用 LaTeX 排版编程技术书籍的一些个人经验

陈硕 (giantchen@gmail.com)

最后更新 2017-10-29

版权声明

多年之前我写过一篇书评《〈Word 排版艺术〉读后感——兼谈与 LaTeX 的比较》[①]，其中写道“如果将来有时间，我把自己用 LaTeX 排书的经验总结一下，让读者在阅读《Word 排版艺术》的基础上，更容易地把知识应用到 LaTeX 排版中去。”我自己排版了《Linux 多线程服务端编程：使用 muduo C++ 网络库》，现在终于可以把账还上了。本文假定读者已经读过 LaTeX 的入门文档 [②] [③] 和书籍 [④]，具备基本的使用技能，这不是一篇入门教程。

排版是一门大学问，我只是一名技术图书的作者，有一些初步的 LaTeX 使用经验。我不是专家，出版印刷的行话也不怎么会说。本文的目的是让有志于用 LaTeX 来排版自己书的人少走一些弯路。换句话说，这篇文章是讲“我是怎么做的”，不是讲“哪种做法最好”。另外，遇到 LaTeX 使用方面的问题请先阅读 FAQ [⑤]，再上 CTeX 论坛 [⑥] 或水木社区 TeX 版 [⑦] 发帖询问，不要给我写信。（我最多能回答我那本书里某个版面是如何排出来的，无法解答你的具体问题。）

最新版下载地址：http://code.google.com/p/chenshuo/downloads/detail?name=typeset.pdf

LaTeX 源文件：http://github.com/chenshuo/typeset

更新记录

2013-02-04 初版

[①] http://blog.csdn.net/solstice/article/details/187233

[②] http://mirrors.ctan.org/info/lshort/chinese/lshort-zh-cn.pdf

[③] http://www.tex.ac.uk/tex-archive/info/latex-notes-zh-cn/latex-notes-zh-cn.pdf

[④] 《LaTeX 入门与提高（第 2 版）》，陈志杰等著，高等教育出版社。

[⑤] http://www.newsmth.net/bbscon.php?bid=460&id=282515

[⑥] http://bbs.ctex.org/forum.php

[⑦] http://www.newsmth.net/bbsdoc.php?board=TeX

目录

第 1 章
环境

1.1 为什么要自己排版？

如果是纯文学著作，完全可以交给出版社去排版。但是对于编程技术书籍，文字之间还穿插代码和图表，那么出版社的专业排版人员很难排出符合程序员审美观的版面，甚至有可能造成技术错误。

- 代码的排版，注意分页和折行。特别是 Python 这种缩进敏感的语言，一般要避免在一个函数内分页，这会造成阅读困难。可以在函数定义之间分页。
- 写作与排版一体化，作者可以适当改写内容，让版面更美观。例如，假如一个函数最后一行那个花括号被挤到了下一页第一行，这就很难看。这时可以考虑临时改变代码的缩进（从 BSD 风格改为 K&R 风格），从而节省一两行空间，让花括号落在本页。另外也可以稍微改变说法，避免段尾孤字成行。
- 文责自负，防止误改。例如我为某本书写的序言，其中提到旧的 boost `RegEx` class 和新的 `regex` class 在线程安全性方面的不同。这本书第二版再用这篇序言的时候被编辑“统一大小写”改为 `RegEx`，整段话也就变得莫名其妙了。类似的还有 $2^{32}-1$ 这种式子很容易被误改为 $232-1$，让人看了一头雾水。

1.2 为什么要用 LaTeX 排版？

- LaTeX 不会自作聪明地自动更正，例如句首字母大写（关键字 double 变为 Double），单词头两个字母大写（`IUnknown` 变为 `Iunknown`），括号匹配（“[begin, end)”变为“[begin, end]”）等等。Word 可以关闭自动更正，但是一旦整个流程（作者、编辑、校对、出片）中有一个人的设置不同，稿子就有可能被误改。
- LaTeX 的源文件是文本格式，可以方便地做版本管理（`diff/merge`），并且很容易利用和编写各种小工具来处理它。相反 Word 的 `.doc/.docx` 文件处理起来就麻烦多了，如果不是不可能的话。想想一句 `grep | sort | uniq -c | sort -n` 要写多少代码。

- LaTeX 可以处理英文断字（Hyphenation），避免一行文字太稀疏。例如 `ConcurrentHashMap` 这种在技术书籍中经常出现的长类名，如果不断字就会造成难看的版面。这也是 Word 排版容易出现的问题。另外 LaTeX 的断行和断页采用动态规划算法，排出来的版面比 Word 的贪心算法更匀称，见后文的例子。
- LaTeX 可以方便地做出交叉引用，引用其他章节图表的页码或编号。LaTeX 原稿可以分散到多个 `.tex` 文件中，便于编辑。如果 Word 也这么做（每章一个文件），那么交叉引用就麻烦得多。但是如果把整本书做成一个 Word 文件，那编辑起来就困难多了，牵一发而动全身。而且有一种如履薄冰的感觉，生怕哪天文件突然就损坏了。

以下展示动态规划与贪心算法的区别。这里版心宽度是 20 个汉字。第一行排满了 20 字，刚刚好。第二行排了 16 字，后接一个长单词，其宽度超过 5 个汉字，因此本行排不下了。Word 和 LaTeX 都会把长单词移到第三行，但是区别在于 Word 不会返回去调整第一行已经排好的那 20 个字（贪心算法只管当前行，排满为止，排不下就另起一行），因此第二行比第一行显得稀疏，版面不匀称。而 LaTeX 则会把整个段落通盘考虑，它会从第一行挪两个字到第二行，让前两行的字距相当，版面更匀称。

Input	一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六 `shared_ptr`。
Word	一二三四五六七八九十一二三四五六七八九十一 二 三 四 五 六 七 八 九 十 一 二 三 四 五 六 `shared_ptr`。
TeX	一二三四五六七八九十一二三四五六七八九十一二三四五六七八九十一二三四五六 `shared_ptr`。

1.2.1 动手之前

作者有能力并且有意愿完整书籍的排版，向出版社提供印刷质量的 PDF 文稿。出版社愿意改变通常的工作流程，采用作者提供的 PDF 文件来校对并印刷。要事先沟通好。LaTeX 不是一个傻瓜化的工具，它需要投入相当的精力去学习，才能排出满意的效果。

1.3 软件工具

1.3.1 操作系统

操作系统应满足三个条件：有好用的中文输入法，有好用的 PDF 阅读器，能方便地用命令行处理文本文件（`grep`、`sort`、`awk` 等）。目前看来符合这个条件的操作系统是 Mac OS X，但是我不可能为了写一本书去买一台新的笔记本电脑。因此我用的是一种混合办法，笔记本上安装 Windows，再在虚拟机中安装 Debian Linux，然后在 Debian 中安装 TeX Live。最后用 Samba 共享文件夹，这样就可以在 Windows 下方便地编辑 Linux 上的文件。而在 Linux 上用 Git 管理 `.tex` 源文件和图片，并且编译出 PDF。

1.3.2 TEX 发行版

Linux 用 TeX Live，Windows 用 CTeX 套装，Mac OS X 用 Mac TeX。中文处理采用 xelatex + xeCJK + ctex 方案[①]，不要采用过时的 CJK 或 CCT 方案。

注意，TEX 本身是非常稳定的，但是中文处理则在不断改进。例如 TeX Live 2010 和 TeX Live 2012 在处理中英文混排方面就有区别，造成“动版”，严重时会影响既有分页。

1. 用一个全局的 facade 来代理 `Foo` 类型对象访问，所有的 `Foo` 对象回调和析构都通过这个 facade 来做，也就是把指针替换为 `objId/handle`，每次要调用对象的成员函数的时候先 check-out，用完之后再 check-in [12]。这样理论上能避免 race condition，但是代价很大。因为要想把这个 facade 做成线程安全，那么必然要用互斥锁。这样一来，从两个线程访问两个不同的 `Foo` 对象也会用到同一个锁，让本来能够并行执行的函数变成了串行执行，没能发挥多核的优势。当然，可以像 Java 的 `ConcurrentHashMap` 那样用多个 buckets，每个 bucket 分别加锁，以降低 contention。

1. 用一个全局的 facade 来代理 `Foo` 类型对象访问，所有的 `Foo` 对象回调和析构都通过这个 facade 来做，也就是把指针替换为 `objId/handle`，每次要调用对象的成员函数的时候先 check-out，用完之后再 check-in [12]。这样理论上能避免 race condition，但是代价很大。因为要想把这个 facade 做成线程安全，那么必然要用互斥锁。这样一来，从两个线程访问两个不同的 `Foo` 对象也会用到同一个锁，让本来能够并行执行的函数变成了串行执行，没能发挥多核的优势。当然，可以像 Java 的 `ConcurrentHashMap` 那样用多个 buckets，每个 bucket 分别加锁，以降低 contention。

[①] http://blog.jjgod.org/2009/11/21/chinese-in-tex-live-2009/

因此我建议不要在排版期间升级 LaTeX 的大版本，这也是我在虚拟机上安装 TeX Live 的原因之一。这样可以轻松地备份整个系统，将来重印需要修订书中某几页的时候可以使用当年的虚拟机映像，版本一致，不必担心其他页面发生“动版”。

1.3.3 PDF 阅读器

推荐 SumatraPDF，它不锁 PDF 文件，可以随时覆盖，并且自动刷新。

1.3.4 离线备份

写书是一项耗时的任务，数据备份是必不可少的，防止误操作和硬件损坏带来的不可弥补的损失。除了 §1.4 讲的源文件版本管理之外，各种图片、表格，以及生成 PDF 也要及时备份，最好是离线（offline）备份。可用以下这些网盘：

- Amazon Cloud Drive
- Dropbox
- Google Drive
- Microsoft Sky Drive

这几个网盘都有 Windows 和 Mac 客户端，这也是我使用 Windows 桌面的原因之一。

1.4 版本管理

我用 Git 管理 `.tex` 文件和其他输入文件，并且同步到 Github 的私人仓库，就像这篇文章一样。

1.4.1 理想的工作流程

作者和编辑都使用版本管理软件，就像开发软件那样工作。LaTeX 就好比是编译器，`.tex` 文件是源程序，`.pdf` 文件是编译的结果。作者和编辑都可以修改源程序，并且通过版本管理软件来 merge 结果。考虑到作者和编辑不在同一个内网，因此一般要用公共的版本库，例如 Github。GitHub 的私有 repository 可保证数据安全。

1.4.2 现实的工作流程

编辑往往既不会 LaTeX 也不会 Git，那就之好采用原始方案，作者提供 PDF，编辑加以评注，或者打印出来再用红笔校对。

1.5 .tex 文件组织

.tex 文件一律使用 UTF-8 编码，一来避免各种编码转换的问题（某些人名用字在 GB2312 中没有定义），二来可以直接使用现有的 Linux 命令行工具来处理 .tex 文件。.tex 文件一般可以按章或按节划分，每个文件不超过 1000 行，以利于编辑。再用一个 .tex 文件把它们 \include 到一起。.tex 和图片文件的文件名不要有下划线。

第 2 章
版式

本文使用的长度单位：1 英寸 = 25.4mm = 72.27pt [①] = 72bp。注意 TEX 定义的 pt 和 PostScript（亦即 Adobe 系列软件）定义的 pt 的长度不一样，本书以 TEX 定义的 pt 为准，PostScript 的基本长度单位写为 bp（big point）。

2.1 纸张大小

书籍的纸张大小通常异于常用的打印纸大小（A4 或 B5），因此用 LATEX 排版书籍的第一步是设置好纸张大小。一般常见的 IT 图书的开本（成品书尺寸，可以用尺子量出来）是 185mm × 230mm，即“国际 18 开”；另外一种常见开本是 185mm × 260mm，即“16 开”。本文以“国际 18 开”为例。我们通常可以用 `geometry` 宏包来设置纸张和版心尺寸，例子见 §2.2。

2.2 版心大小

“版心”即正文区，不包含页眉和页脚 [②]。版心的大小可以这样计算：通常正文字号是 10pt [③]，一行按 39 个汉字计算，那么行宽是 390pt。行宽是正文字号的整数倍，这样中文间距不会无故拉宽。行宽不宜过大，否则阅读的时候容易读串行；也不宜过小，否则一行排不下 80 列代码。一般而言，36 ~ 42 字比较适宜，本文定为 39 字，数数上一行:-)。《Linux 多线程服务端编程：使用 muduo C++ 网络库》一行是 37 个汉字，因为这本书厚达 600 页，如果版心太宽容易影响阅读订口的文字。

对于 10pt 的正文字体，LATEX 默认的行距 [④] 是 12pt，这对于英文是合适的 [⑤]，但是对于中文则显得太密了。因此 CTEX 宏包将 `\baselinestretch` 定义为 1.3，这样行距是

[①] pt 是 point 的缩写，中文叫“磅”。

[②] 这是本文的定义，也有将版心定义为包含页眉和页脚的。

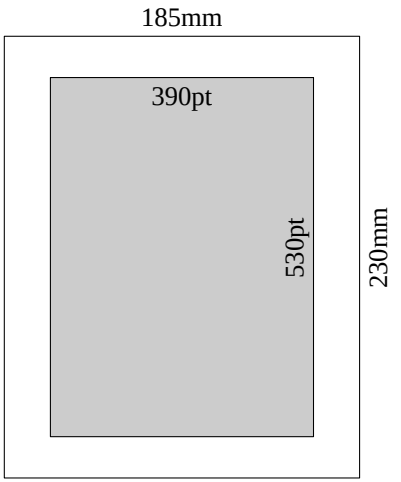
[③] 比五号（10.5pt）字略小，比小五号字（9pt）大。

[④] 行距指的是英文基线（baseline）之间的距离，即中文汉字底部之间的距离，不是两行之间的空白。

[⑤] 因为英文文本多是小写字母，字高远小于 10pt。

1.3×12pt = 15.6pt，阅读起来就比较顺眼了。如果一页排 34 行字，那么版心的高度大约是[①] 34×15.6pt=530.4pt，本文取 530pt[②]。

综上，对于 39 字 × 34 行的版心，其尺寸是 390pt × 530pt，约合 137mm × 186mm。见下图示意。



知道了纸张和版心尺寸，剩下的就交给 geometry 宏包。它同时会设置生成的 PDF 的纸张尺寸。例如 examples/paper.tex。

examples/paper.tex
```
\documentclass[10pt,fancyhdr,UTF8]{ctexbook}
\usepackage[centering,paperwidth=180mm,paperheight=230mm,%
            body={390pt,530pt},showframe]{geometry}
```
examples/paper.tex

注意这里把纸张宽度设为 180mm，这是考虑到装订的位置，这样在电脑上预览的时候左右空白更贴近实际印刷的效果。我们也不必关心版心在纸张中的上下左右位置，居中即可，在印刷的时候有专人负责拼版。当然在正式排版的时候要去掉 showframe 选项。

另外，打印纸一般不会刚好和书籍开本一样大，要想在书印出来之前感受版面效果，可以打印在 A4 纸上，但需要将书页框出来，可用 crop 宏包。例如 examples/paper-crop.tex。

```
$ diff -u paper.tex paper-crop.tex
--- paper.tex           2012-12-29 14:03:02.000 +0800
+++ paper-crop.tex      2012-12-29 14:03:02.000 +0800
 \documentclass[10pt,fancyhdr,UTF8]{ctexbook}
 \usepackage[centering,paperwidth=180mm,paperheight=230mm,%
             body={390pt,530pt},showframe]{geometry}
+\usepackage[a4,center,frame,color=blue]{crop}
```

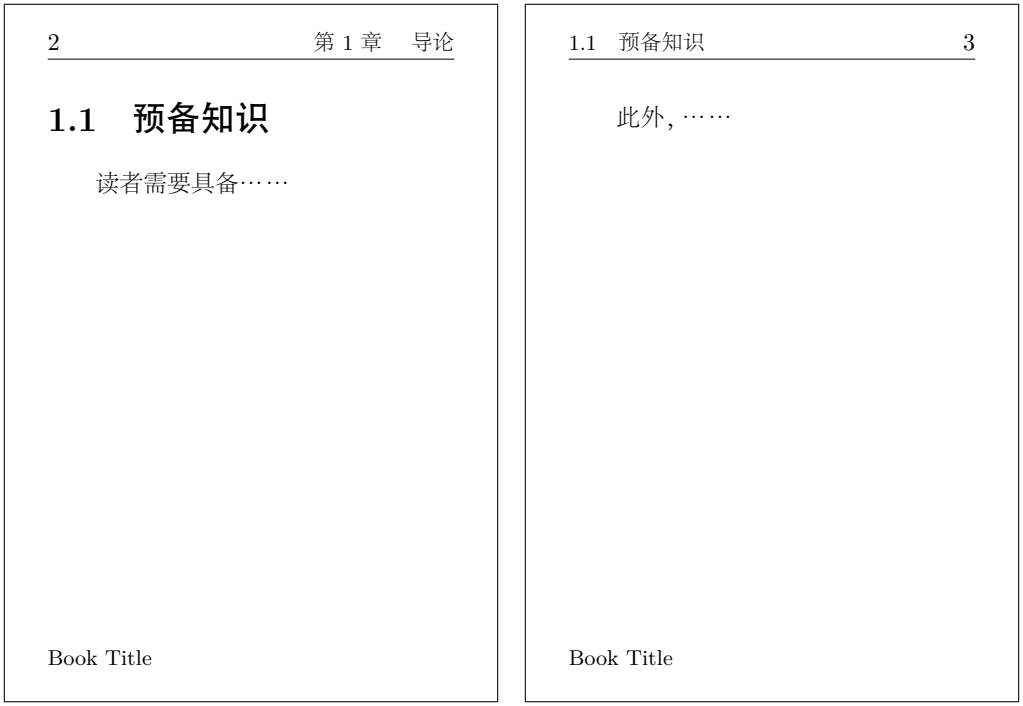
[①]这里说“大约”，因为第一行上方似乎不必留出多余的空白。

[②]对于技术类图书，通常一个自然段不会太长，一页之内几乎总是会遇到分段（整段代码、图表、章节标题）的情况，因此版心高度不必严格是行距的整数倍。

2.3 页眉与页脚

书籍排版不是以一页（一面）为单位，而是以翻开之后的两面（左右页）为单位。LaTeX 默认会设法让左右页的内容一样多，即左右页的最后一行位于同一高度。有时候这会造成难看的版面，特别是有可能留下过多段间空白。一般可以用 `\raggedbottom` 命令来取消这一设定。

页眉的外侧（切口）是页码，内侧（订口）是章节名称，通常是左页（偶数页）放章名，右页（奇数页）放节名[①]。页脚通常可以放书名，这样即便复印其中一面也容易知道出自何处（本文把作者姓名和网址也放到页脚，便于网上传播）。典型的安排如下图所示。



我们一般用 `fancyhdr` 宏包来设置页眉和页脚，常用的设置如下。其中 `RE` 表示偶数页（Even）右侧（Right），`LO` 表示奇数页（Odd）左侧（Left），`LE`、`RO` 的意思想必读者能举一反三猜出来。

```
\pagestyle{fancy}
\fancyhf{}
\fancyhead[RE]{\normalfont\small\rmfamily\nouppercase{\leftmark}}
\fancyhead[LO]{\normalfont\small\rmfamily\nouppercase{\rightmark}}
\fancyhead[LE,RO]{\thepage}
\fancyfoot[LE,LO]{\small Book Title}
```

[①]《Linux 多线程服务端编程：使用 muduo C++ 网络库》p.393 的内侧页眉为空，因为该章第 1 节出现在 p.394。

通常每章的第一页没有页眉，页码放到页脚中央，即 plain 页面格式，如下图所示。

2.4　中文字体

首先，适合屏幕阅读的字体不一定适合印刷书籍。原因可能有两点，一是书籍的印刷分辨率远高于屏幕；二是屏幕会主动发光，而书籍是被动反光。其次，中文字体用三四种（宋、黑、楷）就足够了，不要太花哨。印刷用的正文字体可用方正书宋或华康简宋，视出版社的字体授权情况而定。屏幕阅读可用 Windows 中文字体或 Adobe 中文字体。注意某些 Adobe 中文字体的标点符号位置不正确，例如 Adobe 楷体的全角冒号和分号上下居中，而不是位于左下角（子曰：“食不厌精；脍不厌细。”），使用时需要注意。另外，方正书宋和华康简宋缺少某些繁体字（例如“碁”），可以临时改换为 Adobe 宋体。例如

```
\setCJKfamilyfont{adobesong}{Adobe Song Std}
《C++ 编程规范（繁体版）》由 {\CJKfamily{adobesong} 碁峰} 出版社出版。
```

2.4.1　不要使用中文斜体

中文斜体是非常难看的，千万不要用。为了突出中文术语，可以用**黑体**，例如**与非门**。传统科技书籍也常用*楷体*表示术语和强调，不过黑体目前似乎更流行一些。

2.5 英文字体

英文字体可选择的范围就大多了，可参考 The LaTeX Font Catalogue [①]。

2.5.1 罗马字体

不要使用 Computer Modern Roman，它笔画太细而且衬线略显夸张。英文罗马字体一般可以选 Times Roman [②] 或 ***Palatino*** [③]。例如

- C++ is a general-purpose programming language. [Times Roman]
- C++ is a general-purpose programming language. [Palatino]

2.5.2 无衬线字体

一般用于章节标题和编程语言关键字，例如“this 指针”、“mutable 成员变量”。由于它出现的机会少，用什么字体其实无所谓，默认的 Computer Modern Sans Serif 就行。也可以换为 Helvetica [④]。

2.5.3 等宽字体

一般用于代码，包括变量名、类名等等。不要使用 `Courier New`，它太细而且太宽。可以用 LaTeX 默认的 `Computer Modern Typewriter` [⑤] 或 `Inconsolata`。后者要窄一些，但是双引号略弯。例如

`printf("Hello %s\n", name);` [cmtt]

`printf("Hello %s\n", name);` [Inconsolata]

注意如果要使用 Inconsolata 字体来排版代码，需要防止 LaTeX 替换其中的字符，例如 `operator<<()` 变成 `operator<<()` 等。可以通过 `fontspec` 宏包的 `Mapping` 选项来做到这一点，并搭配合适的 TECkit 映射文件 [⑥]。

如果要排版大量 Java 代码（一行可长达 100 列），可以考虑用窄的无衬线等宽字体，如 TheSansMono Condensed，但似乎没有免费版本。另外，一些英文书籍也用 `Lucida Sans Typewriter` 来排版代码，可以从 Sun JDK 中找到。

[①] http://www.tug.dk/FontCatalogue/

[②] 实际的字体名是 `Nimbus Roman No9 L` 或 `TeX Gyre Termes` 或 `Times New Roman` 等。

[③] 实际的字体名是 `URW Palladio L` 或 `TeX Gyre Pagella` 等。

[④] 实际的字体名是 `Nimbus Sans L` 或 `TeX Gyre Heros` 或 `Arial` 等。

[⑤] 实际字体名是 `Latin Modern Mono`。

[⑥] 见本文源码目录中的 `tex-text-tt.map`，需要用 `teckit_compile` 工具编译为 `.tec` 文件。

2.5.4 特殊字体

URL 可以用窄字体，以节省空间。例如 Ubuntu Condensed 或者 PT Sans Narrow。特殊术语可以用稍微夸张一点的字体凸显，例如“Extract Method 重构”和“Observer 模式”。

2.6 行距与段距

中文区分自然段的方法有三种，传统的方案是段首空两格（indent），现代的方案是两段之间增加空白（\parskip），第三种方案是同时使用前两者。本文采用传统方案，《Linux 多线程服务端编程：使用 muduo C++ 网络库》一书采用第三种方案。设置\parskip 的时候要小心，它会影响所有段落之前的空白，包括章节标题、图表、图表编号等等。

2.7 整段代码

用等宽字体排版代码一般可用 fancyvrb 宏包，然后定义自己的 Code 环境，字号 9pt，行距 0.9，左边留出 3mm。

```
\DefineVerbatimEnvironment{Code}{Verbatim}%
  {fontsize=\small,baselinestretch=0.9,xleftmargin=3mm}
```

用 Computer Modern Typewriter 字体，版心宽度 390pt 时，一行可排 80 列。

```
\begin{Code}
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6         7         8
\end{Code}
```

用 Inconsolata 字体，版心宽度为 390pt 时，一行可排 84 列。版心宽度为 370pt 时可排 80 列。

```
  \defaultfontfeatures[inconsolata]
  {
  Extension = .otf ,
  UprightFont = Inconsolatazi4-Regular,
  BoldFont = Inconsolatazi4-Bold
  }
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6         7         8
```

另外还可以定义 Codex 环境，用于排版带标题（文件名）的代码。

```
\DefineVerbatimEnvironment{Codex}{Verbatim}%
  {fontsize=\small,baselinestretch=0.9,xleftmargin=3mm,%
  frame=lines,labelposition=all,framesep=5pt}
```

我对 fancyvrb 宏包有一些改动，见 verbatim.cls。最终效果如下：

hello.c

```
1 int main()
2 {
3   printf("Hello, world.\n");
4 }
```

hello.c

第 3 章
样式

3.1 转义字符

注意“#”是 TEX 元字符，因此 C# 要写作 C\#。类似的“~”也是元字符，要写作\textasciitilde。这两个字符在 URL 中也经常出现，要特别小心。

C/C++ 代码的标识符中经常出现下划线“_”，例如 boost::shared_ptr、random_shuffle 等。为了避免每次都转义，可以使用 underscore 宏包，将下划线变为普通字符。注意这里 random_shuffle 在行尾断字，那么下划线之后不应该出现连字号“-”，因此应写为 random_\linebreak[0]shuffle。

3.2 斜体

按照排版规范，数学变量名和（非标准）函数名应该用斜体，常量、单位、标准函数名应该用正体。例如：n-body 问题、$\sin x = (\mathrm{e}^{\mathrm{i}x} - \mathrm{e}^{-\mathrm{i}x})/2\mathrm{i}$、5μs。快速排序 n 个元素的数度的平均时间复杂度是 $O(n \log n)$。通过 TCP 发送 n 字节的消息，接收方收到这 n 个字节的事件可能性有 2^{n-1} 种[①]。

3.3 列表

LATEX 默认的列表环境（itemize 和 enumerate）是为多行文字准备的，因此上下间距较大。我一般使用 enumitem 宏包来重新定义列表环境，并且定义几个简单的命令来使用它（\begindot 和 \myenddot，\beginnum 和 \myendnum）。

```
\newcommand\begindot{\begin{itemize}
[itemsep=2pt plus 2pt minus 2pt,%
topsep=3pt plus 2pt minus 2pt,%
parsep=0pt plus 2pt minus 2pt]}
\newcommand\myenddot{\end{itemize}}
```

[①]在 $n-1$ 个字节间隙中，依次插入 $0, 1, \ldots, n-1$ 个“隔板”，一共有 $C_{n-1}^0 + C_{n-1}^1 + \cdots + C_{n-1}^{n-1} = 2^{n-1}$ 种情况。

```
\newcommand\beginnum{\begin{enumerate}
[itemsep=2pt plus 2pt minus 2pt,%
topsep=3pt plus 2pt minus 2pt,%
parsep=0pt plus 2pt minus 2pt]}
\newcommand\myendnum{\end{enumerate}}
```

3.4 章节标题

章节标题无标点，因此 §1.1 是错的。ctex 宏包默认会把章\chapter 和节\section 的标题居中，这种样式显得很古板，章标题可以靠右，节标题和小节标题均靠左。参见本文的 format.cls 文件。

3.4.1 编号

通常单个小节不编号，因此 §2.4.1 和本小节是错的。应要改用\subsection* 命令。

3.5 图表编号

我不使用浮动环境，因此自己定义了 \figcaption 和 \tabcaption 命令来为图表编号。图的标题位于下方，按章编号（图 1-1、图 1-2、图 2-1 等等）。例如

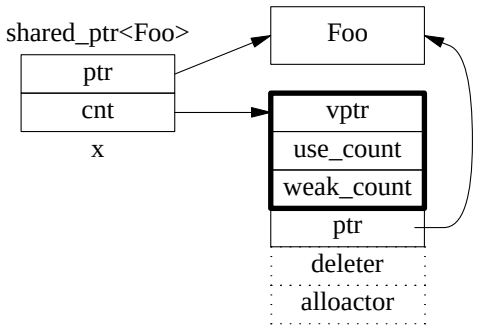


图 3-1 shared_ptr 的数据结构

表的标题位于上方，按章编号（表 1-1、表 1-2、表 2-1 等等）。例如

表 3-1 水平空格命令的长度

命令	长度	用途
\quad	10pt	图表编号与图表标题之间的全角空格
\,	1.67pt	千分空，例如 65 536

3.6 脚注

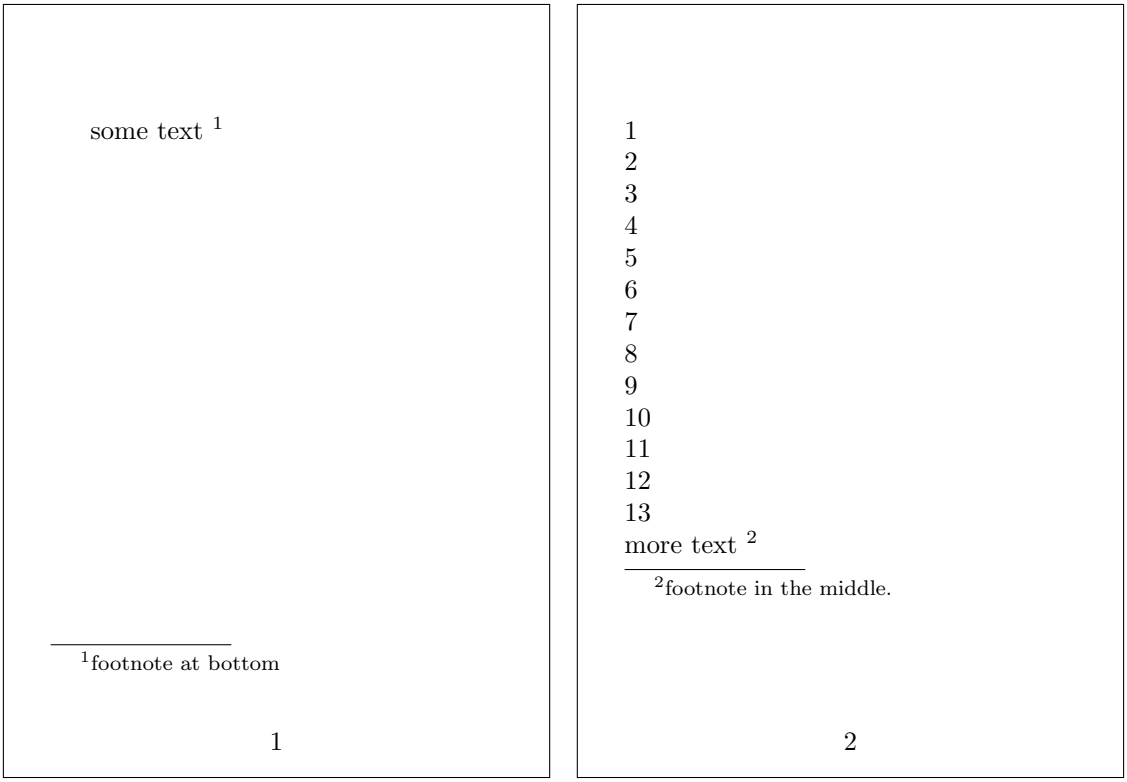
3.6.1 编号

LaTeX 默认是按章重置脚注编号，这么如果重印的时候需要增加一个脚注，势必会影响后续页码，这是《Linux 多线程服务端编程：使用 muduo C++ 网络库》排版的一个教训。因此我意识到脚注应该是按页重置，但是 `\@addtoreset{footnote}{page}` 无效，必须使用 `footmisc` 宏包的 `perpage` 选项，例如 `\usepackage[perpage]{footmisc}`。

LaTeX 默认的脚注编号是数字，这有时会造成误解。例如给长度单位 pt 添加脚注，正文中可能会出现“pt²”，让人误以为是面积单位，因此可以改用带圈数字。

3.6.2 置底

LaTeX 默认的脚注位置不是固定置底，而有可能随页面内容而浮动。例如 footnote-middle.tex。



使用 `footmisc` 宏包的 `bottom` 选项之后，脚注置底。见 footnote-bottom.tex。

some text [1]

[1] footnote at bottom

1

0
1
2
3
4
5
6
7
8
9
10
11
12
more text [2]

[2] footnote in the middle.

2

3.7 参考文献

技术书籍不是学术著作，可不必使用 BibTeX 工具，直接按出版社的格式要求排版参考文献即可。

第 4 章
工具

《Linux 多线程服务端编程——使用 muduo C++ 网络库》这本书是我自己用 LaTeX 排版的，在排版过程中也积累了一些小工具，本章把它们一一展示出来。不少工具都直接基于开源的 iText PDF 库，可从 http://itextpdf.com/ 下载，我用的是 `itextpdf-5.3.0.jar`。另外一些则用到了 Ghostscript，可直接用 `apt-get` 安装。

下载

Groovy 版本位于 https://github.com/chenshuo/typeset/tree/master/tools
Java 版本位于 https://github.com/chenshuo/recipes/tree/master/java/pdf
各个工具的输出示例位于 http://vdisk.weibo.com/s/kT4fL

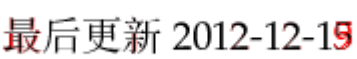
4.1 统计中文字数

LaTeX 没有像 Word 那样自带中文字数统计功能，加上 LaTeX 源文件中有许多控制字符，不能通过文件大小准确获知其中有多少汉字。为此我用 C 写了一个统计中文字数的小工具，名为 `cwc`，即 chinese word counter，这个工具可以处理 GBK、Unicode (UCS-2)、UTF-8 这三种编码的文件。源码位于 https://github.com/chenshuo/recipes/tree/master/utility。以下是一次运行的输出。

```
$ cwc *.tex
    26    368    1729 abstract.tex (UTF-8)
   149   1780    7021 chapEnvironment.tex (UTF-8)
    24    132     737 chapExperience.tex (UTF-8)
   262   2304   12739 chapLayout.tex (UTF-8)
    91    671    3977 chapStyle.tex (UTF-8)
   154   1257    6720 chapTools.tex (UTF-8)
    12      6     192 title.tex (UTF-8)
    41     69     963 typeset.tex (UTF-8)
   759   6587   34078 total
  行数   字数   字节数
```

4.2 PDF 内容对比（diff）

在书籍出版之后，每次印刷都可能修订一些内容，在把新的 PDF 文件交给出版社的同时，也要通知哪些页码有改动，方便出版社印刷。由于 PDF 是二进制格式，无法直接对比新旧 PDF 文件，于是我写了一个 `diffpdf.sh` 小工具用来找出哪些页面的内容有改动。这个工具的思路很土，就是先用 Ghostscript 把 PDF 按页渲染为多个 PNG 文件，然后用 `diff(1)` 比较新旧两个 PDF 渲染出来的这些 PNG 文件是否相同。然后用 Python 脚本（`diffpng.py`）将两个 PNG 文件的不同之处用红色高亮显示出来。例如：



4.3 PDF 截取

为了在网上公布样张，我需要从整书中截取一部分页码，另存为 PDF 文件，这用 `pdftk` 最方便了。例如

```
pdftk book.pdf cat 1-16 output preamble.pdf     # 前言和目录
pdftk book.pdf cat 19-46 output chap1.pdf       # 第 1 章
pdftk book.pdf cat 577-end output appendix.pdf  # 附录
```

4.4 PDF 页码编号

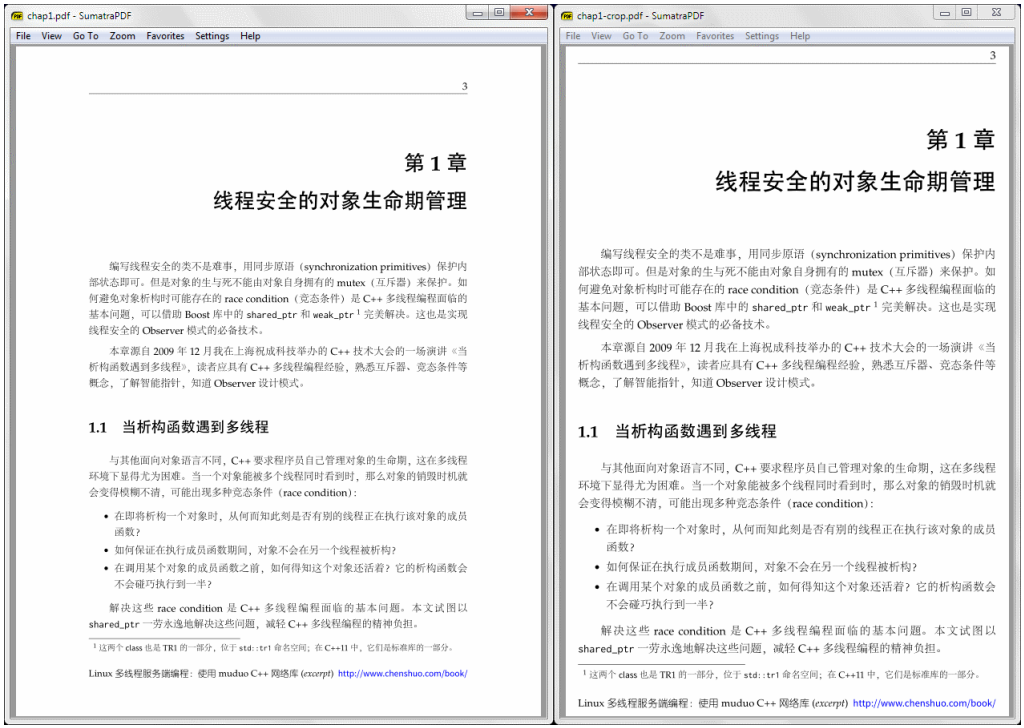
LaTeX 的 `hyperref` 宏包可以为生成的 PDF 设置“逻辑页码”，例如前言目录用罗马数字，正文用阿拉伯数字。不过经过 `pdftk` 截取之后就失效了。为此我编写了 `pagenum.groovy` 工具，用来添加逻辑页码。例如

```
pagenum.groovy '1e,3r' preamble.pdf  # 第 1、2 页是封面，没有页码；第 3 页开始用罗马数字
pagenum.groovy '1n3'   chap1.pdf     # 第 1 章第 1 页的逻辑页码是 3
pagenum.groovy '1n561' appendix.pdf  # 附录第 1 页的逻辑页码是 561
```

4.5 PDF 剪裁（crop）

为了充分利用屏幕空间，也便于在电子阅读器（iPad、Kindle）上阅读校对书稿，我一般会把 PDF 剪裁为版心大小。例如下面左图是原始 PDF，为纸张大小；右图是按固定 Crop Box 剪裁之后的版心。

固定剪裁



剪裁工具是 `crop.groovy`，设好 `CLASSPATH` 环境变量后可直接在命令行运行。其核心是根据版心和纸张尺寸算出左下角和右上角左边，然后剪裁每一页。这个工具不管 PDF 的内容。

如果需要根据页面内容剪裁 PDF，可以使用 Heiko Oberdiek 编写的 `pdfcrop` 工具，地址如下：

- http://www.ctan.org/tex-archive/support/pdfcrop
- http://code.google.com/p/pdfcrop2/

按内容剪裁

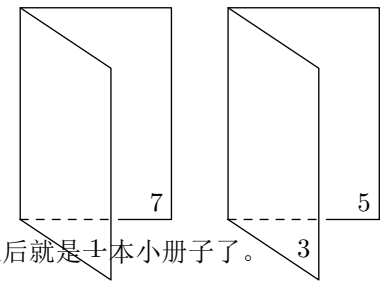
4.6 PDF 拼接（two-up）

有时候想在宽屏上同时阅读左右两页的书稿，除了可以用 PDF 阅读器本身的多页显示功能，我还常常自己做二合一（two-up）。这样得到的 PDF 也可以打印出来看，既节约纸张，而且与原稿是 1:1 大小。生成的 PDF 效果如下图。

二合一工具是 `twoup.groovy`，其核心是算出左右两页在合页中的起始坐标。

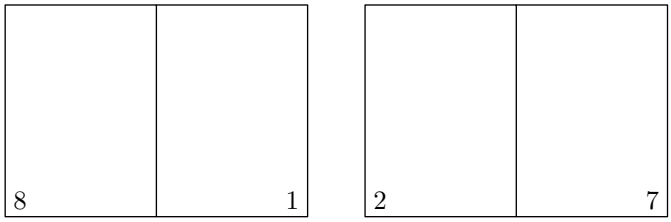
4.7 PDF 小册子（booklet）

有时候我会把一章的内容打印出来，装订成一本小册子，这样读起来有翻书的感觉。为了节约纸张，在打印之前要拼版，这样一张纸双面能打印 4 个页码。例如 8 页内容可以打印到两张 A4 纸上：

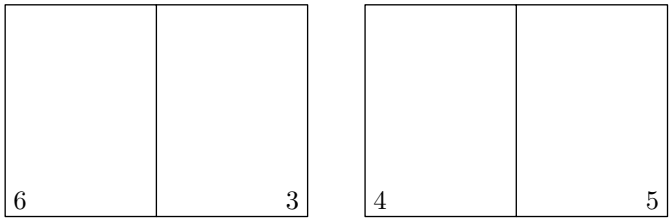


对折、叠好，骑缝订之后就是一本小册子了。

第一张纸的正反两面打印：



第二张纸的正反两面打印：



小册子工具是 `booklet.groovy`，其核心是算出每面纸应该放哪两个页码的原始内容。

装订这种小册子要用骑缝订，可用旋转订书机[①]。一本小册子一般应该控制在 10 页纸左右，即 40 个页码，再厚就订不透了。

4.8 PDF 字体嵌入

出版社拿到作者提供的终稿 PDF 之后，如果没有进一步修改，就会准备印刷了。第一步是让出片公司用激光照排机打印出胶片，即“出片”。这种公司使用的操作系统很可能与作者不同，特别是安装的字体可能不一致。为了防止出现文字乱码或字体错乱，出片公司一般都会要求提供嵌入全部字体的 PDF 文件。

一个办法是修改 Ghostscript 的配置文件，让默认字体全部嵌入[②]。例如：

/usr/share/ghostscript/???/Resource/Init/gs_pdfwr.ps

```
/.standardfonts [
%  /Courier /Courier-Bold /Courier-Oblique /Courier-BoldOblique
%  /Helvetica /Helvetica-Bold /Helvetica-Oblique /Helvetica-BoldOblique
%  /Times-Roman /Times-Bold /Times-Italic /Times-BoldItalic
%  /Symbol /ZapfDingbats
] readonly def
```

/usr/share/ghostscript/???/Resource/Init/gs_pdfwr.ps

还有一个办法是用 Adobe Acrobat Reader 找出 PDF 中没有嵌入的字体，再用 iText 将字体文件嵌入，例子见 http://itextpdf.com/examples/iia.php?id=288。

[①] http://www.amazon.cn/dp/B0080AF0FM http://product.dangdang.com/product.aspx?product_id=1141537002

[②] http://wand.net.nz/~iam4/latex.html

第 5 章
插图

尽量用矢量图，少用点阵图（屏幕截图只适合用于 step-by-step 介绍软件操作）。

在生成 .eps 或 .pdf 图片的时候，最好也同时嵌入所用的字体，这样最后生成的书籍 PDF 的字体是全部嵌入的。事后补救的办法见 §4.8。

5.1 绘图软件

《Linux 多线程服务端编程：使用 muduo C++ 网络库》的排版过程中我使用了多种绘图工具，这里简单列举一下。这里举的插图例子大多数可以在样章 [①] 第 6 章中找到。以下这些软件都可以生成矢量图（EPS 或 PDF 文件），印刷效果较好。

5.1.1 Graphviz

适合绘制依赖关系图，包括 `#include` 头文件的关系图（书 p.132 图 6-1）、简单的 class 继承关系图等。

优点：自动布局。

缺点：如果对自动布局不满意，不易手工调整。

5.1.2 gpic

适合绘制线框图，例如数据结构。本文的图 3-1 就是用 gpic 绘制的。gpic 可参考 Brian W. Kernighan 的《PIC—A Graphics Language for Typesetting, User Manual》和 Eric S. Raymond 的《Making Pictures with GNU PIC》。

缺点：无法输入中文。

5.1.3 METAPOST

适合绘制函数图像，例如书 p.349 图 9-5。

[①] http://vdisk.weibo.com/s/mtupb

5.1.4 Visio

适合绘制较复杂的 class diagram（书 p.132 图 6-2），sequence diagram（例如书 p.170 图 6-12）。需要安装 Visio Stencil and Template for UML 2.2 [①]。

可以输入中文，可以可视化编辑，可以导出为 PDF 文件，经过剪裁去掉白边 [②] 之后可以方便地嵌入 LaTeX 文档。

缺点：多幅内容相近的图片不易统一修改，例如《为什么多线程读写 `shared_ptr` 要加锁?》[③] 中绘制的多幅 `shared_ptr` 内部指针变化图。

5.1.5 Word

适合复杂的表格，例如书 p.161 表 6-1。可以输出为 PDF 文件，剪裁之后以插图的方式嵌入 LaTeX 文档。

5.1.6 Excel

适合绘制点数较少的性能数据图（x-y 坐标系），例如书 p.147 图 6-3。

5.1.7 Gnuplot

适合绘制点数较多的性能数据图，例如书 p.150 图 6-6。

5.2 待续……

[①] http://www.softwarestencils.com/uml/index.html

[②] 用 `pdfcrop` 工具 §4.5 按内容剪裁。

[③] http://chenshuo.googlecode.com/files/CppEngineering.pdf