 framed 宏构造两个运算过程结点：

```
numeric offset; offset := 5mm;
picture b; b := framed(f & "{$s\leftarrow s + i$}", offset);
picture c; c := framed(f & "{$i\leftarrow i + 1$}", offset);
```

需要定义一个宏，用它构造菱形的条件判断结点：

```
vardef diamond (expr text, offset) =
  picture p; p := image(draw textext(text));
  numeric w, h; w := bbwidth(p); h := bbheight(p);
  path q;
  q := fulldiamond xscaled (w + offset) yscaled (h + offset) shifted center p;
  image(fill q withcolor darkgray;
        draw q withpen pencircle scaled 2pt withcolor darkred;
        draw p withcolor white;)
enddef;
picture d; d := diamond(f & "{$i > 100$}", 3 * offset);
```

再构造一个结点表示程序输出：

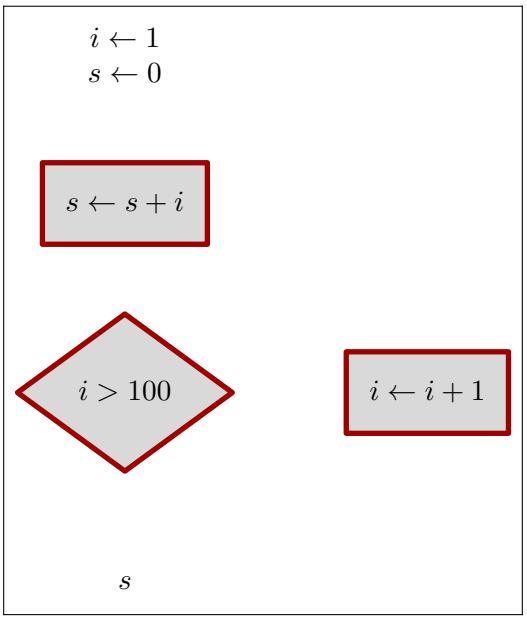
```
picture e;
e := image(draw textext(f & "{$s$}"););
```

保持结点 a 不动，对 b，c，d 和 e 进行定位：

```
b := b shifted (0, -2cm);
c := c shifted (4cm, -4.5cm);
d := d shifted (0, -4.5cm);
e := e shifted (0, -7cm);
```

画出所有结点：

```
draw a; draw b; draw c; draw d; draw e;
```



构造节点 a 的底部中点到 b 的顶部中点的路径：

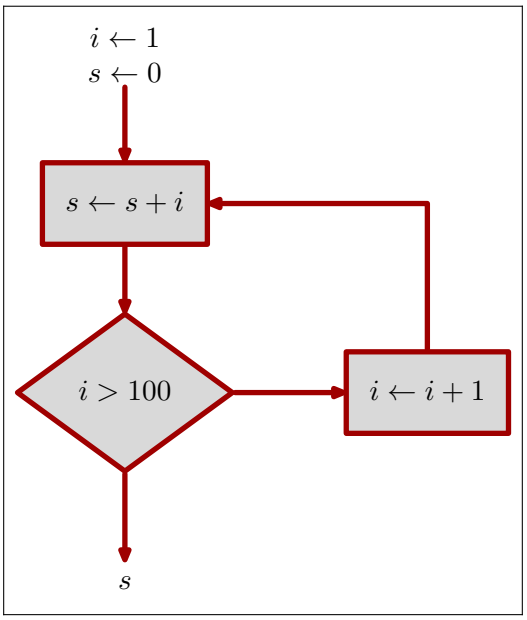
```
path ab;
ab := .5[llcorner a, lrcorner a] -- .5[ulcorner b, urcorner b];
```

其中 llcorner 用于获取路径或画面实例的最小包围盒的左下角顶点坐标。同理，ulcorner，urcorner 和 lrcorner 分别获取包围盒的左上角、右上角和右下角顶点坐标。.5[..., ...] 用于计算两个点连线的中点。用类似的方法可以构造其他连接各节点的路径：

```
path bd;
bd := .5[llcorner b, lrcorner b] -- .5[ulcorner d, urcorner d];
path dc;
dc := .5[lrcorner d, urcorner d] -- .5[ulcorner c, llcorner c];
path cb;
cb := .5[ulcorner c, urcorner c] -- (4cm, -2cm) -- .5[urcorner b, lrcorner b];
path de;
de := .5[llcorner d, lrcorner d] -- .5[ulcorner e, urcorner e];
```

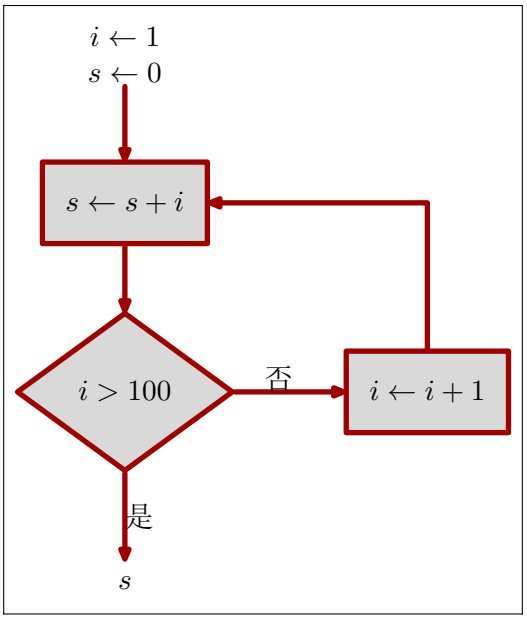
画出所有路径：

```
drawarrow ab withcolor darkred; drawarrow bd withcolor darkred;
drawarrow cb withcolor darkred; drawarrow dc withcolor darkred;
drawarrow de withcolor darkred;
```



最后一步，标注路径：

```
pair no;  no  := point .4 along dc;
pair yes; yes := point .5 along de;
draw thetextext.top("否", no);
draw thetextext.rt("是", yes);
```



11.9 代码简化

11.8 节的代码存在较多重复。可以用条件、循环，宏等形式予以简化。不过，我对它们给出的讲解并不会细致，为的正是走马观花，观其大略。

首先，观察宏 framed 和 diamond 的定义，发现二者仅有的不同是前者用 fullsquare 绘制盒子，后者用 fulldiamond。以重新定义一个更为灵活的宏，用于制作节点：

```
vardef make_node(expr text, shape, offset) =
  picture p; p := image(draw textext(text));
  numeric w, h; w := bbwidth(p); h := bbheight(p);
  if path shape:
    path q;
    q := shape xysized (w + offset, h + offset) shifted center p;
    image(fill q withcolor lightgray;
          draw q withpen pencircle scaled 2pt withcolor darkred;
          draw p;)
  else:
    image(draw p;)
  fi
enddef;
```

由于上述代码使用了 MetaPost 的条件判断语法，以 path shape 判断 shape 是否为路径变量，从而使得 make_node 能构用于构造有无边框和有边框的节点：

```
numeric offset; offset := 5mm;
string f; f := "\framed[frame=off,align=center]";
picture a, b, c, d, e;
a := make_node(f & "{$i\leftarrow 1$\\$s\leftarrow 0$}", none, 0);
b := make_node(f & "{$s\leftarrow s + i$}", fullsquare, offset);
c := make_node(f & "{$i\leftarrow i + 1$}", fullsquare, offset);
d := make_node(f & "{$i > 100$}", fulldiamond, 3 * offset);
e := make_node(f & "{$s$}", none, 0);
```

在 vardef 宏中使用条件语句时需要注意，通常情况下不要条件结束语句 fi 后面添加分号，否则 vardef 宏的返回值会带上这个分号。在其他情境下，通常需要在 fi 加分号。还要注意，我在 make_node 宏中使用 xysized 取代了之前的 xscale 和 yscale，可直接指定路径或画面的尺寸。之所以如此，是因为我们无法确定 make_node 宏的第 2 个参数对应的路径是否为标准图形。

在绘制节点和路径时，存在重复使用 drawarrow 语句的情况，例如

```
drawarrow ab withcolor darkred; drawarrow bd withcolor darkred;
drawarrow cb withcolor darkred; drawarrow dc withcolor darkred;
drawarrow de withcolor darkred;
```

可使用循环语句予以简化：

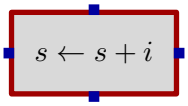
```
for i = ab, bd, cb, dc, de: drawarrow i withcolor darkred; endfor;
```

为了便于获得路径或画面的包围盒的四边中点，定义以下宏：

```
vardef left(expr p) = .5[llcorner p, ulcorner p] enddef;
vardef top(expr p) = .5[ulcorner p, urcorner p] enddef;
vardef right(expr p) = .5[lrcorner p, urcorner p] enddef;
vardef bottom(expr p) = .5[llcorner p, lrcorner p] enddef;
```

以绘制节点 b 的四边中点为测试用例

```
for i = left(b), top(b), right(b), bottom(b):
  draw i withpen pensquare scaled 4pt withcolor darkblue;
endfor;
```



基于上述宏，可以更为简洁地构造连接各节点的路径：

```
path ab; ab := bottom(a) -- top(b);
path bd; bd := bottom(b) -- top(d);
path dc; dc := right(d) -- left(c);
path cb; cb := top(c) -- (xpart center c, ypart center b) -- right(b);
path de; de := bottom(d) -- top(e);
```

center 是 MetaPost 宏，可用于获取路径或画面的包围盒中心点坐标。xpart 和 ypart 也皆为 MetaPost 宏，用于获取点的坐标分量。

路径标注也可以通过定义一个宏予以简化：

```
vardef tag(expr p, text, pos, loc) =
  if loc = "left": thetextext.lft(text, point pos along p)
  elseif loc = "right": thetextext.rt(text, point pos along p)
  elseif loc = "top": thetextext.top(text, point pos along p)
  elseif loc = "bottom": thetextext.bot(text, point pos along p)
  else thetextext(text, point pos along p)
  fi
enddef;
```

其用法为

```
draw tag(dc, "否", .5, "top"); draw tag(de, "是", .5, "right");
```

11.10 层叠

ConTEXt 有一个以 overlay 为基础的层（Layer）机制。利用层机制，我们可将 MetaPost 图形绘制在页面上的任何一个位置。在学习层之前，我们需要对 overlay 的认识再加深一些。

overlay，词义是「覆盖物」，你可以将其理解为图层。所谓 overlay，实际上不是它覆盖别的元素，而是让别的元素覆盖它，亦即你可以将一些文字置于 overlay 对象之上，故而 overlay 常作为一些排版元素的背景存在。在 **10.4** 节里，已经见过了基于 overlay 构造带圆圈文本的方法，在例子里，用 MetaPost 代码画了一个直径与 overlay 宽度相同的圆，并将其作为 \framed 的背景。事实上，\framed 的背景能够支持多个 overlay 的叠加，见下例。

```
\startuseMPgraphic{一个矩形}
path p; p := fullsquare xyscaled (OverlayWidth, OverlayHeight);
% 定义颜色变量，其 r、g、b 三个成分可被随机扰动
color c; c := white randomized (.7, .7, .7);
draw p randomized 5mm
       withcolor transparent (1, .5 randomized .25, c)
       withpen pencircle scaled 2pt;
\stopuseMPgraphic
\defineoverlay[叠叠-1][\useMPgraphic{一个矩形}]
\defineoverlay[叠叠-2][\useMPgraphic{一个矩形}]
\defineoverlay[叠叠-3][\useMPgraphic{一个矩形}]

\defineframed
  [foo][frame=off, background={叠叠-1, 叠叠-2, 叠叠-3}]
\foo{迎接光辉岁月}
```

迎接光辉岁月

需要注意的是，当 overlay 作为 \framed 之类的盒子的背景时，盒子的尺寸会传递给 overlay，而 MetaPost 代码里，也可以通过 MetaFun 的内置变量 OverlayWidth 和 OverlayHeight 获得 overlay 的宽度和高度，亦即 ConTEXt 里的 MetaPost 代码与 ConTEXt 排版环境存在数据传递机制，该机制使得 MetaPost 图形适配排版元素的尺寸成为可能。

层本质上是一个可作为全页背景的 overlay，可使用绝对坐标或相对坐标对排版元素在页面上定位放置。下例定义了一个层 foo，并在三个不同位置分别放置一个随机矩形——**11.2** 节中「随机晃动的矩形」。

```
\definelayer[foo]
\setlayer[foo][x=0cm,y=0cm]{\useMPgraphic{square}}
\setlayer[foo][x=6cm,y=1cm]{\useMPgraphic{square}}
\setlayer[foo][x=\textwidth,y=2cm,hoffset=-4cm,vhoffset=-.5cm]{\useMPgraphic{square}
\flushlayer[foo]
```

上述代码定义的层 foo，其坐标原点是层被投放的位置，亦即在本行文字的左上角。x 坐标向右递增，y 坐标向下递增。

如果将层的宽度和高度分别设为页面的宽度和高度，并将其设为页面背景，则坐标原点在页面的左上角，且可使用一些预定义位置投放内容。通过 preset 参数可调整层的坐标原点和坐标方向。ConTEXt 预定义的 preset 参数值如下：

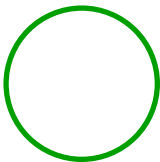
- lefttop
- rightbottom
- middlebottom
- lefttopleft
- righttop
- middle
- middleleft
- lefttopright
- leftbottom
- middletop
- middleright

使用这些参数值时，要注意坐标方向，例如 rightbottom 将层的坐标原点定位于层的右下角，同时 x 坐标方向变为向左递增，y 坐标方向变为向上递增。preset 的用法见下例。该例在层的右下角画了一个绿色的圆，在层的中右位置画了两个小正方形，然后用 \setupbackgrounds 命令将其作为当前页面的背景。

```
% 背景 foo
\startuniqueMPgraphic{circle}
draw fullcircle scaled 2cm withpen pencircle scaled 2pt withcolor darkgreen;
\stopuniqueMPgraphic
\definelayer[foo][width=\paperwidth,height=\paperheight]
\setlayer[foo][preset=rightbottom]{\uniqueMPgraphic{circle}}
% 背景 bar
\definelayer[bar][width=\paperwidth,height=\paperheight]
\setlayer[bar][preset=middleright]{
  \startMPcode
  draw (0, 0) withpen pensquare scaled 12pt withcolor darkred;
  \stopMPcode
}
\setlayer[bar][preset=middleright,x=2cm,y=2cm]{
  \startMPcode
  draw (0, 0) withpen pensquare scaled 12pt
              withcolor transparent (1, .3, darkred);
  \stopMPcode
}
% 设置背景
\setupbackgrounds[page][background={foo, bar}]
```

结语

领域专用语言（DSL）最好是某种宏语言，而非模仿那些通用编程语言的机械化语言。TEX 是排版领域的专用语言，而 MetaPost 是绘图领域的专用语言，它们皆为宏语言，使用它们排版或绘图，本质上是在从事宏编程活动。



宏语言编程并不难，反而比大多数正经的编程语言容易很多，只是宏编程很容易出错。有时你只是手误，写错了一个变量的名字或者符号，宏编译器无法准确捕捉到错误的位置。对此，我的建议是，如果你打算定义一个宏，请不要去尝试将这个宏的代码写完，再去测试，而是先写出这个宏的最简单的形式，哪怕它一开始是空的，也应该代入参数测试一下，然后试着让这个宏逐渐生长，在生长过程中不断测试。简而言之，应当像照顾一株植物一样定义宏。

关于 MetaPost 更为深入和全面的介绍，可参考其手册[19] 以及 Hans Hagon 所写的 MetaFun 手册，后者已在你所安装的 ConTEXt 系统中了，使用以下命令可以找到它。

```
$ mtxrun --search metafun-p.pdf
```

另外，由 Hans Hagon 主持且尚在开发中的 LuaMetaFun 项目可谓是下一代 MetaPost，关于该项目的一些进展和成果，见

```
$ mtxrun --search luametafun.pdf
```

12 幻灯片

现在你大概已经领略甚至可能掌握了 ConTEXt 常用的排版元素，若依然使用 Microsoft PowerPoint 或 WPS 之类的软件制作幻灯片则甚为可惜。其实，只需再掌握寥寥几条排版命令便可以用 ConTEXt 制作出独具风格的幻灯片了。

12.1 纸面尺寸

纸面尺寸是页面的打印尺寸。至此为止，本文档之前所有的示例，在 TEXpage 环境里的，纸面尺寸取决于文档内容的多少，而在 text 环境里的，纸面尺寸默认是 A4，即 21 cm × 29.7 cm。如果你需要将纸面尺寸换成 A5，只需在样式文件中作以下设定

```
\setuppapersize[A5]
```

制作幻灯片通常不需要太大的纸面尺寸。ConTEXt 提供了 S4 尺寸（400 pt × 300 pt），可将其作为幻灯片的纸面尺寸，或者你根据自身需求定义一个纸面尺寸来用，例如

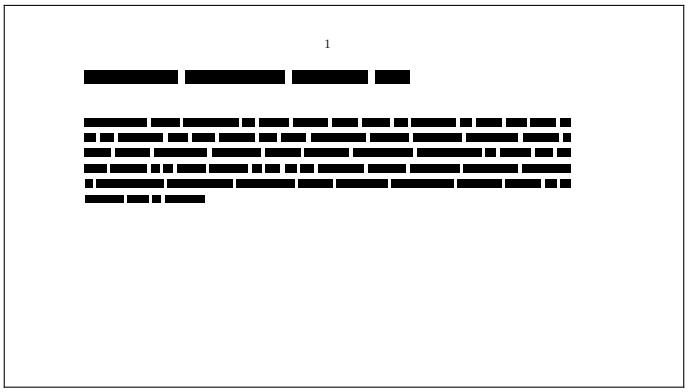
```
\definepapersize[slidesize][width=640pt,height=480pt]
\setuppapersize[slidesize]
```

实际上，纸面尺寸仅对排版结果的打印很重要，倘若只是在计算机（或投影仪）屏幕上观看排版结果，我们总是可以通过调整正文字体大小去应对过大或过小的纸面尺寸，但是纸面尺寸的宽高比例，对于屏幕非常重要。通常，幻灯片的宽高比应当与屏幕的分辨率相适应为最佳。例如，我的屏幕分辨率是 1600 × 900，那么我要制作的幻灯片页面的宽高比也应当是 16 : 9，故而纸面尺寸可设定为

```
\definepapersize[slidesize][width=640pt,height=360pt]
\setuppapersize[slidesize]
```

现在，我们可以制作幻灯片的最原始的形式了，见下例。

```
\usemodule[visual]
\definepapersize[slidesize][width=640pt,height=360pt]
\setuppapersize[slidesize]

\starttext
\title{\fakewords{3}{5}}
\fakewords{50}{100}
\stoptext
```



12.2 页面布局

在 text 环境之前添加以下代码：

```
\showframe
```

可在页面上显示当前的页面布局，结果如图 12.1 所示。可根据图 12.2 理解页面布局中各区域名称和尺寸参数名称。

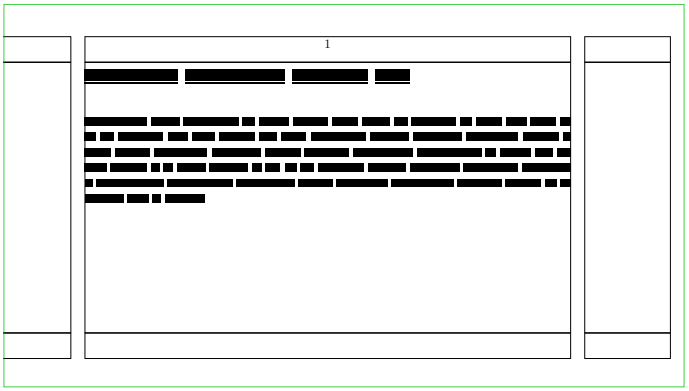


图 12.1 ConTEXt 默认的页面布局

使用 \setlayout 可对各区域尺寸进行调整，例如让版心居中，即图 12.2 所示的 width 和 textheight 确定的区域居中，因为在默认情况下，它有些偏左。我们已经知道，页面的横向尺寸是 640pt，按以下设定即可令版心居中。

```
\setuplayout
  [leftedge=0pt,leftedgedistance=0pt,
   leftmargin=80pt,leftmargindistance=10pt,
   backspace=90pt,
   rightedge=0pt,rightedgedistance=0pt,
   rightmargin=80pt,rightmargindistance=10pt,
   width=460pt]
```

上述设定，实现了页面布局中页面总宽度值 640 pt 的分配，但是要注意一点，backspace 的值等于 4 个以 left 为前缀的参数值之和，虽然在设定了它的各个分量后，但

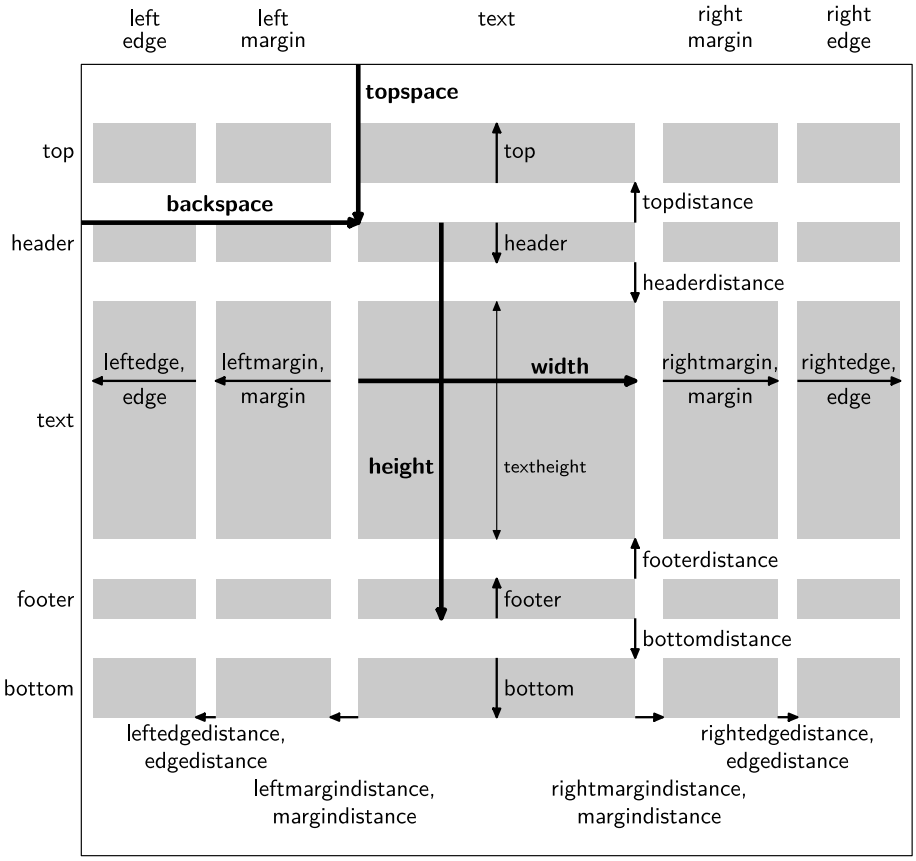


图 12.2　页面布局注解

ConTEXt 在遇到默认的 backspace 值与实际的 left... 参数值之和不符时，会根据 backspace 值进行页面布局。倘若我们不关心 left... 参数，即 ConTEXt 默认对页面左侧留白区域默认如何分配，仅需要让 ConTEXt 知道 backspace=90pt，则上述代码可简化为

```
\setuplayout
  [backspace=90pt,
   rightedge=0pt,rightedgedistance=0pt,
   rightmargin=80pt,rightmargindistance=10pt,
   width=460pt]
```

还可以进一步简化，因为 ConTEXt 对 rightedge 和 rightedgedistance 赋予的默认值原本就是 0，因此有

```
\setuplayout
  [backspace=90pt,
   rightmargin=80pt,rightmargindistance=10pt,
   width=460pt]
```

如果只关心版心的横向居中问题，并不关心左右留白区域的尺寸，最为简单的布局方式是

```
\setuplayout[width=middle]
```

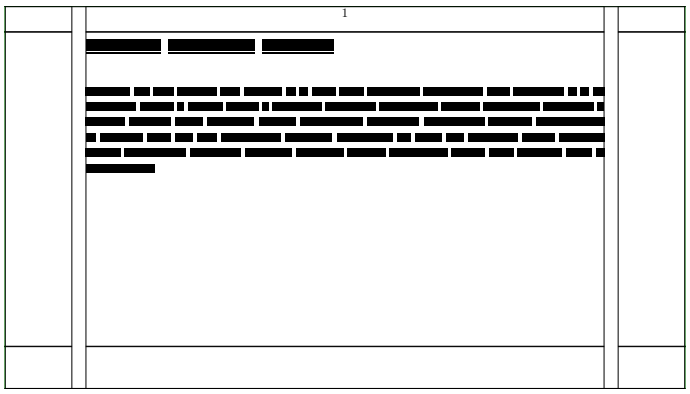
以上设置的是页面横向布局。对于纵向布局，存在与 backspace 类似的问题，即 topspace 的设定。topspace 的值是 top + topdistance，但是在 \setuplayout 中只对后者进行设定是无效的，必须先设定 topspace，然后再将其值分配给 top 和 topdistance，或让直接将 height 设为 middle，让 ConTEXt 自动确定纵向布局。以下纵向参数

```
height=middle,topspace=0pt,bottom=0pt,footer=40pt
```

达到的效果是，页面纵向居中，去除页面顶部和底部留白以节省空间，脚注区域保留。

下面给出一个完整的页面布局设定：

```
\setuplayout[width=middle,height=middle,topspace=0pt,bottom=0pt,footer=40pt]
```



在完成页面布局设定后，别忘记去掉 text 环境之前的 \showframe 语句。此外，我建议你在 text 环境里动手试验一番 \showlayout。

12.3 字体

在 ConTEXt 使用汉字字体，对此估计你已经颇为熟悉了。在幻灯片中，字体可设定为

```
\definefallbackfamily
  [myfonts][ss][latinmodernsans]
  [range={0x0000-0x0400},force=yes]
\definefontfamily[myfonts][ss][simhei][bf=simhei,it=kaiti,bi=simhei]
\setscript[hanzi]
\setupbodyfont[myfonts,ss,16pt]
```

幻灯片所用字体，通常以无衬线字体为主，因为这类字体在屏幕上较为醒目。至于衬线字体和等宽字体，可依情况而定。

12.4 常规样式

幻灯片通常不需要页码，即使需要页码，默认的页码出现的位置和字号皆不合适，故而需消除 ConTEXt 默认在页面顶部设置的页码：

```
\setuppagenumbering[location=]
```

行间距可设为正文字号的 1.75 倍：

```
\setupinterlinespace[line=1.75\bodyfontsize]
```

下例设置了列表的第 1 级样式，具体为 (1) 将 **5.6** 节实现的列表项符号作为默认列表项符号，只是正方形的阴影颜色略加修改；(2) 消除列表项的间距；(3) 在列表项的符号和内容之间插入.5em 的间距；(4) 让列表前后各空 1/4 行。

```
\startuseMPgraphic{square}
numeric u; path p;
u := BodyFontSize;
p := fullsquare scaled .8u;
fill p shifted (.2u, -.2u) withcolor .75white;
fill p withcolor .5[blue, white];
\stopuseMPgraphic

\definesymbol[10][{\lower.2\bodyfontsize\hbox{\useMPgraphic{square}}}]
\setupitemize[1] % 列表第 1 级样式
              [10,packed]
              [distance=.5em,
                before={\blank[quarterline]},
                after={\blank[quarterline]}]
```

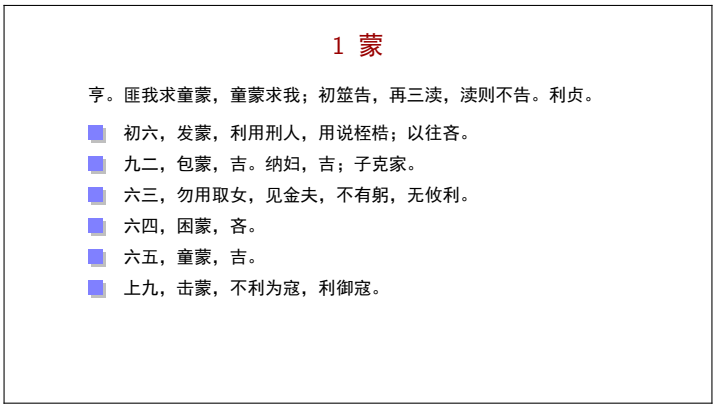
注意，上述代码中的.75white 等效于 rgb 三元组 (0.75, 0.75, 0.75)。另外，若需要二级和三级列表，只需将 \setupitemize 的第 1 个参数换成 2 或 3，然后参照上例设定列表样式。

将幻灯片一级标题设置为居中对齐，并设置字号和颜色，且令其前后各空半行：

```
\setuphead
  [title,chapter]
  [align=center,
    style=\ssb,
    color=darkred,
    before={\blank[halfline]},
    after={\blank[halfline]}]
```

可以在 text 环境里随便写点什么，以察验上述设定是否起效，例如：

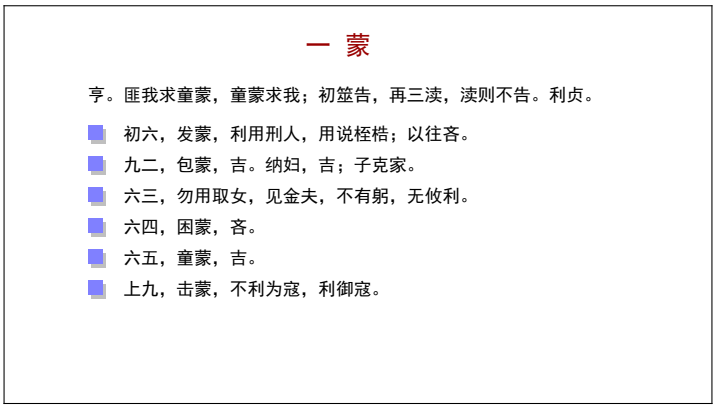
```
\starttext
\startbuffer[foo]
\chapter{蒙卦}
亨。匪我求童蒙，童蒙求我；初筮告，再三渎，渎则不告。利贞。
\startitemize
\item 初六，发蒙，利用刑人，用说桎梏；以往吝。
\item 九二，包蒙，吉。纳妇，吉；子克家。
\item 六三，勿用取女，见金夫，不有躬，无攸利。
\item 六四，困蒙，吝。
\item 六五，童蒙，吉。
\item 上九，击蒙，不利为寇，利御寇。
\stopitemize
\stopbuffer
\dorecurse{3}{\getbuffer[foo]}
\stoptext
```



上述代码使用了 ConTEXt 的 buffer 环境。所谓 buffer，是可以设定名字的缓存，它可以包含 ConTEXt 排版代码。\getbuffer 用于获取缓存内容。

以下设定，可将标题的数字编号更改为中文编号，且将编号样式设定为 inmargin，表示编号不参与居中对齐，只有标题内容参与对齐。

```
\setuphead[chapter][conversion=chinesenumerals,alternative=inmargin]
```



12.5 页脚

在 12.4 节中，我关闭了 ConTEXt 好意提供的页码，但并非不需要页码，而是我期望页码能出现在页面的右下角，且除以总页数，以便清楚幻灯片的进度。

ConTEXt 提供了 \setupfootertexts 命令，可以在页脚左、右区域放入文字、盒子或图形等内容，同时提供了 \setupfooter 用于设定页脚区域的文字样式。另外，\userpagenumber 和 \lastuserpage 可分别用于获取当前页的页码和最后一页的页码。基于这些命令，便可实现能够表示进度的页码，例如

```
\setupfooter[style=small,color=darkred]
% 令页脚左侧区域为空，右侧区域放置用于表示进度的页码
\setupfootertexts[margin][][\hss\userpagenumber/\lastuserpage]
```

结果如图 12.3 所示。

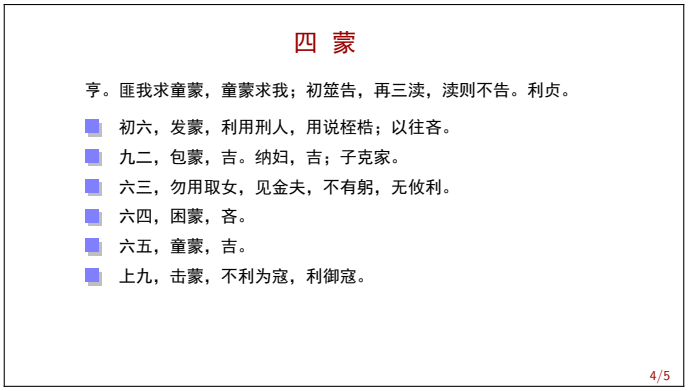


图 12.3　第 4 页幻灯片

也可以用 MetaPost 代码将页码可视化，使之成为进度条，例如：

```
\startuseMPgraphic{processbar}
numeric w, n, s, t; path p; picture q;
w := \overlaywidth; n := \lastuserpage;
s := .5w / (n + 2); t := \userpagenumber - 1;
p := (fullsquare scaled \overlayheight);
pickup pencircle scaled 2pt;
q := image(for i = 0 upto n - 1:
              p := (p shifted (2s, 0)) randomized .5pt;
              if t = i: fill p withcolor darkgreen; fi;
              draw p withcolor transparent(1, 0.5, darkgray);
           endfor;);
draw q xsized w;
\stopuseMPgraphic
```

```
\defineoverlay[process][\useMPgraphic{processbar}]
\setupfootertexts[{\framed[frame=off,offset=0pt,
                           width=\textwidth,height=.8em,
                           background=process,empty=yes]{}}]
```

上述代码唯一需要解释之处是，当 \setupfootertexts 只有一个参数时，该参数的值会被安置在页脚的中间区域。上述代码构造的进度条效果如图 12.4 所示。

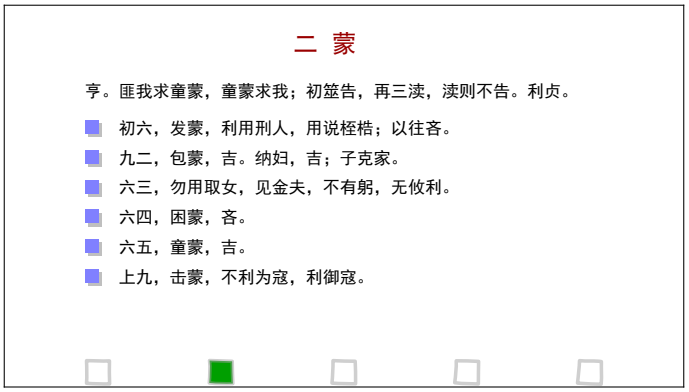


图 12.4 第 2 页幻灯片

12.6 封面

幻灯片的封面可使用 sandardmakeup 环境制作，见下例。

```
\startstandardmakeup[align=center]
\ssd\color[darkred]{如何用 \CONTEXT\ 制作幻灯片}
\blank[2*line]
\ssb\darkred{李某人}
\blank[3*line]
\darkred{2025 年 09 月 01 日}
\stopstandardmakeup
```

之所以用 standardmakeup 环境制作封面，是因为它能构造完全空白的单页，且该页不参与页码计数。

图 12.5 封面

若想让封面赏心悦目，最好的办法是制作一幅与幻灯片所讲主题相适且比例与幻灯片页面相同的图片，将其设为封面背景。假设背景图片为 foo.png，将其设定为封面背景的方法如下：

```
\definelayer[cover][width=\paperwidth,height=\paperheight]
\setlayer[cover][preset=lefttop]
         {\externalfigure[foo.png][width=\paperwidth,height=\paperheight]}
\setupbackgrounds[page][background=cover]
\startstandardmakeup[align=center]
... 省略封面内容...
\stopstandardmakeup
```

图 12.6　全页插图背景

你也可以尝试用 MetaPost 语言画一些图形作为封面背景。这份文档的封面上像迷宫一样的线条便是用 MetaPost 代码画出的。

至于幻灯片最后的致谢页，亦可视为封面，用 `makeup` 环境制作。

结语

一份精致的幻灯片样式，值得你用心设计，但是切记，不要喧宾夺主。质胜文则野，文胜质则史。孔子说，好的幻灯片样式，应该是文质彬彬的。你可以将自己亲手制作的幻灯片样式保存下来，以后可重复使用，也可以分享给他人。不过，制作幻灯片只是帮助你学会页面布局的掌控方法，后者能够帮助你排版更多类型的文档。

13 写书

除了之前章节介绍的所有排版元素，一本书还要有目录、引用、脚注、索引、书签等元素。本章对这些元素给出一些示例性质的介绍，以便你能快速将其应用到你的文档里。详细介绍是不太可能的，原因是，即使我愿意为每个元素写出一章的内容详尽介绍，你也未必会看的，一方面是这些元素本身就枯燥无味，另一方面在一般性的场景中，你也无需了解关于它们的太多细节。不过，对每个元素给出简要介绍后，我会附上更为详尽的介绍以备你日后参考之用。

13.1 结构

一本书，通常是由一组文章构成的，可分为序、前言、目录、正文篇章、跋、参考文献、索引、附录等内容，再加上封面，衬页、扉页等页面。对于书籍制作，ConTEXt 对 text 环境给出了更为细致的划分：

```
\starttext
\startfrontmatter
% 此处放置封面、扉页、序、前言、目录等内容
\stopfrontmatter
\startbodymatter
% 此处为正文篇章
\stopbodymatter
\startbackmatter
% 此处方式跋，参考文献列表、索引等内容
\stopbackmatter
\startappendices
% 此处为附录
\stopappendices
\stoptext
```

对于上述文档逻辑结构，皆为 \definesectionblock 命令[20]定义的区块实例，你也可以用这个命令定义自己的结构。

将一本书的内容全部放在同一份源文件中并不违法，但无论是撰写还是修改必定极为不便，故而通常是构造一份主文件，其内容如下例，而书的各部分内容则存放在一些独立的源文件里，然后用 \input 命令将其载入到主文件中。

```
% 主文件
\starttext
\startfrontmatter
\input cover  % cover.tex: 封面
\input preface  % preface.tex: 序
\stopfrontmatter
```

```
\startbodymatter
\input 01  % 01.tex: 第一章
\input 02  % 02.tex: 第二章
% ... ...
\stopbodymatter
\startbackmatter
\title{参考文献}
\placelistofpublications  % 参考文献列表
\stopbackmatter
\startappendices
\null  % 暂时为空
\stopappendices
\stoptext
```

主文件 \input 命令载入的文件通常位于同一目录。

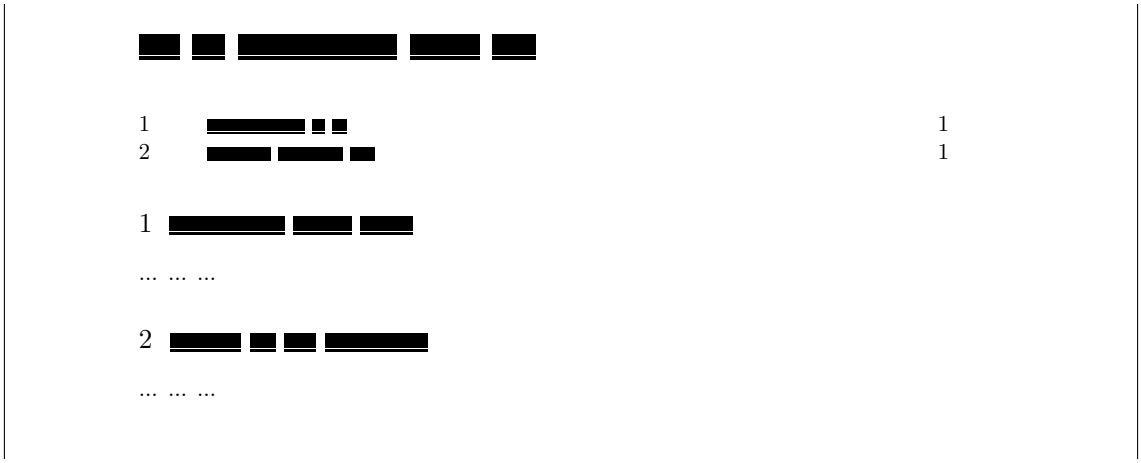
原则上，书籍的所有排版样式皆应在单独的文件中设定，然后使用 \input 或 \environment 命令在 text 环境之前将其载入。例如

```
\environment book-style  % 载入book-style.tex
\starttext
% ... ... ...
\stoptext
```

13.2 目录

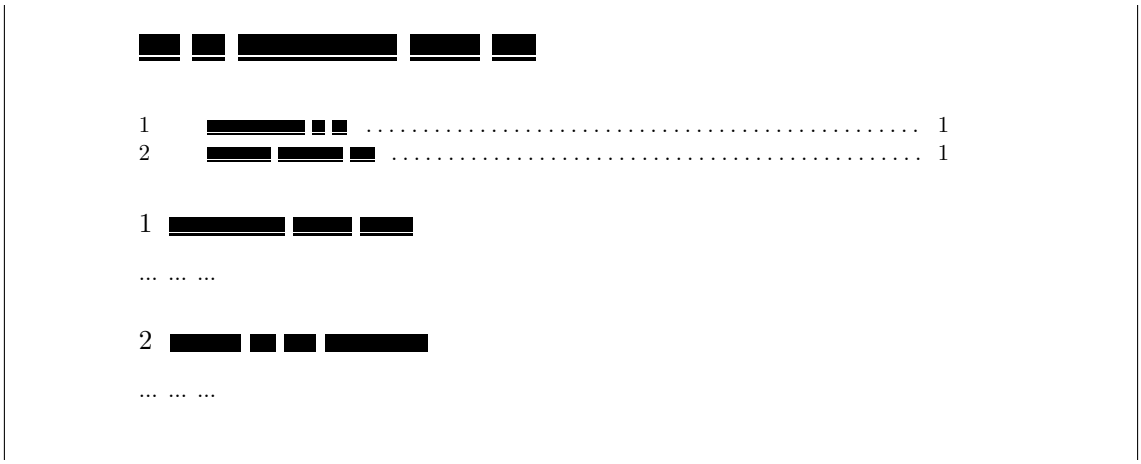
\placecontent 可将全文章节标题及其所在页码等信息汇总为一个列表，以方便读者查阅。例如

```
\usemodule[visual]
\starttext
\title{\fakewords{3}{5}}
\placecontent
\section{\fakewords{3}{5}}
... ... ...
\section{\fakewords{3}{5}}
... ... ...
\stoptext
```



使用 \setupcombinedlist 可设定目录样式，例如设定可在目录中出现的标题级别以及列表样式。若得到常见的目录样式，只需作以下设定：

```
\setupcombinedlist[content][alternative=c]
```



目录列表样式参数 alternative 有 a，b，c，d 四个值可选，默认是 b。本文档目录使用的是 d。

\setupcombinedlist 亦可用于设定目录列表中的标题级别，例如若让章与节出现在目录中，只需

```
\setupcombinedlist[content][list={chapter,section}]
```

注意，当 \placecontent 出现在 \chapter 之后时，生成的目录仅针对该章之内的各节。若是写书，需将 \placecontent 放在 frontmatter 环境，例如

```
\startfrontmatter
\title{目录}
\placecontent
\stopfrontmatter
```

可将全篇章节列入目录。

\setuplist 可用于设定目录列表中的标题样式，例如

```
\setuplist[chapter]
          [alternative=a,
           before={\blank[halfline]},after={\blank[halfline]},style=bold]
\setuplist[section]
          [alternative=d,style=normal,pagestyle=smallbold]
```

上述代码设定的目录样式，与本文档的目录样式已经非常接近了。

无编号标题，例如 \title、\subject 等标题默认不会出现在目录中，不过，ConTEXt 也为它们留出了后门。以 \title 为例，先作以下设定

```
\setuphead[title][incrementnumber=list]
```

然后便可将其添加到目录列表，如下

```
\setupcombinedlist[content][list={title,chapter,section}]
```

需要注意的是，在 frontmatter 环境中放置目录列表时，若使用以下代码

```
\startfrontmatter
\title{目录}
\placecontent
\stopfrontmatter
```

由于此时 \title 已被列入目录列表，因此 \title{目录} 本身会出现在目录列表中。为避免这一问题，需要为目录页定义专用的标题命令。ConTEXt 支持自定义标题命令，例如

```
\definehead[TOC][title]
```

定义了一个新的标题 \TOC，它会继承 \title 的样式。在 frontmatter 环境中使用 \TOC，如下

```
\startfrontmatter
\TOC{目录}
\placecontent
\stopfrontmatter
```

便可让目录页的标题从目录列表中得以逃逸。文档[21]对目录排版给出了更为细致的介绍，可作进一步参考。

13.3 引用

在之前的列表、表格、插图和数学公式等章节中，已简略介绍了 ConTEXt 引用的用法。ConTEXt 的标题也支持引用。例如，本节的标题对应的排版命令是

```
\section[reference]{引用}
```

可以在文章几乎任何一个位置，像下面这样引用本节：

```
我在 \at[reference] 页 \in[reference] 节\about[reference]中的一些内容。
```

> 我在 89 页 13.3 节“引用”中的一些内容。

使用 \textreference 可在文档几乎任何位置插入引用。例如

```
我在此处放置了一个引用\textreference[myref]{一个引用}。
```

> 我在此处放置了一个引用。

```
我在此处使用一个引用，它是 \at[myref] 的「\in[myref]」。
```

> 我在此处使用一个引用，它是 90 的「一个引用」。

13.4 脚注

若需要对书籍内容中一些文字进行注解，可采用脚注形式。\footnote 命令可在待注解的文字之后插入脚注，并自动生成序号。例如

```
我以脚注的形式告诉你一个关于 \CONTEXT\ 脚注的秘密\footnote{\CONTEXT\ 的脚注默认不支持中文断行。}。
```

> 我以脚注的形式告诉你一个关于 ConTEXt 脚注的秘密[1]。

若让脚注里的中文内容能够断行，只需在脚注中使用 \setscript[hanzi] 强行开启中文断行功能。为了简便，可定义一个宏替代 \footnote：

```
\def\zhfootnote#1{\footnote{\setscript[hanzi] #1}}
```

另一种方法是 ConTEXt 超人 Wolfgang Schuster 提供的，用 setups 机制[22]为脚注构造专用的设定，例如：

[1] ConTEXt 的脚注默认不支持中文断行。

```
\startsetups footnote:hanzi
\setscript[hanzi]
\stopsetups
\setupnote[footnote][setups={footnote:hanzi}]
```

13.5 索引

索引通常放在 backmatter 环境，即附在书的正文之后，以便检索在正文某页检索一些关键词。这些关键词在正文中需由 \index 给出。例如

```
\startbackmatter
\title{索引}
\placeindex
\stopbackmatter
```

```
我在此演示 \tex{placeindex}\index[placeindex]{\tex{placeindex}}
和 \type{\index}\index[index]{\type{\index}} 的用法。
```

> 我在此演示 \placeindex 和 \index 的用法。

\placeindex 产生的结果见本文档附录的索引部分。

13.6 书签

与目录类似，对于内容较多的 PDF 文档，提供书签（Bookmark）亦可便于他人阅读。书签通常显示于 PDF 阅读器的侧栏，如图 13.1 所示，点击某个书签便可跳转至其关联的页面。

图 13.1　PDF 书签

为 PDF 文件制作书签，只需在样式文件中添加以下语句：

```
\setupinteraction[state=start,focus=standard]
\setupinteractionscreen[option=bookmark]
\placebookmarks[title,chapter,section][title,chapter]
```

\setupinteraction 用于开启 PDF 的用户交互特性。\setupinteractionscreen 用于设定 PDF 文件被阅读器打开后，以何种形式如何呈现在屏幕上，若其 option 值为 bookmark，则文件打开后，会自动开启阅读器的侧边栏并显示书签；若设置 option 为 max，则文件在被打开后会全屏显示。上述 \placebookmarks 语句的用途是设置可出现在书签栏的标题级别，且仅允许 \title 和 \chapter 的子标题列表可以展开。

需要注意的是，ConTEXt 同样默认无编号标题不被列入书签，但是倘若做以下设定

```
\setuphead[title][incrementnumber=list]
```

则 \title 亦可出现在书签列表中。

还需要注意一点，书签功能取决于你所用的 PDF 阅读器是否支持。此外，你的 PDF 阅读器可能会将 ConTEXt 生成的书签视为索引（Index），而其本身则提供了另一个叫作书签的功能，允许用户手动在侧边栏为文档的某一页建立链接，与 ConTEXt 的书签原理相同。

13.7 双面

为了省纸，书籍通常是双面排版印刷，此时页码的位置要对开设定。如果订口和翻口是非对称的，还需要按双面模式设定页面布局。通常是翻口和底部（地脚）的留白要大一些，以便人们在读书时写注解。经典的双面排印布局是 Van de Graaf 的 Conon 版式，用 ConTEXt 的页面布局可表达为

```
\usemodule[visual]
% 纸张为 A4
\setuppapersize[A4]
% Van de Graaf Conon 版式
\setuplayout
        [backspace=.111\paperwidth,
         width=.667\paperwidth,
         topspace=.056\paperheight,
         height=.778\paperheight]
% 开启双面对开模式
\setuppagenumbering[alternative=doublesided]
\starttext
\dorecurse{30}{\fakewords{50}{150}\par}
\stoptext
```

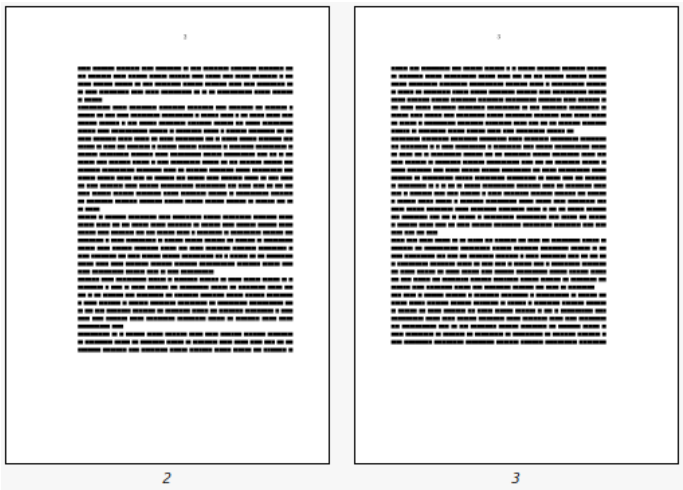


图 13.2　Van deGraaf Conon 版式

13.8 页眉

在页眉区域放置内容，需要用 \setupheadertexts 命令。在单面排印的情况下，该命令支持三个参数，第一个参数是可选的，若不设定，则值默认是 text，表示版心上下的页眉和页脚局部区域，也可以将其值设为 margin，此时表示的是订口和翻口上下的页眉和页脚局部区域。

下例设定了正文区域上方的页眉左侧和右侧区域的内容。

```
\setupheadertexts[页眉左侧][页眉右侧]
```

如果你需要在订口和翻口上方的页眉区域放置内容，可按下例实现。

```
\setupheadertexts[margin][订口上方的页眉][翻口上方的页眉]
```

在双面对开模式下，页眉和页脚的设定则需要五个参数，第一个参数依然是 text 或 margin，至于后四个参数，其中前两个用于设定偶数页的页眉和页脚的左右两侧，后两个则用于设定奇数页的页眉左右两侧。下例是对奇数页和偶数页的正文上方页眉内容的设定。

```
\setupheadertexts
    [奇数页的页眉左侧][奇数页的页眉右侧]
    [偶数页的页眉左侧][偶数页的页眉右侧]
```

下例是偶数页和奇数页的订口和翻口上方的页眉内容的设定。

```
\setupheadertexts
  [margin]
  [奇数页的页眉左侧][奇数页的页眉右侧]
  [偶数页的页眉左侧][偶数页的页眉右侧]
```

页眉区域通常可以放置章节标题以及页码等内容。本文档主要作为屏幕读物，而非印刷读物，故而采用了单面模式，但是页眉是按双面对开模式设定的，如下：

```
\def\CurrentChapter{%
  第 \headnumber[chapter]\ 章\kern 1em\getmarking[chapter]%
}
\setupheadertexts[\CurrentChapter][pagenumber][pagenumber][\CONTEXT\ 蹊径]
```

在单面模式下，奇数页的页眉左侧和右侧显示的分别是当前章名和页码，而偶数页的页眉左侧和右侧显示的分别是书名和页码，但是若开启了双面对开模式，偶数页的页眉便会按上述设定，页码在左侧，书名在右侧。

本文档的页眉设定用了几个之前未曾提及的命令，其中 \headnumber 可以取得当前章节的序号，\getmarking 可以取得当前章节的标题，\kern 命令用于制作指定宽度的间距。此外，我还需要再次提醒，要注意上述宏定义里的注释符，它们的用途是消除其后的换行符，以防在宏展开时这些换行符被当成空格而污染排版内容。

也许你忽然想到，如果我想在正文上方的页眉区域或下方的页脚区域的中部放置内容，该如何实现呢？页眉和页脚的设定命令皆支持单参数形式，可用于在正文上方页眉或下方页脚中部放置内容，下例绘制了页眉底线，并居中放置了一些内容。

```
\setupheadertexts
  [\framed[width=\textwidth, frame=off, bottomframe=on]{页眉内容}]
```

你也能将上例扩展为在页眉的左侧、中部和右侧放置内容，同时带有底线的效果，而无需使用 \setupheadertexts 命令的多参数形式，只是在双面对开模式下，你需要用 \ifodd 命令判断当前页码的奇偶性，然后相应调整左右侧内容，下例可作为参考。

```
\defineframed[framedheader]
             [width=\textwidth, frame=off, bottomframe=on]
\def\myheader{%
  \ifodd\pagenumber
    \framedheader{左侧\hfill 中部\hfill\pagenumber}%
  \else
    \framedheader{\pagenumber\hfill 中部\hfill 右侧}%
  \fi%
}
\setupheadertexts[\myheader]
```

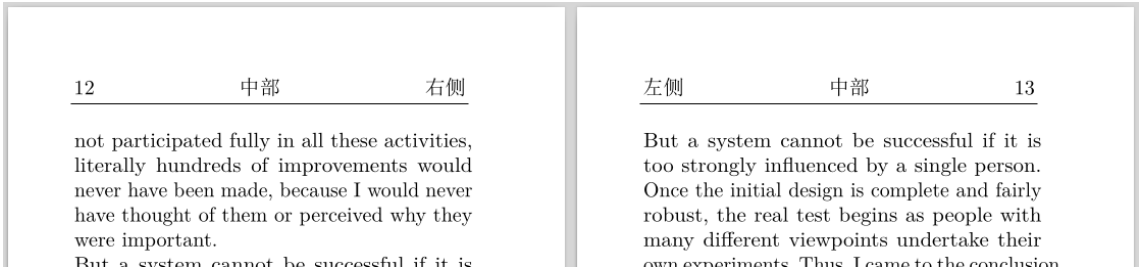


图 13.3　自定义页眉

13.9 页脚

在 **12.5** 节已经用 `\setupfootertexts` 命令为幻灯片设置了页脚区域，该命令的参数形式与 `\setupheadertexts` 相同，故其用法不再赘述。

结语

TEX 系统复杂且难以掌握，主要原因在于排版本身便是极为复杂的事情，页眉区区方寸之地，若想随心所欲设置一些内容，已是颇为不易了，而写书更是工程级别的任务。绝大多数人，一生之中，工作量最大的单个作品大概是自己的学位论文，本科一二十页，硕士四五十页；到了博士阶段，学位论文有百余页，算得上书的级别。换言之，TEX 在写博士论文这种程度的文档时，方能刚刚开始全面展现其功能和优势。换言之，用 TEX 排版一份文档的难度取决于该文档要写什么以及如何写，不可脱离文档自身的内容和形式去看待某个排版系统的优点和缺点。

14 文献

稍微严肃一些的文章，往往会附上一些与文章内容密切相关的参考文献。科研论文更是如此，牛顿都说过自己是站在巨人的肩膀上做事情的。参考文献便是巨人。任何一个 TEX 系统皆不敢对支持参考文献排版这一事宜掉以轻心，否则 TEX 无以为科技文献排版软件之先驱和典范。

14.1 数据格式

大多数 TEX 系统在排版参考文献时，皆需依赖一个名为 bibtex 的外部程序。由 bibtex 基于文献数据文件生成参考文献列表信息，然后交由 TEX 进行排版，这一过程通常是自动进行的，并不需要用户了解和干预。ConTEXt 已在自身内部实现了 bibtex 程序的全部功能，故而不再依赖这个外部程序了。

文献数据文件是纯文本文件，其中包含着论文、专著和手册等文献数据。例如，一篇期刊论文，其信息可表示为

```
@article{knuth-1984-literate,
  title={Literate programming},
  author={Knuth, Donald Ervin},
  journal={The computer journal},
  volume={27},
  number={2},
  pages={97--111},
  year={1984},
  publisher={Oxford University Press}
}
```

再例如一本专著，其信息可表示为

```
@book{knuth-1986-texbook,
  title={The \TeX\ book},
  author={Knuth, Donald Ervin and Bibby, Duane},
  volume={1993},
  year={1986},
  publisher={Addison-Wesley Reading, MA}
}
```

为了学习 ConTEXt 的参考文献排版功能，可将上述文献数据保存到一份空的文本文件里，例如 foo.bib，然后在同一目录下，建立 ConTEXt 源文件 test.tex，其内容为

```
\usebtxdataset[foo.bib] % 加载 foo.bib
\starttext
Knuth 基于文学编程\cite[knuth-1984-literate]技术开发了
\TEX\ 系统\cite[knuth-1986-texbook]。
\blank
\placelistofpublications
\stoptext
```

编译 test.tex，可得以下结果：

Knuth 基于文学编程[1]技术开发了 TEX 系统[2]。

1　D.E. Knuth, “Literate programming”, *The computer journal* 27(2), 97 - 111, 1984.

2　D.E. Knuth and D. Bibby, *The TEX book* (Vol. 1993), Addison-Wesley Reading, MA, 1986.

若将文献数据文件视为文献数据库，则 bibtex 程序或者 ConTEXt 内部实现的同样功能的模块相当于数据库的搜索引擎。在科研工作中，你可以将自己所读的所有文献置于一份 .bib 文件中，然后在写论文或专著时，按需引用。

上述示例所用的文献样式是 ConTEXt 默认样式。在 \placelistofpublications 之前可使用 \usebtxdefinitions 设定文献样式，有三种预定义样式可选，分别为 apa，aps 和 chicago。例如使用 aps 作为文献样式，只需

```
\usebtxdefinitions[aps]
\placelistofpublications
```

Knuth 基于文学编程[1]技术开发了 TEX 系统[2]。

[1]　D.E. Knuth, Literate programming, *The computer journal* **27**(2), 97 - 111 (1984).

[2]　D.E. Knuth and D. Bibby, *The TEX book*, Vol. 1993 (Addison-Wesley Reading, MA, 1986).

预定义的样式，其细节可以调整。例如，以下命令将文献序号宽度设定为 auto，并将其与文献条目的间距设为半个字宽。

```
\setupbtxlist[aps][width=auto,distance=.5em]
```

Knuth 基于文学编程[1]技术开发了 TEX 系统[2]。

[1] D.E. Knuth, Literate programming, *The computer journal* **27**(2), 97 - 111 (1984).

[2] D.E. Knuth and D. Bibby, *The TEX book*, Vol. 1993 (Addison-Wesley Reading, MA, 1986).

14.2　自制样式

ConTEXt 提供的这几种文献样式可能并不符合你的要求，此时你有两个方案，第一个方案是放弃这一切，使用 **5.5** 所述的廉价方案，第二个方案是基于 ConTEXt 的文献样

式机制，自己定义一种你想要的样式。如果你选择了后者，下文可为你提供一个简单的示例，希望有所帮助。

假设要定义一种名为 bar 的文献样式。首先，按照 ConTEXt 的文件命名约定，创建文件 publ-imp-bar.tex，先写入以下内容：

```
\definebtx[bar]
```

上述代码定义了一个命名空间，其上层还有个空间，即 btx。然后你可以根据自己需要定义一些子空间。例如，为 bar 定义子空间 list，再为后者定义子空间 article：

```
\definebtx[bar:list][bar]
\definebtx[bar:list:article][bar:list]
```

实际上，子空间通常无需定义，只有当你需要为某个子空间设定外观时才需要定义它。

list 空间包含了 BibTEX 数据格式中所有的文献条目和字段名。例如，article 空间表示期刊论文条目，在该空间内可以定义期刊论文的各种字段空间，例如

```
\startsetups btx:bar:list:article
  \texdefinition{btx:bar:list:author}\btxperiod\btxspace
  \texdefinition{btx:bar:list:title}\btxperiod\btxspace
  \texdefinition{btx:bar:list:journal}[J]\btxcomma\btxspace
  \texdefinition{btx:bar:list:year}\btxcomma\btxspace
  \texdefinition{btx:bar:list:volume}
  \texdefinition{btx:bar:list:number}\btxcolon\btxspace
  \texdefinition{btx:bar:list:pages}\btxperiod
\stopsetups
```

上述代码定义了期刊论文的构成，其中 \btxcomma，\btxperiod，\btxcolon，\btxspace 分别为西文的逗号，句号，冒号和空格。倘若需要使用中文标点，亦可自行定义，例如：

```
\def\btxchinesecomma{，}
\def\btxchineseperiod{。}
\def\btxchinesecolon{：}
```

接下来，在 publ-imp-bar.tex 文件中定义期刊论文的各个字段，如下

```
\starttexdefinition btx:bar:list:author
  \btxflush{author}
\stoptexdefinition

\starttexdefinition btx:bar:list:title
  \color[darkred]{\bf\btxflush{title}}
\stoptexdefinition
```

```
\starttexdefinition btx:bar:list:journal
  \btxflush{journal}
\stoptexdefinition
\starttexdefinition btx:bar:list:year
    \btxflush{year}
\stoptexdefinition
\starttexdefinition btx:bar:list:volume
    \btxflush{volume}
\stoptexdefinition
\starttexdefinition btx:bar:list:number
  \ignorespaces  % 忽略空白字符
  \btxleftparenthesis  % 左括号 （
  \btxflush{number}
  \btxrightparenthesis  % 右括号 ）
\stoptexdefinition
\starttexdefinition btx:bar:list:pages
  \btxflush{pages}
\stoptexdefinition
```

上述代码定义了样式 bar 的数据格式，其中 \btxflush 命令的作用是从 BibTEX 格式的文献库里获得各条目的字段内容。

若让 ConTEXt 能够按照这种格式将其呈现为文献列表的形式，需要定义一个同名的渲染环境，亦即在 publ-imp-bar.tex 文件中增加以下设定：

```
\definebtxrendering[bar]
```

至此，新的文献样式 bar 便已定义完毕。在 publ-imp-bar.tex 文件所在目录下，建立测试文件 test.tex，其内容如下：

```
\usebtxdataset[foo] % 加载 foo.bib
\starttext
Knuth 发明了文学编程语言 WEB\cite[knuth-1984-literate]。
\blank
\usebtxdefinitions[bar]   % 使用上文定义的参考文献列表样式 bar
\placelistofpublications  % 排版文献列表
\stoptext
```

编译 test.tex，结果如下图。

Knuth 发明了文学编程语言 WEB[1]。

1　D.E. Knuth. **Literate programming**. The computer journal[J], 1984, 27(2): 97 - 111.

14.3 样式微调

上一节定义的 bar 样式，不满足多数中文学术期刊和学位论文的要求，存在的主要问题是，文献序号并非方括号形式且序号与文献条目的间距过大，英文的作者姓名没有将姓放在前面，名字放在后面并缩写。

文献需要的样式可以用 \setupbtxlist 设定。类似于上文设定 apa 样式的文献序号的方法，将 bar 样式的文献序号设定为方括号形式，并缩小序号与文献条目的间距，只需

```
\setupbtxlist[bar][starter={[},stopper={]},width=auto,distance=1em]
```

实际上 \setupbtxlist[bar] 是 \setuplist[btx:bar] 的别名，二者是等效的。\setuplist 命令，我们在 **13.2** 节设定目录样式时已经见识过了。

要设置文献条目中的作者信息的样式，需要为其定义命名空间，例如为 bar 样式定义作者信息的命名空间，需作以下定义：

```
\definebtx[bar:list:author][bar:list]
```

然后便可设定 bar:list:author 的外观，如下：

```
\setupbtx[bar:list:author]
         [authorconversion=invertedshort,
           separator:invertedinitials=\btxspace,
           stopper:initials=\btxspace]
```

参数 authorconversion 用于设定作者姓名格式，其值 invertedshort 可将作者的姓氏前置，名字后置，但该设定只会作用于西文的作者姓名，对中文作者姓名无效，原因是前者的姓名之间有空格作为间隔，故而有前后之分，而中文的作者姓名之间没有间隔。separator:invertedinitials 用于设定姓名之间的间隔符，默认是西文逗号。stopper:initials 用于设定作者姓氏简写字母后的符号，默认是西文句点。上述样式微调的结果如下图所示：

Knuth 发明了文学编程语言 WEB[1]。

[1] Knuth D E. **Literate programming**. The computer journal[J], 1984, 27(2): 97 - 111.

如果你不想为 bar 样式定义作者信息命名空间，也可以直接基于默认的作者信息命名空间设定样式，例如

```
\setupbtx[default:list:author]
         [authorconversion=invertedshort,
           separator:invertedinitials=\btxspace,
           stopper:initials=\btxspace]
```

自定义的文献样式，若未为其定义子命名空间，可以直接用默认的命名空间设定样式，而且这些默认的空间也皆有默认的样式，倘若你不设定，ConTEXt 便按默认的样式渲染该空间表征的文献字段。

14.4 更具一般性

ConTEXt 为文献数据源提供了别名机制，允许你为载入的文献数源据取一个名字，基于该名字和文献样式便可实现特定的文献列表渲染，具体操作见下例

```
\usebtxdataset[bibdata][foo.bib]
\setupbtx[dataset=bibdata] % 为 \cite 命令设定数据源
\usebtxdefinitions[bar]    % 载入 bar 样式
\definebtxrendering[references][bar][dataset=demo]
\starttext
Knuth 发明了文学编程语言 WEB\cite[knuth-1984-literate]。
\blank
% 排版文献列表，也可以用 \placebtxrendering[reference]
\placelistofpublications[references]
\stoptext
```

ConTEXt 的文献管理功能之所以设计得如此复杂，其意图是要支持同一份文献数据，可以用不同的文献样式排版。不过，我觉得这实在是一种过度设计，可以称得上失败。

14.5 第三条路

如果你对 ConTEXt 的文献管理功能颇为不满，实际上还有第三条路可以走，那就是先用 ConTEXt 内置的文献样式管理引用的文献，待完全定稿后，再手工将文献列表排版成你期望的样式。为了降低工作量，你也可以用现代的一些容易使用的脚本语言编写一个简陋的类似 bibtex 的程序作为辅助工具。

结语

Hans Hagen 为 ConTEXt 文献管理功能专门写了一份约 100 页的手册，见

```
$ mtxrun --search mkiv-publications.pdf
```

在不理解本文自定义文献样式所用的各命令及其参数的含义时，可从该手册获得更多的理解……只是这份手册内容凌乱，缺乏概观式的介绍，我建议先阅读文档 [23]，先存下一些不解，然后再阅读该手册。

15 zhfonts

基于第 3 章介绍的汉字字体的设定方法，使用 ConTEXt 足以排版中文文档，但是存在汉字标点符号的间距过大且无法与正文左右边界对齐等问题。若你对此无法忍受，可以考虑使用我写的 zhfonts 模块，另外该模块显著简化了中文字体的设定。这一章的内容应当在第 3 章中出现，但是我不太想过早炫耀我的工作，何况它如此微薄。

15.1 初印象

根据现在你已经掌握的 ConTEXt 知识，理解以下代码，想必已不在话下：

```
\definefallbackfamily
  [foo][rm]
  [latinmodernroman]
  [range={0x0000-0x0400},force=yes]
\definefontfamily
  [foo][rm][nsimsun]
  [bf=simhei,it=kaiti,bi=simhei]
\setscript[hanzi]
\setupbodyfont[foo,12pt]
\setuppapersize[S4]
\setuplayout[width=middle]
\showglyphs  % 显示字形边界盒

\starttext
小知不及大知，小年不及大年 [... 省略很多字...] 此小大之辩也。
\stoptext
```

对应的排版结果如图 15.1 所示。若使用 zhfonts 模块，则上述示例可改写为

```
\usemodule[zhfonts][size=12pt]
\setuppapersize[S4]
\setuplayout[width=middle]
\showglyphs  % 显示字形边界盒

\starttext
小知不及大知，小年不及大年 [... 省略很多字...] 此小大之辩也。
\stoptext
```

对应的排版结果如图 15.2 所示。认真对比图 15.1 和图 15.2 中的标点符号的排版结果，则不难发现 zhfonts 模块解决的问题是标点间距微调。

小知不及大知，小年不及大年。奚以知其然也？朝菌不知晦朔，蟪蛄不知春秋，此小年也。楚之南有冥灵者，以五百岁为春，五百岁为秋；上古有大椿者，以八千岁为春，八千岁为秋，此大年也。而彭祖乃今以久特闻，众人匹之，不亦悲乎！汤之问棘也是已。穷发之北，有冥海者，天池也。有鱼焉，其广数千里，未有知其修者，其名为鲲。有鸟焉，其名为鹏，背若泰山，翼若垂天之云，抟扶摇羊角而上者九万里，绝云气，负青天，然后图南，且适南冥也。斥鴳笑之曰：“彼且奚适也？我腾跃而上，不过数仞而下，翱翔蓬蒿之间，此亦飞之至也。而彼且奚适也？”此小大之辩也。

图 15.1　未使用 zhfonts 的中文标点排版效果

小知不及大知，小年不及大年。奚以知其然也？朝菌不知晦朔，蟪蛄不知春秋，此小年也。楚之南有冥灵者，以五百岁为春，五百岁为秋；上古有大椿者，以八千岁为春，八千岁为秋，此大年也。而彭祖乃今以久特闻，众人匹之，不亦悲乎！汤之问棘也是已。穷发之北，有冥海者，天池也。有鱼焉，其广数千里，未有知其修者，其名为鲲。有鸟焉，其名为鹏，背若泰山，翼若垂天之云，抟扶摇羊角而上者九万里，绝云气，负青天，然后图南，且适南冥也。斥鴳笑之曰：“彼且奚适也？我腾跃而上，不过数仞而下，翱翔蓬蒿之间，此亦飞之至也。而彼且奚适也？”此小大之辩也。

图 15.2　使用 zhfonts 的中文标点排版结果

15.2 安装

首先下载 zhfonts-master.zip 包，如果你能访问 Github 网站，则获取地址为

`https://github.com/liyanrui/zhfonts/archive/refs/heads/master.zip`

否则可以从国内的 Gitee 网站获取，则获取地址为

`https://gitee.com/garfileo/zhfonts/repository/archive/master.zip`

解包后可得目录 zhfonts-master，将其更名为 zhfonts 并移动至

```
$TEXROOT/texmf-local/tex/context/third
```

若无该目录，可自行创建。至于 `$TEXROOT` 的含义，详见 **3.1** 节。

然后执行以下命令刷新 ConTEXt 文件系统，

```
$ context --generate
```

以使 ConTEXt 能够通过 `\usemodule[zhfonts]` 搜索并加载 zhfonts 模块的相关文件。

如果你的系统中装有 git，可在 `$TEXROOT/texmf-local/tex/context/third` 目录执行以下命令安装 zhfonts模块：

```
$ git clone https://github.com/liyanrui/zhfonts.git
$ context --generate
```

日后若要更新 zhfonts，只需将上述的「`git clone ...`」换成「`git pull`」。

15.3 用法

zhfonts 模块默认使用 simsun.ttc，simhei.ttf，simkai.ttf 这三款汉字字体——想必你早已按照 **3.1** 节的介绍安装了它们——，西文字体则使用随 ConTEXt 发行的 Latin Modern 系列字体。

zhfonts 默认使用 11 pt 正体字（宋体 + Latin Modern Serif）作为正文字体：

```
\usemodule[zhfonts]  % 默认正文字号为 11 pt
\startTEXpage[frame=on,offset=4pt,width=12cm]
我能吞下玻璃而不伤身体。I can eat glass, it does not hurt me.
\stopTEXpage
```

我能吞下玻璃而不伤身体。I can eat glass, it does not hurt me.

使用模块参数 `family=ss` 或 `family=tt`，可将正文字体切换为模块参数 `size` 所设定大小的无衬线字体或等宽字体。例如

```
% 以 12 pt 的无衬线字体作为正文字体
\usemodule[zhfonts][family=ss,size=12pt]
\startTEXpage[frame=on,offset=4pt,width=12cm]
我能吞下玻璃而不伤身体。I can eat glass, it does not hurt me.
\stopTEXpage
```

我能吞下玻璃而不伤身体。I can eat glass, it does not hurt me.

15.4 设定

使用 `\setupzhfonts` 可替换 zhfonts 默认的中文和西文字体。例如，将默认的汉字的衬线字族的粗体更换为 NotoSerifCJK-Bold.ttc[1]：

```
\setupzhfonts[serif][bold=notoserifcjkscbold,bolditalic=notoserifcjkscbold]
```

亦可将默认的汉字字族全部替换为其他字体，例如替换衬线字族：

```
\setupzhfonts
  [serif]
  [regular=notoserifcjkscregular,
   bold=notoserifcjkscbold,
   italic=notoserifcjkscregular,
   bolditalic=notoserifcjkscbold]
```

[1] Google 开发的 Noto 字体中的一款，需要你去网络上搜索获取并按照 **3.1** 节所介绍的方式予以安装。

西文字体的替换方式为

```
\setupzhfonts
  [latin,serif|sans|mono]
  [regular=..., bold=..., italic=..., bolditalic=...]
```

zhfonts 并未使用汉字字体中的西文字形，而是将汉字字体作为替补字体「注入」到西文字体，具体原因已在 **3.7** 节给出了说明。

两种不同的字体，混合到一起，可能会出现字形尺寸不一致的情况。若遇到这种情况，可使用 \setupzhfonts 的 fontname@n.n 的参数形式设置汉字的缩放比例。例如

```
\usemodule[zhfonts][size=16pt]
\setupzhfonts
  [serif]
  [regular=@1.5, % 设置默认字体的 rscale 参数值
   bold=notoserifcjkscbold@0.5]
\startTEXpage[frame=on,offset=4pt]
这是汉字，These are English characters.\par
\bf 这是汉字，These are English characters.
\stopTEXpage
```

这是汉字，These are English characters.
这是汉字，These are English characters.

15.5 数学字体

zhfonts 模块默认使用的数学字体是 ConTEXt 默认的 Latin Modern 数学字体。如果你想换成其他数学字体，设定方法与上文所述的方法有所区别，原因是数学字体有些复杂，而且可选性也不太多。文档[24]里已经提供了一些 ConTEXt 自带的数学字体的配置。这些配置用 ConTEXt 的术语来说，即 Typescript。在使用 \setupzhfonts 命令替换默认的数学字体时，你可以直接用某款数学字体的 Typescript 名字指代该字体。例如若将 xits 字体设为数学字体，其 Typescript 名字是 xits，设定命令如下：

```
\setupzhfonts[math][xits]
```

如果你需要使用某款外部数学字体，你需要按照第 **3** 章所述方法将其安装到 ConTEXt 环境里，然后为它写一小段基本的 Typescript。下面以 Google 开发的 Noto Sans Math 字体为例，假设你安装了该字体，那么面向该字体的基本的 Typescript 代码如下：

```
\starttypescript[math][this-is-noto-sans-math]
  \definefontsynonym[MathRoman][file:NotoSansMath-Regular.ttf]
\stoptypescript
```

然后便可以用 \setupzhfonts 设定该字体：

```
\setupzhfonts[math][this-is-noto-sans-math]
```

如果你需要为某款数学字体作细致入微的配置，那只能是精心研究 ConTEXt 的像天书一样的 Typescript 机制了。我没法讲述这部分内容，因为它们超出了我的能力。

结语

zhfonts 模块是基于 ConTEXt 的传统的 Typescript 机制实现的汉字字体和英文字体的混合。若想见识 Typescript 的恐怖之处，只需编译以下内容的 ConTEXt 源文件：

```
\usemodule[zhfonts]
\starttext
\showzhfonts
\stoptext
```

如果你想改进 zhfonts 模块，文档[25] 介绍了 zhfonts 模块的一些关键技术，希望有所帮助。如果你需要汉字直排功能，可使用黄复雄所写的模块[26]。

16 杂技

前面章节所述内容足以应对风格不甚讲究的文档排版任务，例如一些个人文档，如日记、随笔、读书笔记之类。从学习的角度而言，我也很建议先用 ConTEXt 完成此类任务，从而逐渐熟悉其基本用法。但现实中总有一些非常规的要求。一些特定的问题，其解决方法难以独立构成章节，故而我将其汇总为一章，并且自由发挥，想到什么，便写什么。可将这部分内容视为 ConTEXt 在微观视角下的形态。

16.1 标题

使用 \title 命令排版无编号的一级标题时，默认是另起一页，若不希望如此，可作以下设定：

```
\setuphead[title][page=no]
```

在排版手册或书籍时，若文档分成多章，则章的序号的形式通常是「第 x 章」，若构造该形式的序号，若主语言界面[1]为英文，则可以像下面这样设定章序号的前缀和后缀。

```
\setuplabeltext
  [en]
  [chapter={{第\,},{\,章}}]
```

若主语言界面是中文，就按中文界面设定，如下：

```
\mainlanguage[cn]
\setuplabeltext
  [cn]
  [chapter={{第\,},{\,章}}]
```

上述代码中的「\,」，之前讲过，它能够制造一个较小的间距，比它略宽一些的是「\;」，再宽一些便是空格字符。

如果你是用 zhfonts 模块设定字体，该模块里已经为你设置好了章序号的形式，而且这也是 zhfonts 唯一做的额外的事。

比章（chapter）更高级别的标题是部分（part），如果你想顺便设定「第 x 部分」，只需

```
\setuplabeltext
  [cn]
  [part={{第\,},{\,部分}}]
```

[1] ConTEXt 为一些关键词提供了多语言环境，例如英文的 Figure、Table、Chapter 等关键词，在中文界面下分别是图、表、章等。

上述设定的章序号形式，它无法作用于目录，亦即目录里的章序号依然是默认的单纯数字形式。要设定目录中的章节序号，需要用 \setuplist 命令。所谓目录，本质上是嵌套形式的列表结构，每个层级的列表样式皆可用该命令设定。本文档的目录样式，其中章标题所在层级列表的样式设定如下：

```
\def\ChapterNumber#1{第 #1 章\quad}
% 章标题层级的样式
\setuplist
  [chapter]
  [alternative=a,                % 章标题层级使用目录样式 a
   before={\blank[halfline]},    % 章标题之前留半行间距
   after={\blank[halfline]},     % 章标题之后留半行间距
   style=bold,                   % 设置字体为粗体
   width=fit,                    % 设定章序号的宽度为自适应
   pagenumber=no,                % 不显示页码
   numbercommand=\ChapterNumber]
```

上述代码的关键之处在于，通过 numbercommand 能够传入我们自定义的宏，由后者生成我们所期望的章序号。ConTEXt 的很多样式设定命令支持类似的技法。\setuplist 命令各项参数的含义见文档[27]。

16.2 特定页面

如果你设定了页眉或页脚，默认情况下，它们除了会出现在普通页面上，也会出现在各章首页，但有些文档格式要求各章首页不需要出现页眉和页脚。要达到这个效果，需要做以下设定：

```
\setuphead[title,chapter][header=empty,footer=empty]
```

\setuphead 这类样式设定命令，若多次使用，且每次设定不同的参数，则这些设定的结果是叠加的。

如果你的文档是双面排印，每章首页的前一页可能是空白页，这些页面上的页眉和页脚通常也应该消除。解决方法是，为各章首页定义断页策略，例如

```
% 双面排印
\setuppagenumbering [alternative=doublesided]
% 断页策略
\definepagebreak[mychapterpagebreak][yes,header,footer,right]
% 让段页策略生效
\setuphead[chapter][page=mychapterpagebreak]
```

上述的 \definepagebreak 命令参数，可以解读为：定义断页策略 mychapterpagebreak，第一个参数 yes 表示必须断页。后面三个参数的含义是，在右页（right）上放置页眉和页脚，亦即左页无页眉和页脚。

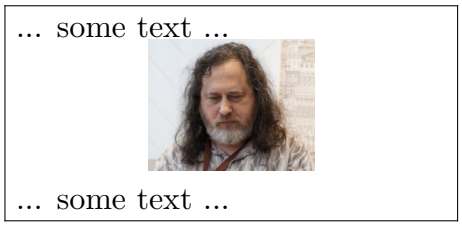
对于各章首页的前页是空页的断页过程，可以理解为，本来是一个页面，现在需要另起一页，前者的内容转移到后者，然后将前者清空，但是可以设定这个转移过程是否要带走页眉和页脚。由于 ConTEXt 的双面排印默认是将每章断页后的内容放在右页的，故而上述代码中的 right 可省略。其他类型的标题的样式，也都支持 page 参数，故而可以用类似的方式为其设定断页策略。

\setuphead 的 page 参数，其值默认是 yes，表示每章都是另起一页，这也是为何 \definepagebreak 的第一个参数是 yes 的原因，因为后者会覆盖 \setuphead 的 page 参数的原有值，故而需要显式提供 yes 以维持每章首页的默认的分页样式。

16.3 奇怪的间距

若只是单纯插入一张图片，令其独占一段且居中，而你不想用 \placefigure 命令，可以像下面这样做，但是所得结果可能是你意想不到的，插图顶部距离上一段太近，而底部又有一些间距。

```
\startTEXpage[width=6cm]
... some text ...\par
\midaligned{\externalfigure[foo.png]}
... some text ...
\stopTEXpage
```



例 16.1　上下间距不对称

这种奇怪的现象，在 **5.6** 节已遇到过了，原因是 \externalfigure 载入的外部图形也是没有基线，结果导致其位置偏上。此时，插图的底线与下一行文字的基线刚好是一行文字的高度，需要让插图下沉半个行高，才能让其上下间距近似对称，见下例。

```
\startTEXpage[width=6cm]
... some text ...\par
\midaligned{\externalfigure[foo.png]}
\blank[-halfline]
... some text ...
\stopTEXpage
```

例 16.2　上下间距对称

\blank 命令能在竖直方向上插入负的间距。如果用 \hbox 将插图包含起来，也可以用 \lower 命令让插图下沉。

\blank 命令是 ConTEXt 特有的。构造竖向间距，更为基本的 TEX 命令是 \vskip，若实现上例等效的负的竖向间距，还需要用 \lineheight 命令获得行高，即

```
\vskip-.5\lineheight
```

构造横向间距的 TEX 基本命令是 \hskip，其用法已在 **3.3** 中涉及，它也支持负的横向间距，例如 a\hskip-1em b 的结果是ba。

16.4 编组

TEX 默认是以一对花括号构造一个编组（Group）。基于编组，你可以将很多字符、插图、表格等元素组织成一个对象，然后将该对象作为参数值传递给 TEX 命令（或宏）。下面以一个简单的宏为例，阐述编组的重要性。

以下代码定义了一个简单的宏 \foo，它接受两个参数，然后给每个参数增加方括号，亦即这个宏的展开结果是它所接受的两个参数分别套上了方括号。

```
\def\foo#1#2{[#1][#2]}
```

如果像下面这样使用 \foo：

```
\foo 路漫漫其修远兮
```

则结果为

```
[路][漫]漫其修远兮
```

如果像下面这样使用 \foo：

```
\foo{路漫漫}{其修远兮}
```

则结果为

```
[路漫漫][其修远兮]
```

通过以上示例，你大概能从直觉上认识到，TEX 会将一个编组视为一个字符，颇似庄子所说的，泰山莫大于秋毫之末。

编组另一个重要的特性是它能构造局部环境，在该环境内所作的任何设定，不会影响编组之外的一切。这个特性，在你需要临时设定一些会影响全局的样式参数时颇为有用。例如，如果你临时想将文本框的颜色设为红色，可以像下面这样做：

```
... {\setupframed[framecolor=red]\inframed{demo}} ...
```

... demo ...

花括号较为单薄，辨识度太低，而且有些情况下无法使用花括号，例如

```
\def\foo{{}
\def\bar{}}
```

也许你会以为 \foo 的展开结果是左花括号，而 \bar 的展开结果是右花括号，但这是不可能的。事实上，\foo 的展开结果是

```
{}
\def\bar{}
```

原因宏定义里出现了未成对的花括号，而 TEX 编译器会努力为其找到配对的花括号。为了解决这类问题，TEX 提供了 \begingroup 和 \endgroup 作为左右花括号的替代。ConTEXt 提供了更为简单的等效命令 \start 和 \stop，故而上述示例可改写为以下的正确形式：

```
\def\foo{\start}
\def\bar{\stop}
```

ConTEXt 的环境命令，例如 \starttext ...\stoptext，本质上就是由 \start 和 \stop 构成的编组。理解了这一点，就可以断定 \placetable 这样的命令，实际上无需用花括号包含 tabulate 这类环境，亦即在文档中插入一个表格，可以像下面这样写。

```
\placetable{标题}
\starttabulate[|c|c|]
\NC 1 \NC 2 \NR
\stoptabulate
```

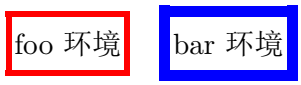
在 **8.3** 节，你已见识过了 \placeformula 省略花括号的写法：

```
\placeformula[formula-foo]
\startformula
\int_0^{+\infty}f(x) {\rm d}x
\stopformula
```

16.5　样式切换

使用 setups 环境能够定义一组样式，便于在文档中根据需要予以切换。下例为盒子定义了两个不同的环境，并演示了环境的切换方法。

```
% foo 样式
\startsetups foo
  \setupframed[framecolor=red, rulethickness=2pt]
\stopsetups
% bar 样式
\startsetups bar
  \setupframed[framecolor=blue, rulethickness=4pt]
\stopsetups
% 切换为 foo 样式
\setup[foo]\inframed{foo 环境}\quad
% 切换为 bar 样式
\setup[bar]\inframed{bar 环境}
```



不过，样式切换也没有太深的玄机，通过定义一些简单的宏，用这些宏封装一些样式设定语句，也能产生类似效用。

16.6 非常规序号

如果一份文档是由多章构成，插图和表格，这些浮动对象的序号默认是每章都会重置，即从 1 开始，且以该章的序号为前缀，例如「图 2.3」，表示第 2 章的第 3 幅插图。有些文档格式要求的是插图和表格的序号贯穿全文，亦即序号从 1 开始，一直递增，直至文档结束，要实现这种效果，可以用以下设定：

```
\setupcaption[figure][way=bytext,prefix=no]
\setupcaption[table][way=bytext,prefix=no]
```

若不想为每个浮动对象的标题设定上述样式，也可以用以下命令统一设定：

```
\setupcaptions[way=bytext,prefix=no]
```

ConTEXt 默认的浮动对象序号的样式是「way=bychapter, prefix=yes」。

插图、表格以及公式的序号，有些文档格式认为以短横线作为间隔符会更好看一些，例如「2-3」。ConTEXt 超人 Wolfgang Schuster 给出的解决方案是

```
\setupcaptions[prefixsegments=chapter,prefixconnector=-]
```

上述设定，可将图片、表格的序号转变为 x-y 形式。至于数学公式，亦可作类似设定，例如：

```
\setupformulas[prefixsegments=chapter,prefixconnector=-]
```

以下代码在局部环境验证了上述方法是否有效。

```
\start
\setupformulas[prefixsegments=chapter,prefixconnector=-]
\placeformula
\startformula
a^2 + b^2 = c^2
\stopformula
\stop
```

$$a^2 + b^2 = c^2 \qquad [16\text{-}1]$$

$$\int_{-\infty}^{x} \frac{dx}{H(x)}$$

有时，一个公式存在不同的形式，有些文档格式要求用字母后缀以示区别，例如 2.3a，2.3b 之类，ConTEXt 的数学环境将这类公式称为子公式，见下例。

```
然后，对可信性 $(BC|D)$ 应用基本函数 $F$，可得：

\startsubformulas
\placeformula[eq:1]
\startformula
(ABC|D) = F\{F[(C|D), (B|CD)], (A|BCD)\}
\stopformula

不过，也可以在一开始认为 $AB$ 是一个命题，这样就可以从另一种顺序推出不同的结果：

\placeformula[eq:2]
\startformula
(ABC|D) = F[(C|D), (AB|CD)] = F\{(C|D), F[(B|CD), (A|BCD)]\}
\stopformula
\stopsubformulas
```

然后，对可信性 $(BC|D)$ 应用基本函数 F，可得：

$$(ABC|D) = F\{F[(C|D), (B|CD)], (A|BCD)\} \tag{16.2.a}$$

不过，也可以在一开始认为 AB 是一个命题，这样就可以从另一种顺序推出不同的结果：

$$(ABC|D) = F[(C|D), (AB|CD)] = F\{(C|D), F[(B|CD), (A|BCD)]\} \tag{16.2.b}$$

如果需要消除字母后缀之前的间隔符号，可以用以下设定：

```
\defineseparatorset[none][]
\setupformulas[numberseparatorset=none]
```

然后，对可信性 $(BC|D)$ 应用基本函数 F，可得：

$$(ABC|D) = F\{F[(C|D), (B|CD)], (A|BCD)\} \tag{16.3a}$$

不过，也可以在一开始认为 AB 是一个命题，这样就可以从另一种顺序推出不同的结果：

$$(ABC|D) = F[(C|D), (AB|CD)] = F\{(C|D), F[(B|CD), (A|BCD)]\} \tag{16.3b}$$

16.7 三段论

ConTEXt 为数学公式提供了 `matrix` 环境，能够像 `tabulate` 表格那样构造矩阵，见下例。在数学公式里，可以用 `\text` 命令构造一个嵌入式的 TEX 环境，有些像抄录（Verbatim）环境里的 TEX 逃逸。

```
\placeformula
\startformula
\startmathmatrix
  \NC \text{若 $A$ 为真，则 $B$ 为真} \NR
  \NC \text{$A$ 为真} \NR
  \HL
  \NC \text{所以，$B$ 为真} \NR
\stopmathmatrix
\stopformula
```

$$\begin{matrix} \text{若 } A \text{ 为真，则 } B \text{ 为真} \\ A \text{ 为真} \\ \hline \text{所以，} B \text{ 为真} \end{matrix} \tag{16.4}$$

不过，matrix 环境只是语法像 tabulate，但二者于本质上是有区别的。tabulate 内容中的每一行可以用 \NC\NR 结尾，例如

```
\starttabulate[|c|]
\NC 1 \NC\NR
\stoptabulate
```

matrix 不可如此，否则会导致矩阵的最后一列是一个空列，例如

```
\placeformula
\startformula
\startmathmatrix
  \NC \text{若 $A$ 为真，则 $B$ 为真} \NC\NR
  \NC \text{$A$ 为真} \NC\NR
  \HL
  \NC \text{所以，$B$ 为真} \NC\NR
\stopmathmatrix
\stopformula
```

$$\begin{matrix} \text{若 } A \text{ 为真，则 } B \text{ 为真} \\ A \text{ 为真} \\ \hline \text{所以，} B \text{ 为真} \end{matrix} \tag{16.5}$$

上例结果中的横线，要比前例略微向右伸出些许，此即矩阵空列所致，而 tabulate 环境无此问题，大概是因为它允许最后一列为空列，以便为表格设定右侧边界线。

16.8 Lua 宏

如果你也觉得 TEX 宏编程是难以掌握的怪物，且觉得能用 Lua 语言为 TEX 增加一些功能要更为容易——在 **5.7** 节里已作基本介绍，那么不妨了解一下 ConTEXt 为此类需

求提供的更为直观的编程接口，下例利用了该接口重新实现了一个可将数字转化为带圈形式的宏。

```
\startluacode
interfaces.implement {
  name = "circled",
  public = true,
  arguments = {"integer"},
  actions = function(x) context('\\char"' .. tostring(2460+x-1)) end
}
\stopluacode
% 测试
\circled{3}\circled{6}\circled{1}
```

③⑥①

上述代码中的 name 是宏名；public 必须设定为 true，表示该接口能以宏的形式调用；arguments 是参数表，integer 表示参数类型为整型数；actions 的值是 Lua 匿名函数，当接口以宏的形式调用时，便会触发该函数。

如果要定义多参数宏，只需依序在 arguments 表里给出类型，例如

```
arguments = { "string", "integer", "boolean", "dimen" }
```

表示接受 4 个参数，类型依序为字符串、整型数、布尔值、尺度。

参数类型也可以是 Hash 表。这类参数通常是用于定义具有选项参数的宏，亦即其参数为方括号形式，见下例。

```
\startluacode
interfaces.implement {
  name = "upper",
  public = true,
  arguments = {"hash"},
  actions = function(x)
    if x.style == "bold" then
      context("{\\bf " .. string.upper(x.text) .. "}")
    else
      context(string.upper(x.text))
    end
  end
}
\stopluacode
% 测试
\framed{\upper[text=demo,style=bold]}
```

DEMO

结语

至此，本文档所介绍的绝大多数 ConTEXt 排版命令，应付常规的技术文档排版任务，绰绰有余。然而，ConTeXt 还有许多更高深的命令，其用法对我而言是超纲了……不过，无论 ConTEXt 提供了多少排版命令，也无法覆盖现实中全部的排版需求，前者是有限的，而后者总是会发生变化。庄子说，指穷于为薪，火传也，不知其尽也。若将排版命令视为薪柴，那么 ConTEXt 之火如何传续呢？答案是，编程。

17 编程

在第 3 章的开始，简单论述了 TEX 与 ConTEXt 以及 LATEX 的关系。TEX 系统的精妙之处在于，它是可编程的，是开放的，它像是给我们提供了一些积木以及一些简单的组合规则，使得我们也能具备建造像 LATEX 和 ConTEXt 这些上层建筑的能力。我并不是说，你也应当再搭建某种上层建筑。实际上即使在日常使用 LATEX 或 ConTEXt 时，TEX 的编程机制依然非常有用，甚至你可以从这些工作里逐步获得建造上层建筑的能力。

我在之前的一些章节里也曾粗略提及了 TEX 的编程机制，诸如自定义一些简单的宏，也对 ConTEXt LMTX 所支持的 Lua 编程机制作了一些介绍。本章尝试对这两种编程机制再作一些专门的探讨。不过，我并没有足够的精力和动力成为 TEX 编程专家，我所能做的仅仅是为你在使用 ConTEXt 时开启一个更为神秘且宏大的视角。

17.1 宏

也许你用的输入法不太方便打出直角引号。在 TEX 里，只需要定义一个简单的宏便可让此事无需劳烦输入法。例如

```
\def\zhqt#1{「#1」}
\zhqt{被中文直角引号包含的文本}
```

「被中文直角引号包含的文本」

对于 TEX 普通用户而言，宏最大的用处是提供简写。是的，你无需编程，便已经能得到宏编程机制的好处了，甚至可以将宏的名字定义为中文：

```
\def\引号#1{「#1」}
\引号{被中文直角引号包含的文本}
```

「被中文直角引号包含的文本」

像 \zhqt 这样的宏称为有参数的宏。在向宏传递参数时，花括号实际上并非必须，例如

```
\def\zhqt#1{「#1」}
\zhqt被中文直角引号包含的文本
```

「被」中文直角引号包含的文本

输出的结果里，只有「被」字会被直角引号包含。

对于有参数的宏而言，用花括号构造的编组，会被 TEX 视为一个整体，与单个字符等效。下面的例子定义了带有 2 个参数的宏，它不仅能给文字加上中文直角引号，而且还能将文字渲染为指定的颜色。

```
\def\zhqt#1#2{「\color[#1]{#2}」}
\zhqt{red}{被中文直角引号包含的文本}
```

「被中文直角引号包含的文本」

在定义有参数宏时，若参数两侧存在某些符号时，则在使用宏时也必须提供相同的符号，这种符号称为宏参数的定界符。例如

```
\def\zhqt[#1]#2{「\color[#1]{#2}」}
\zhqt[red]{被中文直角引号包含的文本}
```

至此，也许你觉得这一切似乎平平无奇。我们定义一个有参数的宏，在使用它时，不过是向其喂入一些文本，再等待它吐出我们期待的文本——该过程称为「宏的展开」——，然而当你尝试让宏「吐出」自身时，就会遇到一件奇怪的事情。例如

```
\def\foo{\foo}
\foo
```

当 TEX 在试图展开 \foo 时，它便会陷入无休止的状态，因为它会对展开所得 \foo 再度展开，亦即 TEX 总是努力将一个宏展开至无所展开时为止。这件奇怪的事情实际上已经触及到编程的本质——递归。我建议你不要轻易尝试上述示例，除非你知道如何关闭一个正在努力工作的程序。

17.2 变量

若想将 TEX 从上一节中 \foo 无尽展开的地狱里解救出来，必须通过寄存器，保存某种状态，从而有机会确定何时截止 TEX 对 \foo 的展开。

早期的 TEX 提供了 256 个可用于保存整数的寄存器。ConTEXt 所用的 luametaTEX 已将这类寄存器的数量拓展为数以万计。可以用 \newcount 命令获得一个尚未被使用的整数寄存器，例如

```
\newcount\mynum
\mynum = 42
{\bf 宇宙的秘密是 \the\mynum。}
```

宇宙的秘密是 42。

可以对整数寄存器中的数据做数学运算，例如加法或减法：

```
\newcount\mynum
\mynum = 42
\advance\mynum by 1  % \the\mynum + 1
\advance\mynum by -1 % \the\mynum - 1
{\bf 宇宙的秘密是 \the\mynum。}
```

宇宙的秘密是 42。

若你有幸学过汇编语言，此刻应该不难感受到一些熟悉的气息。除了整数寄存器，TEX 还有其他类型的寄存器，最为常用的有尺寸寄存器与盒子寄存器。例如，在 ConTEXt 经常使用的一个命令 \textwidth，它实际上是尺寸寄存器，其中存储的值是正文版面的宽度。至于盒子寄存器，在 **10.5** 节便已对其有所领略。

现在可不必关心 TEX 究竟有哪些寄存器以及它们如何使用。不过，若你已经熟悉某种或某些编程语言，我建议你将 TEX 的寄存器视为特定类型的变量。例如整数寄存器，你可以理解为是整型变量。尺寸寄存器，可以理解为实数变量。盒子寄存器，可以理解为结构体或对象。使用 \def 定义的一些无参数的宏，可理解为字符串变量——上一节定义的 \foo 宏是个例外。

17.3 条件

TEX 提供了 \ifnum 命令，可用于比较两个整数的大小或是否相等，例如

```
\newcount\mynum
\mynum = 42
\ifnum\mynum = 42 宇宙的秘密是 \the\mynum。\else\relax\fi
```

宇宙的秘密是 42。

上述代码里所做的比较是，如果 \mynum 寄存器里存储的值为 42，则输出宇宙的秘密，否则什么都不做——可将 \relax 命令理解为它什么都不做……或者无为。若将 = 更换为 < 或 > 则分别可比较 \mynum 中的值小于或大于 42。也许将上述示例中换为更为正经的编程语言来表达，能让你看得更清楚一些，例如用 C 语言：

```
int mynum = 42;
if (mynum == 42) {
        printf("宇宙的秘密是 %d。", mynum);
} else ;
```

TEX 也提供了其他形式的条件命令，诸如 \ifdim、\ifodd、\ifhmode 等，这些条件命令还是在需要它们出现的时候再探讨其用法吧。

17.4 函数

现在，我们有能力拯救从 \foo 里解救 TEX 了，例如

```
\def\foo#1{%
  \ifnum #1 < 42 \advance #1 by 1 \foo{#1}%
  \else\relax%
  \fi%
}
\newcount\TEST
\TEST = 0
\foo\TEST
宇宙的秘密是 \the\TEST。
```

宇宙的秘密是 42。

注意，宏定义里每一行末尾的注释符通常是有必要的，它的作用是防止在宏的展开结果里引入额外的换行符。此刻的 \foo 像是一个可控的递归函数。

通过上述示例，想必你已领悟，在使用有参数的宏时，若其参数是一个命令，也可以不用编组，因为 TEX 会将一个命令视为一个整体传给宏。为了让你更为深刻理解宏的本质抑或它的凶险，我对上例 \ifnum 语句略作改动，如下

```
\ifnum #1 < 42 \advance #1 by 1\foo{#1}%
```

你可以先思考一下结果会当如何，然后验证一下。

倘若你愿意将有参数的宏视为函数或计算过程，也许能感受到 TEX 颇为有趣的地方——宏既可以表达数据，也可以表达代码或运算。在正经的编程语言里，大概只有 Lisp 系的语言拥有类似神韵。

17.5 Lua

在 Lua 语言里——可能你不懂这门编程语言，但它很简单——，也可以定义一个函数，使之类似于 **17.1** 节中 \foo 宏，如下

```
function foo()
    foo()
end
```

ConTEXt 允许我们在 luacode 环境里编写 Lua 代码，例如

```
\startluacode
function foo()
    foo()
end
\stopluacode
```

ConTEXt 也允许我们通过宏去调用 Lua 函数，若调用上述的 foo 函数，只需

```
\ctxlua{foo()}
```

倘若你动手操练了上述代码，应该能看到 ConTEXt 会给出类似以下编译错误：

```
token call, execute: [ctxlua]:3: stack overflow
stack traceback:
... ... ...
```

这是栈溢出错误。因为 foo 函数是无穷递归，而 Lua 解释器在受到无有穷尽的 foo 函数的折磨时，会选择玉石俱焚，而不是像 TEX 那样受困于这种折磨无法解脱。

我们也可以通过让 foo 函数讲述宇宙的秘密的方式，拯救 Lua 解释器。例如

```
\startluacode
function foo(x)
    if x == 42 then
        context("宇宙的秘密是 %d。", x)
    else
        foo(x + 1)
    end
end
\stopluacode
```

然后只需像下面这样调用 foo 函数便可得到宇宙的秘密：

```
\ctxlua{foo(0)}
```

ConTEXt 命令 \ctxlua 与 Lua 代码里的 context 函数，你可以将其理解为 ConTEXt 世界与 Lua 世界的通道的两端。有了这一通道，你可以用 Lua 语言编写程序，然后在 ConTEXt 世界里调用它们。

结语

也许你本身已是一位训练有素的软件开发者，熟悉许多编程语言，即便如此，当你第一次瞥到 TEX 的宏编程机制时，大概印象是，TEX 作为排版软件可谓优秀，但作为编程语言可谓丑陋不堪。当你写的一个宏出错时，也许会觉得自己面临了一场雪崩，没有一片雪花是无辜的。好在 LuaTEX 已问世十余年了，现已进化为更为轻盈的 LuaMetaTEX。在你厌恶宏编程时，总是可以考虑用 Lua 语言编写程序，而 TEX 的宏可以作为界面。不过，我的能力只是为你打开一个通向 TEX 幽深世界的入口。至于你在这个世界里能够走多远，会遭遇什么——那是我看不到的，希望会是一段传奇。

跋

历时二十日，终于写完了。

你现在学会 ConTEXt 了吗？

我还没有学会。

这份文档所介绍的 ConTEXt 知识或许尚不及它全部功能的百分之一，且 ConTEXt 依然在发展着。学无止境，以生之有涯，随知识之无涯，这样不好。弱水三千，仅取一瓢饮，会好一些。

我应该去外面走走了，不然春天又悄悄离去了。

2023 年 4 月

用了大约十二天，更新了这份文档。大概有三分之一的内容是重写的，诸如命令行环境、列表、表格、插图、数学公式、幻灯片、参考文献等内容，基本是重新写了一遍，最后一章「杂技」是新增的，其余章节在局部上有所增删。

此次更新，原本的计划里是想加入几节 Lua 与 ConTEXt 协作的内容，但是后来又放弃了，原因是这部分内容对于常规的文档排版而言并非必须。不过，我在知乎网站上专门为 ConTEXt 创建了一个专栏，其地址在

「https://zhuanlan.zhihu.com/c_1932476336884156084」

该专栏不仅有一些关于 ConTEXt 的文章，也有一些文章试图以实践者的方式讲述 MetaPost 语言。

2025 年 9 月

这次修订工作，主要是修正了文档里的一些书写错误，并对之前部分颇为草率的字句予以细化，主体内容基本上未有增减。在此需要特别感谢 **@Explorer-cc** 的悉心匡正，否则我不会有太多动力去完成这些工作。

对文档的版权略作了修改，补充了一些简单的条款，以满足 Debian 社区对 TeXLive 软件包的授权方面的严谨需求，而我本人对这些条款所能起到的作用实际上一无所知……要感谢 AI 比我懂这方面的事，所补充的条款也是它帮我写的。

第 **7** 章的示例中的插图，我换成了我童年时的偶像——多啦 A 梦。

第 **1** 章的结语部分有较大修改之处。今年风靡一时的养龙虾事件，想必已经让很多人意识到了，命令行并非落后的玩意，反而颇为有用。此外，第 **16** 章的结语也做了修改，为新增加的第 **17** 章做好了铺垫。

第 **17** 章是一份简单的 TEX 编程入门指南，原本我是计划用它呈现 TEX 编程在排版工作中的一些应用，但是在经过一些尝试后，我觉得还是让它尽量简单一些，仅立意于打消你对 TEX 编程的畏惧。

2026 年 4 月

参考文献

[1] Hagen H. ConTeXt LMTX[J]. TUGboat，2019，40(1)：34 – 37.

[2] Hagen H. ConTEXt Mark IV: an excursion，2017.
http://www.pragma-ade.nl/general/manuals/ma-cb-en.pdf

[3] 李延瑞. 听说你讨厌 Bash，2018.
https://zhuanlan.zhihu.com/p/1918388156346107184

[4] 李延瑞. Bash 指南，2018.
https://segmentfault.com/a/1190000017229619

[5] 李延瑞. Bash 之丑陋，2025.
https://zhuanlan.zhihu.com/p/1918533431886853968

[6] Hagen H. Faking text and more，2016.
http://www.pragma-ade.nl/general/magazines/mag-0007-mkiv.pdf

[7] ConTEXt wiki. List of Unicode blocks，2020.
https://wiki.contextgarden.net/List_of_Unicode_blocks

[8] 李延瑞. 现代还有学习 MetaPost 的必要吗？，2025.
https://zhuanlan.zhihu.com/p/1932476924283839013

[9] 李延瑞. MetaPost 的气质，2025.
https://zhuanlan.zhihu.com/p/1932739070951416739

[10] 李延瑞. MetaPost 涂鸦式入门，2025.
https://zhuanlan.zhihu.com/p/1932894857468314368

[11] ConTEXt wiki. Tables，2025.
https://wiki.contextgarden.net/Tables

[12] ConTEXt wiki. Tabulate，2025.
https://wiki.contextgarden.net/Tables/Tabulate

[13] ConTEXt wiki. Natural tables with TABLE，2025.
https://wiki.contextgarden.net/Tables/Natural_tables_with_TABLE

[14] ConTEXt wiki. Mathematics，2025.
https://wiki.contextgarden.net/Mathematics

[15] ConTEXt wiki. Typing，2025.
https://wiki.contextgarden.net/Text_blocks/Environments/Typing

[16] ConTEXt wiki. Verbatim with LuaTEX，2025.
https://wiki.contextgarden.net/Text_blocks/Environments/Typing/Verbatim_with_LuaTeX

[17] 李延瑞. 舞弄 LPEG，2023.
https://zhuanlan.zhihu.com/p/619453867

[18] 李延瑞. 源码凸显，2023.
https://segmentfault.com/a/1190000043405105

[19] Hobby J D. MetaPost Manual，2022.
https://www.tug.org/docs/metapost/mpman.pdf

[20] ConTEXt wiki. \definesectionblock，2025.
https://wiki.contextgarden.net/Command/definesectionblock

[21] ConTEXt wiki. Table of contents，2025.
https://wiki.contextgarden.net/Document_structure_and_headlines/Table_of_contents

[22] ConTEXt wiki. Setups，2025.
https://wiki.contextgarden.net/Input_and_compilation/Setups

[23] ConTEXt wiki. Bibliography and citations，2025.
https://wiki.contextgarden.net/References_notes_and_floats/Bibliography_and_citations

[24] ConTEXt wiki. Mathematics Fonts，2025.
https://wiki.contextgarden.net/Mathematics/Fonts

[25] 李延瑞. zhfonts 备忘录，2023.
https://segmentfault.com/a/1190000043490118

[26] 黄复雄. ConTEXt LMTX 中文竖排插件，2022.
https://blog.xiiigame.com/2022-02-15-ConTeXt-LMTX中文竖排插件

[27] ConTEXt wiki. \setuplist，2025.
https://wiki.contextgarden.net/Command/setuplist

索引

a

alignment 环境 11
alignment，单行对齐
  \leftaligned
  \midaligned
  \rightaligned 10

b

baseline, 基线 10
bbwidth，bbheight：MetaFun 宏 68
编组 16
\blank 22
\bodyfontsize 10
buffer 环境 82

c

center 属性 24
center：MetaFun 宏 68
\char 31
\colored 54
context 函数 32
\ctxlua 32

d

\definecolor 54
\defineconversion 31
\definefallbackfamily 19
\definefont 14
\definefontfamily 18
\definenumber 43
\defineenumeration 50
\definepapersize 77
\definesymbol 30
\definetabulate 40
\definetype 55
\definetyping 55
\dorecurse 11

e

\environment 25
\externalfigure 42

f

\fakewords 8
floatcombination 环境 45
字体切换
  \tf, \bf, \it, \bi 16
MetaPost for 循环语法 73
\framed 57

g

\getmarking 94
\getnumber 43

h

行内公式
  \im
  \dm
  \m 48
\hbox 30
\headnumber 94
横向列表 39
\hfil，\hfill 和 \hss 57
\hskip 15
换行命令 8

i

if/else：MetaPost 条件语法 72, 73
\ifodd 94
image：MetaPost 宏 67
\incrementnumber 43
indenting
  \indentation 49
  \noindent 9
\index 91
\inframed 57

itemize 环境 27

k
\kern 94
空格 30

l
\lastuserpage 83
layer 74
列表项引用 28
\line 56
lines 环境 8
\lower 30
luacode 环境 32

m
MPcode 环境 63
\mainlanguage 38

o
overlay 59, 63, 74

p
picture: MetaPost 类型 67
\placecontent 87
\placefigure 45
\placeindex 91

q
\quad 43

r
randomized 63
reusableMPgraphic 环境 62

s
\setbox 60
\setlayout 78
\setscript 15
\setupbackgrounds 75
\setupbodyfont 18
\setupcaption 38
\setupenumeration 50
\setupfootertexts 83, 95
\setupframed 58
\setuphead 24
\setupheadertexts 93
\setupheads 22
\setupinterlinespace 10
\setuplayout 78
\setuplinenumbering 52
\setuppagenumbering 25
\setuppapersize 77
\setuptabulate 40
\setuptype 55
\setuptyping 55
standardmakeup 环境 84
standard makeup 84
\sym 31

t
TEXpage 环境 7
tabulate 环境 34
textext: MetaFun 宏 66
\textreference 90
\textwidth 42
thetextext: MetaFun 宏 66, 67
\type 52
typing 环境 52

u
uniqueMPgraphic 环境 29, 62
useMPgraphic 环境 62
\userpagenumber 83

v
vardef: MetaPost 变量宏定义 69

x
xysized: MetaFun 宏 72

y
颜色 54