ConTEXt 蹊径

李延瑞

2023 年 3 月 8 日

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，我觉得是因为 ConTEXt 入门文档甚少，其中中文文档则更为罕见。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]，对于具备一些英文阅读能力的人，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[3]，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，但因俗务缠身，后来兴趣又有漂移，便不了了之。

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 。

目　录

第 1 章　不怕命令行

无论是安装还是使用 ConTEXt，皆需要对命令行环境有所了解。原本未有介绍这方面知识的计划，但是考虑到我正在写一份世上最好的 ConTEXt 入门文档，便有了些许动力。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 的需求为要。倘若对命令行环境已颇为熟悉，可直接阅读 **1.5** 和 **1.6** 节。

1.1 任务

使用命令行环境，在文件系统中，创建一个名为 foo 的目录，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」，执行该脚本，查看其输出。

1.2 Windows 命令行

Windows 用户似乎畏惧甚至厌憎命令行环境，甚至很多人认为命令行环境是早已被淘汰的上个世纪的产物，因此要教会他们如何使用命令行环境，通常会有些麻烦，我当勉力为之。

在 Windows 系统中打开一个命令行窗口，有很多种方法，其中最快的应当是使用如图 **1.1** 所示快捷键「Win + R」，打开「运行」对话框，在其中输入「cmd」后点击「确定」按钮或单击「Enter」键，即可打开与图 **1.2** 类似的命令行窗口。

图 1.1　「Win + R」组合键

图 1.2　Windows 命令行窗口

所有要执行的命令皆是在「>」符号后输入，因而该符号名曰「命令提示符」。命令提示符左侧的内容是一个文件目录，名曰「当前目录」或「工作目录」。在命令行环境里，任何命令皆是在某个目录中执行的。在开始尝试输入命令之前，请将输入法切换为英文输入状态。

在命令行窗口中输入命令「d:」，然后单击「Enter」键执行该命令，可将当前目录切换为 D 盘。在命令行环境里，输入命令后，必须单击「Enter」键，方能使命令得以执行。

执行命令「`md foo`」，可在 D 盘根目录创建目录 foo，然后执行命令「`cd foo`」，将当前目录切换为 foo，即 `D:\foo`。

诸如 `D:\` 和 `D:\foo` 这样的表示形式统称为路径，更为准确地说，是绝对路径。有相对路径吗？有。在 `D:\foo` 中执行命令「`cd ..`」，便可将当前目录返回上一级，即 `D:\`。在 `D:\foo` 中执

行命令「cd ..\foo」，则当前目录不会发生变化，因为 D:\foo 的上一级目录的子目录 foo 依然是 D:\foo 。形如..\foo 这样的路径便是相对路径。

在当前目录 D:\ 中执行命令「cd ..」，当前目录会发生什么变化？不会发生任何变化，因为 D:\ 是 D 盘的最顶层目录，亦即它的上一级目录为空。

掌握了上述命令，便可在 Windows 文件系统中畅游无阻了，然而我们的任务尚未完成。在 Windows 系统中，该任务可描述为，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 脚本即批处理文件——扩展名为「.bat」的纯文本文件。在制作这份批处理文件之前，需要了解「echo」命令的基本用法。

「echo」命令，如同我们对着幽深的山谷呼喊而产生回声的过程，它读取一段文本，然后在命令行窗口中原样输出。例如，

```
D:\foo> echo 不怕命令行
不怕命令行
```

似乎「echo」命令是一个什么都不会做的命令，这样的命令有什么用呢？它有一个用途是，利用输出重定向，将一些内容写入文本文件。例如，

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

上述命令通过命令输出重定向符「>」将「echo」原本会输出到命令行窗口的文字「@echo off」重定向为输出到文件 D:\foo\foo.bat 。倘若该文件事先并不存在，上述命令会自动创建它。

现在，使用「echo」命令向文件 D:\foo\foo.bat 增加一行文字：

```
D:\foo> echo echo 不怕命令行 >> foo.bat
```

注意，向指定文件追加内容，需要使用「>>」，倘若使用「>」，则文件原有内容会被全部替换。

现在，略有纪念意义的时刻到了，你可能已经创建了人生中第一份 Windows 批处理文件，执行它吧！

```
D:\foo> .\foo.bat
不怕命令行
```

注意，上述命令使用了相对路径的第二种形式，路径中的「.」表示当前目录。事实上，在 Windows 命令行环境里，这种形式的相对路径可以忽略，亦即上述命令可以改为

```
D:\foo> 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 系统默认先从当前目录中搜索文件，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，执行以下命令可以查看：

```
D:\foo> echo %PATH%
```

顺便指出，这是「echo」命令的另一种用途。

使用以下「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：

```
D:\foo> 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起效。在 Windows 开始菜单里的搜索栏，输入「cmd」并单击「Enter」键提交，然后鼠标右键单击如图?? 所示的搜索结果，在弹出的菜单中选择「以管理员身份运行」。以这种方式开启的命令行窗口方可执行上述的「setx」命令。

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

```
D:\foo> c:
C:\> cd windows\system32
C:\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/ ，藉此避免让层峦叠障的 Windows 窗口截图占据本章太多篇幅。

1.3 Linux 终端

在 Linux 系统中，命令行环境叫作「终端（Terminal）」。终端中可以运行多种 Shell，最为常见的是 Bash Shell。这些 Shell 往往大同小异。终端可以嵌入窗口运行，也可以在没有窗口的情况下运行[1]，我们通常使用的是前者。

Linux 系统发行版众多，每个发行版有其不同的打开终端窗口的方式且因 Linux 用户往往对终端较为熟悉，因而不再赘述如何打开终端窗口。现在，假设终端窗口已经开启。首先，进入 $HOME 目录，创建子目录 foo：

```
$ cd $HOME
$ mkdir foo
```

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

```
$ 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，让它像程序一样运行：

[1] 通常情况下，可使用快捷键「Ctrl + Alt + F1～F6」切换到无窗口的终端。「Ctrl + Alt + F7」也对应一个终端，它通常被 Linux 窗口系统占用。

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

运行 foo.sh：

```
$ ./foo.sh
不怕命令行
```

将「$HOME/foo」添加至系统「PATH」变量并使之生效：

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

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

```
$ cd /tmp
$ foo.sh
不怕命令行
```

倘若想对 Bash Shell 有更多一些的了解，可以阅读我的拙作「写给高年级小学生的《Bash 指南》」[4]，它介绍了 Bash 的诸多有趣之处，也许会让你喜欢命令行，而不仅仅是不怕它。

1.4 macOS 终端

我没用过 macOS 系统，在该系统中打开终端窗口，可按 macOS 官方帮助文档介绍的方法进行，详见：

https://support.apple.com/zh-cn/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/mac

至于 macOS 终端的用法，因 macOS 和 Linux 皆为 Unix-like（类 Unix）系统，二者终端环境的用法近乎相同，唯一有些区别的是，从 macOS Catalina 版开始，macOS 默认使用的 Shell 不再是 Bash，而是 zsh。因此，在设置「PATH」变量时，命令需要变为

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

1.5 安装 ConTEXt LMTX

ConTEXt 的最新版本是 ConTEXt LMTX，目前尚在开发和试验阶段，功能虽不稳定，但对于入门而言并无妨碍。从 https://wiki.contextgarden.net/Installation 下载对应操作系统类型的 ConTEXt LMTX 安装包，按照该网址的相关介绍进行安装即可。授人以鱼，不如授人以渔，安装过程所需要的全部知识皆已在上文介绍完毕。不过，我总担心 Windows 用户无法安装成功，下文以 Windows 64 位系统为例，给出完整的安装过程。

首先，从链接 http://lmtx.pragma-ade.com/install-lmtx/context-win64.zip 获得面向 Windows 64 位系统的安装包 context-win64.zip 并假设将其放在 D:\ 解开，所得目录为 context-win64，其结构如下：

```
D:\context-win64
├─── bin
│    ├─── mtx-install.lua
│    ├─── mtxrun.exe
│    └─── mtxrun.lua
├─── installation.pdf
├─── install.bat
└─── setpath.bat
```

将目录 context-win64 改名为 context，然后打开命令行窗口，依次执行以下命令：

```
> d:
> cd context
> install.bat
```

批处理文件 install.bat 能够自动从网络上下载 ConTEXt LMTX 的所有文件，并将其安装在 D:\context 目录。安装时长取决于网络下载速度。由于服务器在境外，文件下载速度缓慢，可能需要很久方能安装完毕。当安装过程结束后，目前需要再次执行 install.bat，将 ConTEXt LMTX 更新到最新版本。这些都是 https://wiki.contextgarden.net/Installation 未告诉我们的。此外，即使安装过程中断，再次运行 install.bat 可继续安装，而不会导致前功尽弃。

验证 ConTEXt LMTX 是否安装成功的方法是，在 D:\context 目录中执行以下命令：

```
> 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，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 的文本文件编译成 PDF 文件了。

1.6 投机取巧

倘若因为网络不畅甚至不通等问题无法成功安装 ConTEXt LMTX，可以考虑下载我打包的整个 ConTEXt LMTX 环境，目前仅有 Windows 64 位和 Linux 64 位版本，下载地址为：

- Windows amd64 版本：https://pan.baidu.com/s/1r5gMqsk-peGQhOmcp_dNlw?pwd=eejf
- Linux amd64 版本：https://pan.baidu.com/s/1iBHN-8Zroyd9rGOjRpQrYg?pwd=r2hp

以 Windows 系统为例，从上述链接获得压缩包，以 `D:\` 作为解压缩目录，解压后可得 `D:\context` 目录，然后按照上一节所述方式修改系统 `PATH` 变量即可得到完整的 ConTEXt LMTX 环境，日后依然可执行 `D:\context\install.bat` 对其进行更新。我已将该过程录制为视频，网络链接为 https://www.bilibili.com/video/BV1q84y1c7Je/ 。

1.7 小结

在一个桌面操作系统中，命令行环境能完成的工作，图形界面通常也能完成，但对于有些任务，命令行环境比图形界面更为高效，反之亦然，原本无需厚此薄彼，但尺有所短，寸有所长，Windows，Linux 和 macOS 也是如此。

- Windows 系统的命令行环境，前身是 MS DOS 系统，长期未得进化，现在用起来，捉襟见肘。现代的 PowerShell 更为适用，但因 ConTEXt 目前不能直接在 PowerShell 环境里完成安装，故而上文未曾言及。

- macOS 无论图形界面还是命令行环境皆胜于 Windows 和 Linux，但其缺点是在不违法的情况下，它只能运行于苹果公司的计算机，费用不菲。

- Linux 系统的图形界面不及 Windows 和 macOS，但命令行环境优于 Windows[1]，与 macOS 相当，但 Linux 最显著的特点是，它能帮助每一个使用计算机的人，让他深刻觉察自己以往所宣称的那样热爱自由是否真实。

[1] 自 Windows 10 开始，微软在 Windows 系统中构建了 Linux 子系统（WSL），允许用户在 Windows 系统中使用 Linux 命令行环境，但是该环境无法利用 Windows 和 Linux 的图形界面功能。

第 2 章　沙盘游戏

先不要急于学习使用 ConTEXt 写论文，出专著，虽然它极为擅长这类任务，但是我还是希望先将它当成一个可爱的朋友，慢慢去熟悉它，而不是功利主义视角下的工具。

2.1 新手村

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，可以使用任何一个文本编辑器来做此事。foo.tex 文件内容如下：

```
\startTEXpage[offset=1cm]
Hello \ConTeXt!
\stopTEXpage
```

执行以下命令[1]，调用 `context` 程序，可将 foo.tex 编译成 PDF 文件 foo.pdf：

```
$ context foo.tex
```

在使用 `context` 命令编译 ConTeXt 源文件时，可省略文件扩展名「`.tex`」，因此下面的命令与上述命令等效：

```
$ context foo.tex
```

现在，想必你已经敏锐地觉察到了，凡是开头为反斜线「\」开头的英文单词，都是排版命令，的确如此。

排版命令 `\startTEXpage...\stopTEXpage` 称为 `TEXpage` 环境，我将其称为新手村。这一对排版命令所包含的内容，例如「`Hello \ConTeXt!`」会被排版于一个恰好能包含它的矩形排版空间。`frame` 参数用于控制边框是否开启，其值为 `off` 或省略对它的设定时，关闭边框。`offset` 参数可用于对排版空间进行扩大或缩小，如示例 2.1 所示，排版空间从中心向四周被扩大 2.5 mm。

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

示例 2.1　新手村

之所以称 `TEXpage` 环境为新手村，是因它足够简单，将其用于观察大多数排版命令的效果时无需关心版面的天头、地脚、订口、翻口、版心、页码等元素的设定。

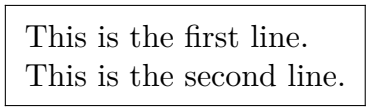
[1] 从本章开始，一直使用 Linux 风格的命令提示符 `$`。

2.2 伪文字

ConTEXt 有一个可视化模块，提供了伪造单词的排版命令，使用该命令可以生成一些黑色的长短随机的矩形块，可将它们当成文字，从而在沙盘上可以更加随心所欲一些。

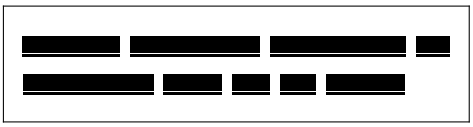
如示例 **2.2** 所示，排版两行文字，每行由 5 个单词组成。示例 **2.2** 给出了伪文字的效果。同样是两行文字，每一行由 3～5 个单词构成。

```
\usemodule[visual] % 加载可视化模块
\startTEXpage[frame=on,offset=2.5mm]
This is the first line.\\ % 强制换行
This is the second line.
\stopTEXpage
```



示例 2.2 两行真实文字

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



示例 2.3 两行伪文字

2.3 注释

在示例 **2.2** 的源代码中，「%」及其后面的同一行文字，是 TEX 注释文本，它们会被 context 程序忽略，故而不会出现在排版结果中。注释是为了便于人类阅读 ConTEXt 源代码而存在的。

2.4 换行符

在示例 **2.2** 和 **2.3** 的源代码中，「\\」是强制换行命令，倘若将其删除，即使在源代码中将文字分为两行，例如

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

排版结果依然是一行，而且换行符会被 context 程序视为一个空格，不妨亲自动手试验一下。

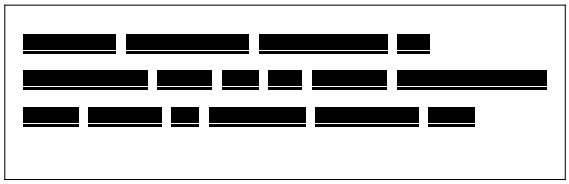
在使用换行符的情况下，即使两行文字在源代码中处于同一行，例如

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

排版结果依然是两行。

\crlf 也能用于对文字强制进行断行。倘若不希望对每一行都输入断行命令，也可以考虑使用 lines 环境，请参考示例 **2.4**。

```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm]
\startlines
\fakewords{3}{5}
\fakewords{4}{7}
\fakewords{5}{9}
\stoplines
\stopTEXpage
```

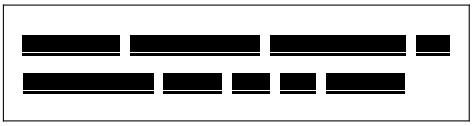


示例 2.4　排版多行文本

2.5 分段

观察示例 **2.5**，虽然排版结果依然是两行，但实际上它是两段。\par 是分段命令。

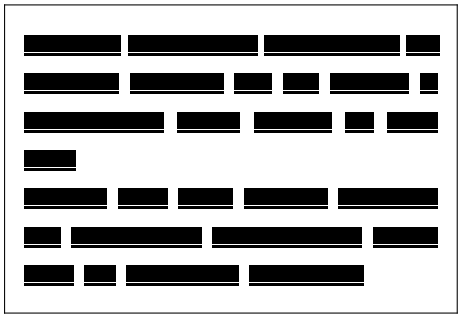
```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm]
\startlines
\fakewords{3}{5}\par
\fakewords{3}{5}
\stoplines
\stopTEXpage
```



示例 2.5　分段

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。示例 **2.6** 使用空行进行分段，并通过限定「新手村」的宽度为 6 cm，从而在促狭的空间里展示了多行伪文字构成的段落。

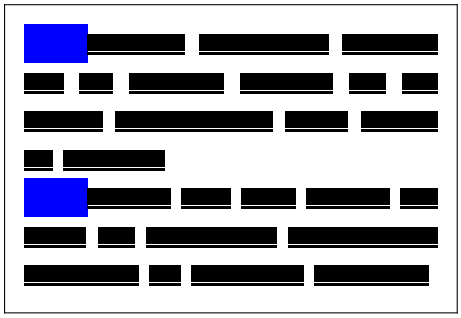
```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\fakewords{9}{15}

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



示例 2.6　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，示例 **2.7** 实现了该需求，排版结果中段落开头的蓝色矩形区域表示文字缩进后产生的空白区域。尺寸 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。还有一个常用的尺寸单位 ex，它是英文字母 x 的高度。

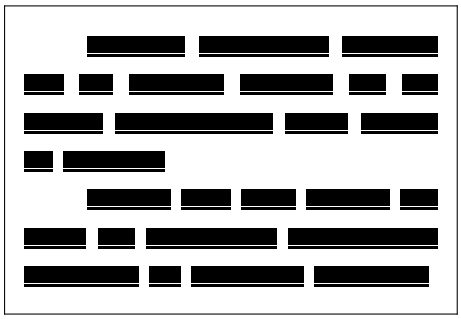
```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\setupindenting[first,always,2em]
\fakewords{9}{15}

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



示例 2.7 设置段落首行缩进

根据文档「Faking text and more」[5]的提示，可将段落缩进区域的颜色定义为白色，从而可避免蓝色矩形块对视觉的干扰，见示例 **2.8**。

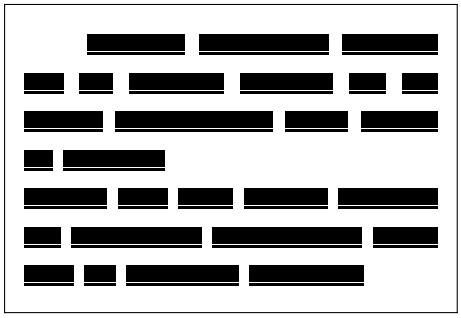
```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\setupindenting[first,always,2em]
% 将缩进区域的颜色由默认的蓝色改为白色
\definecolor[fakeparindentcolor][white]
\fakewords{9}{15}

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



示例 2.8 消除段落缩进色块

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`。示例 **2.9**，第二段消除了第二段文字的首行缩进。

```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\setupindenting[first,always,2em]
\definecolor[fakeparindentcolor][white]
\fakewords{9}{15}

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

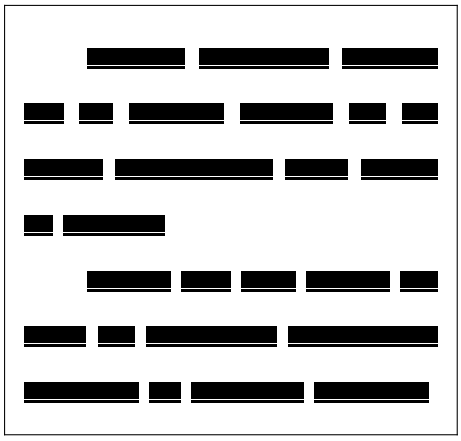


示例 2.9 消除第二段的首行缩进

2.6 行间距

ConTEXt 默认的段落内各行文字的间距是 2.8 ex，约等于 ConTEXt 默认的正文字体大小 12 pt。可使用 `\setupinterlinespace` 对行间距进行调整。如示例 **2.10** 将行间距设为 1.75 倍默认行距。

```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\setupindenting[first,always,2em]
\definecolor[fakeparindentcolor][white]
\setupinterlinespace[1.75]
\fakewords{9}{15}

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



示例 2.10　多行文本构成的段落

不过，我发现示例 **2.10** 的行间距设定语句若放在 \startTEXpage 语句之前则无效[1]，也许是当前的 ConTEXt LMTX 版本存在 bug。

不过，还有一种段落内行间距设定方法。等到掌握第 **3** 章中讲述的设定 ConTEXt 字体的方法，在你已经清楚正文字体的大小时，例如 11 pt，可采取以下方式设定 1.75 倍行距：

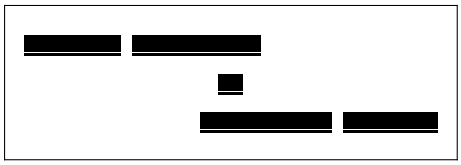
```
\setupinterlinespace[line=19.25pt]
```

之所以是 19.25 pt，是因它等于 1.75×11 pt。上述代码设定的是一行文字的最大高度。在后文中，本文档皆使用这种方式设定段落内行间距。

2.7　对齐

有时，希望将一行文字居左、居中或居右放置，可分别使用 \leftaligned，\midaligned 和\rightaligned 进行排版，请参考示例 **2.11**。

```
\usemodule[visual]
\startTEXpage
  [frame=on,offset=2.5mm,width=6cm]
\leftaligned{\fakewords{1}{2}}
\midaligned{\fakewords{1}{2}}
\rightaligned{\fakewords{1}{2}}
\stopTEXpage
```

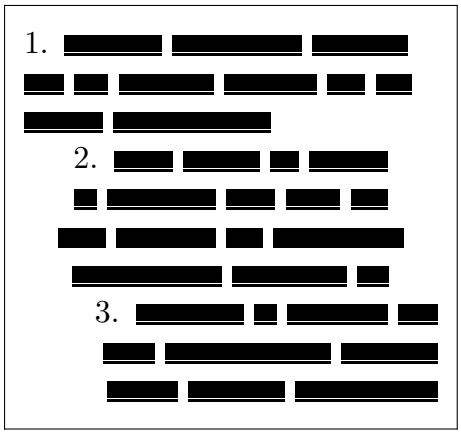


示例 2.11　多行文本构成的段落

如果是一段文字需要居左、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。示例 **2.12** 展示了段落的三种对齐形式。

[1] 在 **4.3** 节中会介绍 \starttext ... \stoptext，直接以默认行距倍数的方式设置行间距，若设定语句位于 \starttext 之前则同样无效。

```
\usemodule[visual]
\startTEXpage[frame=on,offset=2.5mm,width=6cm]
\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
\stopTEXpage
```

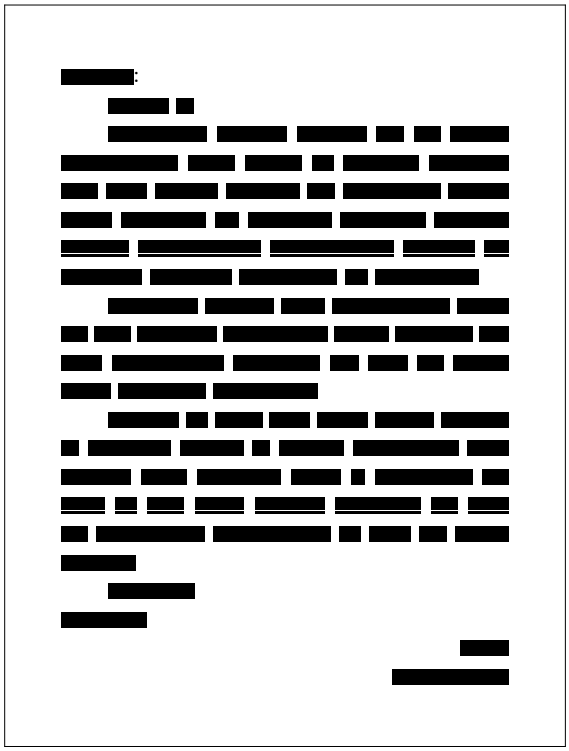


示例 2.12 段落对齐

2.8 写一封谁也看不懂的信

现在，我们已经有能力用伪文字排版一封谁也看不懂内容的书信了，见示例 **2.13**。也许现在你很想用汉字给朋友写封信，并告诉他，这封信的排版是你亲手用 ConTEXt 制作的。我劝你暂时最好放弃这个想法，写英文书信是完全可以的。

```
\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
\fakewords{20}{50}\par
\fakewords{20}{50}\par
\fakewords{20}{50}\par
\fakewords{1}{1}\par
\noindent\fakewords{1}{1}\par
\rightaligned{\fakewords{1}{1}}\par
\rightaligned{\fakewords{1}{1}}
\stopTEXpage
```



示例 2.13 一封谁也看不懂的信

2.9 小结

恭喜你，走出了新手村。在下一章学会安装和使用汉字字体，便可以闯荡 ConTEXt 世界了。

第 3 章　汉字

让 TEX 系统支持汉字，这个任务曾经很容易对新手学习和使用 TEX 的热情造成毁灭性打击。后来，随着 XETEX 和 LuaTEX 等现代 TEX 引擎的出现，这项任务的难度已经大幅降低了。ConTEXt LMTX 所用的 TEX 引擎是 luametaTEX，对 LuaTEX 进行了精简并作为它的继任者而继续发展。可能你现在并不理解这些术语，但也不必担心，如同驾驶一辆 ConTEXt 汽车，无需知道它的引擎（发动机）是哪个工厂生产的，其性能参数又是如何。

3.1　安装字体

首先，需要找到一款汉字字体文件，否则只有学仓颉，自己造字。倘若你或你的朋友的计算机中装有 Windows 系统，可从如图 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。有了它们，足以胜任常规的排版工作。

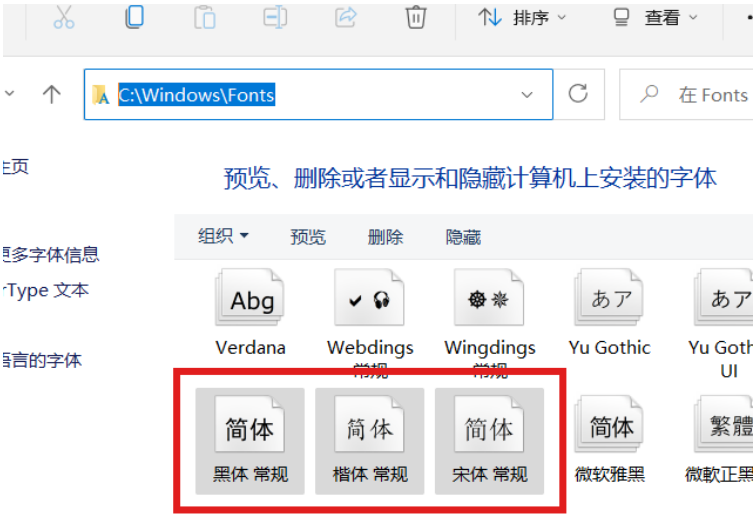


图 3.1　`C:\Windows\Fonts`

在开始安装字体之前，需要给出一个术语，TEX 根目录。以 Windows 系统为例，若 ConTEXt 安装在 `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。同理，对于 Linux 和 macOS 系统，若 ConTEXt 安装在 `$HOME/context` 目录，则 `$HOME/context/tex` 为 TEX 根目录。

为了便于描述，从现在开始，以虚构的类 Unix 环境变量风格的 `$TEXROOT` 指代 TEX 根目录，不再使用具体路径，因为后者严重依赖于操作系统和用户偏好而缺乏一致性。此外，路径中的目录间隔符也统一使用类 Unix 风格进行表示，即使用「/」而非 Windows 风格的「\」，例如 `$TEXROOT/texmf` 表示 TEX 根目录的子目录 `texmf`。

向 ConTEXt 安装宋体（simsun.ttc）、黑体（simhei.ttf）和楷体（simkai.ttf）的具体过程如下：

(1) 将上述字体文件复制到 `$TEXROOT/texmf-local/fonts/truetype/msfonts` 目录，若该目录不存在，则自行创建；
(2) 执行「`context --generate`」命令，刷新 ConTEXt 的文件数据库：

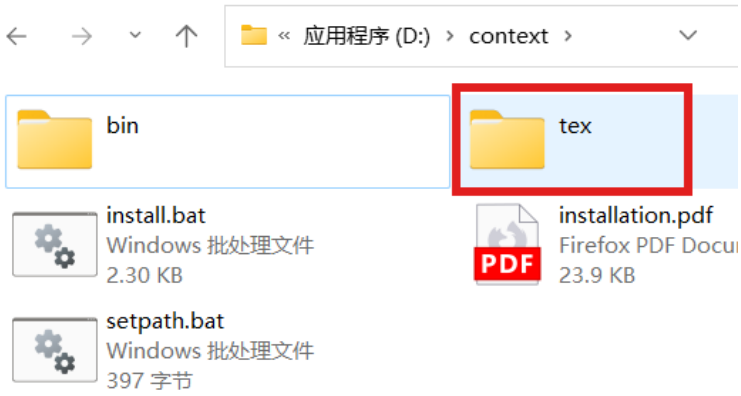


图 3.2 TEX 根目录

(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，可以通过查询字体文件名确认字体是否安装成功，例如查询 simhei.ttf：

```
$ mtxrun --script fonts --list --file simhei.ttf
familyname    weight    style     width     variant    fontname    filename      ...

simhei        normal    normal    normal    normal     simhei      simhei.ttf
```

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。上述命令也可用于查询 simkai.ttf 的信息，但是无法用于查询 TTC 字体 simsun.ttc，可能是因 TTC 字体文件较为特殊，但无论如何，这是 `mtxrun` 的 `fonts` 脚本的一个 bug，希望日后能被修复。

查询 simsun.ttc 对应的字体名，可以使用以下命令：

```
$ mtxrun --script fonts --list --all simsun
identifier        familyname    fontname    filename      subfont    instances

nsimsunnormal     nsimsun       nsimsun     simsun.ttc    2
nsimsunregular    nsimsun       nsimsun     simsun.ttc    2
simsun            simsun        simsun      simsun.ttc    1
simsunnormal      simsun        simsun      simsun.ttc    1
simsunregular     simsun        simsun      simsun.ttc    1
```

`subfont` 栏表明 simsun.ttc 里包含了两种字体，一种是 `simsun`（宋体），另一种是 `nsimsun`（新宋体）。此外，请对 `identifier` 和 `familyname` 栏略加留意，**3.6** 节需要这些信息。

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 字体的定义与使用

示例 **3.1** 演示了如何在新手村里为 ConTEXt 定义大小为 12 pt 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[songti][name:simsun at 12pt]
\definefont[heiti][name:simhei at 12pt]
\definefont[kaiti][name:kaiti at 12pt]
\songti 潜龙勿用。\\
\heiti 见龙在田，利见大人。\\
\kaiti 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。

示例 3.1　三种汉字字体

需要注意的是，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解析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，因此不应急切宣判 ConTEXt 不适合处理中文文档。我的计算机 CPU 是 `Intel i5-4300U @ 1.90GHz`。这份文档写至此处，已有 20 页内容，其源文件单次编译时间不到 3 秒。我感觉 ConTEXt 处理中文文档，并不是很慢，但是应当承认，肯定是比基于 pdfTEX或 XHTEX等引擎的 TEX 要慢一些。

3.3 中文断行

现在尝试用宋体字排版一段中文，见示例 **3.2**。结果表明，仅仅能在 ConTEXt 中使用汉字字体是不够的，因为 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，是一个无法分隔的单元。

```
\startTEXpage[frame=on,width=5cm,offset=4pt]
\definefont[songti][name:simsun at 12pt]
\songti
潜龙勿用。见龙在田，利见大人。%
君子终日乾乾，夕惕若厉，无咎。
\stopTEXpage
```

潜龙勿用。见龙在田，利

示例 3.2　中文段落无法断行

需要注意示例 **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）。倘若将该示例中的注释符去掉，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，从而让 ConTEXt 觉得，它面对的是两个较长的单词，它们之间可以断行，结果见示例 **3.3**，可以断行，但属于误打误撞。

```
\startTEXpage[frame=on,width=5cm,offset=4pt]
\definefont[songti][name:simsun at 12pt]
\songti
潜龙勿用。见龙在田，利见大人。
君子终日乾乾，夕惕若厉，无咎。
\stopTEXpage
```

潜龙勿用。见龙在田，利
君子终日乾乾，夕惕若厉，

示例 3.3 中文段落无法断行

现在，我们可以灵机一动，既然换行符被 TEX 引擎视为空白字符从而使得 ConTEXt 误打误撞对汉字进行了一次断行，那么倘若在汉字之间手工插入空格字符，ConTEXt 能否实现汉字断行呢？答案是，可行，见源文档显现了空格字符（以 ␣ 表示）的示例 **3.4**，只是空格的存在，导致排版结果中的汉字分布较为蓬松。

```
\startTEXpage[frame=on,width=5cm,offset=4pt]
\definefont[songti][name:simsun␣at␣12pt]
\songti
潜␣龙␣勿␣用。␣见␣龙␣在␣田，␣利␣见␣大␣人。
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。

示例 3.4 中文段落手工插入空格进行断行

位于 ConTEXt 底层的 TEX 系统提供了粘连（Glue）机制，我们可以用它定义给定宽度且有些许弹性的空格。例如，定义一个宽度为 0，最大可伸展 2 pt 且不可收缩的粘连：

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

然后用该粘连代替空格，插入到汉字之间，便基本可实现汉字断行，结果见示例 **3.5**，需要注意的是，TEX 会自动忽略排版命令之后尾随的空格，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。

```
\startTEXpage[frame=on,width=5cm,offset=4pt]
\definefont[songti][name:simsun␣at␣12pt]
\def\foo{\hskip␣0pt␣plus␣2pt␣minus␣0pt}
\songti
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。
\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。

示例 3.5 中文段落手工插入粘连进行断行

由于没有人愿意像示例 **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]
```

上述过程大费周章，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，从而避免了像下面这样输入繁琐的排版命令：

```
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，便可以自己创造一个可与 ConTEXt 媲美的宏包。现在，已经隐隐知道了ConTEXt 的一些真相了吧。

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

示例 3.6　孙悟空的信

从现在开始，在提供示例时，为了节省篇幅，我有时会省略一部分经常重复使用的代码，仅给出关键代码以突出重点，在有兴致动手试验这些代码时，需要你自己动手添加被省略的代码。

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，可分为四族：衬线（Serif）、无衬线（Sans Serif）、等宽（Monospace）以及数学。前三个字族基本上又分别细分为正体（Regular 或 Normal）、粗体（Bold）、斜体（Italic）和粗斜体（BoldItalic）。

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。示例 **3.7** 演示了如何切换各种字体以及 `{...}` 的用法。在默认情况下，TEX 引擎将一对花括号所包含的内容视为一个整体，TEX 术语称之为编组（Group）。编组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:

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

示例 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。x 级比正文字体小，a 级比正文字体大。示例 **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

示例 3.8 字体大小的 7 种级别

此外，还需要注意的是，尺寸单位 em 的含义。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，这是针对当前所的字体环境而言的。例如，在示例 **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。这是因为，在示例 **3.9** 中，设定段落首行缩进时，em 的值是基于 ConTEXt 默认的正文字体尺寸确定的，其值为 12 pt，而在示例 **3.9** 中，em 的值是基于我们自定义的字体 `\songti` 的尺寸确定的，其值为 9 pt。

```
\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。

示例 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。

示例 3.10　段落缩进距离是 18 pt

3.6　定义汉字字族

使用 `\definefontfamily` 可将你喜欢的一些汉字字体定义为字族，用以代替 ConTEXt 的默认字族，然后使用 `\setupbodyfont` 启用你定义的字族。示例 **3.11** 将 10.5 pt 的宋体作为正文默认字体[1]。注意，字族的定义和启用，需在新手村之外进行，否则无效。不过，`\setupindenting` 却只能在新手村之内起效，这让我有些不解，但是这就是新手村的规矩，只能接受。

```
\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。

示例 3.11　定义正文字体并启用

示例 **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，我

[1] 10.5 pt 可对应 MicroSoft Word 中的五号字。

们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。

由于我们的 ConTEXt 环境里的 nsimsun 字族只有正体 nsimsunregular，没有其他字体，因此倘若在示例 3.11 中，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。
```

ConTEXt 会因找不到相应的字体而排出空页。若要解决该问题，需要使用 \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]
```

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。现在我们可以对它说，再见，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，可以扬眉吐气了，因为你已经能够自由地在 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 给出了选项，在排版结果中，是上面那行文字里的英文单词美观，还是下面那行？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 可用一种字体中的部分字形替换另一种字体的相应字形，其用法与 \definefontfamily 类似，只是需要在参数中指定字形覆盖范围。示例 3.12 使用 lmroman10 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 nsimsun 字族中每

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

种字体（包括替补字体）的相应字形。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][lmroman10]
  [range={basiclatin,latinsupplement},force=yes]
```

指定了 Unicode 码位区间 [0x0000, 0x00FF] 中的字形作为替换字形。Unicode 码位区间的名字见文档「List of Unicode blocks」[6]。

```
\definefallbackfamily
  [myfonts][rm][lmroman10]
  [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

示例 3.12　字形替换

3.8 小结

当你读到此处，我想悄悄告诉你，ConTEXt 里最难的知识，你已基本掌握。现在，你完全可以在新手村的春天里扬眉吐气了。

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，若有兴趣，不妨拜读。我可以很诚实地说，该手册我只读过寥寥数页。由于你已经安装了 ConTEXt，这份手册就在你的 ConTEXt 环境里，执行以下命令可以找到它：

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

请顺便浏览一番这份手册所在的目录里的其他 PDF 文件吧。

第 4 章　让文章有它该有的样子

走出新手村，我们的第一个任务是，让一篇文章有它该有的样子。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……还要有页码，当然最重要的是，要有段落——新手村里我们的老朋友。对于大多数文学创作工作者而言，这些已经足够了，这就是一篇文章该有的样子。至于科技工作者通常所需要的列表、表格、插图、数学公式等形式，是在文章该有的样子的基础上进一步的构造，现在不必急于理会。

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，排版命令依序是

```
\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，排版命令依序是

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

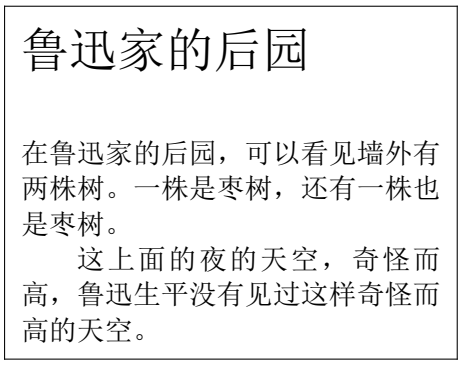
应该不难看出两种标题各自的次级标题降级规律。不建议使用级别层次太深的标题，否则会让读者觉得身陷迷宫，通常前三级标题足够使用。若是写一篇散文，标题只需要用 `\title`。若是写一本小说，只需用 `\title` 制作书名，用 `\chapter` 制作章名。

4.2　写一篇散文

示例 **4.1** 虚构了一篇散文，我只给出了关键的源代码——为它构建完整的可编译的源代码，对你应该不是难事。该示例设置了段落首行缩进距离，并且使用 `\title` 创建了文章标题。目前尚无作者的名字，因为它会导致你无法看到 ConTEXt 标题之后的段落，首行是不缩进的，这是西文的排版习惯。在使用标题前，只需对标题作如下设定，便可强迫 ConTEXt 必须对每个标题后的第一个段落进行缩进。

```
\setupheads[indentnext=yes]
```

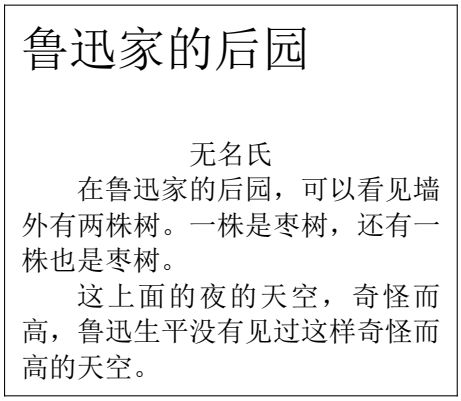
```
\setupindenting[first,always,2em]
\title{鲁迅家的后园}

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。

这上面的夜的天空，奇怪而高，鲁迅生平
没有见过这样奇怪而高的天空。
\stopTEXpage
```



示例 4.1　散文示例 1

现在可以为文章增加作者信息了，虽然他叫无名氏，见示例 **4.2**，只是作者距离正文太近了。不要尝试使用一些空行去增大该距离，因为 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。

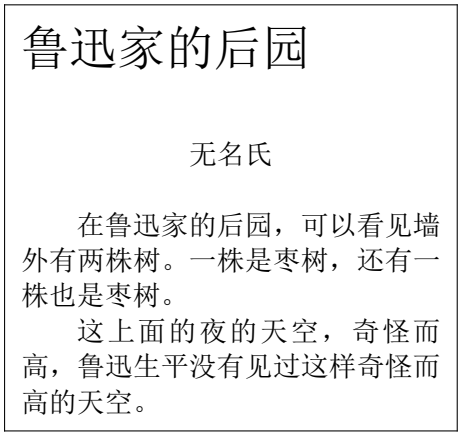
```
\setupheads[indentnext=yes]
\setupindenting[first,always,2em]
\title{鲁迅家的后园}
\midaligned{无名氏}
% 省略了正文内容
\stopTEXpage
```



示例 4.2　散文示例 2

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，通常可以使用 \blank 命令。例如，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]。

```
\midaligned{无名氏}
\blank[line]
% 省略了正文内容
```



示例 4.3　散文示例 3

倘若需要将标题居中，而非默认的居左，只需使用 \setuphead单独为 \title 设定样式：

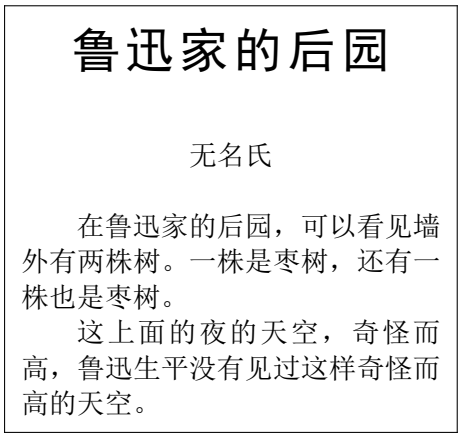
```
\setuphead[title][align=middle]
```

若汉字字族里已经设定了粗体，也可以将标题的样式设为粗体，并指定它的大小级别：

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

示例 4.4 将上述设定综合起来，排版结果已经中规中矩了，只是标题里的汉字的分布有些疏松，这是 \setscript[hanzi] 命令在汉字之间插入的粘连的伸长特性被 ConTEXt 激活导致的，而它们之所以被激活，大概是 ConTEXt 过于追求精确文字居中对齐而导致的，令 \setuphead 的参数 align 的值为 {middle,broad} 可以让 ConTEXt 在文字居中对齐方面宽松一些[7](p86)，结果可让汉字的分布变为紧密，见示例 4.5。

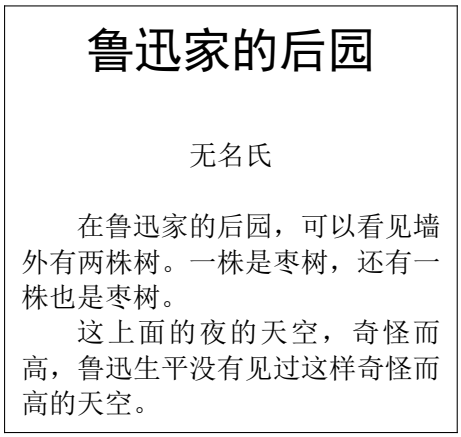
```
\setupheads[indentnext=yes]
\setuphead[title][style=\bfc,align=middle]
\setupindenting[first,always,2em]

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



示例 4.4 散文示例 4

不知 ConTEXt 从哪个版本起，有了一个新的对齐方式 center，它与 {middle,broad} 等价。请记住这个知识，因为以后可能会经常需要设定居中对齐。

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

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



示例 4.5 散文示例 5

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 ConTEXt 排版命令了。事实上，真正的 ConTEXt 世界用起来要比新手村更为简单，只需使用 \starttext ...\stoptext 环境取

代新手村即可，另外，一切设置排版样式的命令皆可放在 \starttext 之前。在 text 环境里，我们通常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，使用 context 程序进行编译。

```
% 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]
% 行距: 1.5 倍的正文字体尺寸, 即1.5 * 10.5pt
\setupinterlinespace[line=15.75pt]
% 正文
\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。

这上面的夜的天空，奇怪而高，我生平没有见过这样的奇怪而高的天空。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。

... ... ...
\stoptext
```

4.4 内容与样式分离

或许你现在已经对每次排版时，会担心日后需要重复输入许多代码，它们主要用于设定排版所用的字体，标题的对齐方式、字体的大小粗细、段落缩进、行间距等。无需有此担心，因为任何一种 TEX 都支持排版内容与样式的分离。

如图 **4.1** 若将 **4.3** 节中的代码中位于 \starttext 之前的部分保存为文件，例如 foo-env.tex，然后将这部分代码之后的所有代码也保存为文件，例如 foo.tex。让 foo-env.tex 和 foo.tex 位于同一目录，然后在 foo.tex 的开头添加

```
\input foo-env
```

最后，编译 foo.tex，所的结果必定与 **4.3** 节中的代码的编译结果相同。\input 是 TEX 层面的命令，它的作用载入指定的扩展名为.tex 的文件，但扩展名可省略。ConTEXt 提供了 \input 等效且用法相同的命令 \environment，以体现所载入的文件仅涉及排版样式的设定。

图 4.1 内容与样式分离

一旦熟悉了如何实现排版内容与样式的分离，在使用 ConTEXt 排版愈发复杂的内容时，你的样式文件 foo-env.tex 的内容便会日益丰富。长此以往，你总是能攒出一些样式文件，供不同的文档形式使用。排版内容总是千变万化，但样式却总是寥寥数种且极易与他人分享，这便是 TEX 的优点，但前提是，你需要清晰地理解你所设定的任何一个排版样式。事实上，这也正是学习任何一种 TEX 系统的乐趣所在。

4.5 页码

如果你亲自动手编译了 **4.4** 的 foo.tex，应当能看到，页眉是有页码的，如图 **4.2** 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，这并不合乎中文的排版习惯，需要对页码进行样式设定。

图 4.2 页码

首先，文章标题所在页面，通常不需要页码。该要求，只需在标题样式将页眉和页脚置空，即

```
\setuphead
 [title]
 [header=empty,
  footer=empty,
  % 应该还有其他设定吧
  ...=...]
```

然后，修改页码投放位置，例如放在页脚右侧：

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

将上述设定酌情添加到样式文件里，然后编译 ConTEXt 源文档，查看其效果吧。日后，倘若是排版书籍，还需要对标题和页码样式的设计作更多的调整。

4.6 小结

现在，你已经可以用 ConTEXt 写日记或随笔了。倘若动手尝试了 `\chapter`，你甚至能用 ConTEXt 写一本文集，只是风格过于朴素。若想让排版结果更为精致，ConTEXt 博大精深，通常总有途径能够实现你的想法，前提是你需要更加用心。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」，引用于此，与君共勉。

第 5 章　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 Todo List

在偷偷使用 ChatGPT 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

示例 5.1　待办事项

现在你已经学会了列表的用法了，剩下的，仅仅是设定它的样式。此外，你也学到了如何写大写的罗马数字，至于小写的，将 `\Romannumerals` 换成 `\romannumerals` 即可。

5.2 无序号列表

示例 **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。该示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，可更换另一种列表项符号。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

示例 5.2　改变列表项符号

5.3 有序号列表

将无序号列表的数字参数换为 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

示例 5.3　有序号列表

有时，需要序号形式是带括号的数字，可作以下设定：

```
\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

示例 5.4　数字带括号的有序号列表

其中「stopper=」是将参数 stopper 置空，达到的效果是消除列表项序号的西文句号后缀「.」。

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 自定义符号列表

你可能会觉得 ConTEXt 为无序号列表提供的列表项符号无法体现你的气质，ConTEXt 说，Do it yourself! 下面我用 MetaPost 代码绘制了一个小正方形，并令其边线略微受到随机扰动，然后将该图形定义为列表项符号：

```
\startuseMPgraphic{foo}
  path p;
  p := fullsquare scaled 8pt randomized 1pt;
  draw p withpen pencircle scaled 2pt
         withcolor darkred;
\stopuseMPgraphic

\definesymbol
  [10][{\lower.2ex\hbox{\useMPgraphic{foo}}}]
\startitemize[10]
\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

示例 5.5　自定义符号的无序号列表

现在你不懂上述代码的具体细节也无妨，观其大略即可，知道有一种可以画画的语言，叫 MetaPost，它的代码可嵌入 ConTEXt 环境作为插图使用，便已足够。

对于有序号列表，有时会需要使用带圈的数字作为序号。虽然 Unicode 有相应的带圈字符的码位，例如 ①，②……对应的 TEX 命令是

```
\char"2460, \char"2461……
```

但是这些带圈数字都是文字，并非数字。带圈的数字，无非是数字外面画个圆圈。画圆圈，用 MetaPost 做此事，完全是不费吹灰之力，然后借助 ConTEXt 的 overlay 机制[<overlay>]，将圆圈作为 inframed[<inframed>] 的背景，再用 inframed 套住列表项序号即可，详见

```
\startuseMPgraphic{foo}
  path p;
  p := fullcircle scaled 12pt;
  draw p withpen pencircle scaled .4pt
         withcolor darkred;
\stopuseMPgraphic
\defineoverlay[rsquare][\useMPgraphic{foo}]
\def\fooframe#1{%
  \inframed[frame=off,background=rsquare]{#1}%
}

\defineconversion[foo][\fooframe]
\startitemize[foo][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

示例 5.6 数字带圆圈的有序号列表

你可能依然不知道上述代码具体细节，不必着急，以后会经常和它们打交道，逐渐便可熟悉。

5.5 间距调整

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

示例 5.7 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

示例 5.8　消除列表项之间的空白

5.6 小结

真正让你的领导失望并抱怨的应该是，他想要一份 Microsoft Word 文件，而你提交给他的却是只能看不能随手改动的 PDF 文件。

第 6 章　参考文献

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。

6.1 BIBTEX

大多数 TEX 系统在排版参考文献时，皆需依赖 bibtex 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 现在不需要依赖 bibtex 程序，它已自身内部实现了 bibtex 的全部功能。本文档在谈及 BIBTEX时，主要指其文献数据格式。

BIBTEX 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，其数据格式为

```
@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，其数据格式为

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

现在，为了学习 ConTEXt 的参考文献排版功能，可将上述文献数据保存为一份文本文件，例如 foo.bib，将该文件作为文献数据文件。

在 foo.bib 相同目录下，建立 ConTEXt 源文件，例如 foo.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 foo.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 texbook* (Vol. 1993), Addison-Wesley Reading, MA, 1986.

6.2 文献列表样式

上一节示例中的文献列表是 ConTEXt 默认样式，除此之外，还有其他三种预定义的文献列表样式：apa，aps 和 chicago，可在 `\placelistofpublications` 之前使用 `\usebtxdefinitions` 进行切换。例如使用 aps 样式，

```
... ... ...
\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 texbook*, Vol. 1993 (Addison-Wesley Reading, MA, 1986).

对于预定义样式，可以进行调整。例如，缩小文献序号后的空白间距，

```
\setupbtxlist[apa][distance=.5em,width=fit]
```

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 texbook*, Vol. 1993 (Addison-Wesley Reading, MA, 1986).

6.3 自定义文献列表样式

也许 ConTEXt 提供的参考文献列表样式并不符合你的需求，因此你会忍不住想自己定义一种样式。对此，我的看法是，文献列表样式当由期刊编辑部或专著出版商负责定义，个人无需如此刻意。不过，倘若你正是此类机构工作人员，需要为 ConTEXt 定义符合自己单位要求的文献样式，下文仅能为你提供一个简单的示例，希望对你有所帮助。

现在，假设我们要定义一种名字叫作 foo 的文献列表样式。首先，需要按照 ConTEXt 的文件命名约定，建立两份文件，一份是 publ-imp-foo.lua，一份是 publ-imp-foo.tex。

在 publ-imp-foo.lua 里写入以下内容：

```
local specification = {
  name      = "foo",
  categories = {},
}
local categories = specification.categories
categories.book = {
  required = { "title" },
  optional = {"author", "editor",
              "year",
              "edition", "series", "volume", "number",
              "address", "publisher",
  },
}
return specification
```

上述代码是 Lua 代码，定义了一个表。ConTEXt 可通过该表明白 BIBTEX 文献数据里哪些信息是必须的，哪些信息是可选的。

在 publ-imp-foo.tex 文件中写入以下内容

```
\startbtxrenderingdefinitions[foo]
\definebtx
  [foo]
  [default=default,specification=foo]

\definebtxrendering
  [foo]
  [specification=foo,numbering=yes]
\stopbtxrenderingdefinitions
```

上述代码定义了 foo 环境及其默认样式。此外，ConTEXt 实际上是通过上述代码获知你在 publ-imp-foo.lua 文件里定义的数据表。

下面在 publ-imp-foo.tex 文件为期刊论文定义文献列表样式：

```
\startsetups btx:foo:list:article
  \btxdoif {author} {
    \color[blue]{\btxflush{author}}\btxperiod\btxspace
  }
  \btxdoifelse {title} {
     \color[red]{\btxflush{title}}\btxperiod\btxspace
  }{
     No title\btxperiod\btxspace
  }
  \btxdoif {journal} {
    \color[magenta]{\btxflush{journal}}\btxcomma\btxspace
    \btxflush{year}, \nospace
    \btxdoifelse {volume} {
      \btxspace
      \btxflush{volume}
      \btxdoif {number} {
        \ignorespaces
        \btxleftparenthesis
        \btxflush{number}
        \btxrightparenthesis
      }
    }{
      \btxdoif {number} {
        \btxlabeltext{default:number}
        \btxspace
        \btxflush{number}
      }
    }
    \btxdoif {pages} {
     \nospace\btxcolon\btxspace\btxflush{pages}
    }
    \btxperiod
  }
  \removeunwantedspaces
\stopsetups
```

这是本文档截止至此最为复杂的 TEX 代码，但是所表达的内容非常简单，仅仅是针对期刊论文设定文献列表应当出现作者、文章名称、期刊名称、时间、卷号、期号以及页码等信息。其中，\btxflush 命令可将 ConTEXt 从 BIBTEX 数据格式中解析出的信息输出到排版结果中；\btxcomma，\btxperiod 之类的命令皆为西文逗号，句号等标点符号；\btxspace 是西文空

格；\btxdoif 之类的命令是判断 ConTEXt 对 BIBTEX 数据格式的解析结果中是否包含某项。

在 publ-imp-foo.lua 和 publ-imp-foo.tex 文件同一目录下，建立测试文件，例如 foo.tex，其内容为

```
\usebtxdataset[foo.bib] % 加载 foo.bib
\starttext
Knuth 发明了文学编程语言 WEB\cite[knuth-1984-literate]。
\blank
\usebtxdefinitions[foo]
\placelistofpublications
\stoptext
```

用于验证上述定义参考文献类表样式 foo 是否可用。结果为

Knuth 发明了文学编程语言 WEB[1]。

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

6.4 小结

参考文献的样式，我总觉得是学术界拿来唬人的东西。写一篇论文，即使附上几十篇参考文献，用 ConTEXt 列表进行排版，与写文章内容的所耗费的时间相比，完全可以忽略不计。然而在学术界，参考文献如何排版，却成了一门学问。倘若你半个小时依然未能学会如何为 ConTEXt 定义参考文献样式，我建议随便选一个 ConTEXt 预定义的样式使用。待文章最终定稿后，再手工用 ConTEXt 列表将参考文献制作成你需要的样式。

第 7 章　数学环境

TEX 原本是 Knuth 专为排版数学类书籍而开发的，亦即排版数学公式是 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。

7.1 两种模式

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，意思是有钱人都精通数学。例如 `$a^2 + b^2 = c^2$` 便是正文模式的数学公式，其排版结果为 $a^2 + b^2 = c^2$。TEX 数学公式的行间模式是放在一对双美元符号之间，例如

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

其排版结果为

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

但是，在 ConTEXt 里，行间模式建议用以下形式的语法

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

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

至于原因为何，想必你已经知道了，ConTEXt 对 `\startformula ... \stopformula` 环境的行间公式默认提供了居中对齐以及前后空白支持。为什么 ConTEXt 不对 `$$...$$` 提供类似的支持呢？因为 ConTEXt 需要以一种统一的方式控制行间公式前后的空白大小，例如消除行间公式前后空白，只需

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

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

若基于 `$$...$$` 标记则很难实现该方式。

7.2 公式编号

如同插图和表格，行间数学公式也有一个放置命令 \placeformula。例如

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

$$\int_0^{+\infty} f(x)\,\mathrm{d}x \tag{7.1}$$

ConTEXt 行间公式支持引用，例如

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

类似插图和表格，使用 \in[...] 进行引用。例如 \in[math-example]，结果为「7.1」。

7.3 定理和证明

ConTEXt 未提供可直接用于排版数学定理和证明的命令，但是我们可以借助枚举环境定义它们。我们对枚举环境事实上并不陌生，因为早已见识过它的一个特例：列表。

使用 \defineenumeration 可以定义一种新的枚举特例。例如

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

然后便可使用该特例：

```
\starttheorem
凡人皆有一死，凡人皆须侍奉。
\stoptheorem
```

结果为

定理 1

凡人皆有一死，凡人皆须侍奉。

接下来，让定理序号与定理内容的第一行处于同一行，让版面更加紧凑（serried）：

```
\defineenumeration[theorem][text=定理,alternative=serried]
\starttheorem
凡人皆有一死，凡人皆须侍奉。
\stoptheorem
```

定理 2　凡人皆有一死，凡人皆须侍奉。

现在，只需将定理的宽度设为 broad 或 \textwidth，便可让定理编号和定理内容不至于如此隔阂，顺便将定理内容的字体切换为粗体：

```
\defineenumeration[theorem][text=定理,alternative=serried,width=broad,style=\bf]
\starttheorem
凡人皆有一死，凡人皆须侍奉。
\stoptheorem
```

结果为

定理 3　凡人皆有一死，凡人皆须侍奉。

由于枚举环境皆支持引用，因此上述定义的定理可免费继承该功能。例如

```
\starttheorem[千面神定理]
凡人皆有一死，凡人皆须侍奉。
\stoptheorem

如定理 \in[千面神定理] 所述……
```

定理 4　凡人皆有一死，凡人皆须侍奉。

如定理 4 所述……

类似地，也基于枚举环境定义证明，只需去掉序号，并在证明结尾居右放置 □ 符号。例如

```
\defineenumeration
  [proof]
  [text=证明,alternative=serried,width=broad,
   number=no,closesymbol={$\square$}]
\startproof
因为人的生命是有限的，为人民服务是无限的。我们应当将有限的生命投入到无限的为人民服务中去。
\stopproof
```

证明　因为人的生命是有限的，为人民服务是无限的。我们应当将有限的生命投入到无限的为人民服务中去。 □

7.4 小结

现在你已经基本学会了 ConTEXt 数学公式排版。与 ConTEXt 数学排版专家相比，你缺乏的可能主要是如何熟练地输入具体的数学公式。若想在这方面能有所精进，可参考 ConTEXt Wiki 的「Math」页面[8]。

第 8 章　插图

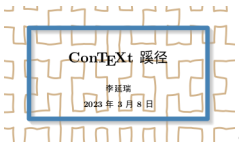
在某些情境下，一图可胜千言。ConTEXt 在插图方面，不仅支持常见的 JPEG，GIF 和 PNG 等位图格式，也支持 PDF，SVG 以及 MetaPost 等矢量图格式。

8.1 位图

所谓位图，直观上的认识是，对其进行放大或缩小，图像会失真。照片和屏幕截图，都是位图。常见的几种位图格式文件的扩展名是「.jpg」（或「.jpeg」），「.gif」和「.png」。ConTEXt 在处理插图时，若发现插图文件的扩展名是这些扩展名之一，便会以位图的形式将图片插入版面相应位置。

假设在 ConTEXt 源文件同一目录里有一幅位图 ctxnotes.png，以下代码，

```
\externalfigure[ctxnotes.png]
```



便可将 ctxnotes.png 作为插图，即 。显然，这是插图，但很可能并非是我们想要的插图形式。我们想要的是，独占一行且居中放置的插图，这个要求这并不难实现：

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

如果需要让插图更大一些，例如宽度为 8 cm，高度按图片原有比例自动放大，只需

```
\midaligned{\externalfigure[ctxnotes.png][width=8cm]}
```

通常更建议使用相对尺寸，例如 0.3 倍的版心宽度（满行文字的最大宽度）：

```
\midaligned{\externalfigure[ctxnotes.png][width=.4\textwidth]}
```

如法炮制，给图片加上标题也很容易：

```
\midaligned{\externalfigure[ctxnotes.png][width=.4\textwidth]}
\midaligned{\tfx \ConTeXt\ 学习笔记封面截图}
```

ConTEXt 学习笔记封面截图

如果你还希望插图能有编号，对于篇幅较小的文章，手工输入即可，建议在编号后，使用 \quad 插入一个字宽的空白作为间隔，因为普通的空格只有半个字宽。例如

```
\midaligned{\externalfigure[ctxnotes.png][width=.4\textwidth]}
\midaligned{\tfx 图 1\quad\ConTeXt\ 学习笔记封面截图}
```

图 1　ConTEXt 学习笔记封面截图

如果你担心，插图太多，手工输入图片编号难免会出错，可以使用 ConTEXt 的计数器功能，让 ConTEXt 为你自动递增图片序号。首先，定义一个计数器：

```
\definenumber[myfig]
```

然后，每次在给插图添加加标题时，将该计数器增 1，并获取它的当前的值：

```
\midaligned{\externalfigure[ctxnotes.png][width=.4\textwidth]}
\incrementnumber[myfig]
\midaligned{\tfx 图 \getnumber[myfig]\quad\ConTeXt\ 学习笔记封面截图}
\blank[line]
\midaligned{\externalfigure[ctxnotes-2.png][width=.4\textwidth]}
\incrementnumber[myfig]
\midaligned{\tfx 图 \getnumber[myfig]\quad 涂鸦版 Hilbert 曲线}
```

图 1 ConTEXt 学习笔记封面截图

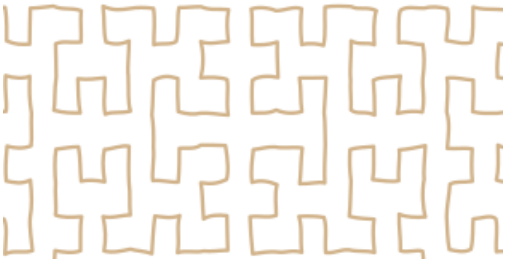


图 2 涂鸦版 Hilbert 曲线

8.2 矢量图

所谓矢量图，对其进行放大或缩小，图像不会失真。PDF 和 SVG 格式，皆为矢量图格式，文件扩展名分别为.pdf 和.svg，将它们作为文档插图，方法与位图相同，例如

```
\externalfigure[ctxnotes.pdf][width=.7\textwidth]
```

8.3 宏

超弦理论认为，宇宙是十维的，其中有六个维度蜷缩在卡拉比-丘空间，人类目前观测不到。我不知道这个理论是否正确，但是在 TEX 系统中，的确能将让一些维度蜷缩在一个空间里，这个空间叫作「宏」。

在 **3.3** 节，我们曾经定义过一个宏：

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

当我们使用 \foo 时，TEX 引擎会将其展开为「\hskip 0pt plus 2pt minus 0pt」。用类似的方法，可以让 **8.1** 节中的制作插图的代码蜷缩在一个带参数的宏里，例如

```
\definenumber[myfig] % 定义插图编号计数器
\def\placemyfigure#1#2{%
  \midaligned{#2}
  \incrementnumber[myfig]
  \midaligned{\tfx 图 \getnumber[myfig]\quad #1}
  \blank[line]
}
```

之后，在文章里放入插图会更为容易，例如

```
\placemyfigure
  {涂鸦版 Hilbert 曲线}
  {\externalfigure[ctxnotes-2.png][width=.4\textwidth]}
```

你可能并不能完全理解 \palcemyfigure 的定义，但是基于上一节的一些示例，应该能猜出其要义。这已经足够了，日后倘若你觉得有些经常重复使用的排版代码，它们只存在少许差异，便可

尝试为它们定义一个宏，用宏的参数代替差异。现在也许你已经隐隐感觉到了，ConTEXt 的排版命令也是 TEX 宏。

8.4 \placefigure

事实上，ConTEXt 提供了比我们定义的 \palcemyfigure 更为强大的命令 \palcefigure，其用法为

```
\placefigure[插图摆放位置][引用]{插图标题}{\exteranlfigure[...][...]}
```

例如，将 ctxnotes-2.png 作为插图，居中放置，引用为「Hilbert 曲线」，标题为「涂鸦版 Hilbert 曲线」，只需

```
\placefigure
  [][Hilbert 曲线]
  {涂鸦版 Hilbert 曲线}
  {\externalfigure[ctxnotes-2.png][width=.3\textwidth]}
```

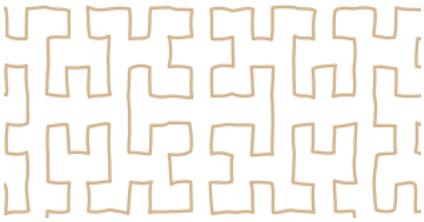


Figure 1　涂鸦版 Hilbert 曲线

8.4.1 插图标题样式

对于 \placefigure 的结果，可能你已经觉得有些不满意了。在中文排版中，图片的编号前缀不应该是 Figure，而应该是「图」，此外，编号也没必要粗体，而且标题字号应当比正文字体小一级。没有办法，ConTEXt 一切默认的样式，皆针对西文排版。不过，我们可以通过以下命令，将插图标题样式设置成我们所期望的样式：

```
[space=on]
\setupcaption[figure][style=\tfx,headstyle=\rm]
\setuplabeltext[en][figure={图}]
```

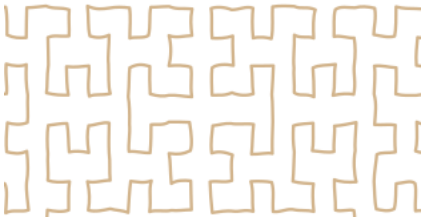


图 1　涂鸦版 Hilbert 曲线

上述设定的样式，已基本符合我们的要求。根据排版结果，很容易能猜出来，\setupcaption 的 style 参数用于设定插图字体样式，headstyle 则用于设定插图编号样式。至于 \setuplabeltext，与 ConTEXt 的语言界面有关，但现在不必涉及太多细节，仅需知道，它可将插图标题的前缀「Figure」替换为「图␣」。不过，依然存在一个细微的问题，标题里的汉字之间的粘连的伸长特性又被 ConTEXt 触发了，导致汉字分布有些疏松。该问题的解决方法与第 4 章的示例 4.5 相似，用将对齐参数设为 center，即

```
\setupcaption[figure][style=\tfx,headstyle=\normal,align=center]
```

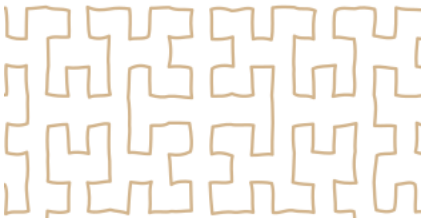


图 1 涂鸦版 Hilbert 曲线

8.4.2 插图位置

\placefigure 的第一个参数用于设定插图摆放位置。当该参数为空时，ConTEXt 默认插图居中放置。有时为了节省排版空间，需要将插图居左或居右放置，该需求可通过参数 left 或 right 实现。例如，

```
% 居左
\placefigure[left][...]{...}{...}
% 居右
\placefigure[right][...]{...}{...}
```

与 Mircro Word 这种字处理软件相比，ConTEXt 的居中插图缺乏文字环绕功能，若想实现该功能，需对在 TEX 层面掌握如何控制段落形状。

在 ConTEXt 世界里，插图实际上是浮动对象（Float Object）的特例。所谓浮动对象，即你以为插图应该在文档的某个位置出现，但实际上 TEX 引擎会根据版面的拥挤程度，修改插图的位置。例如，在文档的某一页的底部，若剩余空间已经不够放置一幅插图，则 TEX 引擎会努力在下一页为插图寻找一个更合适的位置，但是原本应该位于插图之后的正文内容会出现在插图之前。若是禁止插图浮动，只需

```
\placefigure[force][...]{...}{...}
```

还有一个参数 here，强迫性比 force 要弱一些，只是建议 TEX 引擎尽量让插图保持在原位置。

除上述参数之外，\placefigure 还有许多控制插图摆放位置的参数，但并不常用，欲知其详，请参考 ConTEXt Wiki 页面「Floating Objects」[9]。

8.4.3　引用

\placefigure 的第二个参数用于设定图片的引用标记。在正文中，使用 \in[...] 便可得到插图编号。例如

```
如图 \in[Hilbert 曲线] 所示……

\placefigure
  [][Hilbert 曲线]
  {涂鸦版 Hilbert 曲线}{\externalfigure[ctxnotes-2.png][width=.3\textwidth]}
```

如图 1 所示……

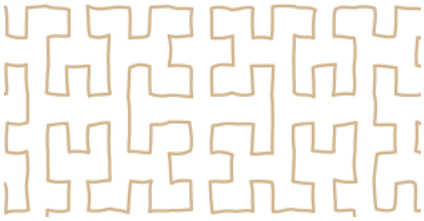


图 1　涂鸦版 Hilbert 曲线

8.5　阵列

有时为了节省排版空间，需要将两幅或更多幅插图并排放置，如图 **1.1** 和 **1.2** 所示。该效果可使用 floatcombination 环境构造插图阵列来实现。例如，首先构建一行两列的插图阵列：

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

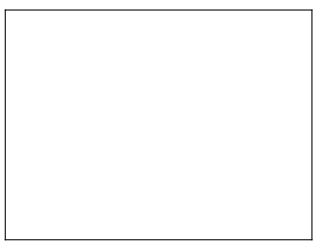


Figure 1

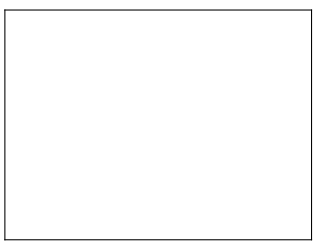


Figure 2

然后将所得阵列作为插图，便可得到居中放置的插图阵列：

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

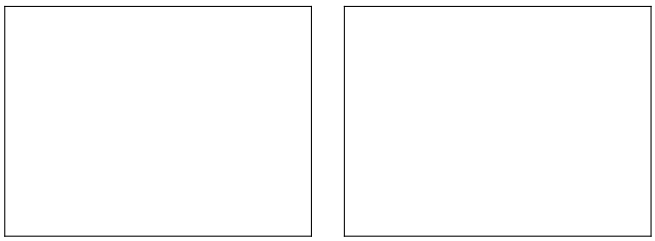
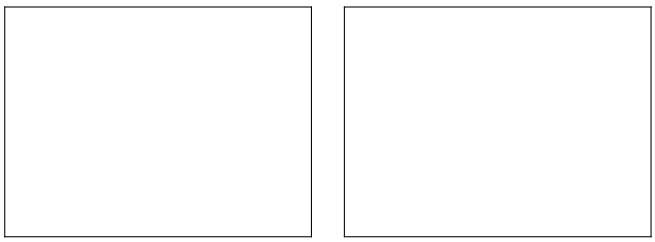


Figure 1 **Figure 2**

上述示例用了 \placefigure 一个小技巧：当 \placefigure 的第一个参数含有 none 时，可以消除插图编号和标题。此外，你应该发现了，\placefigure 的参数为空时，ConTEXt 会以一个矩形框表示插图，还应当注意到，方括号形式的参数，通常是可以省略的。

若 \placefigure 的第一个参数含有 nonumber 时，可以消除插图编号，仅保留标题。因此，上述实现图片阵列的方法稍加变换，便可实现由多幅插图组合为一幅插图的效果：

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

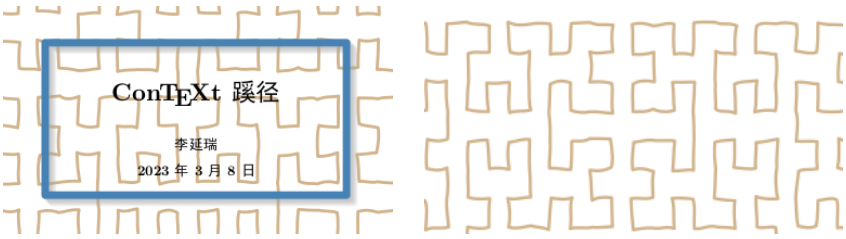


a b

Figure 1

基于 combination 环境可实现与上例等效的插图阵列，只是所用代码略多一些，但形式上更为结构化且应用范围更广。例如

```
\placefigure{}{
  \startcombination[nx=2,ny=1]
  \startcontent \externalfigure[ctxnotes.png][height=3cm] \stopcontent
  \startcaption a \stopcaption
  \startcontent \externalfigure[ctxnotes-2.png][height=3cm] \stopcontent
  \startcaption b \stopcaption
  \stopcombination
}
```



a　　b

Figure 1

8.6 图片目录

前文所有示例，插图所用图形文件皆需与 ConTEXt 源文件位于同一目录。为了让文件目录更为整洁，我们在 ConTEXt 源文件所在目录下，构建了一子目录，例如 figures，专门用于存放图形文件。为了让 ConTEXt 能够找到图形文件，在构造插图时，需要向 `\externalfigure` 提供图形文件的相对路径：

```
\externalfigure[figures/图形文件]
```

若不想每次插图时如此麻烦，可以通过以下命令将图形文件所在目录告知 ConTEXt：

```
\setupexternalfigures[directory={./figures}]
```

8.7 MetaFun

在 ConTEXt 中，还有一种插图形式，MetaPost 绘图代码，这些绘图代码被嵌入在 ConTEXt 的 MetaFun 环境里。例如，使用 MetaFun 环境 `useMPgraphic`，以 MetaPost 语言绘制一个边线被轻微随机扰动的矩形：

```
\startuseMPgraphic{metapost 图形}
path p;
u := 7cm; v := 3cm;
p := fullsquare xyscaled (u, v) randomized 0.07u;
drawpath p;
drawpoints p;
\stopuseMPgraphic
\placefigure{\MetaFun\ example}{\useMPgraphic{metapost 图形}}
```

Figure 1 MetaFun example

上述示例在排版插图标题时，涉及 TEX 宏在使用时即宏调用时的一个细节。例如，\TeX 之后跟随一个或多个空格，即 \TeX，这些空格会被 TEX 引擎吞掉，不会显示在排版结果中，原因是默认情况下，空格是 TEX 引擎需要知道宏的名字的结束符。如果需要在一个宏调用之后插入空格，需要对空格进行转义，即 \␣，亦即反斜线后跟随一个空格。

8.8 小结

所谓插图，不过是个头较大的文字罢了。

第 9 章　表格

ConTEXt 提供了多种表格形式，我们不需要全都学会，可以先学会最为简单的形式 Tabulate，等到将它用到山穷水尽也无法表达你想要的表格时，再考虑其他形式是否够用。简单的未必不好，强大的未必更好，既简单又符合自己需求的，永远都是最好的。

9.1 基本用法

首先，构造一个 2 行 3 列的表格，第 1 行的内容是 `1 2 3`，第二行的内容是 `4 5 6`，排版代码和结果如下：

```
\starttabulate
\NC 1 \NC 2 \NC 3\NC\NR
\NC 4 \NC 5 \NC 6\NC\NR
\stoptabulate
```

1	2	3
4	5	6

结果第 3 列跑到版面最右侧了。这是因为我们尚未定义表格各列的对齐方式。对齐方式不外乎三种，左、中、右，Tabulate 分别使用缩写 `l`，`c` 和 `r` 指代它们。例如，若令表格第 1 列居左，第 2 列居中，第 3 列居右，只需

```
\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

现在看上去像表格了，但是由于表格尚无边框线，无法看出表格各列内容的对齐状态。

想必你已经猜测出了，`\NC` 用于在表格的某一行构造一个单元格，上述示例中，表格内容的每一行最后一个 `\NC` 实际上是多余的，ConTEXt 会忽略它，但是你可以将它理解为表格的单元格的边界。`\NR` 用于构造一个新行，即下一行。上述示例里，表格只有两行，实际上第 2 个 `\NR` 也是多余的，只是为了形式上更整齐而保留，ConTEXt 会忽略它，你可以将它理解为表格一行的结束。

现在，将 `\NC` 替换为 `\VL`，便可画出单元格的左右边界线，即

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

1	2	3
4	5	6

如果在表格每一行的开始放上 \HL，可画出表格各行横线，即

```
\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

可能你已经发现了，表格的竖线将被横线截断了。不必担心是你的问题，而是 Tabulate 主要用于排版横线表，例如图 9.1 所示的在科技论文中常用的三线表。

一	二	三	四	五
甲	乙	丙	丁	戊
one	two	three	four	five
1	2	3	4	5

图 9.1 三线表

不过，要让表格的横线和竖线完全相交并不困难，只需将单元格之间的纵向间距参数 distance 设为 0pt 或 none：

```
\starttabulate[|l|c|r|][distance=none]
\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

9.2 间距调整

若希望单元格的宽度更宽一些，需要在列格式中设定 w 参数，例如令单元格宽度为 1 cm，只需 w(1cm) 即可。例如

```
\starttabulate[|lw(1cm)|cw(1cm)|rw(1cm)|][distance=none]
\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

如果希望所有的 Tablutate 实例的 distance 参数皆为 none，可使用 \setuptabulate 进行设定：

```
\setuptabulate[distance=none]
```

若希望单元格的竖向间距大一些，可使用 \TB 命令插入空行进行调整。例如插入 2mm 高的空格：

```
\starttabulate[|lw(1cm)|cw(1cm)|rw(1cm)|][distance=none]
\HL
\VL 1 \VL 2 \VL 3\VL\NR
\HL
\TB[2mm]
\HL
\VL 4 \VL 5 \VL 6\VL\NR
\HL
\stoptabulate
```

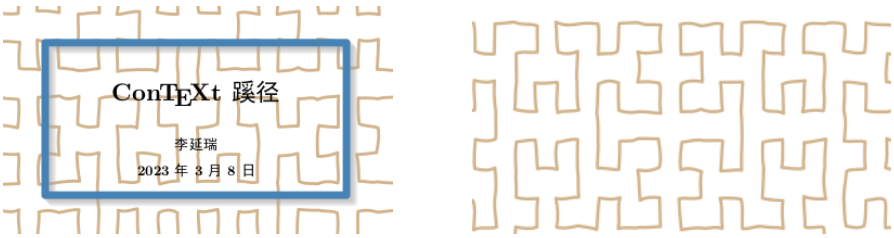
1	2	3

4	5	6

\TB 也可以使用相对尺寸，例如 2*line，line，halfline 和 quarterline 分别为一行文字的高度的 2 倍，1 倍，1/2 倍和 1/4 倍。

由于插图不过是个头较大的文字，因此基于表格理应能实现 8.5 节所述的排版插图阵列。的确可以如此，例如

```
\def\figA{\externalfigure[ctxnotes.png][height=3cm]}
\def\figB{\externalfigure[ctxnotes-2.png][height=3cm]}
\placefigure{}{
  \starttabulate[|cw(6cm)|cw(6cm)|]
  \NC \figA \NC \figB \NC\NR
  \NC a \NC b\NC\NR
  \stoptabulate
}
```



a b

Figure 1

9.3 \placetable

类似于插图，表格也有一个放置命令 \placetable，其用法与 \placefigure 相似。例如

```
\placetable[here][表格示例]{简单的表格}{
  \starttabulate[|cw(2cm)|cw(2cm)|cw(2cm)|][distance=none]
  \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

Table 1 简单的表格

对于中文排版，表格的标题，也需要自己定制，默认的设置并不符合我们的习惯。首先，将表格编号前缀设定为

```
[space=on]
\setuplabeltext[en][table={表}]
```

然后将表格标题编号设为正体，字号比正文字号小一级，放置于表格上方，并居中对齐：

```
\setupcaption
  [table]
  [headstyle=\tf,style=\tfx,location=top,align=center]
```

表 1　简单的表格

1	2	3
4	5	6

9.4 不传之秘

在绘制表格的横线和竖线时，线条粗度可通过参数 rulethickness 进行设定。例如，将线条粗度设为 2 pt：

```
\starttabulate[|c|c|c|c|c|][rulethickness=2pt]
\HL
\NC 一 \NC 二 \NC 三 \NC 四 \NC 五 \NC\NR
\HL
\NC one \NC two \NC three \NC four \NC five \NC\NR
\NC 1 \NC 2 \NC 3 \NC 4 \NC 5 \NC\NR
\HL
\stoptabulate
```

一	二	三	四	五
one	two	three	four	five
1	2	3	4	5

但是，如果我们只想让表格的顶线和底线是粗度 2 pt，中间那条横线让它是 Tabulate 的默认粗度，该如何实现呢？

对于该问题，也许你翻遍 ConTEXt 的 Wiki 或手册，都找不到答案，因为答案在 ConTEXt 的 tabl-tbl.mkxl 文件里。使用以下命令可搜索该文件：

```
$ mtxrun --script base --search tabl-tbl.mkxl
```

需要注意的是，该文件中关于 \TL，\LL 和 \BL 的注释应该是错的。要解决上述问题，需要先了解以下细节：

- 表格线粗度默认大概是 0.4 pt；
- 横线 \HL 有着细致的类别划分，从表格的顶线到底线，依次为顶线 \TL，第一条横线 \FL，中间的横线 \ML，最后一条横线 LL，底线 \BL；

- 若要设定表格横线的不同粗度，则横线必须按照类别使用，不可使用 \HL；
- \HL[1]和 \VL 可以接受两个参数，一个是表格线既定粗度的倍数，另一个是表格线颜色。

解决方法是，首先将上述示例修改为

```
\starttabulate[|c|c|c|c|c|][rulethickness=2pt]
\TL
\NC 一 \NC 二 \NC 三 \NC 四 \NC 五 \NC\NR
\FL
\NC one \NC two \NC three \NC four \NC five \NC\NR
\NC 1 \NC 2 \NC 3 \NC 4 \NC 5 \NC\NR
\BL
\stoptabulate
```

由于该表格内容只有三行，因此只有顶线，第一条横线和底线，亦即无中间横线和最后一条横线。为了更加充分演示问题是如何解决的，可以让该表格的内容再丰富一些：

```
\starttabulate[|c|c|c|c|c|][rulethickness=2pt]
\TL
\NC 一 \NC 二 \NC 三 \NC 四 \NC 五 \NC\NR
\FL
\NC 甲 \NC 乙 \NC 丙 \NC 丁 \NC 戊 \NC\NR
\ML
\NC one \NC two \NC three \NC four \NC five \NC\NR
\LL
\NC 1 \NC 2 \NC 3 \NC 4 \NC 5 \NC\NR
\BL
\stoptabulate
```

现在要保持 \TL 和 \BL 为既定粗度 2 pt，将 \FL，\ML 和 \LL 的粗度设置为 0.2 倍的既定粗度，及 0.4 pt，顺便试验一下颜色是否真的可用，见表 **9.1** 对应的代码，结果只有 \ML 变成了双线，其他皆符合预期。

为何 \ML 如此不配合呢？我猜也许它本来就是在绘制双线，因为 Tabulate 支持表格分页断开，即一个表格若处于页面底部且不能完全被当前页面容纳时，ConTEXt 可将其断开，一部分在当前页面，另一部分在下一页面。为了让断开后的表格完整，\ML 必须是双线。若将表 **9.1** 对应代码中的 \ML 换成 \HL，结果同样是双线。若不需要双线，可将 \ML 皆换为 \FL 或 \LL。

为了避免上述莫名其妙的问题，若只是令表格顶线和底线变粗，不必设定 rulethickness 参数，而是修改顶线和底线的粗度，令其他表格线的粗度皆为默认值。

[1] 包括 \TL，\FL，……，\BL。

```
\starttabulate[|c|c|c|c|c|][rulethickness=2pt]
\TL
\NC 一 \NC 二 \NC 三 \NC 四 \NC 五 \NC\NR
\FL[0.2,red]
\NC 甲 \NC 乙 \NC 丙 \NC 丁 \NC 戊 \NC\NR
\ML[0.2,blue]
\NC one \NC two \NC three \NC four \NC five \NC\NR
\LL[0.2,magenta]
\NC 1 \NC 2 \NC 3 \NC 4 \NC 5 \NC\NR
\BL
\stoptabulate
```

一	二	三	四	五
甲	乙	丙	丁	戊
one	two	three	four	five
1	2	3	4	5

表 9.1　修改表格线粗度和颜色

9.5 小结

除了在设定表格线粗度时不尽人意之外，Tabulate 堪当日常之用。它还有一些功能，本章尚未涉及，诸如跨栏，分页，段落等，这部分功能在后续章节介绍其他排版元素时，将作为搭配示例予以介绍。

待到 Tabulate 用至捉襟见肘之时，可使用「终极表格」，其文档在你的 ConTEXt 环境里，可通过以下命令搜索：

```
$ mtxrun --script base --search xtables-mkiv.pdf
```

第 10 章　盒子

在 5.4 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 Knuth 的《The TEX book》，对盒子的研究之深已经让我望风而拜，但是未必熟悉 ConTEXt 的盒子，故而本章仍有部分内容值得一睹。

10.1　TEX 盒子

前面已经多次暗示和明示，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，但是懂一些引擎层面工作原理，未必会有用处，但是你现在也并不能确定将来会不会成为一名 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，盒子是很重要的部件。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

示例 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}
}
```

在示例 5.5 中，\hbox 用于包含一副使用 MetaPost 语言绘制的矢量插图，关键代码如下：

```
\startuseMPgraphic{foo}
... ... ...
\stopuseMPgraphic
\lower.2ex\hbox{\useMPgraphic{foo}}
```

其中 useMPgraphic 环境中的 MetaPost 代码会被 \useMPgraphic{foo} 提交给 TEX 引擎。TEX 引擎会调用 metapost 程序将 MetaPost 代码编译成 PDF 格式的图片，然后交给 ConTEXt。最后由 ConTEXt 将图片插入到 \useMPgraphic{foo} 所在的横向盒子里。\lower .2ex 可令位于其后的横向盒子下沉 0.2 ex。长度单位 ex 类似于 em，也是一个相对尺寸，它是当前所用字体中的字母 x 对应的字形的高度。从现在开始要记住，横向距离用 em，竖向距离用 ex，不过这只是建议，并非 TEX 世界的法律。

还有一种横向盒子 \line，它的宽度是当前段落的宽度，可让其中的文字向两边伸展并与边界对齐，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，想必你想起了之前我们曾在文章标题的样式设定时，用 {middle,broad} 抑制的 ConTEXt 触发汉字间隙的伸长特性，该特性与\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。

\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

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

结果为 \inframed{...}。倘若使用 \framed，例如

```
\type{\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]{...}}}
```

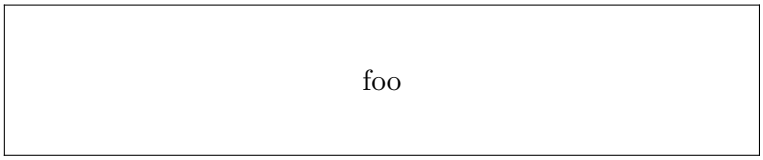
结果为 `\framed[location=low]{...}`。上述示例中的 \type 用于排版代码，它会自动将字体切换为等宽字体（\tt）原样显示它所接受的文字。顺便多言，若是排版一段代码，可使用 \starttyping[option=TEX] ... \stoptyping。

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：

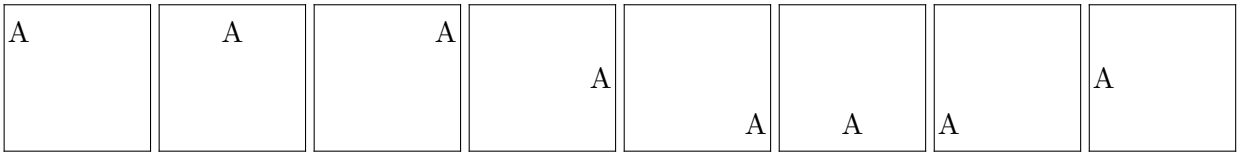
```
\hbox to \textwidth{\hfill\framed[width=10cm,height=2cm]{foo}\hfill}
```



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=1.95cm,height=1.95cm]
  \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}
}
```



\setupframed 所作的设定会影响到其后的所有 \framed，但是不会影响到其所处编组[1]之外的 \framed。

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，可将一些代码片段的排版结果作为 \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!

通过上述示例，现在你应该能有有七八分明白 **5.4** 节中带圈数字是如何实现的了，其关键代码如下：

```
\startuseMPgraphic{foo}
  path p;
  p := fullcircle scaled 12pt;
  draw p withpen pencircle scaled .4pt
         withcolor darkred;
\stopuseMPgraphic
\defineoverlay[rsquare][\useMPgraphic{foo}]
\def\fooframe#1{%
  \inframed[frame=off,background=rsquare]{#1}%
}
```

无非是将一个圆圈图形作为 \inframed 的背景罢了，而且对圈内的文字的长度有限制，字数略多一些，便出圈了，例如 \fooframe{123}，结果为 123。不过，只需对上述代码略加修改，

```
  p := fullcircle scaled \overlaywidth;
```

[1] 还记得 TEX 编组吗？即 {...}。

便可解除该限制。技巧是，用 overlay 的实际宽度作为单位圆的放大倍数，故而可保证圆圈的直径即圈内文字的长度，例如 `\fooframe{我有一个大房子}`，结果为 我有一个大房子。

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，但是在并非如此。例如

```
\inframed{ajk\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。

首先，我们需要明白，西文排版，一行文字是有基线的，但基线并非位于该行文字的最底端，而是距最底端有一段较小的距离，否则所有西文字母都在同一条线上了。在上述示例中，`\framed` 的底线便是其外围的 `\inframed` 所在行的基线。西文字体在设计时，每个字形（Glyph）是有基线位置，因此在排版时可根据实际的基线位置对字形进行对齐。一行文字的基线到该行的最底端的这段距离称为深度。一行文字的基线到该行最顶端的这段距离称为高度。因此，一行文字的真正高度等于深度 + 高度；在 ConTEXt 中，它等于我们使用 `\setupinterlinespace` 设定的尺寸。

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：

```
\placetable[force]{box0 的几何信息}{\tfx
  \starttabulate[|c|c|c|]
  \HL
  \NC 宽度 \NC 高度 \NC 深度 \NC\NR
  \HL
  \NC \the\wd0 \NC \the\ht0 \NC \the\dp0 \NC\NR
  \HL
  \stoptabulate
}
```

表 10.1 box0 的几何信息

宽度	高度	深度
94.34161pt	12.63184pt	4.91237pt

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 我看到一棵樱桃树。

10.6 小结

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。

第 11 章　学一点 MetaPost

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。只是不要妄图通过这区区一章内容掌握 MetaPost，因为关于它的全部内容，足够写一本至少三百多页的书籍了。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。使用 MetaPost 语言编写的代码可被 mpost 程序编译成 PS 格式的图形文件。自 LuaTEX 开始，mpost 的核心功能集成到了 LuaTEX 中，从此以后，在 TEX 环境中使用 MetaPost 语言作图便不需依赖外部程序了。

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，但它们又有三种不同用途：

- useMPgraphic：每被使用一次，对应的 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，其 MetaPost 只会被编译一次。

大多数情况下，建议选用 uniqueMPgraphic，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。

另外需要注意，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。例如，用粗度为 2 pt 的圆头笔用暗红色绘制一条经过 $(0,0)$，$(3\,\mathrm{cm},0)$，$(3\,\mathrm{cm},1\,\mathrm{cm})$，$(0,1\,\mathrm{cm})$ 的封闭路径，

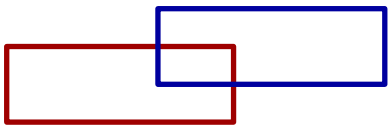
```
\startMPcode
pickup pencircle scaled 2pt;
draw (0, 0) -- (3cm, 0) -- (3cm, 1cm) -- (0, 1cm) -- cycle withcolor darkred;
\stopMPcode
```

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

```
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;
```



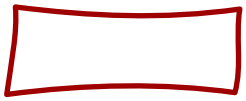
路径 p 被向右平移了 2 cm，继而被向上平移动了 0.5 cm。

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

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

可以使用 MetaFun 宏 randomized 对路径进行随机扰动。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]{光辉岁月}
```

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：

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

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

```
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);
```

`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

```
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);
```

江山如此多娇 江山如此多娇 江山如此多娇 江山如此多娇

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

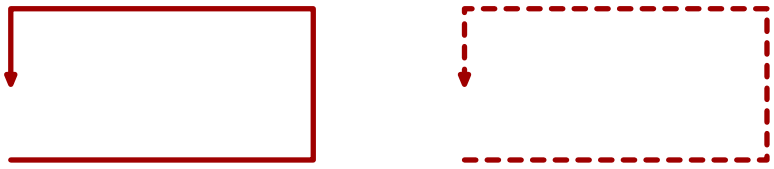
```
string s; s = "{\bf 江山如此多娇}";
draw (0, 0) withpen pensquare scaled 11pt withcolor darkred;
draw textext(s, (4cm, 0)) withcolor darkred;
```

江山如此多娇

11.5 方向路径

MetaPost 宏 drawarrow 可绘制带箭头的路径。例如

```
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;
```



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：

```
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));
```

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

```
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;
```

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

```
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;
```

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

```
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;
```

归妹愆期，迟归有时

上述实例为文字构造的边框太紧了，综合利用现有所学，让它宽松一些并不困难：

```
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;
```

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：

```
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;
```

使用上述定义的宏：

```
draw framed("{\bf 亢龙有悔}", 5mm);
```

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 画一幅简单的流程图

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);
```

还需要定义一个宏，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);
```

最后，再构造一个结点表示程序输出：

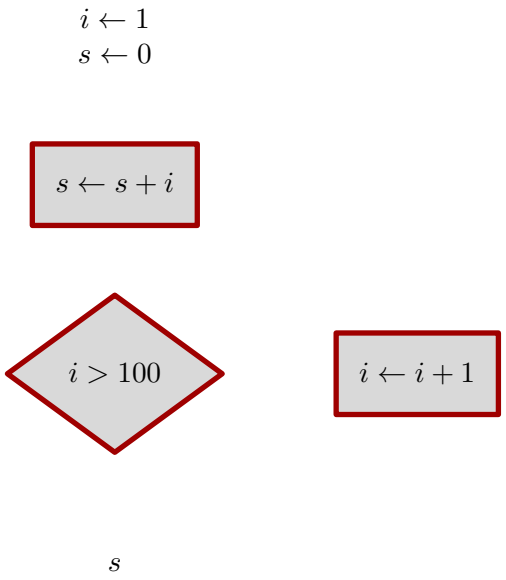
```
picture e;
e := image(draw textext(f & "{$s$}"););
```

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);
```

现在可以画出所有结点了，即

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



现在开始构造连接各结点的路径。首先构造结点 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];
```

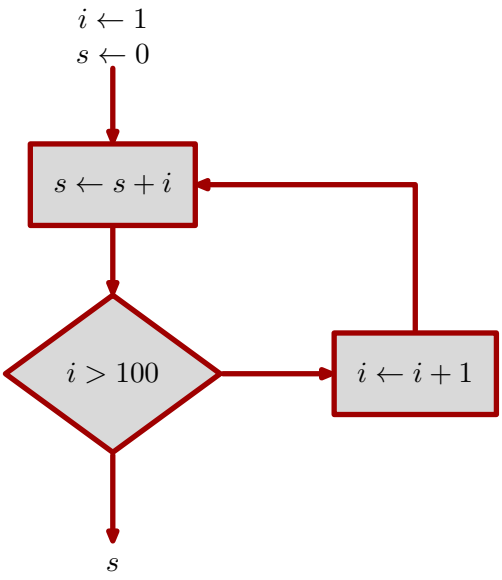
其中 llcorner 用于获取路径或画面实例的最小包围盒的左下角顶点坐标。同理，ulcorner，urcorner 和 lrcorner 分别获取包围盒的左上角、右上角和右下角顶点坐标。.5[..., ...] 用于计算两个点连线的中点。

用类似的方法可以构造其他连接各结点的路径：

```
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];
```

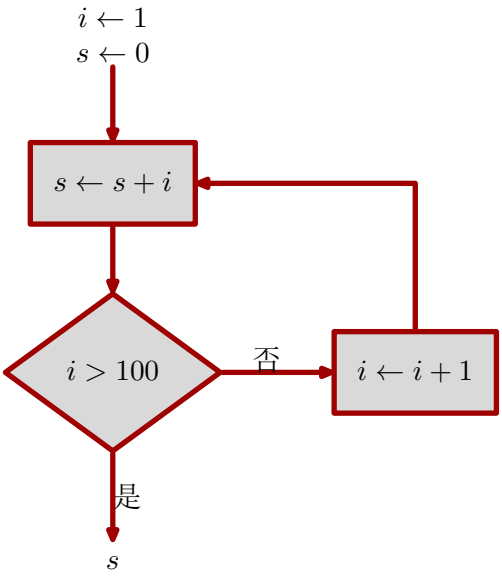
画出所有路径：

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



最后一步，路径标注：

```
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 节中用于绘制程序流程图的代码存在许多重复。我们可以尝试使用条件、循环，宏等形式对代码进行简化，但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`。因此，可以重新定义一个更为灵活的宏，用于制作结点：

```
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结点：

```
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$}", fulldiamond, offset);
e := make_node(f & "{$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。

在绘制结点和路径时，存在重复使用 `draw` 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：

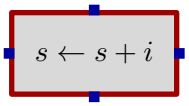
```
for i = ab, bd, cb, dc, de:
  draw i withcolor darkred;
endfor;
```

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;
```

以绘制结点 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结点的路径：

```
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 层层叠叠

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。

11.10.1 overlay

实际上 overlay 环境是一个图形「栈」。通过它，可实现图形（包括文字）的叠加。例如，对于以下 MetaPost 图形：

```
\startuseMPgraphic{一个矩形}
path p; p := fullsquare xscaled 4cm yscaled 1cm;
draw p randomized 3mm
       withcolor transparent (1, .5 randomized .3, red)
       withpen pencircle scaled 2pt;
\stopuseMPgraphic
```

上述代码中，`transparent (1, .5 randomized .3, red)` 用于构造在一定程度上不确定透明度的红色，透明度在 [0.2, 0.8] 之间，亦即 `randomized` 不仅可作用于路径，也可作用于数字或颜色值。由于上述 MetaPost 图形环境包含着随机内容，将其作为插图，在一个 overlay 中多次使用，每次使用都会产生不同的结果，在 overlay 中它们会被叠加到一起：

```
\startoverlay
  {\useMPgraphic{一个矩形}}
  {\useMPgraphic{一个矩形}}
  {\useMPgraphic{一个矩形}}
\stopoverlay
```

之前在为 `\framed` 定义 overlay 时，overlay 中只有单个图形，亦即我们并未充分利用 overlay 的特性。以下代码定义的 `\framed` 的背景便包含了三个叠加的 overlay：

```
\defineoverlay
  [叠叠]
  [\startoverlay
   {\useMPgraphic{一个矩形}} {\useMPgraphic{一个矩形}} {\useMPgraphic{一个矩形}}
   \stopoverlay]
```

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

迎接光辉岁月

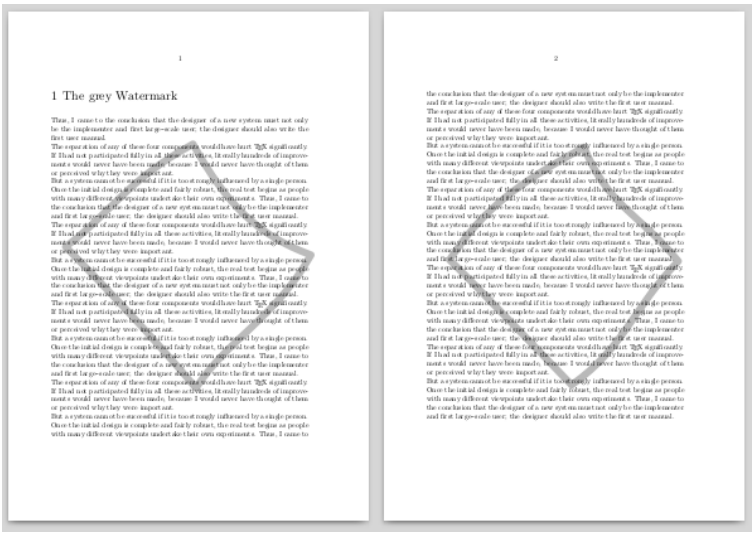
\framed 的也可以将多个 overlay 作为背景，以下代码与上例等效：

```
\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,重叠-2,重叠-3}]
\foo{迎接光辉岁月}
```

这意味着，\framed 的背景，本质上也是一个 overlay。

基于 overlay，也可为文档设置水印。例如，若在 \starttext 之前添加以下代码：

```
\startuseMPgraphic{square}
path p; p := fullsquare xscaled 4cm yscaled 1cm;
draw p randomized 3mm
       withcolor transparent (1, .5 randomized .3, red)
       withpen pencircle scaled 2pt;
\stopuseMPgraphic
\defineoverlay[watermark][\useMPgraphic{square}]
\setupbackgrounds[page][background=watermark]
```



11.10.2　层

ConTEXt 的层，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。例如，定义一个层 foo，在三个不同位置分别放置一幅 MetaPost 图形，并将其作用于当前页面：

```
\definelayer[foo]
\setlayer[foo][x=0cm,y=0cm]{\useMPgraphic{一个矩形}}
\setlayer[foo][x=6cm,y=1cm]{\useMPgraphic{一个矩形}}
```

```
\setlayer
  [foo]
  [x=\textwidth,y=2cm,hoffset=-4cm,vhoffset=-.5cm]{\useMPgraphic{一个矩形}}
\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。例如以下代码可在当前页面的左上角和右下角位置各放一个圆圈：

```
\startuniqueMPgraphic{circle}
draw fullcircle scaled 2cm withpen pencircle scaled 2pt withcolor darkgreen;
\stopuniqueMPgraphic
\definelayer[foo][width=\paperwidth,height=\paperheight]
\setlayer[foo][x=0cm,y=0cm]{\uniqueMPgraphic{circle}}
\setlayer[foo][preset=rightbottom]{\uniqueMPgraphic{circle}}
\setupbackgrounds[page][background=foo]
```

需要注意的是，层被使用一次后，便会被清空，因而将其作为页面背景，仅对当前页有效。

通过 preset 参数可调整层的坐标原点和坐标方向。在上述代码中，rightbottom 可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。

ConTEXt 预定义的 preset 参数值有

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

使用这些参数值时，要注意坐标方向。例如，若 preset=middleright，其 x 坐标方向变为向左递增，而 y 坐标方向依然保持向上递增，以下代码可予以验证：

```
\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
}
\setupbackgrounds[page][background=bar]
```

11.11 小结

也许你意犹未尽，甚至觉得我语焉不详。切莫怪我，我原本仅仅是介绍如何使用 MetaPost 绘制一个边框受随机扰动的盒子。不过，在大致掌握了本章所述的内容的基础上，关于 MetaPost 更多的知识，特别本章所有你觉得语焉不详之处，皆可阅读它的手册[10]以增进认识。此外，ConTEXt 开发者 Hans Hagen 所写的 MetaFun 手册除了讲述 MetaFun 之外也大量介绍了 MetaPost 的基本知识和技巧，该手册在你的 ConTEXt 系统中，使用以下命令查找：

```
$ mtxrun --script base --search metafun-s.pdf
```

参考文献

[1] Hagen H. ConTeXt LMTX [J]. TUGboat，2019，40(1)：34 – 37.

[2] Hagen H. ConTEXt Mark IV: an excursion. http://www.pragma-ade.nl/general/manuals/ma-cb-en.pdf，2017.

[3] 李延瑞. ConTEXt 学习笔记：Using MkIV. http://mirrors.ctan.org/info/context-notes-zh-cn/ctxnotes.pdf，2009.

[4] 李延瑞. 写给高年级小学生的《Bash》指南. https://segmentfault.com/a/1190000017229619，2018.

[5] Hagen H. Faking text and more. http://www.pragma-ade.nl/general/magazines/mag-0007-mkiv.pdf，2016.

[6] ConTEXt wiki editors. List of Unicode blocks. https://wiki.contextgarden.net/List_of_Unicode_blocks，2020.

[7] Hagen H. ConTEXt Reference. http://pmrb.free.fr/contextref.pdf，2013.

[8] ConTEXt wiki editors. Math. https://wiki.contextgarden.net/Math，2022.

[9] ConTEXt wiki editors. Floating Objects. https://wiki.contextgarden.net/Floating_Objects，2021.

[10] Hobby J D. MetaPost Manual. https://www.tug.org/docs/metapost/mpman.pdf，2022.