

X_YLA_TE_X 指南

万泽

2015 年 1 月 22 日

前言

我试图收集整理各方面来的 **xelatex** 相关的信息，包括自己的实践经验总结，作为后学者的指南手册。

一切在尊重版权的基础上出于爱好兴趣和相信自由分享的价值原则上进行。

在乎内容和在乎外表总是一个永恒的两难问题啊。读者在使用 **tex** 排版系统的时候应该早早的就给自己一个定位，你是更在乎内容呢还是更在乎华丽的表现形式。

如果你在在乎的是更流畅的书写内容，那么本书的 **org** 模式详解部分可能是你很感兴趣的。采用这种模式就不可避免的在生成资料的外表上有点损失，不管是 **html** 的 **css** 深度定制还是 **tex** 相关 **sty** 深度定制都是如此。当然这种模式并不是说不可以进行生成资料的深度定制，只是因为 **html** 和 **tex** 源代码已经是二级生成文档了，所以进一步的修改或者对 **org** 文本的进一步定制都不能太多。对于这部分读者推荐使用 **emacs** 编辑器，将 **org** 模式作为你的作业环境。

如果你在在乎的是更华丽的表现形式或者说出版级别的版面设计，那么本书的 **org** 模式详解部分你可以不阅读，同时本书讨论的一些高级课题都可能会对你有帮助，同时由于 **tex** 排版系统真要深入进去有极深的内容，这些我也无能为力了。那么对于你的高要求目的（出版级别的版面设计）可能本书都稍显肤浅和入门性质了。对于这部分读者推荐就直接使用 **tex** 文件作为工作环境，编辑器推荐较为高级的编辑器，比如 **texmaker** 等等。

全书主要分为五大块：

1. **xelatex** 基础

这部分内容会就 **xelatex** 或者 **latex** 的基础知识内容做全面的整理归类排版，争取做到言简意赅并面面俱到和实用性很强的手册性质。

2. **xelatex** 进阶

更深入地讨论 **xelatex** 的排版问题。

3. org 模式详解

这部分是对着重内容的作业模式的集大成，在前面的讨论的基础上，继续深入到 **org** 模式，去讨论更加快捷的内容输入方案。

4. xelatex 高级篇

这部分讨论了 **xelatex** 在不同领域（比如化学，**tikz** 制图，幻灯片，试卷等等）的应用，这部分作业模式只可能是基于 **tex** 文档的作业模式了，那么也就是在完全追求华丽的表现效果的目的之上了，同时这些领域可能会有其他工具提供更好的用户体验或者更华丽的表现效果，这是值得我们注意的。

5. 附录

里面有些内容你可能会很感兴趣。

本文的 **github** 地址是：

<https://github.com/a358003542/xelatex-guide-book>

目 录

前言	i
目录	iii
I xelatex 基础	1
1 背景知识	2
1.1 TeX	2
1.2 关于 Knuth 教授	2
1.3 LaTeX	2
1.4 XeLaTeX	3
2 beginning	4
2.1 查看宏包帮助文档	5
2.2 从 documentclass 说起	5
2.2.1 一般的可选项	6
2.3 书籍的通用结构	7
2.4 页码控制	8
3 页面布局	9
3.1 geometry 宏包详细讨论	9
3.2 多栏环境	11
3.2.1 分栏环境备用	11
4 字体	13
4.1 字体的五种属性	13
4.1.1 字型编码	13
4.1.2 字族	13

4.1.3 字型系列	14
4.1.4 字形	14
4.1.5 字号	14
4.2 调整五种属性	15
4.2.1 调整字型编码	15
4.2.2 调整字族	15
4.2.3 调整字型系列	15
4.2.4 调整字形	15
4.2.5 调整字号	16
5 字体配置	18
5.1 fontspec 宏包详解	18
5.1.1 基本的命令	18
5.1.2 font features 的讨论	19
5.1.3 Color	20
5.1.4 字体大小	20
5.1.5 词间距——WordSpace	20
5.1.6 标点后间距——PunctuationSpace	20
5.1.7 断字符号——HyphenChar	20
5.1.8 字母之间的距离——LetterSpace	21
5.1.9 最后两个命令	21
5.2 设置数学字体	21
5.3 xeCJK 宏包详解	22
5.3.1 只能在加载宏包时填入的选项	22
5.3.2 可以在 xeCJKsetup 命令中设置的选项	22
5.3.3 xeCJK 宏包的一些命令	22
5.3.4 设置 CJK 字符大小	23
5.4 UTF-8 编码	23
6 特殊的符号	24
6.1 去掉符号命令后面的空格	24
6.2 基本的特殊符号	24
6.2.1 \	24

6.2.2 {和}	24
6.2.3 %	24
6.2.4 ~	25
6.2.5 \$	25
6.2.6 #	25
6.2.7 ^ 和 _	25
6.2.8 &	25
6.2.9 单引号和双引号	26
6.2.10 破折号和连字号	26
6.2.11 温度的度	27
6.2.12 省略号	27
6.3 更多特殊符号	27
7 初步中文化	30
8 颜色	31
8.1 颜色的心理学	31
8.2 颜色配置	32
8.2.1 定义新的颜色	32
8.2.2 用软件查看颜色	32
8.2.3 改变文章的背景颜色	32
8.2.4 改变字体的颜色	33
8.3 xcolor 宏包简介	33
8.3.1 单个颜色调百分比	34
9 段落	35
9.1 换行和分段	35
9.2 断字	35
9.3 段落中的行距	36
9.3.1 baselineskip	36
9.3.2 baselinestretch	37
9.4 段落间距	37
9.5 段首缩进	38
9.5.1 段首缩进量调整	38

9.5.2 缩进还是不缩进	38
9.6 段落对齐	38
9.7 flushright 环境空白间距问题	39
10 页眉页脚设计	40
10.1 观察	40
10.2 全部信息归零	40
10.2.1 plain 样式重置	40
10.2.2 选择默认样式为 empty	41
10.3 继续定制 plain 样式	41
10.3.1 fancyhf 命令的可选项	41
10.3.2 本文页眉页脚配置代码	42
11 章节标题设计	45
11.1 本文章节标题设计	45
11.2 titleformat 命令说明	45
11.3 章节编号数值修改	46
11.4 章节编号深度修改	46
11.5 章节编号形式修改	46
11.6 章节标题上插入脚注	47
11.6.1 text	47
12 目录设计	48
12.1 目录深度控制	48
12.2 前言加入目录	48
12.3 目录行间距拉大	49
12.4 目录编号和标题间距调整	50
13 封面设计	51
13.1 一个简单的封面	51
14 引用	53
14.1 文章内部引用	53
14.2 hlabel 命令	53
14.3 flabel 命令	54

14.4 hyperref 宏包简介	54
14.4.1 如何插入超链接	54
14.4.2 链接字体颜色控制	54
15 脚注	55
15.1 脚注标签修改	55
15.2 一页脚注重新编号	55
15.3 脚注中的距离	55
15.4 重定义脚注文本格式	56
15.5 表格里的脚注	56
16 旁注	58
17 尾注	59
17.1 endnotes 宏包说明	59
17.2 endnotes 宏包的使用	66
17.2.1 endnote 命令	66
17.2.2 showendnotes 命令	66
18 文字强调	67
18.1 emph 命令	67
18.2 重新定义 emph	67
18.3 underline 命令	67
18.4 ulem 宏包	67
18.4.1 ulem 宏包源码	68
18.5 reduline 命令	68
19 插入列表	70
19.1 基本使用	70
19.1.1 itemize 环境	70
19.1.2 enumerate 环境	70
19.1.3 description 环境	71
19.2 enumerate 环境标签的修改	71
19.3 itemize 环境标签的修改	72

20 插入图片	73
20.1 图片标签的修改	73
20.1.1 取消图片标签	74
20.2 设置图片寻找文件夹	74
20.3 图片格式的讨论	74
20.3.1 支持的图片文件格式	74
20.3.2 搜索图片后缀名控制	74
20.3.3 eps 图片格式	74
20.3.4 svg 图片格式	77
20.3.5 jpg 图片格式	78
20.4 图片格式选择总结	78
20.5 导入 pdf 页面	78
20.6 本文插入图片的一些 DIY	78
21 插入表格	80
21.1 基本情况的讨论	80
21.2 booktabs 宏包	81
21.3 一个三线表模板	82
21.4 booktabs 宏包详解	83
21.4.1 线条粗细	83
21.4.2 cmidline 命令	83
21.5 拉宽一行行的距离	84
21.6 带颜色的表格	85
22 插入代码	86
22.1 小代码	86
22.2 稍微大点的程序代码	86
22.3 fancyvrb 宏包	87
22.4 minted 宏包	87
22.4.1 mint 命令	88
22.4.2 minted 环境	88
22.5 tcolorbox 和 minted	89
23 特殊文字环境	91

23.1 语录和引用环境	91
23.1.1 语录环境演示	93
23.1.2 引用环境演示	93
23.2 诗词歌赋环境	94
23.3 notecard 环境	94
23.3.1 notecard 环境演示	95
24 插入摘要	97
25 参考文献	98
26 数学相关	99
26.1 数学环境	99
26.2 什么时候用数学环境	99
26.3 数学的 label 和 ref	100
26.4 加入文本	100
26.5 常用数学符号	101
26.5.1 空心粗体实数集合	101
26.5.2 上标和下标	101
26.5.3 平方根符号	102
26.5.4 表达式上划线或者下划线	102
26.5.5 表达式上或下有个大括号	102
26.6 向量	102
26.7 空白距离	102
26.8 特殊的函数	102
26.9 多行数学环境	103
26.10 分数	103
26.11 二项系数	104
27 注释	105
II xelatex 进阶	107
28 长度量	108
28.1 单位	108

28.2 默认的长度量	108
28.3 定义自己的长度量	109
28.4 修改长度量	109
28.5 使用长度量	109
28.5.1 显示长度量的值	109
28.6 calc 宏包介绍	109
29 计数器	110
29.1 latex 默认的计数器	110
29.2 使用计数器	110
29.3 计数器变成带圈的数字系列	111
29.3.1 用 tikz 画的方法	111
29.4 定义自己的计数器	112
29.5 修改计数器的值	112
29.5.1 计数器设值	112
29.5.2 计数器加一	112
29.5.3 计数器加多少	112
29.6 计数器和其他计数器绑定	112
30 新的命令或环境	114
30.1 新的命令	114
30.2 新的环境	115
31 盒子和 glue	116
31.1 基本知识	116
31.2 盒子命令介绍	118
31.2.1 makebox	118
31.2.2 raisebox	119
31.2.3 resizebox	119
31.2.4 parbox 和 minipage	121
31.3 带颜色或者线框的盒子	122
31.3.1 framebox	122
31.3.2 colorbox	122
31.3.3 fcolorbox	122

31.4 保存盒子	123
31.4.1 savebox	123
32 线条	124
33 大文档管理	125
III org 模式详解	126
34 org 转 tex 初步配置	127
34.1 title 命令	128
34.2 设置目录深度	128
35 org 模式对应 latex 命令详解	130
35.1 .emacs 文件及时生效	130
35.2 org 文件的基本操作	131
35.3 org 对应 latex 基础	131
35.4 特殊的符号 (org 模式中)	131
35.4.1 连字号	132
35.4.2 更多的符号和中文化问题	132
35.4.3 \ 符号	133
35.5 数学环境	133
35.6 换行命令	134
35.7 插入列表	134
35.8 插入引用	134
35.9 插入脚注	136
35.10 特殊的文字环境	136
35.10.1 粗体	136
35.10.2 斜体和 emph 命令	137
35.10.3 下划线	138
35.10.4 短代码环境	138
35.10.5 verse 诗歌环境	139
35.10.6 quote 引用环境	140
35.10.7 居中环境	140

35.11 插入一横线	141
35.12 插入图片	141
35.12.1 加上标题	142
35.12.2 加上标签	142
35.13 上标和下标	143
36 org 转 html 和 tex 的协调问题	144
36.1 原有环境命令属性的修正	145
36.2 文档内非要使用自定义的命令	145
37 插入表格	147
38 插入代码	148
38.1 简单的代码	148
38.2 更华丽的代码	150
38.2.1 SRC 环境的一些选项	152
38.3 插入代码文件	153
38.4 执行代码	153
39 org 模式其他额外的特性	154
39.1 快速输入	154
IV xelatex 高级篇	156
40 tikz 制图	157
41 化学相关	158
41.1 什么时候是原子	159
41.2 什么什么键	160
41.3 键的角度和其他参数控制	161
41.3.1 角度的三种表达方式	162
41.4 高中的化学反应式的绘制	162
42 exam 类	163
43 beamer 类	164

V 附录	165
A 我的作业心得分享	166
B ubuntu 知识	168
B.1 通过 U 盘安装 ubuntu	168
B.2 初入 ubuntu	168
B.3 linux 命令介绍	169
B.4 通过 ppa 安装软件	170
B.5 软件推荐	171
B.6 rime 输入法	172
B.6.1 备份和还原	173
B.6.2 把 QQ 拼音上你的词库导入 rime	173
B.6.3 定制 rime 输出特殊符号	174
B.7 ubuntu 美化	175
B.8 去除登录界面的白点	176
B.9 配置 L ^A T _E X 编写环境	176
B.10 新宏包的安装	177
B.10.1 阅读宏包代码	177
B.11 texmaker 技巧	177
B.11.1 自动补全命令	177
B.11.2 自定义命令	178
B.11.3 保存你的设置文件	178
B.12 用 rime 快速输出 L ^A T _E X 命令	178
B.13 其他 T _E X 编辑器评测	178
B.13.1 gedit 的一些技巧简要说明	179
B.14 安装字体	180
B.14.1 找字体文件	180
B.14.2 放置字体文件	180
B.14.3 命令行安装字体	180
B.14.4 用 fontmanager 安装字体	181
B.15 ubuntu 系统字体的配置	181
B.15.1 或者更简单的方法?	183

B.16 ubuntu 的备份和还原	183
B.16.1 更简单的备份?	184
B.17 更多 ubuntu 知识	184
B.18 注释	184
C git 基础	185
C.1 结合 github 进行项目管理	185
C.1.1 基本命令	185
C.1.2 日常改动提交流程	187
C.1.3 本地仓库管理文件丢失	187
C.2 安装	187
C.3 第一次使用的配置	188
C.4 初始化仓库	188
C.4.1 github 上删除项目	188
C.5 远程仓库文件到本地	188
C.6 本地文件夹进入索引	189
C.7 将索引跟踪的文件提交本地仓库	189
C.8 本地仓库提交到远程仓库	189
C.9 远程仓库的改动更新到本地	190
C.10 日常改动提交流程	190
C.11 本地仓库管理文件丢失	190
D git 进阶	192
D.1 查看本地仓库更改记录	192
D.2 查看本地仓库目前的状态	192
D.3 设置快捷命令	193
D.4 加上标签	193
D.5 返回功能	193
D.5.1 返回某个特定的版本	193
D.5.2 丢弃本地操作	194
D.6 分支功能	194
E emacs 入门	195
E.1 最基本的快捷键操作	195

E.2 lisp 语法简介	195
E.2.1 设置变量	196
E.2.2 定义函数	197
E.3 最基本的配置	197
E.3.1 加入 smex 插件	197
E.4 一些命令一览	200
E.5 DIY 之门	201
F 其他问题的讨论	203
F.1 TeX 一些常用命令简介	203
F.1.1 relax 命令	203
F.2 TeX 中执行系统的命令	203
F.3 TeX 中写文件和读文件	203
F.4 TeX 中的程序结构	204
F.4.1 条件控制语句	204
F.5 TeX 生成前面有零的数字序列	205
F.6 文本中的上标还有下标	206
F.7 多张图片并列显示	206
F.8 生成字体所有已有的字形	207
F.9 方便手机上观看的 pdf	208
F.10 临时改变页面布局	209
F.11 xverbatim 宏包说明	210
F.12 xverbatim 宏包的使用	215
F.13 注释	216
参考文献	218

xelatex 基础

背景知识

TeX

以下完全按照 [wikibook](#) 中的 [latex](#) 翻译的。[wikibook-latex](#)

TeX 是一个底层的标记式的编程语言，**Donald Knuth** 发明的排版系统，可以用来排版出很漂亮的文章。当初 **Knuth** 看到自己的文章和书籍被排版的丑陋不堪，于是在 1977 年开发了这个排版引擎，这个引擎深深地改变了出版业，大力扩展了数字打印设备的潜能。1989 年 **TeX** 支持了 8 位字符，然后 **TeX** 的开发就被冻结了，只限 **bug** 的修复。**TeX** 作为一种编程语言，是支持 **if-else** 结构的：你能够在里面执行数学运算（他们在编译文件的时候被执行），等等。。不过你会发现要做其他的还是很困难的除了排版文字。**TeX** 对于文章的结构和格式提供了良好的解决方案，使得它成为一个强大的神器。**TeX** 是出了名的稳定，可以运行在各种计算机上，几乎没有 **bug**。**TeX** 的版本是按照 π 的序列扩展的，目前到了 3.1415926。

关于 Knuth 教授

Knuth 教授是 **T_EX** 排版系统的发明人，出于内容精简的目的，关于 **Knuth** 教授的生平信息删去了，有兴趣的请参看[wiki-高德纳](#)。

LaTeX

LaTeX 是一个宏包，其目的是使作者能够利用一个预先定义好的专业页面设置，从而得以高质量地排版和打印他们的作品。**LaTeX** 最早是由 **Leslie Lamport** 编写的，并使用 **T_EX** 作为其排版系统引擎 [[lshort](#)]。

XeLaTeX

关于 XeLaTeX 第一是文档是 UTF-8 编码的，第二是它对各种字体多语言输出文章的解决方案是最完美的，第三是 LaTeX 里面能够用的命令它一般都能用，第四是编译生成 pdf 文件使用的命令是 `xelatex` 什么什么 `tex` 文件，第五是需要知道它内置引擎现在一般是 `xdvipdfmx`。

beginning

在 \LaTeX 的代码中最重要的是理解各种各样的命令的功能，正是这些各种各样的命令让你输入的文字显得与众不同。比如说我现在在打很长的一段文字， \LaTeX 会自动换行的，而我在这里按下 **Enter** 键，实际上并没有换行的效果。理解这一点很重要， \LaTeX 不同于微软的 **word** 软件或者其他 **openoffice** 之类，不是采用的所见即所得模式，我在这里打的是奇奇怪怪的东西，但是最后显示出来的却可能是很美观的东西。 \LaTeX 的一个设计理念就是所想即所得，它甚至有点偏执地要求你组织好你自己的文章的结构，而这正是 \LaTeX 的爱好者所推崇的。

同样在代码中你空一个格或者空很多个格都是没有区别的，都是一个空格^①。

在 \LaTeX 中空一行和空很多行的效果是一样的，都是空一行，表示另起一段。

\LaTeX 的命令用到了一些特殊的符号，所以你就不能按照常规用到它们了，这些符号如下：

\$ % ^ _ & { } ~ \

更详细的说明请参见后面的特殊符号6

\LaTeX 的命令是 **case sensitive** 的。也就是命令是区分大小写的。

现在我把最基本的代码说明一下， \LaTeX 的代码的通用格式是这样的，`\` 开头，然后跟上命令符号，然后跟上 `[]`，方括号中放的是该命令的可选参数，然后跟上 `{}`，花括号里面跟的是该命令的必填参数。具体如下：

`\command [optional parameters]{parameters}`

前面第一行代码是：

`\documentclass [11pt,oneside]{book}`，意思是描述文章模板的类型为 **book**

^① 现在中文之间的空格不会显示了，不清楚是 **xelatex** 还是 **xecjk** 宏包处理，这样挺好的。

，也就是一本书，除此之外还有 `article`，`report`，`slide` 类型等，更详细的讨论参见 `documentclass` 说明[2.2](#)

然后我们看到第二行代码：

```
\usepackage{什么宏包}
```

这个 `usepackage` 命令后面跟上你想加载的库文件，等你使用 `LATEX` 久了，就会接触到更多的宏包的。

后面的代码：

```
\begin{document}
```

文章内容

```
\end{document}
```

描述文档开始，文档结束。在文档结束命令之后，你写的任何东西都会被 `LATEX` 忽略掉。文档环境里面就写着你的文章的内容。

查看宏包帮助文档

这个我先讲了，实际上沉下心来阅读文档是最好的学习 `LATEX` 的方法了。比如我要查看 `xeCJK` 文档，就在终端中输入：

```
texdoc xeCJK
```

在 `texmaker` 的帮助菜单下面有个功能类似的小插件。

从 documentclass 说起

文档刚开始是 `preamble` 区域，放着文档的一些配置，从 `begin{document}` 开始进入正文区，出了 `end{document}` 这句话之后后面写的什么程序都不管了。`document` 是一个环境，后面我们会接触很多的环境的。

`documentclass` 命令的必选参量有 `article`，`report`，`book`，`slides`，`beamer` 等，一般了解这几个就够用了。他们之间有很多细微的差别，这个后面慢慢了解。

分节命令带星号表示该分节不进入目录，也不编号。

文档的章节分级结构如下：

```
\part {partname}
\chapter {chaptername}
\section {sectionname}
\subsection {subsectionname}
\subsubsection {subsubsectionname}
\paragraph {paragraphname}
\subparagraph {subparagraphname}
```

一般 `paragraph` 和 `subparagraph` 分节不怎么使用，就在文档中一行一行空出来即可。还有 `subsubsection` 这个分节也不常使用，因为 `section` 之下有 `subsection` 已经很好地满足了思想的分级结构。`paragraph` 命令可以构建出类似 `description` 环境的效果，不同的是后面不缩进，装载内容容量更大。

所以结合前面 `book`, `article` 分类我在这里为了简单起见约定如下：文档中一个小段落就是 `subparagraph` 不需要用命令再标识一次，几个段落构成一个 `paragraph`，这从原则上就是某一个问题的阐述，也就是一个 `section` 级别，当我们对某一个课题反复思考之后，积累的资料越来越多，然后我们发现某几个 `section` 可以合并起来，这样出现了 `section` 和 `subsection` 两个级别。目录只需要显示 `section`，如果是大型文档有 `part` 的时候可以考虑加入少量的 `subsection`，这样目录才不至于过于庞大反而失去了实用性。这所有的 `section` 潜在的一个大的分类是 `chapter`，但是这里不需要写出来，因为这个时候整个文档的级别是 `article`。也就是通常所见的小容量的书小册子，某一个专门课题的讨论就按照 `article` 来处理。如果上升到某一个学科不同课题的讨论，那么上面 `article` 隐藏的 `chapter` 写出来，然后将他们合并为 `book` 类，这个时候这个 `book` 潜藏的最高级别为 `part`。如果是不同学科的合并书籍那么级别就上升到 `part` 了。目录在 `book` 类的时候有 `part` 写上 `part`，然后 `chapter` 和 `section` 都显示出来，结构也不会太复杂的。

当然以上讨论只是泛泛而论，你需要根据自己的实际情况来，但总的原则是自己心里应该有一个划分标准，毕竟一本书最有价值的部分就是目录了，如果一本书的目录结构是乱七八糟的那么这本书不值一看。

一般的可选项

10pt, 11pt, 12pt 设置文档所使用字体的大小，默认是 10pt[\[wikibook-latex\]](#)

。

`a4paper` , `letterpaper`... 定义纸张的大小, 此外还有 `a5paper`, `b5paper`, `executivepaper`, `legalpaper` 等。

`fleqn` 设置该选项将使数学公式左对齐, 而不是中间对齐。

`leqno` 设置该选项将使数学公式的编号放置于左侧而不是右侧。

`titlepage`, `notitlepage` 指定是否在文档标题后开始新一页, `article` 文档类不开始新页, `report` 和 `book` 开始。

`onecolumn`, `twocolumn` 指定 LaTeX 以单栏或双栏方式排版文档。

`twoside`, `oneside` 指定 LaTeX 排版文档为双面或单面格式, `article` 和 `report` 默认为单面, `book` 默认为双面。

`openright`, `openany` 定义 `chapter` 开始时仅在奇数页或者随意, `book` 类默认 `openright`, `report` 默认 `openany`, `article` 没有 `chapter`。

书籍的通用结构

通常一本书是由好几部份构成的, 包括封面、扉页、书名页、目次、序、内文、补充或参考资料、版权页。

出版的书籍的封面和扉页这里我们不考虑。电子书籍就从书名页开始说起。也就是我们的 `maketitle` 命令。这个时候也可以认为书名页作为通常意义上的封面。`maketitle` 可以生成多页, 你可以考虑把版权页也算在里面, 因为出版的书籍那个版权页刚好在背面, 而电子书籍我觉得版权页还是放到最后面合适一些, 当然多页封面你还可以自己加点名言警句页, 这个看自己喜好了。

然后接下来是序言部分, 自序或者他序都可以。自己编的电子书籍就是自序了, 自序内容不宜过长, 相当于论文的摘要部分, 用最简短的话让别人对这本书有一个大概的印象。

接下来如果有 `listoftables` 和 `listoffigures` 的就放到这里, 这个看个人喜好, 似乎在一般都喜欢放在书后面吧。

然后是 `tableofcontents`, 目录。

然后是正文部分, 包括引言, 前言等。

然后是 **appendix**，附录部分是用来提示一些与内容有关而不便载于正文里的资料。

然后是索引：是针对这本书中重要资料如人名、地名、概念等的查检。将本文的重要概念列出，并注明出现在文中的页次。它是依一定的方法排列，通常中文是按字体的笔划多寡决先后顺序，西文则按字母的顺序排列，以便检查。通常附录是直接资料，索引则是提供查询资料的线索。（索引主要是方便纸质书检索，我觉得如果一本书如果电子化了还是很容易搜索了，可以不加索引了。）

页码控制

frontmatter 命令跟在 **begin document** 后面，接下来页码为罗马数字。

mainmatter 命令放在正文开始的前面，表示页码的阿拉伯数字开始计数。

appendix 命令表示附录开始，后面各章节改为字母标记。页码没有变化

backmatter 命令放在参考文献或者索引的前面。章节编号关掉，页码没有变化。^①

^① **backmatter** 不能在 **appendix** 前面，请参考[这个网站](#)。

页面布局

页面布局最好用`geometry`宏包调节。

geometry 宏包详细讨论

页面布局尺寸由 `geometry` 宏包指定，页面布局包含很多参量也就是 `geometry` 的选项，请看下图3-1：

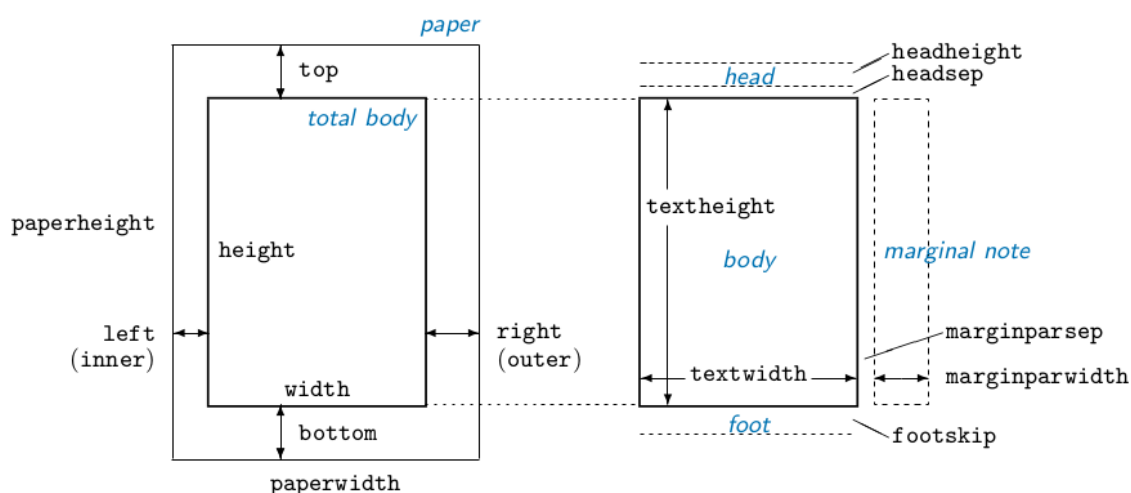


图 3-1: geometry 选项 1

`geometry` 提供的纸张类型很多，从 `a0paper` 一直到 `a6paper` 都有，还有 `b1paper` 到 `b6paper` 系列等等。纸张类型指定了后面的 `paperwidth` 和 `paperheight` 就都确定了。

我们先来看横向参量，`paperwidth` 是纸张的宽度，`textwidth` 是正文宽度，`marginparsep` 指旁注和正文之间的间距，`marginparwidth` 指旁注宽度，`left` 指左

边空白宽度, `right` 指右边空白宽度。如果 `book` 类型是 `twoside` 的, 那么 `left` 最好命名为 `inner`, 也就是类似出版书籍靠里面的那段空白宽度, 类似的 `right` 最好命名为 `outer`, 靠外面的那段空白宽度。其中默认情况下 `width=textwidth`, 如果加入选项 `includemp=true`, 那么:

`width=textwidth+marginparsep+marginparwidth`。然后还有:

`paperwidth=left+width+right`。(1)

再来看竖向参量, `paperheight` 为纸张高度, `textheight` 是正文高度, `top` 为上面的空白高度, `bottom` 为下面的高度, 默认 `top` 包含页眉高度 `headheight` 和页眉于正文间的一段小空白 `headsep`, `bottom` 包含页脚高度(2) `footskip` 和下面的一段空白。所以你的 `top` 至少要大于 `headheight` 高度。然后中间的区域高度为 `height`。默认情况下 `height=textheight`, 请看下图3-2

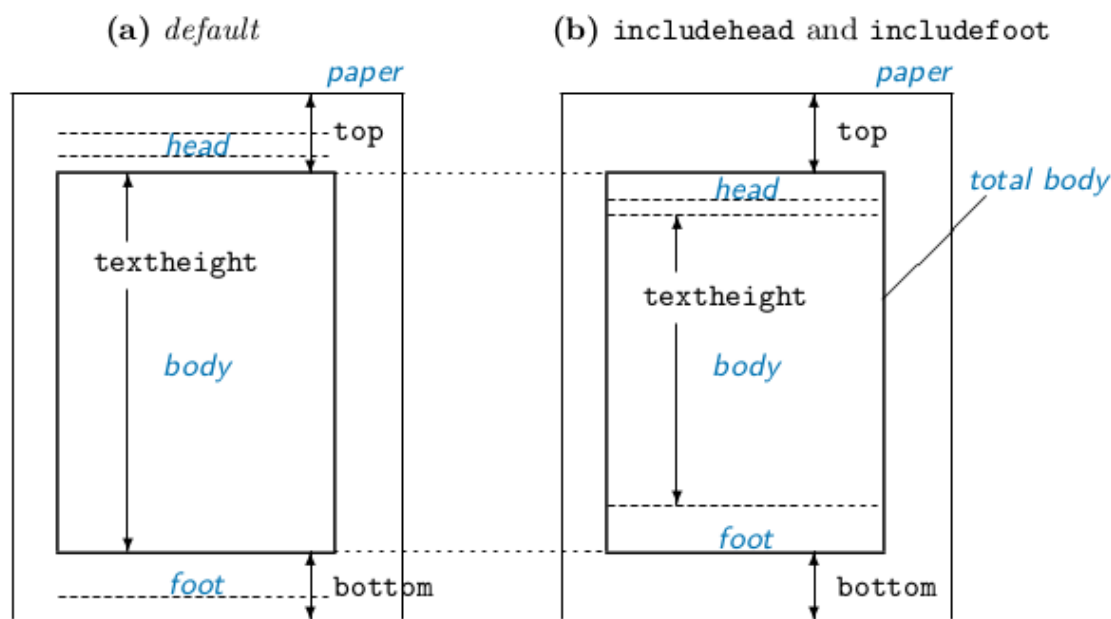


图 3-2: geometry 选项 2

如果加上 `includehead=true` 选项, 那么就和上图右边描述的类似, 页眉部分计入 `height`, 类似的有 `includefoot=true`, 那么页脚部分也计入 `height`。

`geometry` 的机制是以上讨论的横向或者竖向参量指定足够的数量之后, 剩下的可以自动计算得到。没有明确指定的参量虽然可以通过计算得到, 但是在后面似乎是不能够作为变量使用的?

多栏环境

多栏环境推荐使用 **multicol** 宏包。⁽³⁾ 这个宏包很厉害，支持两栏到十栏的环境。就作为通用的一般形式如下：

在这里每一栏的宽度表示为 **linewidth**，这个可以用来控制放进去的图片宽度。这里用 **columnbreak** 命令手动调整栏的跳转。你也可以不用 **columnbreak** 命令，而让 **T_EX** 自动计算栏的高度和分布等。不过似乎用 **columnbreak** 只能近似控制，并不是那种完全严格的跳转命令。这是最基本的应用，请参看多张图片并列显示这一小节。[F.7](#)

如果你希望整个文档都分为两栏，那么在前面 **documentclass** 命令可选项里加上 **twocolumn** 即可。这里的分栏环境是分两栏，然后加入的分栏线。就是在分栏环境中用 **setlength** 调整长度量 **columnseprule** 为 0.4pt^①。觉得这个命令应该重新修改下，直接 **\columnseprule** 就是加分栏线，然后后面跟个可选参数表示线的宽度。

还有一个长度量 **columnsep** 表示栏之间的间距宽度，一样用 **setlength** 调节。一般没啥好调整的。分栏环境就这样简单说下吧。

分栏环境备用

在实际应用过程中，常常遇到分两栏的情况，但是这两栏宽度是可以自由调整的。我找了一下似乎并没有类似 **beamer** 类 **columns** 环境那样方便的存在，只好通过 **minipage** 命令简单地实现了类似的效果。实际应用我是通过 **texmaker** 自定义快捷输入代码模块功能快速输入进文档的。

```
\noindent
\begin{minipage}{\textwidth}
\begin{minipage}{0.38\textwidth}

\end{minipage}\hfill
\begin{minipage}{0.6\textwidth}
```

^① 参考了[这个网站](#)

\end{minipage}

\end{minipage}

<i>t</i> (分)	<i>s</i> (英尺)
0	0
1	1200
2	4000
3	9000
4	9500
5	9600
6	13000
7	18000
8	23500
9	24000

表 2-1

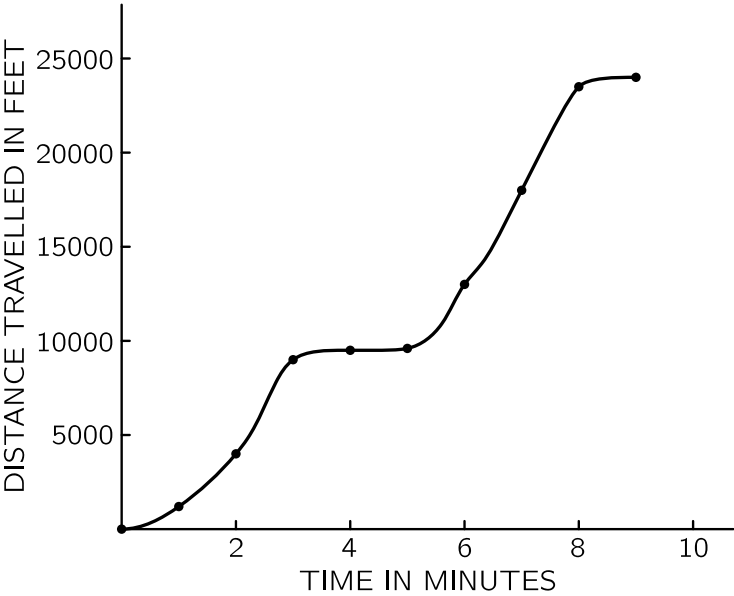


图 3-3: 汽车的距离 -时间曲线

解决方案不是很完美，但勉强够用了。

字体

我们知道 `xelatex` 的机制可以调用系统内的任意字体，当然系统没有的字体就要自己安装了，请看[B.14](#)。

字体的五种属性

\LaTeX 的字体有五种属性，这五种属性是：字型编码，字族，字型系列，字形，字号，即：`encoding,family,series,shape and size`。

字型编码

字型编码即各个个别的字在一个字型里头的排列顺序以及安排方式。原始的 \TeX 字型编码我们就称为 `OT1(Old TEX text encoding)`，这是预设的，如果都不指定字型编码，那所使用的就是 `OT1` 编码。在目前新一代的字型编码里头，字的安排方式及内容和 `OT1` 不一样，例如 `T1` 等。

字族

字族分为三大类，`roman or serif (rm)`，`sans serif (sf)` 和 `monospace (tt)`。⁽⁴⁾

serif Serif 中文译为「有衬线字体」，衬线即是印刷字体在每个笔划起始与终止处，加上短线或三角突起等，以便于快速辨认字符，利于阅读，为印刷专用字体。旧版 Windows 与较旧的 Linux 发行版以此为预设显示字体，而英文新版则改为 Sans-Serif，中文新版则是：当字体大于某一程度时，则将 Serif 的明体或宋体，以 Sans 的黑体取代。

Serif 字体著名的有：Times New Roman、DejaVu Serif、宋体、明体等。

sans-serif Sans-Serif 中文译为「无衬线字体」，专用于荧幕、简报、艺术字体、展示等，较美观，但不适于长时间阅读。多数英文语系的作业系统多以此为预设字体，而采用此种字体为预设的中文作业系统，以 Mac 系统最为著名。

Sans-Serif 字体著名的有：DejaVu Sans、Helvetica、Verdana、圆体、黑体等。

monospace Mono 意思是「单一的」，space 意思是「空间」，中文翻为「等宽字体」。等宽字是打字机时代下的遗产，每个英文字母皆设计为同一宽度，以便于排版；现代为节省不必要空间的浪费，依字母形状分配其宽度，如 **m** 与 **i** 其宽度便不相同，不相信可以拿尺来量看看。**Monospace** 现多用于终端机显示、程序码表示等。

Monospace 字体著名的有：Andale Mono、DejaVu Sans Mono、Courier、AR PL New Sung Mono。

字型系列

正常用的是 medium, **m**。粗体是 bold, **b**。然后还有 Bold extended, **bx**。还有 Semi-bold, **sb**。。和 Condensed, **c**

字形

字形有正常的 normal, **n**。还有意大利斜体 Italic, **it**。斜体 Slanted, **sl**。和 Small Caps, **sc**。

字号

比如说本文设置的就是 11pt。

调整五种属性

调整字型编码

X_YLaTeX 只处理 UTF-8 编码，那个调整字体编码的 `inputenc` 和 `fontenc` 宏包都不要用了。

调整字族

有两种方法设置，一种是命令式的，一种是环境式的。`roman font family` 是默认的字族，一般不需要明确设置。

命令式	环境式	描述
<code>\textrm{text}</code>	<code>{\rmfamily text}</code>	roman 字族
<code>\textsf{text}</code>	<code>{\sffamily text}</code>	sans-serif 字族
<code>\texttt{text}</code>	<code>{\ttfamily text}</code>	monospace 字族

表 4-1: 调整字族

调整字型系列

默认的 `medium`，一般不需要设置，然后还有一个 `bold`，即字体加粗。

命令式	环境式	描述
<code>\textmd{text...}</code>	<code>{\mdseries text...}</code>	正常字体粗细
<code>\textbf{text...}</code>	<code>{\bfseries text...}</code>	bold 粗体

表 4-2: 调整字型系列

调整字形

默认是 `upright shape`，常用的字形如下：

命令式	环境式	描述
<code>\textup{text...}</code>	<code>{\upshape text...}</code>	正常字形
<code>\textit{text...}</code>	<code>{\itshape text...}</code>	意大利斜体
<code>\textsl{text...}</code>	<code>{\slshape text...}</code>	斜体
<code>\textsc{text...}</code>	<code>{\scshape text...}</code>	small caps

表 4-3: 调整字形

调整字号

相对字号调整

LaTeX 里面自带的调整相对字号命令如下：

命令	输出
<code>{\tiny test line}</code>	test line
<code>{\scriptsize test line}</code>	test line
<code>{\footnotesize test line}</code>	test line
<code>{\small test line}</code>	test line
<code>{\normalsize test line}</code>	test line
<code>{\large test line}</code>	test line
<code>{\Large test line}</code>	test line
<code>{\LARGE test line}</code>	test line
<code>{\huge test line}</code>	test line
<code>{\Huge test line}</code>	test line

表 4-4: 调整字体大小命令

然后我们看下图，不同字号下这些命令确切的大小⁽⁵⁾：

size	10pt (default)	11pt option	12pt option
\tiny	5pt	6pt	6pt
\scriptsize	7pt	8pt	8pt
\footnotesize	8pt	9pt	10pt
\small	9pt	10pt	11pt
\normalsize	10pt	11pt	12pt
\large	12pt	12pt	14pt
\Large	14pt	14pt	17pt
\LARGE	17pt	17pt	20pt
\huge	20pt	20pt	25pt
\Huge	25pt	25pt	25pt

表 4-5: 字体具体大小

绝对字号调整

上面的命令基本上够用了，并不鼓励使用绝对字号。不过有的时候还是有用的，比如旁注环境需要同时调整字体大小和行距，使用这个命令注意不要对文本中某一小段文字使用，否则会造成行距的不一致。比如这个旁注使用的命令是：

```
\newcommand{\sidenote}[1]{\marginpar{
  \fontsize{10pt}{20pt}\selectfont #1}}
```

其中 `fontsize` 就是调整字号的命令，第一个参量是字体的大小，第二个参量是行距。然后后面 `\selectfont` 必须写上，就理解为表示对后面的字体进行操作吧。类似的还有 `fontencoding`, `fontfamily`, `fontseries`, `fontshape` 命令，这些本文略过。

字体配置

`xelatex` 字体配置有名的三件套就是：`xltxtra`，`fontspec` 和 `xunicode` 这三个宏包。其中 `xltxtra` 宏包是专门处理 XeLaTeX 的一些问题的，它会自动加载后面的 `fontspec` 和 `xunicode` 宏包。`xunicode` 是处理某些特殊字符的。注意 `xltxtra` 宏包在加载时我放到所以字体配置宏包的后面，否则会出现错误。下面就字体配置相关宏包详细说明之。

fontspec 宏包详解

当然来自 `fontspec` 宏包的帮助文档。后面关于宏包的相关信息来源于自身帮助文档我就不说明了。`fontspec` 宏包主要用于英文字体的设置，中文字体的设置建议用 `xeCJK` 宏包来管理。 XeLaTeX 只能使用 `opentype` 和 `truetype` 字体。

基本的命令

`fontspec` 宏包最基本的应用就是用 `setmainfont` 来设置文档的默认 `roman` 字族字体的，`setmainfont` 原来的名字叫 `setromanfont`。^①`setsansfont` 是设置文档默认 `sans-serif` 字族字体的，`setmonofont` 是设置 `monospace` 字族字体的。然后我们看到前面都有一个可选项 `Mapping=tex-text`，这个说是让 XeLaTeX 文字分布更好的，可能和正确断行有关系，不大确切。

`fontspec` 宏包有一个和这个宏包名字一样的命令，这个命令非常的基本，大约相当于引擎的入门口，我估计前面三个命令实际上是建构在 `fontspec` 命令之上的。`fontspec` 命令的作用不光临时改变字体。还可以加上很多可选项，比如字体尺寸，颜色等等。总之 `fontspec` 命令的优先级要高于前面的三大默认字体设置命令。请看下面的例子：

^① 参考了这个网站

```
{\fontspec[Scale=4,Color=magenta]{Ubuntu} this is a test.}
```

this is a test.

另外还有一个有用的命令就是 `newfontfamily`，这个命令相当于把 `fontspec` 命令包起来再新建一个命令。新建的那些字体命令就像 `\sffamily` 之类的命令一样使用。，在后面讲到引入特殊符号的时候我们还会接触到这个命令。

```
\newfontfamily{\ubuntu}[Scale=3]{Ubuntu}
```

以上基本命令总结如下：

```
\fontspec [ font features ] { font name }
\setmainfont [ font features ] { font name }
\setsansfont [ font features ] { font name }
\setmonofont [ font features ] { font name }
\newfontfamily{ cmd }[ font features ] { font name }
```

上面的 `font name` 在安装字体的时候说明清楚了，比如说用 `fc-list` 命令调出来的宋体, `SimSun:style=Regular` 中字体名字就是宋体或者 `SimSun`。而在 `font-manager` 里面比如第一行显示 `Comic Sans MS Regular`，字体名就是 `Comic Sans Ms`。接下来主要讨论 `font features` 的内容，所讨论的内容以上几个基本命令都适用。由于 `xeCJK` 的 `fontfeatures` 可选项是继承自 `fontspec`，所以下面的讨论也适用于 `xeCJK` 宏包。

font features 的讨论

一个字体的字族之下还分为不同的字形，默认的字形设置可能并不能满足你的要求。有一些字体甚至没有粗体或者意大利体这样的字形。一般的玩家就选用默认字形设置足有了，最多在 `mainfont` 哪里设置下 `boldfont` 和 `italicfont`。比如类似下面这样配置：

```
\setCJKmainfont[BoldFont=Adobe 黑体 Std,
  ItalicFont=Adobe 楷体 Std]
  {Adobe 宋体 Std}% 影響 rmfamily 字體
```

具体字形有：

BoldFont = font name

ItalicFont = font name

BoldItalicFont = font name

SlantedFont = font name

BoldSlantedFont = font name

SmallCapsFont = font name

Color

前面的例子我们看到了 **Color** 属性的定制，在这里推荐适用 **xcolor** 宏包。**X_YLaTeX** 默认使用的是 **xdvipdfm**，不支持透明颜色。然后 **xcolor** 宏包对于颜色的讨论请参看颜色一节[8](#)

字体大小

在前面的那个例子也用到了 **Scale** 选项，选个数字来整体调整这个字体的尺寸大小。还有两个常量值，一个是 **MatchLowercase**，一个是 **MatchUppercase**。就是变成小写字母一样的或者大写字母一样的大小。

词间距——WordSpace

觉得默认的设置更有弹性吧，一般的玩家没必要动它。

标点后间距——PunctuationSpace

从零开始可以加点距离。

断字符号——HyphenChar

就是要换行了选择从哪里断字的符号，比如设置 **HyphenChar** = +，那么标明 +，就从哪里断字。默认的是 \-。

字母之间的距离——LetterSpace

从零开始加点距离，而且只适用于小写字母，感觉很累赘。而且这个 **feature** 只适用于 **X_YTeX**。

最后两个命令

最后还有两个命令 **defaultfontfeatures** 和 **addfontfeatures**。有什么优先级：**addfontfeatures>fontspec>defaultfontfeatures**。觉得太花哨了暂时应该用不到。

设置数学字体

觉得默认的数学字体也很好看的，就不折腾了。

这里简单提一下，数学没有太多额外需要加载的宏包，其中 **amssymb** 和 **amsmath** 这两个宏包必备，基本上大家都加载了的。然后数学上或者物理上等对于国际单位和文本中小的数字（但是需要输入上标下标等的）推荐用 **siunitx** 宏包处理。

然后通过 **\everymath{\displaystyle}** 这个语句可以让文本内的数学形式，比如分数显得更大更好看一点。其他问题这里都略过了。

```
\RequirePackage{amssymb,amsmath}
\RequirePackage{fontspec}
%si unit
\RequirePackage{siunitx}
\sisetup{
math-micro = \text{\mu},
text-micro = \mu
}
\everymath{\displaystyle}
```

xeCJK 宏包详解

xeCJK 宏包只处理 CJK 字符在这里指中文字，也就是英文字还是用前面的 fontspec 宏包来处理。

只能在加载宏包时填入的选项

CJKnumber = <true | false>

默认是 false，如果设置为 true，那么可以适用 CJKnumber 命令。比如这个命令 \CJKnumber{1} 的输出结果是：一。还有一个命令比如：

\CJKdigits{1545} 的输出结果是：一五四五。

indentfirst = <true | false>

章节下面第一段首行缩进不缩进，默认是 true，缩进。这个没啥好修改的，一般都统一缩进吧，如果有某个段落你不想缩进加上 noindent 命令就是了。

可以在 xeCJKsetup 命令中设置的选项

CJKmath = <true | false>

默认是 false，如果设置为 true，那么可以在数学模式下输入 CJK 字符。

PunctStyle = {quanjiao|.....|plain}

其他选项请参看文档，这个是设置标点处理格式的。在本文字体配置第 3 行那里你可以看到我设置为 plain，也就是是什么就是什么，不做处理。默认是 quanjiao。

xeCJK 宏包的一些命令

现在对照 fontspec 宏包对 xeCJK 新建的只针对 CJK 字符处理的命令说明如下，其中各个命令的用法和 fontfeatures 都是类似的。

setCJKmainfont 类似于 setmainfont。

setCJKsansfont 类似于 setsansfont。

setCJKmonofont 类似于 setmonofont。

CJKfontspec 类似于 `fontspec`。

newCJKfontfamily 类似于 `newfontfamily`。

defaultCJKfontfeatures 类似于 `defaultfontfeatures`。

addCJKfontfeatures 类似于 `addfontfeatures`。

setCJKmathfont 估计应该类似于 `setmathrm`。

设置 CJK 字符大小

可以通过如下命令：

```
\defaultCJKfontfeatures{Scale=1.2}
```

来调整所有 CJK 字体的属性，比如上面代码的意思就是所有 CJK 字体都放大 1.2 倍。比如你在 `documentclass` 里面设置 10pt，那么实际 CJK 字符是 12pt。⁽⁶⁾

UTF-8 编码

`texmaker` 软件并没有这个问题，不管怎么样，加上这样一行代码没什么害处喽。确保文档以 `utf-8` 编码打开和保存。在文档开头加上如下代码：

```
% !Mode:: "TeX:UTF-8"
```

本文只关注于 UTF-8 编码。

特殊的符号

去掉符号命令后面的空格

值得一提的是命令如果后面跟上一个花括号，后面的字符紧跟花括号，那么命令显示的符号和后面的字符就没有空格了。比如这里 `& a` 之间就有一个空格，加上花括号`\&{ }a`，`&a` 就没有了。

基本的特殊符号

`\`

`\`，我们知道命令的开头表示就用它，所以在文档中是不能直接使用的，如果需要显示`\`，需要输入 `\textbackslash` 来显示。

`{和}`

`{和}`，同上，作为命令的格式。如要显示前面加上`\`符号。

`%`

`%`，我们知道这个符号在文档是用来标记注释信息的开始的，所以在文档也不能直接使用。在前面加个`\`即可。

~

~，这个符号在文档中产生一格空白，如要显示在前面加上 \ 符号。这样显示的波浪号有点小。还可以进入数学模式下输入 \sim，在 **texmaker** 左边的关系符号一栏中也可以找到。所以为了美观的话就用数学模式吧。然后在 **Rime** 输入法里面我们看到还有一种符号是全角下的波浪号～。我比较了下这个和前面两个都不相同，简单起见用这个全角的波浪号也是可以的。

\$

\$，美元符号之所以不可以用是因为它标记了数学模式的开始和结束。比如说我要输入字母 π ，就是在两个美元符号中间输入 \pi 即可。如果直接从输入法输入希腊字母的符号 π ，这个也是可以正常显示的，只是没有前面数学模式下的美观。我查了一下，数学模式下的那个 π 也是 **unicode** 区域里面的那个希腊字母，我想只是字体问题了。^①

#

#，这个符号没怎么接触，**latex123** 说是定义巨集的，前面加上 \ 即可。

^ 和 _

^ 和 _，这两个符号表示进入数学模式之后进入上标和进入下标。这两个符号在文本中后面还要跟一对花括号，否则显示会出问题。

&

&，这个符号表示表格中的分隔符。要显示前面加上 \ 即可。

^① [这个网站](#)用来查找 **unicode** 和 **L^AT_EX** 命令的对应关系

单引号和双引号

你可以用 `\textquotedblleft` 来表示左双引号，用：
`\textquotedblright` 来表示右双引号。用 `\textquoteleft` 来表示左单引号，用 `\textquoteright` 来表示右单引号。

然后还有敲键盘的方法，要左边和右边区别对待，比如说左单引号就是点击 **ESC** 下面那个键，右单引号是点击分号右边那个键；而左双引号是点两次 **ESC** 下面那个键，右双引号是按住 **shift**，然后点击分号右边那个键。

一边单引号和双引号都用全角模式，中文的输入法调试好会很方便输入的，这不存在问题。至于英文左单引号和右单引号好像一般不怎么区分。

破折号和连字号

连字号就是一个这个 `[-]`，可以直接在键盘上输入。
 短破折号就是两个 `-` 连续输入成为 `→ [-]`
 长破折号就是三个 `-` 连续输入成为 `→ [—]`
 负号如果在数学模式下是 `→ -1` 如果有时嫌麻烦就直接 `-1` 应该可以接受把。

此外中文全角短的连字号有 `[-]`，**unicode** 码是 **U+FF0D**；长破折号是 `[—]`，有两个 **unicode** 码 **U+2014** 的符号组成。值得注意的是长破折号的一个，`[—]`，和前面的全角短连字号也是有所不同的，似乎稍微长一点，一般不怎么用吧，但我发现有些书排版有时用到这个符号。

两个连字号的显示

比如要显示终端选项的时候有两个短连字号，如果你直接输入两个 `-` 就变成短破折号了。如果你要有这样的显示效果，`--`，那么你需要中间加一个花括号^①。下面是三个连字号的例子：

`-{}-{}-`

① 参考网站

温度的度

在 **texmaker** 左边的关系符号哪里我们可以看到一个小圆圈，输入命令是 `\circ`，当然需要在数学模式下。现在就需要让这个圆圈位于上标即可。前面的符号介绍我们提及 `^`，就是进入上标显示，上标的内容在 `^` 后面的那个花括号内，于是我们就知道三十度怎么画了。30°，注意 **texmaker** 左边有快捷键。点起来很方便的。具体代码是：`$ 30^{\circ}`。当然你可以直接用 **rime** 输入法打出这个温度的度：°，°C。在输出% 候选项哪里。

更加专业的做法：

输入温度的度的更加专业的做法利用 **siunitx** 宏包的 `degreeCelsius` 命令输出：

```
30\si{\degreeCelsius} and 45\si{\degree}
```

```
30°C and 45°
```

省略号

this... that...

三个点... `\ldots` 命令...

老实说没看出什么区别，所以省略号就跟着点三个点也是可以的，别用中文句号就行了。

更多特殊符号





上面是 **Ubuntu** 的图标符号，可不是一张图片哦，是一个字符。这个图标符号只在 **Ubuntu** 字体里面才有。现在让我们将思路先理清一下。首先是 **Unicode** 码，这个只是一个理论上的编码规则，具体的实现是字体。但是每一个字体都只专注于某一个领域，并没有把 **Unicode** 所有的码都画出字形来，那么系统是如何显示字体的呢？系统是安装了很多字体，如果一个字体并不包含它要显示的 **Unicode**，它就搜索打开下一个字体文件，找


相关的 Unicode 的字形。只有从字体文件中具体找了这个 Unicode 的字形，才有办法将其显示出来。⁽⁷⁾

我的加入新符号的配置代码如下：

```

\newfontfamily{\libertine}[Scale=1.5]{Linux Libertine 0}
\newfontfamily{\ubuntu}[Scale=3]{Ubuntu}
\usepackage{newunicodechar}
\newunicodechar{@}{{\libertine{@}}}
\newunicodechar{®}{{\libertine{®}}}
\newunicodechar{©}{{\libertine{©}}}
\newunicodechar{®}{{\libertine{®}}}
\newunicodechar{}{{\ubuntu{}}}
    
```

值得提醒的是目前系统的文档里面那个 Ubuntu 图标都不能正确显示出来，而我们在 pdf 中却能显示出来。我们来看看代码。⁽⁸⁾

首先第 1, 2 行前面以及谈及了，主要是看这个新引入的宏包 `newunicodechar`。好吧，这个宏包就只有一个命令，这个命令就是这个宏包的名字。其实我们能够猜到，这个命令的作用就是将这个字符变成类似 \TeX 命令的东西，然后替换为后面的那一串，而后面的那一串单独提出来也是能够正常显示的。如 .

所以过程还是很简单的，但是整个过程让人紧张，如果一篇文章有很多这样的特殊符号，那不要写上长长的这样一串？我想到的第一个解决方案就是最好有这么一个字体包含所有的 Unicode 的字形，然后我试着用 `fontforge` 将一些字形复制粘贴调大小混到一起，然后发现这个方案既满足不了美观的要求也满足不了效率的要求。后来认识到也许我多虑了，如果一个符号经常出现，那么那个字体设计者肯定会优先考虑把它加进去的。

这里有一段代码生成该字体所有已有的字形^{F.8}。当然你可以直接在字符映射表里面选

择只显示已有字形查看，不过有时并不方便。生成那个 pdf 查看之后可以复制粘贴，到那个字符映射表里面搜索，还是很方便的。

初步中文化

主要是些原 \LaTeX 常量是英文单词，然后改成中文字即可。看看代码就清楚了。

```
\renewcommand\contentsname{目 ~ 录}
\newcommand\econtentsname{Contents}
\renewcommand\listfigurename{插图目录}
\renewcommand\listtablename{表格目录}
\renewcommand\bibname{参 ~ 考 ~ 文 ~ 献}
\renewcommand\indexname{索 ~ 引}
\renewcommand\figurename{图}
\renewcommand\tablename{表}
\renewcommand\partname{部分}
\renewcommand\appendixname{附录}
\renewcommand{\abstractname}{摘 ~ 要}
\renewcommand\today{\number\year 年 \number\month 月 \number\day 日}
```

值得一提的是不同的文档类包含不同的命令，比如有的文档类可能没有定义 `today` 这个命令，比如 `article` 类不包含参考文献命令等。

颜色

颜色的心理学

颜色心理学是一门学问，这里不会深究，只是在文档里面字体或者背景使用什么颜色是一门大学问。这里主要参照[这个网站](#)简单说下。本文定义的几个颜色也是参考的这个网站。

红色 一种激奋的色彩。刺激效果，能使人产生冲动，愤怒，热情，活力的感觉。

绿色 介于冷暖两中色彩的中间，显得和睦，宁静，健康，安全的感觉。它和金黄，淡白搭配，可以产生优雅，舒适的气氛。

橙色 也是一种激奋的色彩，具有轻快，欢欣，热烈，温馨，时尚的效果。

黄色 具有快乐，希望，智慧和轻快的个性，它的明度最高。

蓝色 是最具凉爽，清新，专业的色彩。它和白色混合，能体现柔顺，淡雅，浪漫的气氛
(像天空的色彩:)

白色 具有洁白，明快，纯真，清洁的感受。

黑色 具有深沉，神秘，寂静，悲哀，压抑的感受。

灰色 具有中庸，平凡，温和，谦让，中立和高雅的感觉。

关于具体选择什么颜色，我也纠结了很久，但最后无果而终，可选择颜色太多了，我说不上什么意见。

颜色配置

首先推荐使用 `xcolor` 宏包。在[这个网页](#)里，谈到原 `color` 宏包的所有特性基本上 `xcolor` 都支持，同时又加了很多新特性，相当于扩充集吧。

这里放着我之前的一些颜色配置的代码。

```
\usepackage{xcolor}

%===== 在网上找的配黑色文字比较好的背景色 =====%
\definecolor{bgcolor-co}{RGB}{255,255,255}
\definecolor{bgcolor-dd}{RGB}{255,255,200}
\definecolor{bgcolor-tp}{RGB}{215,255,240}
\definecolor{bgcolor-bf}{RGB}{240,218,210}
\definecolor{defaultbgcolor-0}{RGB}{199,237,204} %for eye

%\pagecolor{defaultbgcolor-0}
```

定义新的颜色

上面颜色配置代码里有很多 `definecolor` 命令就是定义新的颜色的，然后在后面要用到颜色的地方使用你这里定义的新的颜色名字就可以了。第一个花括号就填着你定义的新颜色的名字。第二个花括号填着你要定义的颜色模式，比如 `RGB`, `rgb`, `HTML`, `cmyk` 等。最后就是填着对应的模式的数值。其中 `HTML` 模式不要 `#` 号。

用软件查看颜色

你可以用手机照相，或者某个在某个网页某个文档上截图。然后用 `Gcolor2` 软件来捕捉某个点的颜色。

改变文章的背景颜色

上面代码最后的 `pagecolor` 命令就是设置整个文档背景颜色的。我试着在 `minipage` 模式下使用也会改变整个文章的背景颜色。

改变字体的颜色

有两个命令，`textcolor` 和 `color` 命令。

```
\textcolor{colorname}{some text}
{\color{colorname} some text...}
```

我更喜欢 `color` 命令。

xcolor 宏包简介

`xcolor` 宏包虽然是 `color` 宏包的扩展集，但对于我这个颜色知识盲来说多少有点不知所云，可能专业弄颜色的对那些颜色模式的增加还有颜色混合的表达扩展觉得很感动吧。如果你对颜色有更高需求，请详细阅读 `xcolor` 宏包文档，这里不赘述了。

就一般用户还是用 `definecolor` 命令吧，支持的模式有 `gray`, `rgb`, `RGB`, `HTML`, `cmymk`。其中只要加载 `xcolor` 宏包就能使用的颜色名字如下：

颜色	效果	颜色	效果
black		olive	
blue		orange	
brown		pink	
cyan		purple	
darkgray		red	
gray		teal	
green		violet	
lightgray		white	
lime		magenta	
yellow			

表 8-1: 直接可以使用的颜色名字

这个表格里面的小格子涂上颜色是使用的 `xcolor` 宏包里面的 `cellcolor` 命令，就是在表格相应的格子位置使用这个命令就可以了。上面表格类似的一行我写出来吧：

```
yellow & \cellcolor{yellow}\\ \bottomrule
```

如果你加载 xcolor 宏包时填上其他选项，比如 svgnames 等，就会有更多其他颜色的名字可以直接使用了。具体请参看xcolor 官方文档。

单个颜色调百分比

虽然颜色混合我弄不大明白，不过单个颜色调百分比^①还是很有用的。比如 gray 灰色后面跟个!20，就表示 20% 的灰。请看下面不同百分比的红色。




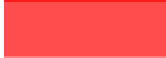

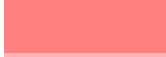


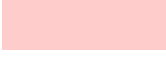
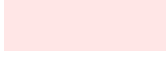
颜色	效果	颜色	效果
red		red!90	
red!80		red!70	
red!60		red!50	
red!40		red!30	
red!20		red!10	
red!0			

表 8-2: 不同百分比的灰色

值得一提的是 xcolor 宏包还支持一种表格颜色交替模式，看上去不错。请看带颜色的表格这一小节21.6。

^① 不清楚和谁调？白色？

段落

换行和分段

一个意思写一个段落，意思没说完就不要分段，在文章中一个段落的书写就是直接写就是了， \LaTeX 会自动处理好一切的。简单的分段的做法就是空一行。这样分段首行缩进仍然存在，表示不同的段落。

如果你只是想换行而不分段，那么用命令：

`\`

可以满足你的要求。

`\` 命令还有一个用法，比如后面跟上 `[10pt]`，表示在原有行距的基础上再加上额外的空白距离，参数也可以是负数。

换行不空一行在 `tex` 文档中的效果只是加上一个空格，你可以前面那行后面加上 `%` 来取消这个空格。

断字

如果你编译的时候出现了 `overfull hbox` 什么的错误，一般是断字出现问题了，有字越过文档边界了。命令 `\` 是强制换行。还有一个换行命令 `\linebreak`，这个命令后面还可以跟个可选项，但意义我不太明确。一般就是观察生成的文档，然后选择合适的位置插入 `linebreak` 命令。这个断行命令断行之后，前面那行的文字将会扩展充满整行，在越界小距离的情况下还是可行的，但是距离大了这一行扩展导致字间距过大也不是十分美观。

一般断字问题主要是英文单词的断字，你可以在你想要断字的位置加上 `\-`，这样系统会自动判断，如果这里适合断字它就会在这里断字，这种断字方法不会加上符号 `-`。其实还

有一种直接的断字方法，那就是在你想要断字的文字加上-\\，这样会显示符号-。第三种断字方法就是在你觉得可能要断字的地方加上符号-，这样系统会自动判断在那里断字，然后也会显示符号-。

段落中的行距⁽⁹⁾

```
\setlength{\baselineskip}{22pt}
```

上面的代码就是设置行间距的。一行一行之间的间距也是一个 glue，我们知道 glue 有基本的 space 和伸缩量。行距的基本 space 由命令 `baselineskip` 控制，伸缩量有 `baselinestretch` 命令。^①

一般行间距的调整用宏包 `setspace`，似乎这个宏包对文档其他内容比如脚注旁注等都进行调整，但是如果你自己定义脚注格式就不太依赖这个宏包了。不过用法也简单，如下所示：

```
\RequirePackage[doublespacing]{setspace}
```

此外还有选项 `onehalfspacing`，如果你试了 `onehalfspacing` 和 `doublespacing`（当然还有默认的不拉伸间距的情况），你都感觉不太满意。这个时候推荐你使用 `linespread` 命令，在刚开始的配置内容里写上即可，不需要额外的宏包加载。其中 `\linespread{1.3}` 的效果相当于前面的 `onehalfspacing`，如果是 1.6 则相当于 `doublespacing`，然后当然，1.4，1.5 等等都随你设置了。

值得注意的是 `linespread` 命令还会对表格有拉伸效果，而 `setspace` 宏包没有这个效果。所以如果用 `setspace` 宏包而不是 `linespread` 调整行间距，那么才需要对 `arraystretch` 量进行一些修改，提供额外的表格行间距。

baselineskip

段落中行之间上下间距^②由三个命令控制：`baselineskip`，`lineskip` 和 `lineskiplimit`。简单的说明就是首先间距是 `baselineskip`，但是如果上面的盒子伸的太下或者下面的盒子伸的太高，那么他们就可能会碰到一起。`lineskiplimit` 控制的就是

① 实际上这个多少有点揣测的意思。

② 一行一行就是横向的盒子，行间距就是横向的盒子之间的上下间距。

盒子之间最小间距，比如 0pt。当 `baselineskip` 减去上面的盒子的深度 `depth` 再减去下面盒子的高度 `height` 然后得到的值比 `lineskiplimit` 小，那么跳转方案就会选择 `lineskip` 模式。也就是上面盒子最低的点和下面盒子最高的点之间的距离是 `lineskip` 那么多。⁽¹⁰⁾

我感觉自己设置一个 `baselineskip` 已经很满足要求了，尤其个别字用 `Huge` 命令的时候效果也还行。

baselinestretch

① `baselinestretch` 量相当于行间距 `glue` 的伸缩量，也就是对前面的 `baselineskip` 做一定的伸缩。这个命令要使用的话格式如下：

```
\renewcommand{\baselinestretch}{伸缩量}
```

这个命令等价于：

```
\linespread{伸缩量}
```

不过这不是故事的全部。如果要在文档中改变行间距必须采用如下的形式：

```
{\linespread{伸缩量}\selectfont sometext \par}
```

其中伸缩量和前面说的一样是：设为 1.3 就是 1.5 倍行距，1.6 就是双倍行距，其他自由发挥。除了要封闭环境否则对其他文本有影响外，还需要加上 `selectfont` 命令，此外最后还必须加上 `\par` 命令，否则会没有效果。

段落间距

```
\setlength{\parskip}{1.6ex plus 0.2ex minus 0.2ex}
```

段落间距也就是一段和一段之间的空白距离。上面就是本文段落间距的设置代码。`parskip` 是一个 `length` 量。其中距离设置设置一个固定量和一个加量还有一个减量，`wikibook` 中说很有用，不太清楚。

① 参看的[这个网站](#)

段首缩进

段首缩进量调整

```
\setlength{\parindent}{\textpt * \real{2}}
```

上面代码就是设置段首缩进量的，其中 `parindent` 是一个 `length` 量。具体就是两个字符的意思。这里预先需要加载 `calc` 宏包^①，为了让设置的距离更加相对化，我新建的一个长度量 `textpt`：

```
\newlength{\textpt}
\setlength{\textpt}{12pt}
```

本文的默认字体大小值是 `12pt`，然后传递到了 `textpt` 这里。

缩进还是不缩进

在 `xeCJK` 宏包加载的时候默认章节下面第一段缩进打开了，然后后面基本上都段段有缩进，也没怎么管了。如果你希望某一段不缩进就用 `noindent` 命令吧，类似的有 `indent` 命令。如果你希望都不缩进，我觉得简单点的办法就是把缩进量设置为 `0` 吧。

段落对齐

`flushleft` 环境为左对齐环境，`flushright` 环境为右对齐环境，`center` 为居中环境。类似的命令样式有 `raggedleft`，`raggedright` 和 `centering`。这些命令还可以控制表格图片（也许是一切盒子？）的位置。简单示例如下：

```
\begin{flushright}
\fbbox{右边才是王道。}
\end{flushright}
```

^① 注意乘法乘以小数的时候需要用 `real` 命令处理。

右边才是王道。

flushright 环境空白间距问题

`flushright` 环境有一个大型的空白间距，如果你不喜欢这么大的空白间距正如前面谈及的，可以使用 `raggedright` 等命令来调整，不过 `raggedright` 等命令似乎更适合对图片表格等盒子调整位置（`raggedright` 命令不太好用，有时不知怎么不起作用，推荐就用 `hfill` 命令让其他内容居右，唯一要提醒的是 `hfill` 命令是针对当前行把内容往右推。）

那么在段落内如何实现类似 `word` 那种一行之类居右对齐？

```
this is a test line
```

```
{\vspace{-\parskip}\hfill some text}
```

这里推荐先空一行分段，然后用 `hfill` 命令将文本推到右边去，然后用 `vspace` 调整下距离。这里选择 `-\parskip` 这样得到的行距和其他行距是一致的。

然后我注意到开辟新的竖向模式

，比如换行之后，不空一行用 `smallskip`，`medskip` 等，然后不用 `vspace` 命令调整也是可以的。因为 `smallskip` 也是一种 `vspace` 命令的调整模式，但是行距和其他的不太一致，所以在这里推荐就空一段，然后负的 `parskip`。当然如果可以接受不用 `vspace` 调整也是可以的。

页眉页脚设计

页眉页脚设计推荐用 **fancyhdr** 宏包。

观察

我什么都没设置，生成的文档有章名的那一页中间有页码，其他页右上角有页码。有章名的那一节可能样式是 **plain**，其他页可能是默认的 **myheadings** 样式。

全部信息归零

plain 样式重置

fancyhdr 宏包提供了一个命令 **fancypagestyle** 来重新或者定义一个新的页眉页脚样式。现在我将 **plain** 样式所有信息全部清除，宁愿没有也不愿出现其他别样的信息。

```
\fancypagestyle{plain}{  
  \fancyhf{}  
  \renewcommand{\headrulewidth}{0pt}  
  \renewcommand{\footrulewidth}{0pt}}
```

该代码第一行是提出要重新定义 **plain** 样式，然后里面包含新的定义。第二行是所有的页眉页脚信息换为空值，第三行是将页眉那条线宽度设为 **0pt**，也就是不显示了，第四行类似是页脚那根线不显示了。

选择默认样式为 empty

使用命令：

```
\pagestyle{empty}
```

也就是明确指定页眉页脚样式为已经有的样式 **empty**，即什么都没有。现在文章所有的页眉页脚信息都被清除了。现在摆在我们面前有两条路，一条是继续 **DIY plain** 样式，然后全部页面设置为 **plain** 样式。一条是 **plain** 继续归零，然后自己 **DIY** 一个新的样式并使用这个样式。我在这里选择第一条道路了。

继续定制 plain 样式

我不太喜欢那一条横线，有需要的请自己将前面的什么 **rulewidht** 命令横线宽度设为 **0.4pt** 左右。

我是一个喜欢简单的人，**fancyhdr** 宏包里面还有很多命令：

```
\lhead[<even output>]{<odd output>}
\chead[<even output>]{<odd output>}
\rhead[<even output>]{<odd output>}
\lfoot[<even output>]{<odd output>}
\cfoot[<even output>]{<odd output>}
\rfoot[<even output>]{<odd output>}
\fancyhead[selectors]{output you want}
\fancyfoot[selectors]{output you want}
```

这些命令都可以用 **fancyhf** 命令来达到，所以我不做介绍了，有需要的请参考 **fancyhdr** 宏包文档。

fancyhf 命令的可选项

fancyhf 命令的一些可选项如下表所示，比如偶数页的页眉左边位置的表示就是 **EHL**，奇数页页脚中间位置就是 **OFC**。本文没有区分偶数页和奇数页，而且也不想有过多的信息，比如就想页眉左边和页脚右边有点内容，那么可选项为 **HL** 和 **FR**。

字母	意义
H	页眉 (head)
F	页脚 (foot)
L	左边 (left)
C	中间 (center)
R	右边 (right)
E	偶数页 (even)
O	奇数页 (odd)

表 10-1: fancyhf 可选项字母意义

本文页眉页脚配置代码

```
%===== 页眉页脚格式 =====%
\RequirePackage{fancyhdr} % 页眉页脚
\pagestyle{fancy}
\renewcommand{\sectionmark}[1]
{\markright{第 \CJKnumber{\arabic{section}} 节 ~~#1}{}}

\fancypagestyle{plain}{%
    \fancyhf{}
    \renewcommand{\headrulewidth}{0pt}
    \renewcommand{\footrulewidth}{0pt}
    % \renewcommand{\chaptermark}[1]
    %{\markboth{第 \CJKnumber{\arabic{chapter}} 章 ~~#1}{}}
    % \fancyhf[HL]{\ttfamily \footnotesize \leftmark }
    \fancyhf[HR]{\ttfamily \footnotesize \rightmark }
    \fancyhf[FR]{\thepage}
    \fancyhfoffset[R]{\marginparwidth+\marginparsep}}
\pagestyle{plain}
```

首先是加载 `fancyhdr` 宏包^①。接下来是页面风格 `fancy`，不先这样做后面的 `sectionmark` 的重定义无效，具体原因不明。

然后接下来重定义 `sectionmark`，这个 `sectionmark` 我不太明白，十分有意思的它在 `exam` 文档类里面照样有效，但是如果不设置 `\pagestyle{fancy}` 它就无效，然后放在新定义的 `plain` 样式里面也会出错。

下面就是重新定制 `plain` 样式，首先是：

```
% \renewcommand{\chaptermark}[1]
%{\markboth{第 \CJKnumber{\arabic{chapter}} 章 ~~#1}{}}
% \fancyhf[HL]{\ttfamily \footnotesize \leftmark }
```

为了文档简洁这些代码我都注释掉了，不过你可能用的着。这里 `leftmark` 就是目前的章名，`fancyhf` 带上选项 `HL` 就是定义文档页眉左边的内容。然后通过修改 `chaptermark`（如果你章节标题不自己 `DIY` 的话，这里修改 `chaptermark` 你会看到变化的。）会影响 `leftmark` 的内容。这里的 `markboth` 也是配套的。

```
\fancyhf[HR]{\ttfamily \footnotesize \rightmark }
\fancyhf[FR]{\thepage}
```

这里的 `rightmark` 就是目前的节名，通过修改 `sectionmark` 会影响它的内容。`sectionmark` 的重新定义如前所示是和 `markright` 命令配套的。

我起初试图如下重新定义 `chaptername` 和 `sectionname`，然后重新定义 `thechapter` 和 `thesection` 也就是当前章节编号。在这里先将编号用 `arabic` 命令转化为 `1,2,3...` 的形式，然后用 `CJKnumber` 命令转化为一, 二, 三... 的形式。可是目录形式也跟着改变了，还有图片编号。影响范围太大了，所以就直接用上面代码的形式组合了。

`FR` 也就是右边页脚处，哪里简单填上 `\thepage` 即表示当前页码。具体格式不用设置，请参看页码这一小节2.4。

^① `exam` 类有自己的页眉页脚定制方法，和 `fancyhdr` 宏包有冲突。

```
\fancyhfoffset[R]{\marginparwidth+\marginparsep}
```

最后这段代码可要可不要，作用是让页眉页脚扩展到旁注的宽度那里。放在这里，如果设置 `marginparwidth` 和 `marginparsep` 都是 `0pt` 的话，也没有什么影响的。

章节标题设计

推荐使用`titlesec`宏包进行章节标题设计，当然还有其他的宏包可以设计出更加花哨的章节标题这里忽略。

本文章节标题设计

```
\usepackage{titlesec}
\titleformat{\part}{\huge\sffamily}{}{0em}{}
\titleformat{\chapter}{\LARGE\sffamily}{}{0em}{}
\titleformat{\section}{\Large\sffamily}{}{0em}{}
\titleformat{\subsection}{\large\sffamily}{}{0em}{}
\titleformat{\subsubsection}{\normalsize\sffamily}{}{0em}{}

```

本文章节标题设计非常简单，几乎就是 `titleformat` 命令各个选项的空值。

`titleformat` 命令说明

`titlesec` 宏包还提供了其他一些命令，不过一切设置都可以通过 `titleformat` 命令来获得：

```
\titleformat{\chapter}[shape]{格式}{label}
{sep}{before-code}[after-code]

```

第一个花括号是选择你要修改的目标，也就是是章啊，还是节啊，还是小节。从 `part` 到 `subparagraph` 都可以的。

后面是 `shape`，一个可选项，没看懂要干嘛的。估计也不太重要吧。

第二个花括号是重点，里面放着格式命令，比如设置字体字族，字体大小，颜色等都可以的。这个格式影响后面要讲的 `label` 标签还有标题文本。^①

第三个花括号是标签，空着就是没有标签。你也可以——比如说对于 `section` 的标题——填上 `\thesection` 表示节的编号。

第四个花括号是标签和后面标题文字之间的空隙，这里因为没有 `label` 所以设为 0 了，如果有 `label` 还是加点距离。

第五个花括号和后面的可选项是什么标题盒子之前和之后的代码，这里忽略。有兴趣请参看文档。

章节编号数值修改

`part`, `chapter`, `section` 等这些都是 `counter` 量，通过 `setcounter` 命令直接修改——比如说 `section`——为 2，那么接下来 `counter` 自动计数是从 2 开始，下一个 `section` 编号是 3。

章节编号深度修改

通过设置 `secnumdepth` 这个 `counter` 的量可以设置章节编号深度。

```
\setcounter{secnumdepth}{1}
```

默认是 2，编号到 `subsection`，你可以设置为 3，`subsubsection` 都有编号，或者设置为 1，`section` 有编号。我想设置为 0 的话 `section` 也没有编号了。

章节编号形式修改

`thepart`, `thechapter`, `thesection`, `thesubsection` 等你可以称他们为标签吧，也就是通常我们看到的编号那部分内容。通过对这些命令重定义就可以对他们进行修改

^① 这里对齐命令，`vspace` 命令等都是可以用的？

了。

```
\renewcommand{\thesection}{\arabic{section}.}
```

那么就会有 1. 的形式。

章节标题上插入脚注

一般使用章节标题命令就是`\section{text}`，前面还是可以加个可选项的，这个可选项的文本将作为目录中实际显示的文本，而后面则是实际执行的文本。比如：

```
\subsection[text]{\color{red} text}\footnote{脚注}}
```

text^①

① 脚注

目录设计

一般在封面之后插入目录，用 `tableofcontents` 命令即可。本文没有对目录做太多的修改，默认的目录格式挺好的。唯一使用的技巧是新建一个 `common-format` 环境，用这个将除了目录和封面之外所有的其他内容都包括进去。

```
\newenvironment{common-format}{ %  
    \setlength{\parskip}{1.6ex plus 0.2ex minus 0.2ex}  
    \setlength{\parindent}{\textpt * \real{2}}}  
{}
```

目录深度控制

```
\setcounter{tocdepth}{1}
```

`tocdepth` 是一个 counter 量，默认是 2，显示到 subsection。这里为了是目录更加简洁，设置为了 1 即显示到 section。0 显示到 chapter，-1 显示 part，-2 就什么都没有了。

前言加入目录

本文定义了两个命令为了做到这点：

```
\newcommand{\addchtoc}[1]{ % 目录中加入新章节  
    \cleardoublepage  
    \phantomsection
```



```

\addcontentsline{toc}{chapter}{#1}}

\newcommand{\addsectoc}[1]{ % 目录中加入新的 section

\phantomsection

\addcontentsline{toc}{section}{#1}}
    
```

两个命令大致类似吧，一个是针对 **chapter** 的，一个是针对 **section** 的。其中 **chapter** 的需要加上一个 **cleardoublepage** 命令，其具体解释参看[这个网站](#)。意思是首先结束本页，然后将所有图片和表格都显示出来。在两面 **twoside** 模式里，要确保下一页从奇数页开始，必要时插入空白页。

后面的 **phantomsection** 命令^①是由 **hyperref** 宏包提供的，需要加上它使链接有效。[\(11\)](#)

然后 **addcontentsline** 命令就是将目前章节加入目录，第一个花括号里面的选项有：**toc**, **lof**, **lot**。也就是目录，图目录和表目录。第二个花括号里面如果是 **toc** 的话就是 **part**, **chapter** 之类的，如果是 **lof**，好吧，一般加入 **label** 就可以了，还没接触这种情况，说不上什么。最后那个花括号里面放着要在目录上面显示的文字。

本文目录的前面加入如下命令即可：

```
\addchtoc{目录}
```

同理，参考文献处理情况类似。值得提醒的是 **section** 的时候注意换命令。然后 **part**, **section** 之类的命令带个星号 ***** 表示不编号不进入目录，这个前面说过的。

目录行间距拉大

我试图建立一个通用的格式环境横跨目录的时候会失效，只好将其分开。不过这样做带来一个好处，那就是你在目录前面设置一些格式只对目录起作用。比如：

```
\addtolength{\parskip}{8pt}
```

这里对段落之间的间距增加了一点宽度，这个会影响一条条目录之间的行间距。

^① 这个命令名字真不好记。。

目录编号和标题间距调整

如果你的目录编号，也就是前面的 `label` 和标题内容有点重叠，那么可以用 `tocloft` 宏包来自动调整目录编号和标题间距。具体代码如下：

```
\RequirePackage{tocloft}%  
\renewcommand{\numberline}[1]{%  
  \@cftbsnum #1\@cftasnum~\@cftasnumb%  
}
```

well，具体看不懂，不过把这段代码加上之后效果确实不错，请参考[这个网站](#)。

封面设计

基础的封面就是用 **title** 输入题目，用 **author** 输入作者，用 **date** 命令输入日期，默认是输出当时编辑的日期的。**author** 里面可以用 **and** 命令连接几个作者或者用 **** 命令换行。然后用 **maketitle** 命令插入一个封面即可。

一个简单的封面

为了说明基本知识我使用本文早期的一个简单的封面作为例子讲解。

```
%===== % 封面設計 =====%
\makeatletter
\renewcommand\title[1]{\def\@title{#1}}
\renewcommand\author[1]{\def\@author{#1}}
\newcommand\email[1]{\def\@email{#1}}
\newcommand\version[1]{\def\@version{#1}}
\newcommand\editor[1]{\def\@editor{#1}}

\renewcommand{\maketitle}{
  \begin{titlepage}
  \begin{flushleft}

    \vspace*{\stretch{1}}
    {\Huge\sffamily \@title}\\[10pt]
    {\sffamily\large 作者: \@author}\\
```

```

\vspace{\stretch{1}}
{\sffamily 编者: \@editor}\[10pt]
{\sffamily 邮箱: \href{mailto: \@email}{\@email}}\

\vspace{\stretch{1}}
{\large\ttfamily 版本号: \@version}\[10pt]
{\large\ttfamily 完成日期: \today}\

\end{flushleft}
\end{titlepage}
}
\makeatother

```

这个代码一些小的细节我就不说明了，现在就主要内容说明一下。这个封面主要是通过重新定义 **maketitle** 命令来完成的，然后前面还加上了一些新的输入命令。其中 **makeatletter** 和 **makeatother** 一前一后表示他们夹著的内容 @ 这个符号就是一个符号，这样 ‘@abc’ 和 ‘abc’ 是不同的。

def 命令是 **T_EX** 的原始定义命令，比如说：

```
\newcommand\email[1]{\def\@email{#1}}
```

就是定义 **@email** 命令，然后这个命令的输出就是输出 **#1**。**#1** 也就是第一个参数。也就是 **email** 命令接受到的参数。

然后封面设计就是进入 **titlepage** 环境进行一些排版操作即可。这里涉及到的居左对齐，**vspace** 等命令就不多说了。

我的建议是一般对封面没什么要求的就用默认的风格或者类似前面的稍作 **DIY**，如果要做出出版级的华丽封面，而且要每本书都不同的话，建议还是用 **tikz** 绘图或者其他绘图软件来处理，然后把图片导入即可。毕竟 **L^AT_EX** 的设计本意只是日常文档排版。

引用

文章内部引用

某一个特别的章节图片或者表格等需要被引用时，你就在它哪里加上 `label` 命令。然后就可以用 `ref` 命令在文章内部建立链接引用他们了。值得一提的是 `texmaker` 的提示功能非常好。建立 `label` 的时候方便你管理，`section` 部分前面就加一个 `sec:` 前缀，图片加一个 `fig:` 前缀，表格加个 `tab:` 前缀等。然后 `label` 中文英文都是可以的，方便自己管理最好写的很明晰。

比如在这里我插入了一个参考文献的引用，是用的是 `cite` 命令。

`\cite{lsshort}` 请参见文献 [\[lsshort\]](#)

还有一个 `pageref` 命令和 `ref` 命令差不多，文档上写法都类似，不同的是在文章上显示的是页码。

hlabel 命令

```
\newcommand{\hlabel}[1]{\phantomsection \label{#1}}
```

本文定义了一个 `hlabel` 命令，就是在前面加上了 `hyperref` 宏包提供的精确定位命令 `phantomsection`^①。一般 `ref` 图片或者 `section` 或者 `subsection` 等文档片段都不需要这么做。就是比如有时一大段 `section` 里面有十几个段落，占了几页的篇幅，那时你如果直接用 `label` 命令建立链接，这几个小段的链接都会挂在 `section` 开头那里。如果使用这个 `hlabel` 命令那么就会精确定位到你想要的位置。

^① 对这个命令具体作用还不太清楚。

flabel 命令

我在写 `endnotes` 宏包的时候新建了一个命令，还不清楚在外面能不能用。一般为了省心不怎么使用吧。`flabel` 命令有两个必填参数，第一个参数是真实建立的 `label` 标签，第二个参数影响你后面用 `ref` 命令引用这个标签显示的文字。

```
\DeclareRobustCommand*\flabel}[2]
{\phantomsection \def\@currentlabel{#2}
\label{#1}}%label styl customization
```

hyperref 宏包简介

如何插入超链接

这里插入一个超链接到 `google`, [google](https://www.google.com/)
 首先要在前面加载 `hyperref` 库文件

```
\usepackage {hyperref}
```

然后在你想要插入超链接的地方使用命令：

```
\href {https://www.google.com/} {google}
```

链接字体颜色控制

```
\usepackage[colorlinks=true,linkcolor=blue,
            ulrcolor=red,citecolor=blue]{hyperref}
```

colorlinks=true 是把颜色打开，如果是 `false` 那么都是默认的黑色了。后面的选项主要影响文字颜色，还有更多颜色设置。**linkcolor** 影响目录颜色和脚注和内部引用，**ulrcolor** 影响对外链接，**citecolor** 影响对文献的链接。**anchorcolor** 超链接源码。

`hyperref` 宏包目前我的 DIY 只限于一些颜色设置，有兴趣的请自己研究[文档](#)。

脚注

加入脚注还是很有用的^①。具体方法就是：

```
\footnote{这就是一个脚注}
```

脚注标签修改

重定义 `thefootnote` 命令会影响每一条脚注前面的 `label` 样式，比如默认的 `1.`。

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

一页脚注重新编号

我喜欢每一页脚注都重新编号，觉得那些数字反正也没什么意义。具体实现如下代码所示。

```
\usepackage{perpage}
\MakePerPage{footnote}
```

脚注中的距离

- `\footnotesep`，长度量，控制脚注与脚注之间的距离。

^① 这就是一个脚注

- `\skip\footins`，长度量，控制正文和脚注之间的距离。

重定义脚注文本格式

```
\renewcommand\@makefnmark[1]
{\vspace{5pt}
\noindent
\makebox[20pt][c]{\@makefnmark}
\fontsize{10pt}{12pt}\selectfont #1}
```

这里重新定义了（在 `sty` 文件中）`\@makefnmark` 命令，命令的参数就是每一条脚注的内容的。都是一些格式的调整，主要的改动是加入了一个盒子放着脚注的标签（`\@makefnmark`）。

表格里的脚注

[参考了这个网站](#)，具体实现过程很简单，就是表格放入 `minipage` 环境下即可直接使用脚注命令了。

```
\begin{minipage}{6cm}
  \begin{tabular}{|l|c|c|}
    \hline
    A & 1 & 2 \footnote{This is a footnote.} \\
    \hline
    B & 2 & 1 \\
    \hline
    C & 3 & 3 \\
    \hline
  \end{tabular}
\end{minipage}
```

A	1	2 ^{<i>a</i>}
B	2	1
C	3	3

^{*a*} This is a footnote.

旁注

\LaTeX 自带的有 `marginpar` 命令就是插入旁注的（你需要把页面布局的旁注部分设置好。）我也接触过另外一个宏包的 `marginnote` 命令，虽然多了一个可选项可以调整旁注竖向位移，但是带来的麻烦更多，有时候页尾的旁注会向下越界，如果我想写了一个旁注，又在下面继续写一个旁注命令也会出现重叠出错。而 `marginpar` 就没有这样的问题。写了一个 `marginpar`，后面可以继续再开一条 `marginpar` 命令。至于竖向位移的问题，使用 `vspace` 命令调整即可。

下面是我新建的一个 `rightnote` 命令，以供参考。

```
\newcommand{\rightnote}[1]{\marginpar{  
  \fontsize{10pt}{12pt}\selectfont #1}}
```

在 `marginpar` 命令里面进行各种 \LaTeX 命令都是可以的，比如改变字体大小，字体颜色，插入图片等等。

尾注

尾注也就是每一章后面的注释部分，这种排版形式显得更加专业。而且如果尾注写的好参考文献一章或者其他附录部分都可以不用写了。

本来已经有一个 `endnotes` 宏包了，但是这个宏包不支持超链接，然后后来我按照[这个网站](#)的说明操作加入另外一个 `sty` 文件之后是能够正常双向链接的，但是为了解决一个问题引入两个宏包，而且这两个宏包还存在交互问题只是让问题更加复杂了。

我试着慢慢将这两个宏包融合起来，遇到了很多困难，因为我对 `tex` 原初命令并不是十分熟悉，最终似乎解决了这一问题。其中有些细节我也不大清楚，下面就这个新的 `endnotes` 宏包详细说明。

endnotes 宏包说明

整个宏包还是很简洁的，满打满算八九十行。宏包的文件在 `github` 源码的 `texmf` 文件夹里面。

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01]
\ProvidesPackage{endnotes}[2013/11/29 v0.12 endnotes]
```

首先是一些文档的注释说明信息，这里就不写了。

然后是标准的宏包 `sty` 文件的开头样式，这里填上 `endnotes` 宏包即可。

```
\RequirePackage{etoolbox}
\DeclareRobustCommand*{\flabel}[2]%
```

```
{\phantomsection \def\@currentlabel{#2}
\label{#1}}%
```

这里使用 `RequirePackage` 命令引入了 `etoolbox` 宏包，主要是为了引入了 `expandonce` 命令，这个命令是确保后面的一个命令只运行一次。

然后接下来新建了一个命令 `flabel`，这个命令是为了文本中的尾注标记符号和尾注中的标记符号区分开来，比如文本中的尾注标记符号是上标的，而尾注中的标记符号则不是。`flabel` 命令里面使用了 `hyperref` 宏包的精确定位命令 `phantomsection`，因为 `hyperref` 宏包在 `myconfig` 宏包中已经加载了，这里就不加载了。

`flabel` 命令接受两个参数，第一个参数 `#1` 表示真实的 `label`，而第二个参数 `#2` 表示显示的 `label`。所谓真实的 `label` 指等下生成的 `pdf` 中的精确定位 `label` 标记。命令中重新定义了 `\@currentlabel`，也就是显示的 `label`。这样将这两者区分开来，本来他们都是一样的。

```
\gdef\theenmark{\arabic{@EndnoteCounter}}%
\gdef\endnotemark{\textsuperscript{(\theenmark)}}%
\gdef\endnotemarkback{(\theenmark)}%
```

然后上面三行定义了一些对外的接口命令，方便 DIY。`theenmark` 命令是尾注标记符号的计数器部分，这里使用了 `arabic` 命令转化了。`endnotemark` 命令是文本中尾注标记符号的形式，这里简单加上了 `()` 然后使用 `textsuperscript` 命令将其转化成为上标形式，之所以使用这样的形式，是因为使用数字可以没有任何限制，而使用其他奇怪的符号标记要为可能超过界限操心。而 `endnotemarkback` 命令则是尾注中的标记符号形式，这里和前面的区别就是没有转化成为上标。

```
\newlength{\enoteindent}
\setlength{\enoteindent}{2.5\textpt}
```

这段代码的意思是设置了一个长度量 `enoteindent`，然后将其设置为 2.5 倍 `textpt` 长，这个 `textpt` 之前说过的是为了让文档相对化将之前 `documentclass` 设置的 12pt 传递过来的，你也可以在这里设置成一个你喜欢的长度。这个长度影响等下尾注标签的缩进量。

接下来下面的代码首先用 **newwrite** 命令新建了文件，**tex** 中的文件操作命令如下：

```
\newwrite\tempfile
\immediate\openout\tempfile=lists.txt
\immediate\write\tempfile{this is test}
\immediate\closeout\tempfile
```

这里的 **newwrite** 是新建一个写文件，然后后面 **immediate** 命令表示立即执行，**openout** 表示打开某个文件，这里打开了新建的这个写文件 **\tempfile**，后面的等号跟着你想要的这个新文件的文件名，等下电脑中新建出来的这个文件就是这个文件名。

然后 **write** 命令是往这个文件里面写入某个文本信息。接下来是 **closeout** 命令把这个文件关闭，只有文件正常关闭了写的内容才能从缓冲区真正进入系统中的文本。

```
\newwrite\@entout

\newcounter{@EndnoteCounter}%
\newcounter{@AllEndnoteCounter}%
\newcounter{@ShowEndnoteCounter}%
\setcounter{@EndnoteCounter}{0}
\setcounter{@AllEndnoteCounter}{0}
\setcounter{@ShowEndnoteCounter}{1}

\newif\if@entopen \global\@entopenfalse
```

接下来是新建了三个计数器 **@EndnoteCounter** 等，其中 **@EndnoteCounter** 标记每一章尾注内部的计数，而 **AllEndnoteCounter** 标记了所有的尾注的计数，这个影响真实 **label**。**ShowEndnoteCounter** 标记了每一个分章的章数（这里主要指按照每一章，不过实际操作并不局限于此。）的计数。然后用 **setcounter** 命令给这些计数器一个初值。

上一段代码最后还新建了一个条件控制变量，具体请参看 **tex** 条件控制语句一小节 [F.4.1](#)。这里的 **global** 命令是让接下来的定义或者声明是全局性的。

```

\def\@openent{%
\def\@entname{\jobname
\ifnum\value{@ShowEndnoteCounter}<10 0\fi
\ifnum\value{@ShowEndnoteCounter}<100 0\fi
\arabic{@ShowEndnoteCounter}}%

\immediate\openout\@entout=\@entname.ent%
\global\@entopenttrue
}%

```

上一段代码新建了一个命令 **@openent**，这个命令等下要完成打开新建的 **ent** 文件任务，然后正如前面谈及的文件操作，在 **openout** 命令打开文件的时候，这里对具体生成的 **ent** 文件的文件名进行了一些 DIY。其实我可以将文件名简单设置成为 **jobname-1**，**-2** 之类的形式，之所以弄成以上这种形式多少是我在之前探索过程中弄出的一种 **jobname001**，**jobname002** 这种形式，然后方便使用 **linux** 系统的 **cat** 命令来将他们汇总，后来修改修改着这个功能发现不需要了，但这个技巧还是很有用的就没删除了。

下一段的解释先跳过，看到下下一段定义的 **endnote** 命令。

```

\begingroup \catcode '|=0 \catcode '['=1
\catcode']=2 \catcode '\{=12 \catcode '\}=12
\catcode'\|=12
|gdef|@writeent[|immediate|write|@entout[
{\footnotesize
\hspace{-2\parindent}\makebox[\parindent-3pt][l]
{\flabel{|\@linknameb}{|\@showlinknameb} \ref{|\@linkname}}
\leftskip=\parindent |expandonce|@include \par } ] ]%
|endgroup

```

上面这段代码就是定义了那个 **@writeent**，但是以一种十分绕脑袋的形式，我也没完全弄明白。。首先看到 **catcode** 命令，这个是改变 **tex** 系统内部字符列表的命令。在 **tex** 中字符分为以下几类：

0. 跳脱符号，一般是 \

1. **group** 开始符号, 一般是 {
2. **group** 结束符号, 一般是 }
3. 数学符号, 一般是 \$
4. 表格分隔符, 一般是 &
5. 一行结束符号, 一般是 <return>
6. 参数符号, 一般是 #
7. 上标符号, 一般是 ^
8. 下标符号, 一般是 _
9. 忽略符号, 一般是 <null>
10. **space**, 一般是 <space> 和 <tab>
11. 字母, 一般是 a-z 和 A-Z
12. 其他, 不是其他分类就在此类
13. 激活 (Active) 符号, 比如 ~
14. 注释符号, 一般是 %
15. 无效 (Invalid) 符号, 一般是 <delete>

也就是说 **tex** 引擎看到 \ 符号就会认为接下来将是一个命令, 然后 **catcode** 命令可以改变这个列表。比如上面这段代码中, `\catcode '|=0` 就将 | 符号归为 0 类, 这样 **tex** 看到 | 也会跟看到 \ 一样处理。后面的类似, 于是我们有:

| 类似 \ [类似 {] 类似} {} \ 都是其他

gdef 命令是 **global** 和 **def** 命令的组合缩写, 这里定义了下面要用到的 **@writeent** 命令。就是立即向打开的那个 **ent** 文件写入花括号内的信息。其中的符号变化法则上面谈及了, 于是这里的 **\footnote** 之类的都不会处理, 而是直接写入 **ent** 文件之中。但是因为我们在这里要建立 **label** (等下插入 **ent** 文件之后, 在尾注那里要标记 **label** 和 **ref** 命令。) 因此 **label** 和 **ref** 内的参数必须立即运算出来。在这里有一些尾注的格式美化命令, 比如用 **makebox** 命令调整尾注标记符号格式等, 这里介绍略过。

现在主要看到从下一段代码中定义的 `@include` 命令接受的是 `endnote` 命令传递过来的文本，本来在下一段代码中新建一个 `group` 然后用 `let` 命令将 `endnote` 命令中所有的 `protect` 命令改成了 `string`（字符串），应该是可以了。实际上操作中确实各个命令没有展开直接以文本进入 `ent` 文件了，但是环境命令还是展开了，于是这里加上了 `etoolbox` 宏包提供的 `expandonce` 命令，也就是控制只展开一次，这样环境命令也成功传递过去了。这里的具体实现细节我实际上也没弄太明白。

```

\DeclareRobustCommand*\endnote[1]{%
\stepcounter{@EndnoteCounter}%
\stepcounter{@AllEndnoteCounter}%
\edef\@linkname{enote:\arabic{@AllEndnoteCounter}}%
\edef\@linknameb{enote:\arabic{@AllEndnoteCounter}b}%
\edef\@showlinknameb{\endnotemark}%
\edef\@showlinkname{\endnotemarkback}%
\ignorespaces\flabel{\@linkname}{\@showlinkname}%
\ignorespaces\hspace{-0.5ex}\ref{\@linkname b}%
\if@entopen \else \@openent \fi%
\begingroup%
\let\protect\string%
\gdef\@include{#1}%
\endgroup%
\@writeent%
}

```

显然你看到这里定义的命令 `endnote` 就知道这是这个宏包的主体部分，程序一般遇到的就是两个命令，`endnote` 引入尾注，`showendnotes` 插入尾注。首先这个命令将两个主要计数器加 1，其中 `@EndnoteCounter` 计数器是标记一章尾注内部的计数的，这里加 1 了就开始从 1 开始，然后后面这里的处理技巧是如果你需要插入尾注了，那么就将 `@EndnoteCounter` 这个计数器归为零。这样做和将 `@EndnoteCounter` 和 `chapter` 或者 `part` 绑定起来的好处就是更加自由，你不需要考虑什么，甚至在某一个 `section` 部分内，插入 `showendnotes` 命令就插入了尾注，然后内部计数又重新开始。

接下来定义了四个命令，我们看到这里有一个 `edef` 命令，`edef` 命令和 `def` 命令类似，区别在于定义某个命令的时候立即展开，比如在这里我 `edef` 了一个 `@linkname` 命令，那么后面的 `@AllEndnoteCounter` 当时是什么立即赋值计算出来。因为这里要建立 `label` 的名字比如是独一无二的，然后看到后面 `@showlinkname` 和 `@showlinknameb`，这是在决定文档中尾注标记符号和尾注中标记符号的样式，这里名字弄翻了，因为之前我在编写的时候也不大确切，然后后面就交换了一下。

然后我们注意到这个 `endnote` 命令后面每一行都有一个%，这是不得已而为之。`endnote` 命令在它所在地地方我需要插入 `label` 和 `ref`，然后每一行换行之后都会增加一个空格。只好用百分号消去。

这里插入的 `ignorespaces` 命令为了消去空格可能是多余的，然后使用 `flabel` 命令植入 `label`，前面谈及了第一个参数是真实的 `label`，第二个参数是显示的 `label`。还有我们需要把 `label` 命令和 `ref` 命令的关系弄清楚，`label` 只是在文档中植入一个记号，而 `ref` 命令才会在文档中显示某些字符。我在测试的时候发现尾注标签上标记号有点偏后，然后用 `hspace` 命令稍作调整，注意我们调整距离尽可能使用 `ex` 或者 `em` 这样的相对距离。然后就是使用 `ref` 命令。

接下来一行语句的意思就是 `ent` 那个文件打开了吗？打开了什么也不做，没打开打开这个文件，使用之前定义的 `openent` 命令打开。

然后我们进入一个 `group`，`tex` 命令中 `\begingroup \endgroup` 之间夹著一个内环境。这个内环境的作用就是为了把 `endnote` 命令中的文本内容传递出去，为什么要这么做？我们的思路是将 `endnote` 中的文本写入到某个 `ent` 文件中去，然后继续写，直到要插入尾注了再引入这个 `ent` 文件即可。思路很简单，但是要实现起来并不简单。

这里最大的问题是 `endnote` 命令里面的命令和环境都要不执行，当作文本一行写入进去。因为 `tex` 默认参数都要执行计算出来的，这就是问题的所在。这里面的细节甚是复杂，我也没完全弄明白。我们再看到上面那段代码定义的 `@writeent` 命令。

```
\DeclareRobustCommand*\showendnotes{%
\immediate\closeout\@entout%
\global\@entopenfalse

\input{\@entname.ent}
```

```

\setcounter{@EndnoteCounter}{0}
\stepcounter{@ShowEndnoteCounter}%

}

\endinput

```

这是宏包的最后一段了，非常容易理解了。定义了一个 `showendnotes` 命令，命令首先将那个打开的 `ent` 文件关闭，这样之前写入的内容都写进去了。然后改变那个条件变量表明文件关闭了。然后引入尾注。然后 `@EndnoteCounter` 计数器归零，这个前面谈到过。然后 `ShowEndnoteCounter` 计数器加一。然后结束宏包使用 `endinput` 命令。

endnotes 宏包的使用

在加载好 `endnotes` 宏包之后，用法还是很简单的。

endnote 命令

想要在那里插入尾注部分注释就用 `endnote` 命令即可：

`\endnote{这是一段尾注}`。比如这里⁽¹²⁾。

showendnotes 命令

在你想要显示尾注时，使用 `showendnotes` 命令即可。那么到目前为止，所有尾注内容都会显示出来：

```
\showendnotes
```

一般上面还加上一个标题，表示注释内容就在这里。如果整个文档都不使用 `showendnotes` 命令，那么也不会有什么错误。

文字强调

emph 命令

emph 命令一般是英文换成斜体，中文换成楷体（也就是前面设置的意大利字形的字体）。不过不同字族会有不同的表现。如果在强调环境之内有强调（一般没有这种情况把。）那么文字又会换成常规形态。**emph** 命令是 **L^AT_EX** 自带的最基本的用于文字强调的方法。

重新定义 emph

本文档中的 **emph** 命令被重新定义了：

```
\renewcommand\emshape{\color{red}}
```

比如：我觉得字体设为红色更加起到强调作用

underline 命令

这个也是 **L^AT_EX** 自带的命令，就是加上下划线，不过在中文中并不能正确换行。所以往下看。

ulem 宏包

下面的内容来自 **ulem** 宏包。^①

^① 现在 **ulem** 中文可以正确换行了。

ulem 宏包源码

ulem 宏包在 ubuntu 系统中源码的位置是：

```
/usr/share/texlive/texmf-dist/tex/generic/ulem
```

ulem 宏包提供如下命令：uline, uuline, uwave, sout, xout, dashline, dotuline。主要用于文字的强调，其中 uline 是下划线，dotuline 是加点强调，uwave 是波浪线。其他命令请参看 ulem 宏包文档。

这是一段很长的测试文字，主要用于说明中文情况下加入下划线并且能够正确的换行。用的是 uline 命令。

注意如果单纯加载 ulem 宏包，原有的 emph 命令也会成为类似 uline 命令的效果，也就是加下划线。可以后面跟上命令 \normalem，也就是 emph 命令还是原来的处理效果。

reduline 命令

这个命令是 ulem 官方文档提供的。其中除了 reduline 命令名字可以自己 diy 之外，你能改动的就是第二行了。rule 部分简单说明下，第一个可选项是竖向位移，第一个参量是线条横向宽度，我为了好理解就设置成了 1em，似乎这个值设置成其他的值影响也不大。第二个参量是线条竖向高度。

```
\newcommand\reduline{\bgroup\markoverwith
  {\textcolor{red}{\rule[-0.5ex]{1em}{0.4pt}}}\ULon}
```

为了好玩类似的我还用 tikz 画了一个小三角形和一个小圆，如果各位看官有更喜欢的标记用于文字强调，类似的用 tikz 画就是了，可见 ulem 这个扩展命令实在是棒极了。

```
\newcommand\utriangle{\bgroup\markoverwith
  {\raisebox{-0.4em}{\makebox[1em][c]{\tikz[blue]

```

```
{\draw(0,0) -- (0.1,0) -- (0.05,0.05) --cycle;}}}}
\ULon}
```

```
\newcommand\ucircle{\bgroup\markoverwith
{\raisebox{-0.6em}{\makebox[1em][c]{\tikz[red]
{\draw(0,0) circle (0.05) ;}}}}
\ULon}
```

插入列表

`itemize` 和 `enumerate` 环境其实也支持 `item` 后面跟上可选项的形式，只是从格式上他们常常出界，不建议使用。

基本使用

`itemize` 环境

```
\begin{itemize}
\item 这是一个列表
\item 这又是一个列表
\end{itemize}
```

- 这是一个列表
- 这又是一个列表

`enumerate` 环境

```
\begin{enumerate}
\item 这是一个列表
\item 这又是一个列表
\end{enumerate}
```

1. 这是一个列表

2. 这又是一个列表

description 环境

```
\begin{description}
\item[鸭子] 是一种动物
\item[苹果] 是红色的
\end{description}
```

鸭子 是一种动物

苹果 是红色的

enumerate 环境标签的修改

enumerate 环境中各个 item 标签自动生成是依赖各个对应的计数器，然后通过 labelenumi, labelenumii, labelenumiii 和 labelenumiv 这几个命令控制的，也就是重新定义这些命令，标签样式就被修改了。这几个中最常用的是第一级标签 labelenumi，下面就这个命令给出例子。

首先我希望这个 enumerater 环境的 item 计数从 0 开始，然后我希望它的格式是 <0>。

<0> 这是一个列表

<1> 这又是一个列表

上面形式的实现就是在这个 enumerate 环境中加入了这样两行代码：

```
\setcounter{enumi}{-1}
\renewcommand{\labelenumi}{<\arabic{enumi}>}
```

估计计数器使用的时候都会先加一，所以为了初值为 0，只好先设置为 -1 了。

itemize 环境标签的修改

`itemize` 环境里面当然没有计数器，不过它类似的也有：

`labelitemi`, `labelitemii` 等命令，通过重定义这些命令就可以影响第一级第二级等标签样式。

插入图片

要插入一个图片， \LaTeX 文档开头那里要加载库文件：

```
\usepackage{graphicx}
```

然后在你想要插入图片的地方输入如下命令：

```
\includegraphics [scale=1]{图像名字}
```

插入图片有很多参数可以设置，这里就最有用的宽度和高度设置说明一下：

具体就是 **height** 表示高度，**width** 表示宽度。然后等于多少 **in**，英寸。参数放在可选参数那里。不过我觉得下面这个参数设置挺实用的，如下：

```
\includegraphics[width=\textwidth, keepaspectratio]{test.png}
```

其中 `\textwidth` 表示让图片和文字一般宽，然后 `keepaspectratio` 参数意思是缩放的时候保持宽高比不变。

图片标签的修改

本文的图片环境标题通过输入 `caption{text}` 即实现如下形式的图片标题：图 1-1: `text`。其中图这个字是通过重定义 `figurename` 命令来完成的。

```
\renewcommand\figurename{图}
```

而后面本来默认的是 1.1 这样的形式，但是常见的都是 1-1 这样的形式，通过重定义 `thefigure` 命令即可达到目的。

```
\renewcommand{\thefigure}{\arabic{chapter}-\arabic{figure}}
```

取消图片标签

有时候通过 `caption` 插入图片标题，前面的标签你不想要。这个时候就算将 `the-figure` 命令设置为控制还是会有“图:”这两个文字，如果这两个你都不想要。推荐用 `caption` 宏包的 `caption*` 命令，那个命令完全没有标签，写成什么样子显示就是什么样子。

设置图片寻找文件夹

如下的命令设置，这样 `TeX` 会自动在这些文件夹寻找文件，也就是你在 `include` 图片时直接写上图片文件名即可。这个语句是支持多个文件夹设置的。`\graphicspath{{figures/}}`

图片格式的讨论

支持的图片文件格式

使用 `xelatex` 和 `graphicx` 宏包，目前我测试支持的图片格式有：`eps`，`png`，`jpg`，`pdf`。

搜索图片后缀名控制

经测试上面的 `eps`，`png`，`jpg`，`pdf` 格式的图片后缀名默认都支持，也就是只要写上前面的文件名即可。

eps 图片格式

如果你使用矢量绘图软件画的图片，推荐用 `eps` 或者 `pdf` 格式。下面给出一个 `eps` 图片，是 `pyx` 宏包画出来的，然后用 `inkscape` 软件用 `300dpi` 另存为 `pdf` 格式，并用 `inkscape` 软件用 `300dpi` 另存为 `png` 格式。

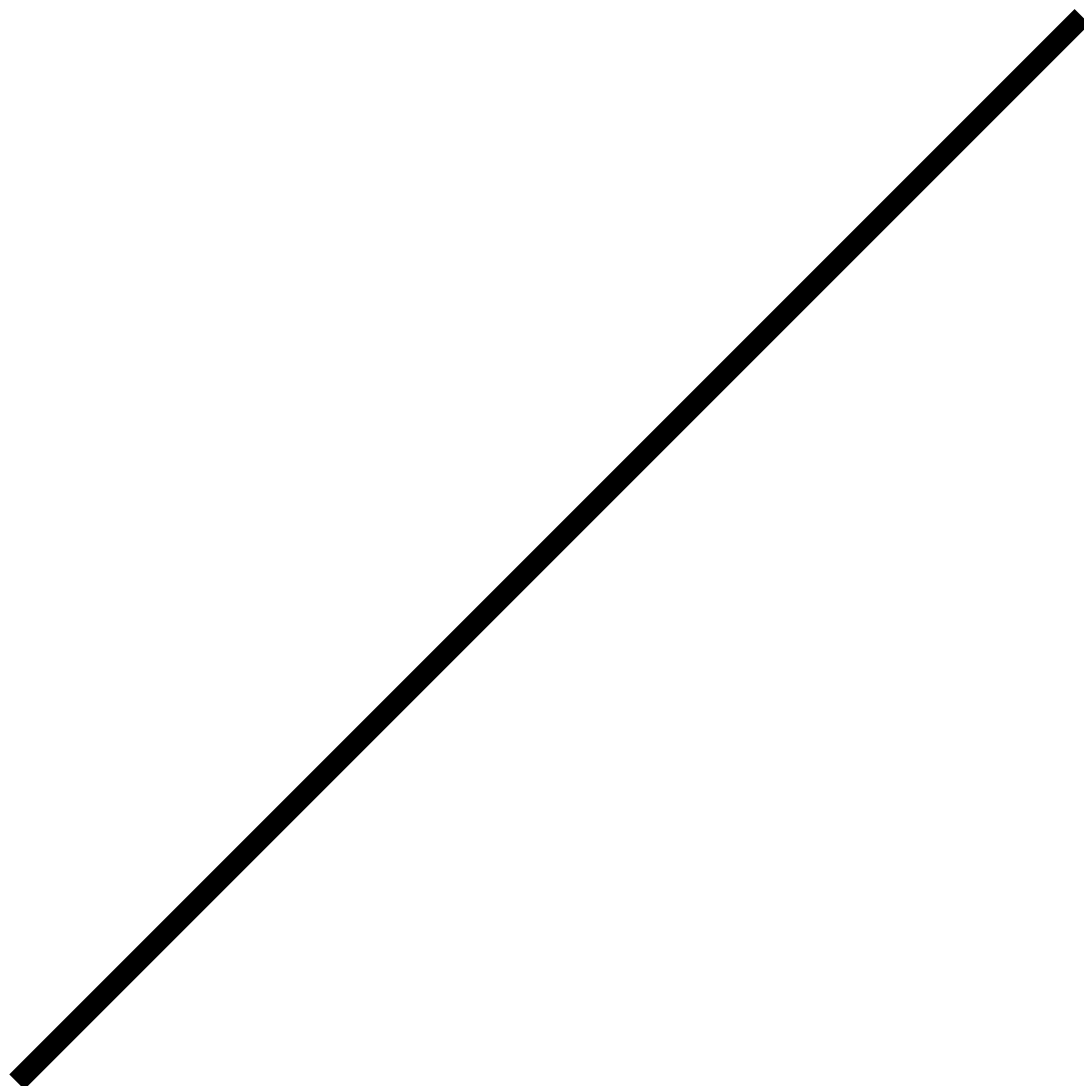


图 20-1: line.eps

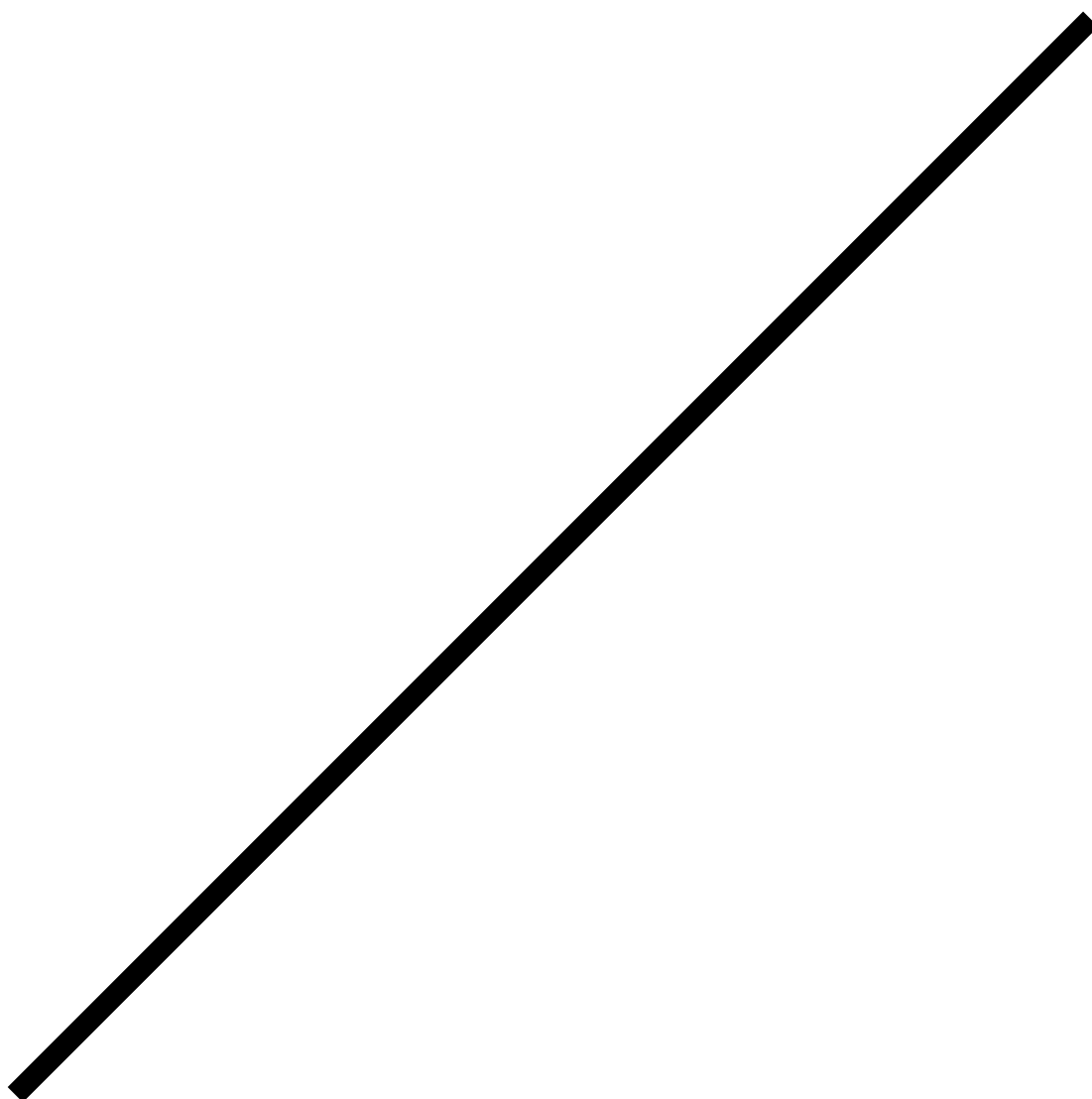


图 20-2: line.pdf

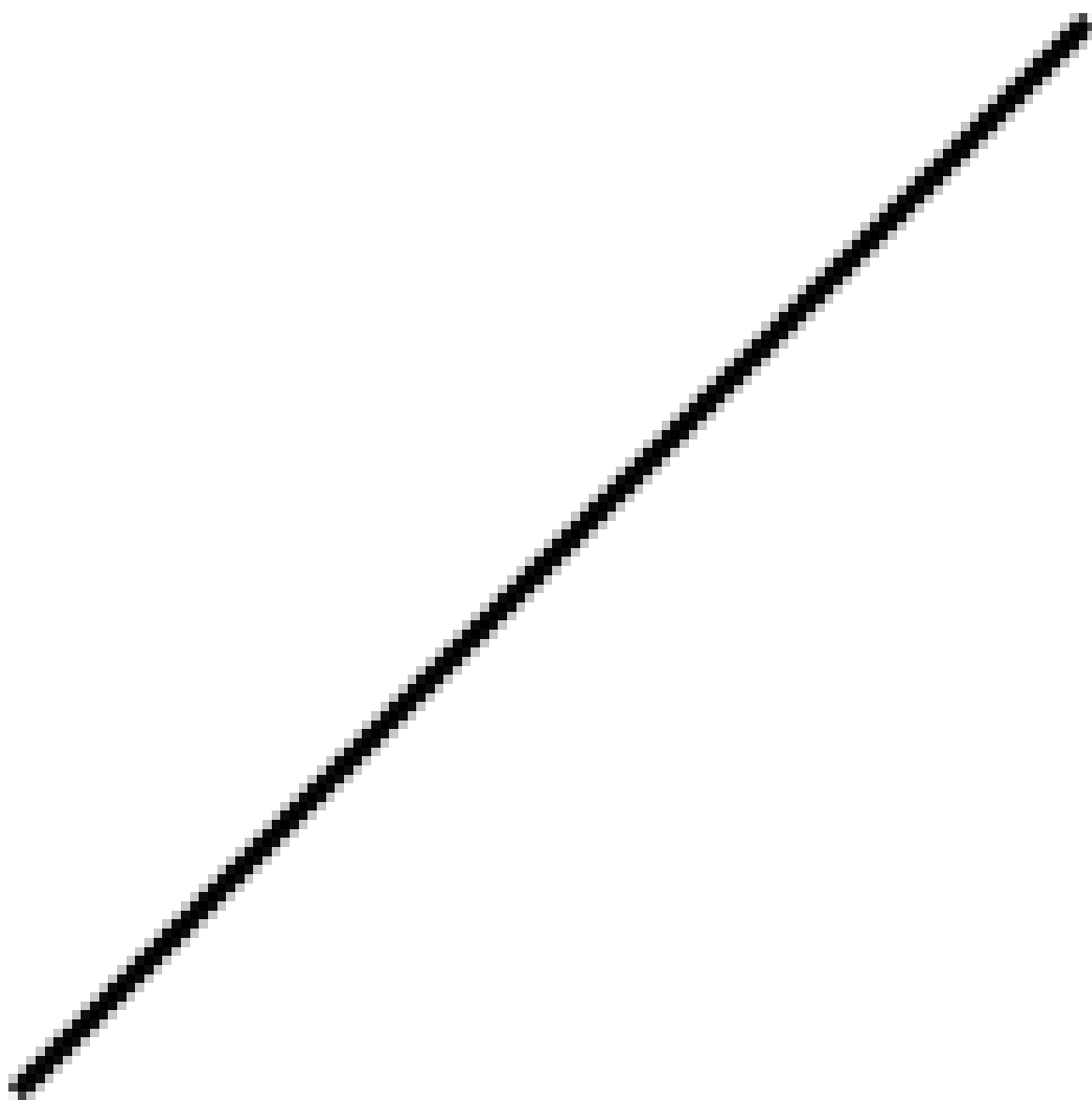


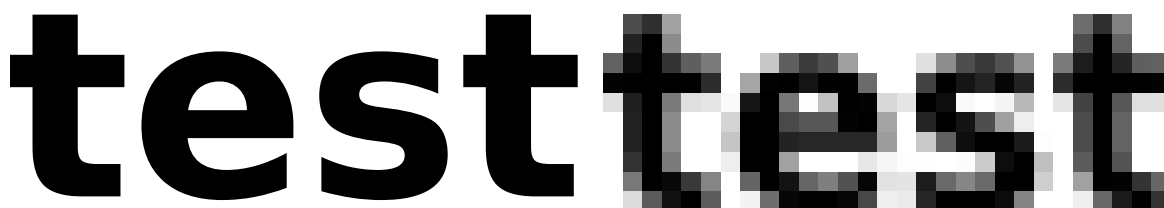
图 20-3: line.png

可以看出来如果选择 **png** 格式图片将会变得很模糊，而 **eps** 格式和 **pdf** 格式图片清晰度都差不多的，推荐用 **pdf** 格式。

svg 图片格式

有时在网上另存为出来的图片格式是 **svg** 图片格式，是一种什么可缩放矢量图形。下面是一个小测试，原 **svg** 图片是一个很小的文字，然后放大。其中 **pdf** 和 **png** 都是

inkscape 软件用 300dpi 导出的，可以看出 pdf 格式明显更胜一筹。



svg 转化成 pdf 格式 in300dpi

svg 转化成 png 格式 in300dpi

jpg 图片格式

他们说相片最好用 jpg 格式，一般相片什么的都较大，所以也不存在缩放问题。

图片格式选择总结

结论是如果图片本来很大不存在缩放问题，那么能够使用的就直接使用，其他不支持的图片格式考虑生成 png 图片，记得 dpi 至少设到 300 以上，一般到 1200dpi 是绝对够用了。如果源图片很小，需要缩放，那么 eps 和 pdf 文件格式不用做进一步处理，其他文件格式优先考虑 pdf 文件格式，导出的时候我们看到 300dpi 已经绰绰有余了。

导入 pdf 页面

这里要讲的是使用 pdfpages 宏包来实现多个 pdf 页面的导入。

```
\usepackage[options]{pdfpages}
```

默认是插入所有 pdf 页面，还有一些选项请参看文档。最常用的选项就是：

```
pages={1,{},8-10}
```

上面的意思是插入第一张页面，然后插入一个空页面，然后插入第八张到第十张。

本文插入图片的一些 DIY

```

\newenvironment{fig}[2][1]
    {\begin{figure}[H]
    \centering
    \includegraphics[scale=#1 , keepaspectratio]{#2}}
    {\end{figure}}
\newenvironment{linefig}[2][1]
    {\begin{figure}[H]
    \centering
    \includegraphics[width=#1\linewidth ,
    totalheight=0.95\textheight , keepaspectratio]{#2}}
    {\end{figure}}

```

本文新建了两个新的图片环境命令，第一个 **fig** 环境默认使用 **scale=1**，也就是那些较小的图片的居中对齐效果较好，如果图片较大可以考虑 **linefig** 环境，这个环境会让图片宽度对齐到 **linewidth**（之所以选择 **linewidth** 是因为在多栏环境下也会自动适应到那个栏的宽度。）。

```

\begin{linefig}[0.8]{svg 图片测试.png}
\label{fig:svg 图片测试.png}
\caption*{\footnotesize svg 转化成 png 格式 in300dpi}
\end{linefig}

```

这样插入图片代码就很简洁了。记住可选参数在参数中是排前面的，这里是缩放比的意思，方便调整图片大小。

插入表格

基本情况的讨论

一般情况下一些小的表格就用 **tabular** 环境处理即可。下面看这个例子：

```
\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
l & l 表示该列格子左对齐 \\
\hline
c & c 表示该列格子居中 \\
\hline
r & r 表示该列格子右对齐 \\
\hline
\end{tabular}
\caption{tabular 参数}
\label{tab:tabular 参数}
\end{table}
```

例子显示如下：

l	l 表示该列格子内容左对齐
c	c 表示该列格子内容居中
r	r 表示该列格子内容右对齐

表 21-1: tabular 参数

在 `table` 环境那里我加了一个可选参数 `h`，意思是在这里就在这里。这个表格还有图片环境都是什么浮动体。我们看到他们可以加上 `caption` 命令从而有一个标题，然后 `table` 和 `figure` 后面有个可选参数来控制这个浮动体的位置。默认是 `tbp`。不过我喜欢用 `h`，也就是在这里如果可能。有的时候 `h` 的表现效果可能不太让你满意。那么你可以尝试 `float` 宏包，它提供了 `H` 参数，会更加强制地控制浮动体，`H` 的意思是一定要在这里。

这段代码中 `centering` 命令是让表格居中。`caption` 命令是加上标题，`label` 命令是方便引用。

最重要的就是 `tabular` 环境，前面 `|` 符号表示画一个竖线，也就是每一列刚开始画一条竖线，你也可以不画表示每一列开始不画竖线。然后是字母 `c`，表示每一列第一个格子居中对齐，类似的还有字母 `l(left)` 和 `r(right)`。还有一种格式 `pwidth`，表示该格子具有 `width` 的宽度，然后里面的文字自动断行。

`hline` 命令表示画一条横线。`&` 这个特殊符号表示进入下一列。`\\` 表示进入下一行，后面跟上可选项距离，那么下一行的高度将拉高。

booktabs 宏包

如果你只是想要一个简洁明了的表格，目前大家公认的好的表格标准就是三线表式，你也可以称之为 `booktabs` 风格吧。简单来说有以下规则：① 不要垂直线；② 不要双横线；③ 每一行的各个格子都有足够的空间；④ 一律左对齐；⑤ 三线，`toprule`，`midrule`，`bottomrule`。(13)

在这里我们还有个捷径，不一定要手工将表格代码全部敲出来。首先我们找一个表格软件，`libreoffice` 的或者 `gnnumeric` 都行。然后将数据输入进去保存好。然后我们将数据选择复制，打开网站：

<http://www.tablesgenerator.com/>

在那个网站的表格的开头哪里按下 `Ctrl+v`。这个网站还有一些设置可以调整，稍微摸索下就知道了。唯一要说的是那个 `activate/deactivate custom grid edit` 选项，可以选择绘制某几根线框显示。在这里不做调整，直接选择最右边的 `booktabs table style`。然后把多余的行或者列删除掉，就可以点击下面的 `generate` 了。生成的代码如下：

```
\begin{table}[h]
\begin{tabular}{@{}ll@{}}
```

```

\toprule
参数      & 描述      \\ \midrule
l          & 左对齐      \\
c          & 居中        \\
r          & 右对齐      \\
p{width}  & 一定宽度自动换行 \\ \bottomrule
\end{tabular}
\end{table}
    
```

表格的显示效果初步如下: (14)

参数	描述
l	左对齐
c	居中
r	右对齐
pwidth	一定宽度自动换行

接下来就是微调整了, 比如说上面的花括号没有正常显示, 还有想要居中, 开头加上 **centering** 命令, 还有 **caption** 标题命令, 还有 **label**。修改之后结果如下:

参数	描述
l	左对齐
c	居中
r	右对齐
p{width}	一定宽度自动换行

表 21-2: tabular 参数 -2

一个三线表模板

这是一个简单的三线表模板, 你可以在 **texmaker** 或者其他编辑器里面设置好快速插入代码片段的功能。然后快速输入一下代码块, 然后填上数据, 这样就很快捷了。

```

\begin{table}[H]
\centering
\label{tab:}
\caption{}
\medskip
\begin{tabular}{@{}ll@{}}
\toprule
选项 & 说明 \\ \midrule
H & 页眉 (head) \\
O & 奇数页 (odd) \\
\\ \bottomrule
\end{tabular}
\end{table}

```

booktabs 宏包详解

上面已经算是一个 **booktabs** 风格的表格了，为了进一步深度定制这里对 **booktabs** 宏包做一些说明。

线条粗细

前面我们看到了 **booktabs** 宏包新加了三个命令 **toprule**, **midrule** 和 **bottomrule**。这三个命令后面都可以跟个可选项调整线条粗细。后面跟的参数 **1pt** 就表示线条粗 **1pt**，没有加法的意思。上面的例子中 **toprule** 和 **bottomrule** 设置为 **1.2pt**，比原来的稍微粗了一点。嫌麻烦不改动也可以，觉得原来的也还好。

cmidline 命令

cmidline 类似于原来的 **cline** 命令，简单来说就是你希望第几个到第几个格子画一条线。这个命令后面一样可以跟个描述粗细的可选项。请看下面的例子：

```
\begin{table}[H]
\centering
\begin{tabular}{@{}lll@{}}
\toprule
slices & \multicolumn{2}{l}{abs.error(slices)}
\\ \cmidrule(l){2-3}
& avg. & max \\
<5000 & 116 & 625 \\
5000-10000 & 209 & 1807 \\
10000-15000 & 297 & 2133 \\
>15000 & 317 & 1609 \\
\bottomrule
\end{tabular}
\end{table}
```

显示效果如下：

slices	abs.error(slices)	
	avg.	max
<5000	116	625
5000-10000	209	1807
10000-15000	297	2133
>15000	317	1609

表 21-3: cmidrule 例子

拉宽一行行的距离

```
\renewcommand{\arraystretch}{1.3}
```

通过上面这个命令，就在导言区整体设置就行了。文章的表格一行行距离稍微拉宽了一点，看上去更加美观了些。关于这个命令和段落行距之间的关系之前有所谈及，这里不赘述了。请看[9.3](#)

将上面这个命令放入某一个表格环境下就是对单独某个表格的修改。

带颜色的表格⁽¹⁵⁾

`xcolor` 宏包提供了一种颜色交替的表格模式，还挺好看的。好吧，我对颜色搭配不太擅长。首先需要在加载 `xcolor` 宏包时填上 `table` 可选项，即`\usepackage[table]{xcolor}`。

实现本表步骤	备注
<code>table</code> 环境下加上 <code>rowcolors</code> 命令	让整个表格交替显色
<code>rowcolors</code> 第一个选项	决定颜色从那一行开始显示
<code>rowcolors</code> 第二个选项	奇数列的颜色
第三个选项	偶数列的颜色
本表具体代码	<code>\rowcolors{2}{}{lightgray!50}</code>
第一行用 <code>rowcolor</code> 命令控制	在表头行上面

表 21-4: 带颜色的表格

这样形式的表格建议不用三线表式了，反倒是纯颜色样式不带线条更美观。

插入代码

小代码

有的时候一行之内的小代码就不需要大动干戈用 `verbatim` 环境，用 `\verb+` 这里放着小代码 `+`，这里的 `+` 号可以换成其他任何的符号表示小代码开始和结束，除了 `*` 和空格不行。带星号的 `verb` 有其他用途，在里面空格以符号显示出来了，比如：

```
_this_is_a_test_.
```

这个可能对某些强调缩进格式的程序语言有用。

在 `Verbatim` 环境或者 `verb` 命令之内，字体是 `ttfamily`。

某一行小代码有时候需要缩小字体从而防止字出界，这里似乎没有办法 `DIY` 出一个新的命令，只能在 `verb` 命令之外套个改变字体的环境。

```
{\footnotesize \verb+this is a test line in verb command+}
```

```
this is a test line in verb command
```

稍微大点的程序代码

在环境 `verbatim` 之间的任何文本是什么就是什么，不执行任何 `LATEX` 命令，包括所有的空白和断行，如下：

```
(defmacro with-gensyms (syms &rest body)
  '(let ,(mapcar #'(lambda (s)
    '(',s (gensym)))
    syms)
    ,@body))
```

同样这里也外面包围了一个 `footnotesize` 环境改变字体大小。

fancyvrb 宏包

`fancyverb` 宏包提供了一个 `Verbatim` 环境，比如下面的配置代码的说明如下。更多说明请参见 `fancyvrb` 宏包文档。

```
\fvset{numbers=left,frame=lines,tabsize=4 ,baselinestretch=2,
      xleftmargin=6pt, fontsize=\footnotesize , numbersep=2pt}
```

表 22-1: Verbatim 环境一些设置

配置	说明
<code>numbers=left</code>	左边显示数字
<code>frame=lines</code>	框框是两条线
<code>tabsize=4</code>	<code>tab</code> 符号是四个空格
<code>baselinestretch=2</code>	行间距拉伸
<code>xleftmargin=6pt</code>	左边间距 6pt
<code>fontsize=\footnotesize</code>	字体大小设置
<code>numbersep=2pt</code>	数字和框框间距

minted 宏包

`minted` 宏包也是利用 `pygments` 来自动进行语法染色，它还提供了一个 `listings` 环境，不过并不能正确分页，所以让我们忘了那个 `listings` 环境（或者现在问题解决了？），然后就其他部分，自动语法染色，`fancyvrbDIY` 选项的继承，还新增了背景颜色的设置，这些都是极好极好的。。

mint 命令

minted 宏包提供了 mint 命令来进行 inline 的 code 输入，这正好弥补了原 verb 命令功能过于简陋的问题。

```
\mint{tex}+\test code+
```

初步的测试情况如下，

```
\test code
```

，注意：mint 命令里面不能再写上 mint 命令。

minted 环境

```
\begin{minted}{c}
#include <stdio.h>
```

```
int main(void)
{
    printf("hello, world\n");
    return 0;
}
\end{minted}
```

```
#include <stdio.h>
```

```
int main(void)
{
    printf("hello, world\n");
```



```

return 0;
}

```

然后 `minted` 宏包加上背景颜色会出现 `hbox` 的 `badness` 问题，推荐使用 `tcolorbox` 来加上背景颜色。

tcolorbox 和 minted

我不愿加上数字之类的主要是方便人们在 pdf 上复制，虽然空格并不能保留，不过还是已经很省心了。这里用 `tcolorbox` 宏包的主要任务就是加上背景颜色。

```

%minted
\RequirePackage{minted}%
\RequirePackage[minted]{tcolorbox}%
\tcbuselibrary{breakable}
\newtcblisting{tcbminted}[2][]{listing engine=minted,
minted style=colorful,minted language=#2,
minted options={fontsize=\footnotesize},
colback=blue!5!white,colframe=blue!75!black,
listing only,#1,breakable=true}

\begin{tcbminted}{python}
class Hero():
    def addlevel(self):
        self.level=self.level+1
        self.hp=self.hp+self.addhp

class Garen(Hero):
    def __init__(self):
        self.level=1
        self.hp=455
        self.addhp=96

```

```

self.skill=['不屈','致命打击','勇气','审判','德玛西亚正义']

garen001=Garen()
for i in range(6):
    print(' 级别:',garen001.level,' 生命值: ',garen001.hp)
    garen001.addlevel()
print(' 盖伦的技能有: ',"".join([x + ' ' for x in garen001.skill]))
\end{tcbminted}

```

```

class Hero():
    def addlevel(self):
        self.level=self.level+1
        self.hp=self.hp+self.addhp

class Garen(Hero):
    def __init__(self):
        self.level=1
        self.hp=455
        self.addhp=96
        self.skill=['不屈','致命打击','勇气','审判','德玛西亚正义']

garen001=Garen()
for i in range(6):
    print(' 级别:',garen001.level,' 生命值: ',garen001.hp)
    garen001.addlevel()
print(' 盖伦的技能有: ',"".join([x + ' ' for x in garen001.skill]))

```

特殊文字环境

语录和引用环境

L^AT_EX 原生的有 `quote` 为语录环境，`quotation` 用于超过几段的引用环境，`verse` 用于诗歌环境。

以下讨论参考了[这个网站](#)

`quote` 和 `quotation` 在 `article.cls` 中的原初定义是这样的：

```
\newenvironment{quote}
    {\list{}{\rightmargin\leftmargin}%
     \item\relax}
    {\endlist}
\newenvironment{quotation}
    {\list{}{\listparindent 1.5em%
     \itemindent \listparindent
     \rightmargin \leftmargin
     \parsep \z@ \@plus\p@}%
     \item\relax}
    {\endlist}
```

我改成了如下：

```
%===== 重新定义 quote=====
\renewenvironment{quote}[1][anonymous]{
```

```

\newcommand{\quoteauthor}[1][anonymous]{#1}
\list{}{\rightmargin\leftmargin % 右间距等于左间距
\itemindent 2em
}\item\relax
\ttfamily}
{\
\makebox[\linewidth][r]{\sffamily —\quoteauthor}
\endlist}

% 重新定义 quotation
\renewenvironment{quotation}
{\list{}{\rightmargin\leftmargin % 右间距等于左间距
\itemindent 2em%item 的缩进也就是第一段的缩进
\listparindent \itemindent % 第二段的缩进
}%
\item\relax
\ttfamily}
{\endlist}

```

`quote` 和 `quotation` 都是通过 `list` 环境来实现的，具体不太清楚。这里就我做的修改说明如下：

- `\rightmargin\leftmargin` 这是让右边空白距离等于左边空白距离，左边空白距离默认等于`\parindent`，这个有用我保留了。
- `\itemindent 2em`，这个 `itemindent` 控制的是第一段的段首缩进量，这里设置为 `2em` 即两个字符的意思。
- 代码第七行是设置 `quote` 环境下字体为 `ttfamily`。
- `quote` 环境后面的代码，在结束环境之前，先换行，然后输出了一个盒子，右对齐，`ssfamily`，写着这个格言的作者，因为格言一般都要写上作者，如果不知道那么也写上 `anonymous`，无名氏。这个作者的名字是 `quote` 环境接受一个可选项而来的额。

- `\listparindent` `\itemindent`, `quotation` 多了一个这个设置, 其中 `listparindent` 是控制 `quotation` 里面除了第一段之外其他段落的缩进, 这里就简单设置为跟第一段一样。
- `quotation` 环境的其他修改, `parsep` 是控制段与段之间距离的, 这里取消了, 因为默认的就很好了。

语录环境演示

```
\begin{quote}[无名氏]
some text
\end{quote}
```

“When the winds of change blow, some people build walls
and others build windmills.”

—无名氏

引用环境演示

```
\begin{quotation}
some text

some text
\end{quotation}
```

说 Unix 必将衰败, 或者被其他操作系统挤出市场。可是在今天, 化身 Linux、BSD、Solaris、MacOS X 以及好几种其它变种的 Unix, 却显得前所未有的强大。

Robert Metcalf[以太网络的发明者] 曾说过: 如果将来有什么技术来取代以太网, 那么这个取代物的名字还会叫“以太网”。因此以太网是永远不会消亡的 Unix 也多次经历了类似的转变。

诗词歌赋环境

诗歌环境我还懒得 DIY，默认的效果还可以。值得一提的是在诗歌环境里面不会自动换行，需要手动输入\\。

月下獨酌
作者: 李白
花間一壺酒
獨酌無相親
舉杯邀明月
對影成三人
月既不解飲
影徒隨我身
暫伴月將影
行樂須及春
我歌月徘徊
我舞影零亂
醒時同交歡
醉後各分散
永結無情遊
相期邈雲漢

notecard 环境

这个环境编写主要参考[这个网站](#)。

有一类信息，这类信息和正文关系不太大，但是又很重要适合放在正文中而不是尾注或者脚注中。这类信息需要放在文档右边好不引起人们的注意或者好引起人们的注意。文档中的异常元素通常都有这个效果，在于读者的选择。

具体实现代码如下：

```

\newsavebox{\tempbox}
\newenvironment{notecard}[2][white]
{
  \noindent\ignorespaces%
  \setlength{\fboxsep}{10pt}
  \newcommand{\tempcolor}{#1}
  \begin{lrbox}{\tempbox}%
  \begin{minipage}{#2}
    \setlength{\parindent}{0pt}
    \setlength{\parskip}{1.618ex} % 段落间距
  }
  {
    \ignorespacesafterend%
    \end{minipage}%
    \end{lrbox}%
    \colorbox{\tempcolor}{\usebox{\tempbox}}}

```

这个 notecard 代码环境有点小复杂，主要涉及到 \LaTeX 盒子相关高级知识，这里先跳过去，后面会提及再慢慢讲。

notecard 环境演示

notecard 环境，一种卡片式记忆环境，长高推荐按照黄金比例来。推荐的颜色有：
 black olive blue orange brown pink cyan purple darkgray red gray teal green
 violet lightgray white lime magenta yellow 建议都调淡 30。不过推荐一个文档内部
 一种类型只使用一种颜色，颜色太杂也是不好的。

```

\begin{flushright}
\begin{notecard}{14em}

```

不懂 Unix 的人注定最终还要重复发明一个蹩脚的 Unix。

Usenet 签名, 1987 年 11 月

```

{\hfill —Henry Spencer}

```

```
\end{notecard}
\end{flushright}
```

不懂 Unix 的人注定最终还要重复发明一个蹩脚的 Unix。

Usenet 签名, 1987 年 11 月

—Henry Spencer

```
\begin{flushright}
\begin{notecard}[blue!30]{14em}
不懂 Unix 的人注定最终还要重复发明一个蹩脚的 Unix。
```

Usenet 签名, 1987 年 11 月

```
{\hfill —Henry Spencer}
\end{notecard}
\end{flushright}
```

不懂 Unix 的人注定最终还要重复发明一个蹩脚的 Unix。

Usenet 签名, 1987 年 11 月

—Henry Spencer

插入摘要

`abstract` 环境就是摘要环境，不过 `book` 类不能使用摘要环境，只有 `article` 和 `report` 才有。命令格式如下：

```
\begin{abstract}
```

这是一段摘要文字。

```
\end{abstract}
```

参考文献

这里只讲最基本的参考文献环境知识，其他高级知识如 **bibtex** 知识这里先不谈论。

首先设置 **thebibliography** 环境，然后用 **bibitem** 命令插入文献，这里 99 的意思是编号宽度不超过 99^①。在你想要引用的地方用 **cite** 命令^②。具体格式如下所示：

```
\begin{thebibliography}{99}
\bibitem{latex123} 《大家来学 \LaTeX 》，原版作者：李果正。
\end{thebibliography}
```

bibitem 命令的必填参数是参考文献的标签名字，但不是实际显示的名字，默认是显示的编号。如果你需要控制实际显示的文本名字是什么那么可以用可选项来控制之。如 `\bibitem[what]{latex123}`。这样处理之后文章中用 **cite** 命令简单引用这篇文献，那么显示的文本也将变成 “**what**”，而不是编号。

① 实际显示的编号。

② 之前谈及，比如这里就是 `\cite{latex123}`

数学相关

数学环境

`$ $` 包围起来的是段落中的数学公式；**`displaymath`** 环境显示单独占一行的数学公式，不过没有编号；**`equation`** 环境有编号，**`equation*`** 环境没有编号。

下面依次演示 **`displaymath`**，**`equation`** 和 **`equation*`** 环境的显示效果。其中 **`displaymath`** 环境还可以用 `\[` `\]` 简单表示之，然后 **`equation*`** 环境是 **`amsmath`** 提供的，如果你已经加载了 **`amsmath`** 宏包，那么推荐就使用 **`equation`** 和 **`equation*`** 这两个环境。

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b \tag{26.1}$$

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$$

什么时候用数学环境

在排版物理书籍的时候常遇到一个问题，那就是比如 **10cm** 或者 **10 米** 等等这样的数字加单位组合我们该如何排，特殊的符号或者一串公式甚至一大串公式当然用数学环境排这是没有疑问的，对于这样短小的数字和单位一般通用的做法是如何的呢？

一般的数字当然就是直接写，然后一般的单位也是直接写，比如 **10cm** 啊 **10 米** 啊就

是标准的做法。然后有些特殊的单位符号以及上标下标问题等如果用数学环境写，就会破坏文档这部分内容的协调性，**siunitx**宏包就是来解决这个问题的。

比如说`\num{3d10}`的显示效果就是 3×10^{10} ，比如说`100\si{厘米^3}`的显示效果就是 100厘米³。

值得一提的是原来 **xeCJK** 宏包会让数字和中文（单位）之间自动空一格，这很好，而使用了 **num** 命令或 **si** 命令之后这个自动空格机制被干扰了，你可以选择重新包装这个命令或者手动空一格吧。

siunitx 宏包还有一个好处就是对于某些特殊的符号的输入也提供了支持，比如温度的度°C等等，此外还可以定制一些额外的符号：

```
\sisetup{
math-micro = \text{\mu},
text-micro = \mu
}
```

这样你就可以用`\si{\micro m}`来显示 μm 了。当然了，这个宏包说白了就是对数学环境下某一些符号命令的 **text** 命令的一种包装吧，如果你能够直接输入那个符号就直接输入吧，这就是推荐的做法。

简单说来就是短小的数字或单位等还是不要麻烦数学环境了。更多的内容请参看 **siunitx** 官方文档，还可以参看[这个网站](#)。

数学的 label 和 ref

一般格式是`\label{eq:I:4:1}`

然后引用是`(\ref{eq:I:4:1})`，外面加个括号好看点。

加入文本

使用 **text** 命令可以在数学环境中正常输入一些文本（包括中文，如果你已经处理好中文显示问题了。）

常用数学符号

- 大于等于 \geq `\geqslant`
- 并集 \cup `\cup`
- 属于 \in `\in`
- x' `x'`
- 约等于 \approx `\approx`
- 中间一个小点表示字母相乘 \cdot `\cdot`
- 中间三个小点 \cdots `\cdots`
- 趋近于 \rightarrow `\rightarrow`
- 无穷大 ∞ `\infty`
- 角度 \angle `\angle`
- 正比 \propto `\propto`
- 积分 $\int_0^1 \sin x$ `\int_{0}^{1}\sin x`
- 求和 $\sum_{i=1}^n$ `\sum_{i=1}^n`
- 乘积 \prod `\prod`

空心粗体实数集合

\mathbb{R} `\mathbb{R}`

上标和下标

\wedge $_$ ，这两个符号跟着上标和下标。多个符号的用`{}`括起来。

平方根符号

`\sqrt`, 如果是`\sqrt[n]` 表示 n 次平方根。

$$\sqrt[n]{x^2 + y^2}$$

表达式上划线或者下划线

就是 **overline** 命令和 **underline** 命令。

$$\overline{x + y} \quad \underline{\frac{2}{3}}$$

表达式上或下有个大括号

使用 **overbrace** 和 **underbrace** 命令。

$$\underbrace{a + b + \cdots + z}_{26}$$

向量

vec 命令, **overrightarrow** 和 **overleftarrow** 命令。

vec 命令只适用一个字母的小箭头。

$$\vec{A} \quad \overrightarrow{BC}$$

空白距离

数学环境里面的所有空白距离和换行都被忽略了。需要用命令手动生成。

quad **qqquad**

特殊的函数

```

\begin{gather*}
\arccos \cos \\
\csc \exp \\
\ker \\
\limsup \min \\
\arcsin \cosh \deg \gcd \lg \ln \Pr \\
\arctan \cot \det \hom \lim \log \sec \\
\arg \coth \dim \inf \liminf \max \sin \\
\sinh \sup \tan \tanh
\end{gather*}

```

$$\begin{array}{c}
 \arccos \cos \\
 \csc \exp \\
 \ker \\
 \limsup \min \\
 \arcsin \cosh \deg \gcd \lg \ln \Pr \\
 \arctan \cot \det \hom \lim \log \sec \\
 \arg \coth \dim \inf \liminf \max \sin \\
 \sinh \sup \tan \tanh
 \end{array}$$

多行数学环境

gather 或者 **gather***, **align** 或者 **align***。这样就可以直接用`\\`换行了。带星号表示没有编号, 不确定这两个环境命令是不是都是 `amsmath` 宏包提供的, 不管怎么说加上 `amsmath` 宏包喽。

分数

`\frac{1}{2}`, 或者 `1/2`。

$$\frac{1}{2} \quad 1/2$$

二项系数

注释

- (1) 主要是针对有旁注的情况，在设置为 `true` 之后，那么旁注宽度你定义是多少就是多少，左边距右边距是多少就是多少，然后剩下的正文宽度是一个从量（即自动确定的量）了。
- (2) 这里的概念理清一下，`headheight` 是页眉高度，保证页眉内容能够装的下即可，而 `headsep` 是页眉（一般页眉靠下）和正文间的空白距离，也就是你看到的页眉上面的那段空白有 `top` 减去 `headsep` 再减去 `headheight` 剩下的距离，然后这段空白从视觉上来说还有你的页眉文本上面没有填充的部分。而 `footnote` 命令插入的文本和页码之间的空白间距是由 `footskip` 控制的，然后 `bottom` 减去 `footskip` 还会得到一个值，这个值是页码下面的空白距离。
- (3) 在 `beamer` 类下有一种看上去不错的多栏环境，就是使用的 `columns` 环境，不过只适用于 `beamer` 类的 `frame` 框架下。
- (4) 下面参考了[这个网站](#)
- (5) 不知道你注意到没有，本文用的是 `12pt`，上面表格上的 `huge` 和 `Huge` 字号还是有差异，这里因为我加载了 `moresize` 宏包的缘故。
- (6) 以前我喜欢放大 `CJK` 字体，但是后来发现这样放大之后让数学环境下的字体显得太小很不好看，而且这样整了之后似乎数学环境下的字体也不能通过 `DeclareMathSizes` 命令调节了。
- (7) 主要参考了[这个网站](#)
- (8) 如果你输入了某个 `Unicode` 的字形你目前的 `pdf` 字体并不包括，那么将会产生这样的错误信息：
`WARNING invalid CMap mapping entry`. 然后你编译 `pdf` 之后发现某个字符没有正常显示出来只是一个方框那么可能是目前字体不包括这个字形。
- (9) 参考了[这个网站](#)
- (10) 参看了 `a beginner's book of TEX` 的 `spacing between boxes` 一节。在 `google book` 哪里可以看到这一段，不过似乎这本书网上并不能自由下载。
- (11) 参看了[这个网页](#)
- (12) 这是一段尾注
- (13) 按照 Markus Püschel 的 `small guide to making nice tables` 里面的介绍。
- (14) 你注意到这里有 `@{}`，意思是每一列前面有一段空白，可以被花括号中的字符填充，这里是完全取消

掉那点空白。**booktabs** 的风格是开头那点空白和最后一列最后那点空白全部取消掉。

(15) 本小节除了参看 **wikibook** 之外还参看了[这个网站](#)。

xelatex 进阶

长度量

单位

L^AT_EX 中常见的长度单位如下表所示：

表 28-1: L^AT_EX 中的常见长度单位

单位	说明	换算法则
pt		
mm	一毫米	1mm=2.84pt
cm	一厘米	1cm=28.4pt
in	一英寸	1in=72.27pt
ex	(相对长度) 大约相当于当时字体的 x 高度	
em	(相对长度) 大约相当于当时字体的 M 宽度	

默认的长度量

- **\evensidemargin \linewidth \oddsidemargin \paperwidth \paperheight \textheight \textwidth \topmargin** : 这些长度量都和页面布局有关，很多都在前面页面布局那一节里面说了，这里就不赘述了。
- **\baselineskip \baselinestretch**
\baselineskip \parindent \parskip : 这些长度量和段落格式有关，大多都在前面段落一节说了，这里也不赘述了。
- **\columnsep \columnwidth \tabcolsep \unitlength**: 这几个长度量和

特殊的环境有关，比如 `columnsep` 控制多栏环境之间的宽度，`columnwidth` 控制栏的宽度，`tabcolsep` 控制 `tabular` 环境一列一列之间的间距，`unitlength` 控制 `picture` 环境下的 `unit` 的长度（还不太清楚）。

定义自己的长度量

要新建一个自己的长度量使用 `newlength` 命令即可：

```
\newlength{\lengthname}
```

修改长度量

用 `setlength` 命令直接设值某个长度量的值为多少。

```
\setlength{\lengthname}{20pt}
```

此外还可以使用 `addtolength` 命令来给某个长度值加上一个数值。

```
\addtolength{\lengthname}{20pt}
```

使用长度量

长度量一般在该使用的地方可以直接使用，其中伸缩量可以跟一个 `plus` 量和一个 `minus` 量。此外长度量还支持前面加个实数表示相乘多少的形式。

```
0.7\lengthname
```

显示长度量的值

要在 pdf 文档中显示某个长度量的值使用 `\the` 命令即可：

```
\the\lengthname
```

calc 宏包介绍

前面提及的 `setlength`，`addtolength` 命令只能接受最简单的长度量，在引入 `calc` 宏包之后可以进行一般的中缀数学表达式的运算。更多内容请读者自己翻看[官方手册](#)。

计数器

计数器 (counter) 之前也接触过一些了, 现在基于 [wikibook](#) 和[这篇博文](#)做一些整理工作。

latex 默认的计数器

- **part chapter section subsection subsubsection paragraph subparagraph** 这些计数器我们在前面的章节编号形式修改那一小节中有所接触, 具体请参看章节编号形式修改这一小节: [11.5](#)。
- **equation figure table** 这三个是浮动体环境的计数器, 其中 **figure** 我们在前面的图片标题的修改那一小节有所接触, 具体请参看图片标题的修改这一小节: [20.1](#)。
- **page footnote mpfootnote page** 看得出来是页面的计数器, 还没怎么接触。
footnote 是脚注的计数器, 这个在 **DIY** 脚注的时候会接触到, **mpfootnote** 是 **minipage** 下的脚注的计数器, 这个知识点参看的[这个网站](#)。
- 此外 **enumerate** 环境里面还有专门的计数器: **enumi enumii enumiii enumiv**。**enumi** 记录的是当前 **enumerate** 环境第一级的 **item** 出现的次数, 后面的依次是第二级第三级等等, 不怎么常用。具体请参看 **enumerate** 环境标签的修改这一小节: [19.2](#)

使用计数器

计数器并不可以直接使用, 需要使用以下命令来获得某个特殊形式的值。

value 将计数器转化为数值方便后面的计算, 不能作为字符直接显示。

arabic 将计数器转化为 1 2 3... 的形式，可以直接显示。

alph 将计数器转化为 a b c... 的形式，可以直接显示。

Alph 将计数器转化为 A B C... 的形式，可以直接显示。

roman 将计数器转化为 i ii iii... 的形式，可以直接显示。

Roman 将计数器转化为 I II III... 的形式，可以直接显示。

fnsymbol 将计数器转化为一系列的特殊符号。

计数器变成带圈的数字系列

我想新建一个命令，这个命令可以把计数器转化为 ①②③... 这样的形式，可以直接显示。可以看得出来这个命令会很实用的。

用 tikz 画的方法

```
\newcommand*\circled[1]{%
  \tikz[baseline=(char.base)]\node
    [shape=circle,draw,inner sep=1pt,minimum size=8pt] (char) {#1};}
\newcommand*\circledarabic[1]{\circled{\arabic{#1}}}
```

上面这段代码参考了[这个网站](#)。看得出来代码大体过程就是利用 **tikz** 宏包画出那个符号出来。

这个方法的好处是自由度比较高，根据 **circled** 命令，类似的可以定义 **circledalph** ——圈圈里面带个字母等。这个方法还有一个好处，虽然 ①② 这样的符号 **Unicode** 里面有，但是字体没有，如前面所述，还是需要写上一连串的代码，多少有点费事，而且到后面数字大了总会没有这个字体符号了，就会出错，而这个方法可以避免这些麻烦。推荐使用这个方法。

① 这是一个列表

② 这又是一个列表

定义自己的计数器

用 `newcounter` 命令来定义自己新的计数器。

```
\newcounter{countername}
```

修改计数器的值

系统默认的计数器都是默认设置初始值为 0，如果你需要使用这个计数器，首先加一，然后使用这个计数器。

计数器设值

直接将计数器设为某个值用 `setcounter` 命令。

```
\setcounter{countername}{number}
```

计数器加一

将计数器数值加一使用 `stepcounter` 命令。

```
\stepcounter{countername}
```

计数器加多少

将计数器加上一个数值使用 `addtocounter` 命令。

```
\addtocounter{countername}{number}
```

计数器和其他计数器绑定

在使用 `newcounter` 命令定义新的计数器的时候可以后面跟个可选项，可选项里填着另外一个计数器的名字，每当这个计数器加一的时候你定义的新的计数器就会归为零。具体格式如下：

```
\newcounter{mycounter}[chapter]
```

这样每当 `chapter` 加一也就是到了新的一章，你定义的 `mycounter` 将会重新归零。

新的命令或环境

新的命令

新建命令一般就用 `newcommand` 命令

```
\newcommand{name}[num]{definition}
```

用法都差不多，第一个必填选项是新建命令的名字，一般前面都加一个 `\` 符号。第二个可选选项是新建的这个命令接受的参数，比如你填一个 `2`，那么表示这个命令接受两个参数，在后面 `definition` 中引用的时候 `#1` 表示第一个参数，`#2` 表示第二个参数。后面还可以跟一个可选项表示这个命令的某个参数为可选参数并且可以给出默认值，这个可选参数默认对应后面的 `#1`。

\LaTeX 的 `newcommand` 命令是由原 \TeX 的 `def` 命令 (还有 `edef` `gdef` `xdef` 这里先不管。) 而来的。作用原理有点类似于替换展开的功能。比如：

```
\newcommand{\test}{this is a test line}
```

你定义了这个 `test` 命令，然后后面你输入 `\test`，系统遇到 `\test` 之后就会进行替换操作，就成了后面的 `this is a test line` 了。理论上只是简单的文本替换，后面可是跟上随意的命令参数或者环境。不过这只是理论上，实际编写宏包命令的时候会遇到很多复杂的问题，主要是 `expand` 展开的控制。

还有一个命令 `providecommand` 和 `newcommand` 差不多的，区别就是 `providecommand` 只是提供命令，如果这个命令已经定义了它就什么都不做。而 `def` 一定是完全不动声响的覆盖了。

在编写宏包文件的时候推荐使用 `DeclareRobustCommand` 命令，但是有些任务似乎只有 `def` 或者 `edef` 才能完成，这个我也没弄明白。

新的环境

新建环境命令格式如下：

```
\newenvironment{name}[num]{before}{after}
```

和 `newcommand` 命令其他用法都差不多，区别就在于后面有两个宏替换。这里值得一提的是原 $\text{T}_{\text{E}}\text{X}$ 并没有新建环境这个命令，而是 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 新加进去的。

我们假设我们要新建 `env` 这个环境，那么 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的工作流程是，创建了两个命令，一个是 `\begin{env}` 命令；一个是 `\endenv` 命令。所以简单的来看 `newenvironment` 命令就是把它看作开辟了一个 `group`（原 $\text{T}_{\text{E}}\text{X}$ 里面的概念），然后首先执行了 `before` 中的命令，然后是环境中的内容，然后是 `after` 中的命令，然后退出环境。

这里只是说明一下 `newenvironment` 命令的工作原理，具体我们新建环境还是用 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ 的风格，即用 `newenvironment` 命令，只有在不得已的情况下（通常很复杂的情况）才考虑用 $\text{T}_{\text{E}}\text{X}$ 里面的 `group` 概念，那个时候估计是要使用 `catcode` 命令。

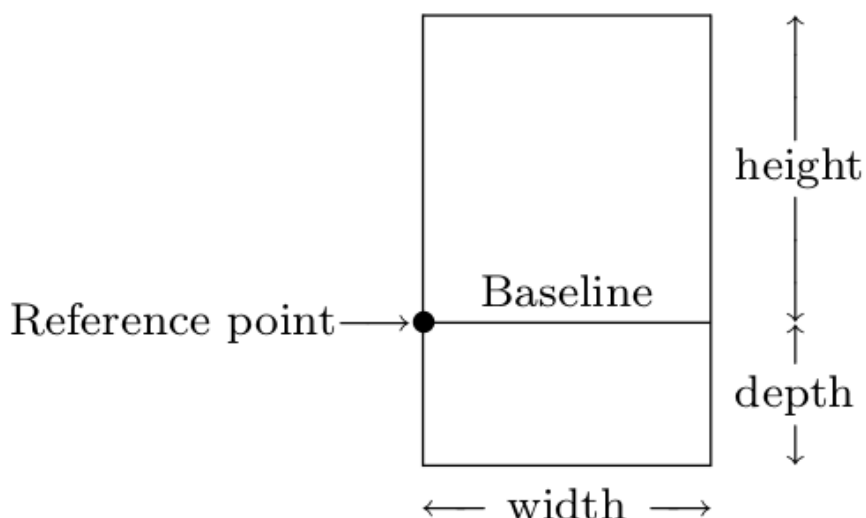
盒子和 glue

这里参考了 Knuth 的 *The TeXbook*，但是我并没有将其写入参考文献，因为只是觉得关于 **box** 和 **glue** 的概念最好参考原初定义，但是并不推荐读者阅读这本书，现在已经是 \LaTeX 时代了，不推荐读者使用原始的 \TeX 命令，一是不太实用，二是兼容性可能不太好。除非你是宏包编写者，但就作为一般的使用者真的没有必要接触那些原始命令了。还参考了 [\[boxes\]](#)

在这里 **box** 翻译为盒子没什么问题，就是这个 **glue** 翻译为胶水或者橡皮都让我不太满意。在后面我都使用的术语是距离或者间距或者干脆用英文 **glue**。从某种意义上讲 **glue** 的直译确实应该译为胶或者胶水，不过觉得距离这个词汇更能够让人们有感觉些： \TeX 排版就是不同的盒子编写和彼此之间距离的设置问题。

基本知识

最小的盒子就是基于 **Unicode** 的字符，这些字符然后组成更大的盒子——单词，然后单词组成更大的盒子——行等等。行是一个盒子，段落也是一个盒子，图片是一个盒子，表格也是一个盒子。而这些盒子按照 Knuth 的描述都是用 **glue** 胶水粘合起来的，或者我们称之为这些盒子之间都存在着空间胶合层。下面就是一个盒子的详细参数：



盒子就是这么一个长方形的区域，如上图所示，它有参数：**height**，**width**，**depth**。**baseline** 和 **reference point** 在后面讲的 **hbox** 和 **vbox** 中会用到。在 **T_EX** 看来，从字体而来的 **Unicode** 字符就是一个最简单的盒子。字体的设计者已经决定了这个字符的高度，宽度和深度以及它在这个盒子里面看起来如何。**T_EX** 就是用这些维度将盒子黏合到一起，并最终决定所有字符的 **reference point** 参考点在页面上的位置。

T_EX 的盒子如果全部涂上颜色，一般是黑色，那么就成了一个黑盒子。这样的黑盒子还有一个名字叫做 **rule box**。也就是线条。这个在后面会谈论到的。

不管是字符盒子还是黑盒子，他们要某是水平排列要某是垂直排列。水平排列要做的就是让这些盒子的参考点在一条水平线上。类似的垂直排列要做的就是让这些盒子的参考点在一条垂直线上。

好吧，介绍两个 **T_EX** 命令：**hbox** 和 **vbox**。**hbox** 命令就是让所有的盒子在一条水平线上，而 **vbox** 命令就是把一些 **hbox** 命令垂直排列，比如下面的代码：

```
\vbox{\hbox{恭}\hbox{喜} \hbox{发}\hbox{财}}
```

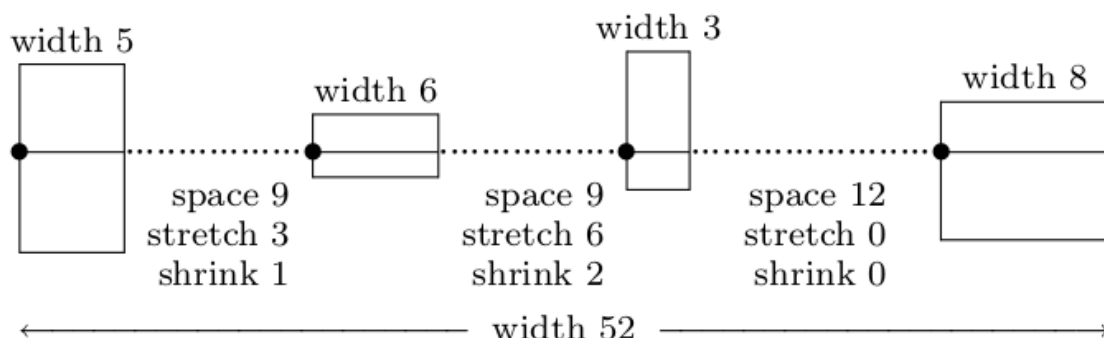
恭

喜

发

财

glue 也就是各个盒子之间的间距。下图是 **glue** 的具体图示：



前面说到盒子的 **reference point** 水平排列，然后他们之间还有叫做 **glue** 的间距。间距有三个属性：正常间距量 (**space**)，拉伸量 (**stretch**)，缩减量 (**shrink**)。比如这个图片中第一个 **glue** 的正常间距是 9 个单位，拉伸量为 3 个单位，缩减量为 1 个单位。而总的情况是正常间距是 $5(\text{box1}) + 9 + 6 + 9 + 3 + 12 + 8 = 52$ 个单位。现在假设一行宽 58 个单位，**T_EX** 就要调整使得这一行盒子的宽度刚好等于 58 个单位，于是还需要增加 6 个单位的宽度，而这 6 个单位的宽度^①需要从这一行所有 **glue** 里的拉伸量中找出来。于是总的拉伸量加法是 $3 + 6 + 0 = 9$ 。也就是 6 个单位的宽度要分成 9 等分再分配给他们，即第一个 **glue** 的拉伸量是 $3 \times \frac{6}{9}$ 。这样第一个间距的总长度就是 $9 + 3 \times \frac{6}{9} = 11$ 。

经过计算所有的 **glue** 间距都确定下来了，那么整个页面布局就确定了。我在这里就戛然而止了，毕竟这里只是对 **box** 和 **glue** 的基本概念的阐明。

盒子命令介绍

参考了这个网站

makebox

```
\makebox[width][alignment]{some text}
```

还有一个 **mbox**，不过 **makebox** 命令更加全面，推荐使用。**makebox** 就是制造一个水平的盒子，注意这个 **box** 里面的文本是不能换行的，也就是一行之内的盒子。第一个可选

^① 为了简单起见这里不考虑缩减量，具体缩减量如何计算我也不大清楚。

项 `width` 指这个盒子的长度，第二个可选项 `alignment` 是里面文本的对齐方式，有 **【l c r s】** 几个选项，**l**表示左对齐；**c**表示居中；**r**表示 `right`；**s**表示两端对齐。

raisebox

`raisebox` 命令一般的用法就是：

```
\raisebox{高度}{内容}
```

表示把某个内容放进一个盒子里然后抬高多少高度，高度值可以是负值则是降低。

resizebox

[参照了这个网站](#)

`resizebox` 命令是由 `graphics` 宏包提供的，一般都加载了吧。比如 `tikz` 绘图的时候如果你只是简单 `scale` 那么 `node` 内容会出现很大的偏差。这个时候如果你使用 `scalebox` 命令则可以将整个图片进行大小变换。

用法如下：

```
\resizebox{宽度}{高度}{内容}
```

```
\resizebox{\linewidth}{!}{内容}
```

```
\scalebox{2}{测试文字}
```

第二个的用法是将这个图片放大到文本宽度，高度填“!”符号的意思是随之而变化。

测试文字

表格放大或缩小

用 `scalebox` 和 `resizebox` 命令除了可以放大或者缩小图片之外，也可以用于放大或者缩小图片，这是值得注意的一件事，因为在排版的时候对表格的大小调整有的时候也是很重要的。

值得一提的是现在在表格里面 `verb` 命令不可以用了，会出错。

```
\begin{table}[H]
\rowcolors{2}{}{lightgray!50}
```

```

\centering
\resizebox{\linewidth}{!}{
\begin{tabular}{@{}ll@{}}
\rowcolor{lightgray!20}
表格放大      &      说明      \\
1      & 2      \\
3      & 4      \\
5      & 6      \\
\end{tabular}
}
\caption{表格放大}
\label{tab: 表格放大}
\end{table}

```

表格放大 说明

1

2

3

4

5

6

表 31-1: 表格放大

parbox 和 minipage

parbox 用法:

```
\parbox[pos][height][contentpos]{width}{text}
```

minipage 用法:

```
\begin{minipage}[pos][height][contentpos]{width}
```

```
text
```

```
\end{minipage}
```

他们的参数都类似，具体说明如下：

pos center, **top** 和 **bottom**，说是控制盒子相对周围文本内容的位置。

height 盒子高度

contentpos center, **bottom**, **top**, **spread**。文本分布，居中，底部，顶部或两端对齐。

带颜色或者线框的盒子

framebox

`framebox` 和 `makebox` 命令类似，除了加上了一个线框。

```
\setlength{\fboxsep}{10pt}
```

```
\setlength{\fboxrule}{5pt}
```

如上面描述的这里有两个长度量，`fboxsep` 控制线框和文字之间的距离，`fboxrule` 控制线框的宽度。值得一提的是 `framebox`，`fbox` 或者 `fcolorbox` 命令的线框的这两个参数都是类似上面的代码控制的。

colorbox

`colorbox`，带颜色的盒子。第一个参数是背景颜色，第二个参数是盒子里面的内容。

```
\colorbox{yellow!50}{this is a test line.}
```

this is a test line.

fcolorbox

`fcolorbox` 和 `colorbox` 的区别就是外面加了一个边框，然后第一个参数是边框的颜色，第二个参数是背景颜色，第三个参数是盒子里面的内容。

```
\fcolorbox{red!50}{yellow!30}{this is a test line.}
```

this is a test line.

保存盒子

savebox

线条

`\rule[depth]{width}{height}` 这三个选项都要填长度量，第一个长度量是线条竖向偏移量，第二个长度量是线条横向长度，第三个长度量是线条竖向宽度。

大文档管理

之前我也用过 `include` 命令来试着将文档分开，但是发现一是减弱了编辑器的各种小插件功能，二是有的时候 `include` 命令会出错（一些奇怪的错误，我也不太明白）。然后那个 `includeonly` 命令我觉得也挺累赘的，如果你使用 `input` 命令，不要引入了前面加个百分号注释掉即可。一个文档就是整体，觉得还是不要分开的好。

org 模式详解

org 转 tex 初步配置

```
;latex 设置
(setq org-latex-default-packages-alist '("(" "" nil)))
```

在.emacs 文件里面加上那么一句，这里 **org-latex-default-packages-alist** 列表里面存储着从 **org** 文件生成 **tex** 文件默认要加载的一些宏包。这个列表是由一系列的三个元素的列表组成，其中第一个元素是要加载宏包的选项，第二个参数是要加载宏包的名字，第三个选项设置为 **t** 或者 **nil**，即要加载或者不加载。这里就简单将 **org-latex-export-to-latex** 命令默认的要加载的一系列宏包全部都清空，因为有些宏包会和接下来我们要用的 **xelatex** 不匹配，然后既然我们经过了前面的学习已经有了自己的一般文档配置，那么简单把我们自己的文档配置加进来即可。

```
#+LATEX_CLASS: article
#+LATEX_CLASS_OPTIONS:[11pt,oneside]
#+LATEX_HEADER: \usepackage{article}

* org 模式入门

** 文档结构

    article 里面的一级结构也就是一个星号可以看作 section,
    两个星号可以看作 subsection, 三个星号可以看作 subsubsection.
```

上面 **org** 文档的 **#+LATEX_CLASS** 用来设置目标 **tex** 文档的 **documentclass**，这里的 **#+LATEX_CLASS_OPTIONS** 对应的是 **documentclass** 命令的可选项，这样就对应了目标代码：

```
\documentclass[11pt,oneside]{article}
```

然后

```
#+LATEX_HEADER: \usepackage{article}
```

对应的 `tex` 文档的代码是：

```
\usepackage{article}
```

这个 `article.sty` 文件正是我们之前一般文档配置的总结。

然后你用 `xelatex` 命令处理生成的 `tex` 文档，我们会发现配置文件都刷过去了没有问题，比如页面布局啊，字体啊等等都是有效的，生成的 `pdf` 已经很是美观了。但是我们还有一些问题要面对，第一个问题是 `org` 的输入模式和 `latex` 的对应问题，第二个问题是如何处理好自建命令和自建环境。

title 命令

默认的 `title` 命令填的是当前 `org` 文件的文件名，具体的 `title` 你可以通过：

```
#+TITLE: mytitle
```

来配置。

类似的还有 `AUTHOR`，`DATE` 对应的是 `tex` 文档的 `author` 和 `date` 命令，这些命令和 `org` 文档生成的 `html` 兼容性也很好。

设置目录深度

在 `org` 文档上加入如下 `OPTIONS` 的配置：

```
#+OPTIONS: toc:2
```

此即相当于在 `tex` 文档上加入了如下代码：

```
\setcounter{tocdepth}{2}
```

org 模式对应 latex 命令详解

如果你使用的是 **emacs** 编辑器，那么打开 **org** 后缀的文件即直接进入 **emacs** 的 **org** 模式，并不需要额外的配置。如果你需要更多的信息请参阅[org 模式官网](#)

接着前面的来，如果你在 **org** 文档前面定义 **latex** 的 **class** 是 **article**，那么一个星号对应的就是 **section**，两个星号就是 **subsection**，三个星对应的是 **subsubsection**。如果 **latex** 的 **class** 是 **book**，那么一个星号对应的 **latex** 命令是 **part**，接下来依次是 **chapter** 和 **section**。

四个星的情况默认转成了列表形式，这不是我们想要的。我们可以在 **org** 文件上加入这样的设置

```
#+OPTIONS: H:4
```

这样 **article** 就支持 **subsubsection**，**book** 就支持 **subsection**，也就是四级结构命令了。

为了以后方便，我们还是把 **org-export-headline-levels** 设置为一个很大的数字吧，一般 5 级结构，也就是五颗星够用了，**article** 到 **subparagraph**，**book** 到 **subsubsection**。

```
(setq org-export-headline-levels 5)
```

.emacs 文件及时生效

好吧，开始对修改 **.emacs** 或者其他插件之后，需要重新启动 **emacs** 来使其生效有点厌倦了。

org 文件的基本操作

org 文件的基本操作其实很简单，就是一些视图控制，比如你的光标落在一个标题上，然后按下 **Tab** 键试试，然后还有一个按键 **Shift+Tab**，其不需要光标一定落在标题上，全局性质的展开或折叠。还有其他一些按键操作和 **emacs** 的按键操作等等先暂时不予考虑，就用鼠标操作吧。

org 对应 latex 基础

现在我们可以开始编写最简单的带有文档结构的 **org** 文件了，写上一段文字，我们会发现，由于各个标记语言之间内部的相似性，**org** 上的一段文字就对应 **tex** 文档上的一段文字，就对应 **html** 文档上 `<p></p>` 包围的一部分也就是一个段落，虽然我们在 **org** 文档上多加入的空格和空行都会保留到目标 **tex** 文档和 **html** 文档上，但我们如同往常一样认为，多余的空行视作一个空行，多余的空格视作一个空格也是可行的（**html** 文档也是多个空格只视作一个空格）。恩，对于此，我们很是满意。

然后关于页面布局的控制，我们利用之前自定义的 **article.sty** 或者 **book.sty** 里面加载的 **geometry** 宏包然后进行设置，一切都工作得很好，而且和 **html** 文档的生成毫不干扰，这很好。

然后如果我们想要使用 **xelatex** 引擎，生成的 **tex** 文档对应的 **sty** 配置文件里面，如之前所说的设置好字体配置，然后用 **xelatex** 处理之，中文显示问题即解决，一切都很好。总之，在你的目标 **tex** 文档中，如果没有使用额外的命令或环境，而只是利用宏包然后在 **sty** 文件上进行配置，那么不会有任何问题，一切都工作得很好。

那么接下来要考虑的一个就是这个很紧要的问题：**tex** 文档的特殊符号的处理。

特殊的符号（org 模式中）

考虑到我们要利用 **org** 文档生成目标 **tex** 文档和 **html** 文档，而这三个文档内部各有自己的原生语义癖好，所以特殊符号如何处理和显示可真是个大问题啊。

第一个问题是 **#** 符号，之前我们接触了类似 **#+TITLE** 等这样的配置写法，似乎这个符号在 **org** 文档中是注释一般的存在，可是并不尽然。如果一行开头是 **#** 符号，后面的

内容不加额外的空格，那么 **#** 符号和后面的内容都会顺利的显示出来（包括 **html** 文档中），唯一要注意的是如果一行开头是 **#** 符号，然后后面跟着空格，那么这一整行都会被视作 **org** 文档内的注释内容，这一整行都不会在生成的目标 **tex** 文档或者 **html** 文档中。

然后接下来的 **tex** 文档中的一些特殊符号在 **org** 文档中都可以直接写上，其输出时都会如下所示进行转换：

\$	<code>\\$</code>
&	<code>\&</code>
%	<code>\%</code>
^	<code>\^{}</code>
{	<code>\{</code>
}	<code>\}</code>
~	<code>\textasciitilde{}</code>
\	<code>\$\backslash\$</code>

虽然上面的转换有些空格可能处理得不是很理想，然后 `\` 符号最好使用 `\textbackslash{}` 命令（字体问题），但考虑到我们这条线路跟来就是更多的关注于内容，所以这些小细节且先略过吧。

值得一提的是这些符号的显示同样对于生成目标 **html** 也会有相应的处理方案，显示得很好。

连字号

我们知道在 **tex** 文档中一个两个连字号在生成 **pdf** 时会处理为特殊的字符，现在这点不需要我们担心了，经过测试，**org** 文档将一个或者两个或者三个连字号原样保留在了 **tex** 文档中，然后 **html** 文档中已经变成了对应的 **unicode** 符号了，和前面谈论的 **pdf** 显示效果是一致的。

更多的符号和中文化问题

在 **sty** 配置文件中设置好即可，然后直接用输入法输入，没什么好担心的。中文化问题在 **sty** 配置文件中配置好，就不用操心了。

\ 符号

\ 符号还需要讨论一下，如果只是希望输入单独这个符号那么就直接输入即可，但是如果 \ 符号后面紧跟着其他字符或符号，即类似 `tex` 文档中命令这样的形式：

```
\test
\test{}
\test[]{}

```

那么这些代码将会原样送入 `tex` 文档中，即 \ 符号没有发生转变（如果 \ 符号后面跟着中文字符不在上面讨论的情况中，即还是普通的 \ 符号。）。

这似乎很方便，可以实现 `org` 文档无缝对接 `tex` 文档，但在 `html` 上就会多了很多奇怪的 `\test` 之类的命令，而且如果我们这样使用 `org` 文档，那还不如直接用 `tex` 文档作为第一编辑代码源呢。所以如上这些 `tex` 文档的命令和环境在 `org` 文档中都不推荐使用（后面还会继续详细讨论这个自定义命令的对接问题），但有一个情况是个例外，请看到下面的数学环境。

数学环境

接下来要讲的严格意义上和 `org` 模式关系不大了，我们在 `org` 文档中添加的原生数学环境大多原样存放在了目标 `tex` 文档和目标 `html` 文档上了，`tex` 文档不用说了，而目标 `html` 文档之所以能够正确显示这些数学环境的 `tex` 代码是因为 `org` 生成的 `html` 目标文档配置了 `mathjax` 这个工具。

但是 `mathjax` 所支持的数学环境 `tex` 代码和原 `latex` 的 `tex` 代码还是有细微差异的，下列命令或环境能够正常支持：

```
$ $ 行内数学环境
$$ $$ 单行数学环境
\[ \] 单行数学环境，没有编号

```

虽然上面的 `\[\]` 环境支持，但 `displaymath` 不支持，这个有点古怪，估计是 `org` 具体生成配置还没优化好和 `mathjax` 工具无关。

然后来自 `amsmath` 宏包的 `equation` 和 `equation*` 环境也能正常支持，不过 `equation` 没有显示编号，估计和 `mathjax` 具体的配置有关系。

然后多行数学环境，即里面可以用 `\\` 来换行的多行数学环境，`gather` 和 `gather*` 还有 `align` 和 `align*` 都正常支持，同样编号没有正确显示，估计和 `org` 生成的目标 `html` 文档的 `mathjax` 配置有关。

至于这些数学环境里面那些命令可用那些命令不可用那就不好一一讨论了，应该常用的 `tex` 原生数学命令和 `amsmath` 宏包里面的数学命令（数学环境）都是支持的。

换行命令

`\\` 命令在 `html` 会被转换为 `
` 命令，因此你可以在 `org` 文档中自如地加上换行 `\\` 命令。但是 `\\[10pt]` 这样的命令就不要用了，在 `html` 上也没有这个概念。换行命令算是一个 `tex` 命令可以进入 `org` 中的特例。

插入列表

插入列表 `org` 模式解决得比较完美了，无序列表项以 `'-'`、`'+'` 或者 `'*'` 开头，其对应的是 `latex` 的 `itemize` 环境；而有序列表则以 `'1.'` 或 `'1)'` 开头，其对应的是 `latex` 的 `enumerate` 环境，然后 `description` 环境 `org` 对于生成的 `html` 文档的 `css` 还有额外的处理，在 `org` 上你只要简单如下输入即可：

```
- the bold font :: some test
```

进一步的深度定义推荐都在 `sty` 文件里面定制。

插入引用

`org` 模式上插入引用也是很方便的，`label` 对应的命令格式如下：

```
<<targetname>>
```

其将转变成为:

```
\label{targetname}
```

然后要引用这个目标地点就使用如下语法:

```
[[targetname]]  
[[targetname][shownname]]
```

这里的 **targetname** 对应的就是你上面定义的地点名字, 然后 **shownname** 就是实际页面显示链接的文本, 如果如第一种形式不设置 **shownname**, 那么生成的 **tex** 代码是使用的 **ref** 命令, 如下形式:

```
\ref{targetname}
```

如果加上 **shownname** 那么使用的是 **hyperref** 宏包提供的 **hyperref** 命令, 如下:

```
\hyperref[shownname]{targetname}
```

需要提醒读者的是: targetname 不能使用中文, **shownname** 可以使用中文。

外部链接使用类似上面的语法:

```
[[http://www.google.com][谷歌]]
```

其对应的 **tex** 代码如下:

```
\href{http://www.google.com}{谷歌}
```

其他形式还有:

```
file:papers/last.pdf
mailto:adent@galaxy.net
```

还有其他一些格式，这里的讨论就略过了。关于这些链接的颜色啊等等 `hyperref` 宏包的配置，请在 `sty` 文件上完成。

相应的生成的 `html` 文档代码也很好，无需多言。

插入脚注

由于 `pdf` 和 `html` 显示风格的不同，实际上脚注并不是那么好协调，在 `html` 上脚注更确切不应该叫做脚注而要叫做尾注。所以 `org` 模式提供了两种插入脚注语法：

```
this is a test line[fn::this is a footnote]

this is a test line[fn:name]

[fn:name] this is a footnote
```

这两种写法在 `html` 上显示稍微有点区别，但无关紧要。总的来说更推荐第一种写法，其直接将脚注内容写进去不用考虑命名和编号细节，而第二种写法还要花心思去考虑脚注具体叫什么名字，浪费脑力，而且我们选择 `org` 模式不就是为了重视内容快速写作吗，第二种脚注写法有点分散写作者注意力的嫌疑。

特殊的文字环境

粗体

```
*bold*
```

两个星号包围的字体将被加粗，对应生成的 `tex` 代码即：

```
\textbf{bold}
```

其对应生成的 **html** 代码如下：

```
<b>bold</b>
```

斜体和 `emph` 命令

在 **org** 模式下，如下两个`/`符号包围的字符信息

```
/环境/
```

输出在 **html** 文档上将是：

```
<i> 环境</i>
```

我们似乎可以把两个`/`符号包围的环境称作斜体环境，如果其对应的 **tex** 代码则是：

```
\emph{环境}
```

tex 的原始代码确实 **emph** 内的字体将用 **italic** 字形处理，不过这个命令经常被换成其他形式，比如 **ulem** 会将其换成下划线等，而我喜欢将其换成红色，觉得更有强调的效果^{18.2}。而 **html** 的 **i** 环境我们其实也可以稍作调整即可以达到同样的效果，比如：

```
i{  
  font-style:normal;  
  color:red;}
```

这样我们可以简单理解两个`/`符号包围的字符是文字强调功能。

下划线

我使用了一段时间 `ulem` 宏包了，除开花里胡哨的命令，`uline` 和 `uwave` 命令是用的最多的。`uwave` 命令以后考虑怎么处理加入去，现在 `uline` 命令 `org` 模式已经很好的支持了。

`org` 文档中的

下划线环境

将会变成 `tex` 文档中的代码：

```
\uline{下划线环境}
```

而其对应的 `html` 代码如下所示：

```
<span class="underline"> 下划线环境</span>
```

这个 `underline class`，`org` 生成的 `html` 已经定义好了，具体就是文字下面加上了下划线效果。

短代码环境

`org` 文档的

```
=verbatim 环境 =  
~ 另外的 verbatim 环境~
```

输出 `html` 文档都会被 `code` 标签包围，即短代码环境。而在 `tex` 文档下有点差异。

第一种情况对应的是 `ttfamily` 字族情况：

```
\texttt{verbatim 环境}
```

第二种情况对应的是 `verb` 命令：

```
\verb~ 另外的 verbatim 环境 ~
```

不得不承认这个 `~` 符号已经选得很好了，但代码内还是可能出现干扰。更多的细节参见下面的插入代码一小节的讨论。

verse 诗歌环境

诗歌环境 `org` 模式处理的很好，会自动在每一行后面加上 `\\` 换行。所以你只需要如下输入：

```
#+BEGIN_VERSE
这里放着
诗歌
#+END_VERSE
```

其生成的 `tex` 代码如下：

```
\begin{verse}
这里放着 \\
诗歌 \\
\end{verse}
```

跟进一步的定制在 `sty` 文件里面配置 `verse` 环境即可。

其对应的 `HTML` 代码是：

```
<p class="verse">
这里放着<br />
诗歌<br />
</p>
```

其对应的 `p.verse class` 在自动生成的 `html` 文档前面已经很粗糙的定义好了——就是简单的缩进了下，可能并不能让你满意。

quote 引用环境

类似的 `quote` 加入语录环境有：

```
#+BEGIN_QUOTE
```

```
这里放着 quote
```

```
引用环境
```

```
#+END_QUOTE
```

对应的 `tex` 代码就是变成 `quote` 环境，没有其他改动了。`html` 代码是对应了一个特定的代码：

```
<blockquote>
```

```
<p>
```

```
这里放着 quote
```

```
引用环境
```

```
</p>
```

```
</blockquote>
```

这个 `blockquote` 标签我还是第一次简单，`html` 文档没有做任何配置，这个标签环境默认就有了缩进举动，有那么一点引用环境的意思了。

居中环境

和上面讨论的情况类似，不赘述了。

```
#+BEGIN_CENTER
```

```
文本居中环境
```

```
#+END_CENTER
```

插入一横线

如上所示，在 **org** 文档中这样的连接号大于等于 5 个，其将输出一横线，对应的 **tex** 代码是：

```
\rule{\linewidth}{0.5pt}
```

对应的 **html** 代码就是大家熟悉的 **hr** 标签了。

插入图片

插入图片用的和前面谈及的插入引用一样的语法，最简单的形式如下所示：

```
[[file:images/test.jpg]]
```

其生成的 **tex** 代码是：

```
\includegraphics[width=.9\linewidth]{images/test.jpg}
```

注意到其并没有加入 **figure** 环境，然后对应的 **html** 代码是：

```
<div class="figure">
<p>
</p>
</div>
```

其已经引入 **figure class** 了。

注意：插入图片就不要使用这样的形式了：

```
[[file:images/test.jpg]][test]]
```

其将变成前面讨论的插入引用的形式，**tex** 文档用 **href** 命令处理，然后 **html** 用 **a** 标签处理。

加上标题

图片加上标题如下所示：

```
#+CAPTION: 女金刚狼
[[file:images/test.jpg]]
```

其生成的 **tex** 代码将变成这样的形式，即引入 **figure** 环境了：

```
\begin{figure}[htb]
\centering
\includegraphics[width=.9\linewidth]{images/test.jpg}
\caption{女金刚狼}
\end{figure}
```

加上标签

```
#+CAPTION: 女金刚狼
#+NAME: fig:test
[[file:images/test.jpg]]
```

其生成的 **tex** 代码如下所示：

```
\begin{figure}[htb]
\centering
```

```
\includegraphics[width=.9\linewidth]{images/test.jpg}
\caption{\label{fig:test} 女金刚狼}
\end{figure}
```

非常遗憾，同在插入引用一小节所提及的，`label` 的中文名标签不支持，所以上面只好随便取一个名字了。

值得一提的是：**#+NAME**：必须在 **CAPTION** 的下面，而且不能单独存在——即没有 **CAPTION** 命令。

关于生成的 **html** 代码这里略过讨论了。

上标和下标

上标和下标在 **org** 文档中就可以这样直接使用：

这是 \wedge {上标}，这是 $_$ {下标}，十分方便。

之所以敢这样使用是因为 **html** 文档中使用 **sup** 和 **sub** 标签即能很好地解决上标和下标问题，而生成的 **tex** 代码如下：

这是 $\wedge^{\text{上表}}$ ，这是 $_{{\text{下表}}}$ ，十分方便。

我们看到这里自动加上数学环境进行处理了。这算是 **org** 模式一个额外的加分点吧。

org 转 html 和 tex 的协调问题

其实 org 的转 html 和 tex 文档的协调问题很好解决，加入

```
#+LATEX: \appendix
```

那么这一行的内容，就只原样输入进 tex 文档中，不进入 html 文档中。类似的有命令：

```
#+HTML:
```

类似的还有

```
#+BEGIN_LATEX
```

这里将原封不动的放入 tex 文档

```
#+END_LATEX
```

但是既然我们选择了 org 模式，那么就不应该过多的添加 tex 原始代码。html 和 tex 这种代码越多你的 org 文档就越复杂（最糟糕的情况可能是等于你编写了两边文档），所以如果一定要用的，那么还是推荐使用

```
#+LATEX: \appendix
```

这种一行的修正。

以下这些命令是基于 tex 文档结构的，可以考虑加进去，加进去了反而可以更好地提醒写作者此处我在写些什么。

```
\mainmatter
\appendix
\backmatter
```

其中 `mainmatter` 命令放在文档内容开始的前面，然后 `appendix` 放在你的附录标题前面，`backmatter` 放在你的参考文献标题前面。其他内容照旧，`html` 没有页码自然不用考虑那些问题。注意这些命令都是针对 `book` 类的。

其中参考文献为了考虑协调性如果没有太多的话就使用简单的列表表达即可。

原有环境命令属性的修正

其只适用于表格，图片，列表，源代码和其他特殊的区块。

```
#+ATTR_LATEX:
```

文档内非要使用自定义的命令

基于简单原则，`sty` 配置文件最好不要引入额外的自定义命令和环境，如果实在不得不使用自定义的命令或环境，同时内容最好是 `html` 和 `pdf` 都共享的（这正是 `org` 模式本身要求的），那么接下来工作就会变得很多了，下面主要讨论这个问题。

在前面讨论的各种情况中，如果 `org` 模式针对 `tex` 代码和 `html` 代码有特定的环境命令或 `class`，那么推荐进一步定制该环境命令或 `class` 即可。下面讨论的情况是除开这些特定的环境命令或 `class`，要使用其他的环境命令或者 `class`，那么在 `org` 文档中，任何

```
#+BEGIN_WHAT
```

```
#+END_WHAT
```

都将输出为 `tex` 文档的如下格式：

```
\begin{what}
```

```
\end{what}
```

对应的 **html** 文档代码是：

```
<div class="what">
```

```
<p>
```

```
what
```

```
</p>
```

然后读者把 **tex** 文档的该 **what** 环境（在 **sty** 文件中）配置好和把 **html** 的该 **what class**（在 **css** 文件中）配置好即可。

插入表格

接下来的插入表格和插入代码，就最简单的应用并没有多少内容，不过这里将它们单独提出来作为一章，是因为在 **org** 模式和在 **emacs** 编辑器下，如何处理表格，生成表格，进行运算，如何处理代码，运算代码，加载其他文件进行运算生成表格，加载其他文件的代码等等有着丰富的内容。这部分内容算是比较高级的东西了，就拖到这后面单独成为一章了。

reference ch03

插入代码

小代码之前讲过了，两个等号或者两个波浪号包围的在 `tex` 文档中一个对应 `ttfamily` 字体，一个对应 `verb` 命令。

简单的代码

简单的代码如下所示，其对应的就是 `tex` 文档的 `verbatim` 环境。

```
#+BEGIN_EXAMPLE
      THIS IS A EXAMPLE ENVIRONMENT
#+END_EXAMPLE
```

不需要华丽表现不需要染色的简单代码就推荐使用 `EXAMPLE` 环境，不过我对 `verbatim` 环境不太满意，主要是前后没有分割直线，没有将代码和文本和清晰的分开。一直使用的是 `fancyvrb` 的 `Verbatim` 环境，那么我们如何把 `org` 模式的 `EXAMPLE` 环境对应的输出为 `Verbatim` 环境呢？

本来以为这个问题会很简单，不过目前要达到这种效果还有点麻烦，因为目前（2015-01-22）的 `org` 模式的 `EXAMPLE` 并没有把这个选项打开，在参考[这个网站](#)的情况下，大致的一个解决方案如下：

1. 首先如果你是手工编译的 `emacs`，那么在 `[lisp]→[org]` 文件夹里找到 `ox-latex.el` 这个文件，在这个文件里看到 `org-latex-example-block` 这个函数，看到 `format` 那里，你就明白为什么默认的输出的 `verbatim` 环境。那么将最后两行删除，即进行如下改动：

```
-      (format "\\begin{verbatim}\\n%s\\end{verbatim}"
-          (org-export-format-code-default example-block info))))
+      (format "\\begin{%s}\\n%s\\end{%s}"
+          org-latex-verbatim-env
+          (org-export-format-code-default example-block info)
+          org-latex-verbatim-env))))
```

这里简单知道`%s` 类似大多数编程语言的字符串处理方式即可，这样环境就改为由 **org-latex-verbatim-env** 来定义了。然后重新编译 **emacs**。

我之前是自己编译的 **emacs**，但并没有采取上面的方式，采取的是下面提及的更新 **org** 宏包的方式。

首先在`.emacs` 文件里面加入如下代码：

```
;more package
(require 'package) ;; You might already have this line
(add-to-list 'package-archives
              '("melpa" . "http://melpa.org/packages/") t)
```

这将为你的 **emacs** 宏包管理提供更多的可用宏包，这些宏包是由 **melpa** 提供的。然后打开 **emacs**，打开 **options** 的宏包管理，你就可以看到更多的宏包选项了，其中有最新的 **org**，安装即可。这些宏包都放在你的主文件夹的`.emacs.d` 文件夹那里。

这里顺便提一下如何通过这个宏包管理插件删除安装的宏包，将光标移动到你想删除的宏包名字那里，按一下字母 **d** 键，你就可以看到前面有个标记 **D**，然后移动光标到其他位置，按下 **x** 键你就会看到 **minibuffer** 有提示了^①。

在`.emacs.d` 文件夹的 **elpa** 文件夹里面，你会看到 **org** 宏包相关的文件夹，同样打开找到 `ox-latex.el` 文件，然后类似上面谈及的做出一些修改，然后注意更新一下 `elc` 文件。具体就用 **emacs** 打开这个文件，然后在 **emacs-lisp** 下面的 **byte-compile this file** 选项。

^① 参考了[这个网站](#)

2. 接下来就是修改`.emacs` 文件，把我们修改好的接口 **org-latex-verbatim-env** 这个变量定义好：

```
(setq org-latex-verbatim-env "Verbatim")
```

好了，你可以测试一下效果了。

更华丽的代码

`org` 模式还提供了 `SRC` 环境，对应的是 `latex` 的更华丽的代码环境：

```
#+BEGIN_SRC python
print('hello')
#+END_SRC
```

其输出可以调成 `listings` 环境：

```
(setq org-export-latex-listings t)
```

不过我不太喜欢 `listings` 环境，其对 `minted` 环境的输出支持也很好（后面将要谈到的选项支持）：

```
(setq org-latex-listings 'minted)
```

`minted` 环境有个问题，就是加上背景色之后分页会有问题，在之前的介绍 `xelatex` 基础语法中的插入代码一小节里谈到的自定义命令 `tcbpython` 和 `tcbbash` 如何引入进来呢，因为 `tcolorbox` 和 `minted` 结合在显示代码这一块很是完美和让人满意了。

我们可以在`.emacs` 文件里进行如下设置：

```
(setq org-latex-custom-lang-environments
      '((python "tcbpython"))
```

```
(bash "tcbbash")
))
```

这样，当你的 SRC 环境后面跟着 `bash`,

```
#+BEGIN_SRC bash
sudo pip3 install pygments
#+END_SRC
```

就会输入 `tex` 代码如下:

```
\begin{tcbbash}
sudo pip3 install pygments
\end{tcbbash}
```

类似的上面我们还定义了 `python` 对应的是 `tcbpython`，也会对应成 `tcbpython` 环境。而我们没有自定义的语言，如果你设置了 `minted` 为默认引擎，比如这样的情况：

```
#+BEGIN_SRC emacs-lisp
some code
#+END_SRC
```

那么将生成这样的 `tex` 代码:

```
\begin{minted}[]{common-lisp}
some code
\end{minted}
```

这已经很好了。至于 `html` 那边的情况，代码都能正常显示，但没有染色，你可以安装 `emacs` 的 `htmlize` 宏包来达到某种染色效果。。但真的好丑陋，我对那边的情况还不太熟悉，所以这里略过不做讨论了。

SRC 环境的一些选项

SRC 环境跟着程序语言种类之后后面还可以跟很多选项，其中有些选项很复杂，还有一些设置很底层。这个有兴趣的慢慢研究手册吧。

暂时就介绍一个选项 **-n**：

```
#+BEGIN_SRC emacs-lisp -n
def main(args):
    return 0
#+END_SRC
```

如果 SRC 环境选择是 **minted** 环境作为输出，那么其参数会对应的设置进去：

```
\begin{minted}[linenos,firstnumber=1]{common-lisp}
def main(args):
    return 0
\end{minted}
```

如果是 **listings** 环境也会支持的很好，但是如果是前面谈及的自定义的 **tcbpython** 环境或 **tcbbash** 环境，代码的每一行前面都将加上数字：

```
1 def main(args):
2     return 0
```

这不是很美观，更好的做法，如果你喜欢加上数字，那么在 **sty** 配置文件的 **tcbpython**（或者你定义的什么名字）的定义里加上：

```
minted options={linenos,numbersep=3mm}
```

即可。

当然假设你定义的 `python` 对应的是 `what` 环境也没问题，反正这里不涉及 `html`，然后你在 `sty` 文件里面配置好这个 `what` 环境即可。

最好就是如下的无参数 `what` 环境形式。

```
\begin{what}
```

```
\end{what}
```

插入代码文件

废话不多说，插入代码文件格式如下：

```
#+INCLUDE: "~/emacs" src emacs-lisp
```

当然是不可能采取这种形式的，就是 `INCLUDE` 命令放在 `SRC` 环境里面，这样将不会被执行而被视作代码。所以插入代码文件只能采取上面的形式，后面跟着 `src`，似乎还有其他可能性，然后接下来是程序语言，当然前面提及的 `SRC` 环境的一些选项也是可以加上去的。

执行代码

这部分内容等下再讨论。

org 模式其他额外的特性

快速输入

`emacs` 深度定制能够开发出很多快速输入技术，而 `org` 模式自带的就有一些快速输入的方法了，已经很好用了。比如你要快速输入 `EXAMPLE` 环境，先输入一个 `<` 符号，然后再输入一个小写字母 `e`，然后按下 `Tab` 键，就能快速输入 `EXAMPLE` 环境，类似的有：

c `CENTER` 环境

e `EXAMPLE` 环境

h `HTML` 环境

H `HTML` 单行命令，即 `#+HTML:`

I `INCLUDE` 单行命令，即 `#+INCLUDE:`

l `LATEX` 环境

L `LATEX` 单行命令，即 `#+LATEX:`

q `QUOTE` 环境

s `SRC` 环境

v `VERSE` 环境

此外你还可以定制自己的快速输入命令和环境，用以上类似的操作；此外在 `emacs` 环境下你可以定制很多快速输入的按键和命令等等，这里先略过了。

todo list ch 05

标签 ch 06

属性日期和时间 ch07 ch08

捕获转发 ch09 议程 ch10

xelatex 高级篇

tikz 制图

这个内容较多，需要慢慢整理出一个 **tikz** 制图参考手册。

更多内容请参见我的这个 **github** 项目，里面专门讨论了 **tikz** 制图，并放了很多实际的例子进去。[tikz-examples](#)

化学相关

化学相关符号甚至非常复杂的有机分子式的显示问题推荐用`chemfig`宏包解决。

最基本的用法就是

```
\chemfig{C-H}
```

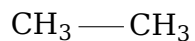


后面更加复杂的化学分子式都是在这个模型的基础上构成的，简单来说就是第一个原子，然后连着键，然后跟着第二个原子。

`chemfig` 会默认调用 `tikz` 宏包，也就是它的绘图功能是在 `tikz` 的基础上开展起来的，所以我们猜到前面所谓的原子 `C` 实际上是一个 `node`，而这里的键就是连接 `node` 的路径。

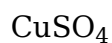
不过和 `tikz` 的 `node` 不同，`chemfig` 这里的“原子”都是默认支持数学模式（这里主要想说的是支持上标和下标模式）的，于是我们画出了乙烷：

```
\chemfig{CH_3-CH_3}
```



当然简单的无机分子表达式也不在话下了，比如硫酸铜，其他的大多类似了：

```
\chemfig{CuSO_4}
```



什么时候是原子

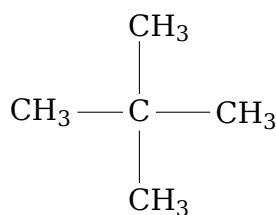
什么时候是所谓的原子，按照 **chemfig** 的参考手册就是程序在扫描的时候如果遇到以下字符则表明原子（团）结束：

- = ~ < > (! * @

其中 - = ~ < > 这些符号的作用是控制键的类型，如果键开始了那么当然原子（团）也就结束了。具体请看什么键这一小节[41.2](#)。

其中 (这个符号表示一个分支原子（团）开始了，那么当然之前那个原子团就结束了。比如说我要画新戊烷：

```
\chemfig{CH_3-C(-CH_3)(-[2]CH_3)(-[6]CH_3)}
```



我们看到括号包围着分支原子团，其实之前的乙烷如果我们也加上一个括号是没有什么影响的，如下所示：

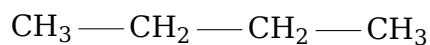
```
\chemfig{CH_3(-CH_3)}
```

```
\chemfig{CH_3-CH_3}
```



可见从处理上讲圆括号包围的部分只是明确讲所谓的“原子”或者说原子团或者说 **node** 标记了出来。后面的处理都是类似的。而分支原子团的用法就是其作用的中心原子还是之前的那个原子，然后键后面跟着的选项这里是控制键的角度的，这里却不讨论。如果不使用分支原子团的圆括号用法，那么当前作用的原子是逐步移动的（似乎是 **tikz** 中 **++** 和 **+** 的区别。）下面是正定烷的例子：

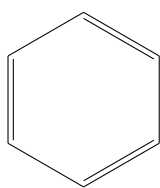
```
\chemfig{CH_3-CH_2-CH_2-CH_3}
```



然后我们还有！ * @ 这几个符号，！是用来变量替换的，具体讨论请看到后面的***** 一小节。

* 星号和有机化合物的环结构有关，比如说我要画一个苯环：

```
\chemfig{*6(====)}
```



这里星号 6 的意思表示一个六元环结构，相关更加详细的讨论在后面的环结构这一小节。

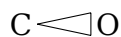
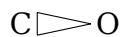
@ 符号的用法更加高级了，后面***** 再讨论。

什么什么键

前面已有所提及，-表示单键，= 等号表示双键，然后波浪号表示三键，然后尖括号和立体化学涉及到的那个键有关。比如演示如下：

```
\chemfig{C~O}
\chemfig{C>O}
\chemfig{C<O}
\chemfig{C>:O}
\chemfig{C<:O}
\chemfig{C>|O}
\chemfig{C<|O}
```





键的类型就这么多了，关于键更加复杂的是键的角度等各个参数的控制。

键的角度和其他参数控制

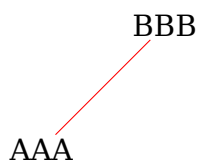
还是从 $\text{A} \text{---} \text{B}$ 这个简单的例子说起，我们看到单键后面可以跟一个方括号的可选参数项，实际上这个可选参数项可以接受五个参数，具体是：

```
\chemfig{A-[角度, 默认长度的乘以因子,A 原子团的第几个原子,B 原子团的第几个原子,
其他 tikz 路径可选参数]B}
```

这些参数的默认值是：角度 0° ，默认长度乘以因子 1 ，后面的都是空值，由 `chemfig` 的其他部分决定。

我们来看这个例子：

```
\chemfig{AAA-[1,2,2,2,red]BBB}
```



这个例子的意思是角度 45° ，然后长度是默认长度的 2 倍，然后从第一个原子团的第二个原子出发指向第二个原子团的第二个原子，最后是 `tikz` 的额外参数 `red`，即键的颜色调为红色。

一般最常用的就是角度参数吧，如果你不希望调角度只希望调后面的某个值，那么参数列表需要加上逗号标记出你正在调整的是第几个参数。

角度的三种表达方式

第一种表达方式应该最常用，0 表示向右的角度 0，然后 1 表示 45°，然后后面的类推。

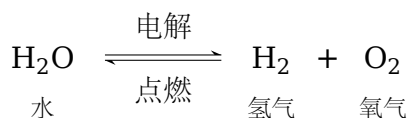
第二种表达方式是绝对角度，:35 就表示 35 度。

第三种表达方式是相对角度，::30，也就是相对之前的角度偏移了多少度。

高中的化学反应式的绘制

对于高中（或初中？）的化学反应式的绘制重要的不是化学分子式，他们遇到的化学分子式一般都比较简单，如NaCl之类的，而是这两个东西，化学分子式下面加上名字和箭头还有箭头上也要有说明文字。

```
\chemname{\chemfig{H_2O}}{\footnotesize 水}
\chemrel[电解][点燃]{<>}
\chemname{\chemfig{H_2}}{\footnotesize 氢气}
\chemsign{+}
\chemname{\chemfig{O_2}}{\footnotesize 氧气}
```



这里 **chemname** 命令可以在你画的化学分子式下面加上说明文字，第一个参数是 **chemfig** 命令，第二个参数是说明文字，然后 **chemname** 还可以接受一个可选参数，控制说明文字和化学分子式之间的距离。

这里 **chemrel** 命令就是加上箭头，其中->就是常见的单向箭头，然后还有<-,<->,<->。而这里的<>表示可逆反应的那种箭头。然后可以接受两个可选参数，第一个是上面的文字，第二个是下面的文字。

这里还有一个 **chemsign** 命令，和直接输入+加号不同，这个命令会在你要输入的符号之前和之后都加上一段空白距离，这个命令可以接受一个可选参数就是调整这个空白距离的。使用这个命令会让你的化学反应式更好看一点，因为我们一般两个化学反应物之间会有一小段距离。

更多内容待续.....

exam 类

详细情况请参看本 [github](#) 中文件夹 [exam 类]。

beamer 类

\LaTeX 中幻灯片的处理是用 **beamer** 类来写的，老实说我认为 \LaTeX 并不适合偏视觉的幻灯片的编写， \LaTeX 更适合内容文字等等较多的时候的情况，那个时候 **word** 都有点力不从心，而 **tex** 标记语法风格结合良好的编辑器是很得心应手的。

所以这部分内容我决定略过了，在本 **github** 里面有个制作幻灯片的文件夹，里面有一点我早期探索的内容，可做读者的参考。

总的来说推荐还是用 **ppt** 之类的编辑软件制作幻灯片吧，这样更加视觉化和有感觉些。

附录

我的作业心得分享

"The opera ain't over until the fat lady sings."

好戏还在后头，等著瞧吧。

我在 Ubuntu 的模板文件夹里面新建了一个 `tex.tex` 文件。方便我用鼠标右键快速创建模板。

第一个模板我打算使用一个通用的配置，写一个 `myconfig` 宏包，用于加载这个通用配置。还有一个 `mytitle` 宏包用于加载封面。然后整个模板就专注于写作，很简洁的样子。

然后我在 Ubuntu 的 `home` 文件目录或者称之为主文件夹下面新建了一个文件夹 `texmf`，然后里面新建了一个文件夹 `tex`，然后里面新建了一个文件夹 `latex`，然后在里面新建了一个文件夹 `myconfig`。`myconfig.sty` 文件就放在里面。然后在终端上运行命令：`sudo~ texhash` 即可。`mytitle` 宏包的安装类似。

`myconfig` 宏包的内容是我以前之前的 XeLaTeX 指南的精华部分，很多冗余删除了。关于 `myconfig` 和 `mytitle` 宏包的编写，我参考了[这个网站](#)。

三个文件的代码在本文后面列出。现在就一些必要的信息说明如下：

首先 `sty` 文档里面 `@` 符号不需要用 `makeatletter` 这个命令了，`@` 在里面已经默认就是一个符号。其次文档的基本结构

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{mytitle}
...
\endinput
```

这个不用多说。

然后就是 `sty` 文档里面所有 `usepackage` 都换成 `RequirePackage`。这样 `sty` 文档兼容性更好，前面没有 `documentclass` 也行。

`myconfig` 宏包的详细讨论我在 `XeLaTeX` 指南一书中会详细讨论。`mytitle` 宏包前面那部分是定义新的输入命令，第二行是用于处理可能出现错误的反应。就照这个模式写。然后 `gdef` 命令后面就跟著新定义的命令的形式，`gdef` 和 `def` 命令的不同是他定义的是 `global` 的，我估计之前我那种写法不行可能就是没有 `gdef`。后面就是基本的封面设计了。

特别提醒在 `Ubuntu` 下右键新建之后，是未命名然后空格然后 `tex`，这种文件名 `XeTeX` 处理似乎不很好。建议改成没有空格的文件名。

ubuntu 知识

通过 U 盘安装 ubuntu

1. 在安装之前请先把硬盘中的资料做一些调整，空出一个大于 20G 的硬盘做将来要安装 **ubuntu** 根目录的地方。然后还需要一个大约为你内存两倍的硬盘分区等下要作为 **linux** 系统的 **swap** 交换分区。
2. 到 **ubuntu** 的官网上下载系统的光盘映像。
3. 用 **ultroiso** 软件（其他具有类似功能的软件也行）（注意 **ultroiso** 最好选择最新的版本，**ubuntu10.10** 之后的一定要用 **9.3** 版本之上的）将该光盘映像写入到你的 U 盘中去。
4. 重启计算机，BIOS 稍作改动使计算机变成从 U 盘启动。
5. 进入安装过程，其他过程都比较直观，就是硬盘分区设置上我们选择高级手动，然后将你分出来的那个 20G 硬盘作为 / 的加载点，并设置格式化成为 **ext4** 日志系统（其他文件系统如 **ext3** 等也行）。然后设置交换分区，安装完成。

初入 ubuntu

- 设置 **ubuntu** 的 **root** 用户，这个以后可能会有用。按住 **ctrl+alt+t** 弹出终端，然后输入：
sudo passwd
设置好密码之后，以后输入：
su
就可以进入 **root** 账户了。

- 安装一个有用的工具：

sudo apt-get install nautilus-open-terminal

这个工具可以让你在视窗背景下鼠标右键之后打开当前目录下的终端，有时还是很快捷的，尤其是遇到一些怪怪的文件名的时候。

- 系统安装好之后第一件要做的事是选一个好的源，然后安装更新。在系统设置 → 软件和更新那里，在“下载自”的哪里就是软件源的服务商，你最好还是自己搜索一个速度最快的源。然后在终端中执行以下命令来升级系统软件包：

sudo apt-get update （更新源）

sudo apt-get upgrade （升级源下已经安装了的软件）^①

- 一些闭源的有用的东西：

sudo apt-get install ubuntu-restricted-extras

这个安装到 **adobe-flashplugin** 的时候会有点卡住，最好还是耐心点吧。或者在 **ubuntu** 软件中心（**Ubuntu** 额外的版权受限程序）中安装也是可以的。

- **ubuntu** 系统清理，运行以下两个命令即可：

sudo apt-get autoremove （清理软件残余）

sudo apt-get autoclean （清理缓存）。

- 删除通过 **apt-get** 安装的软件：

sudo apt-get remove 要删除的软件名

删除相应的软件配置文件：

sudo apt-get purge 软件名

- 长按 **super** 键或者说是 **win** 键会弹出一些 **ubuntu** 桌面常用的快捷键，比如说 **ctrl+win+d** 是显示桌面，按住 **win+f** 搜索文件，**win+a** 搜索程序等。

linux 命令介绍

\$ 和 **~** 当我们打开终端的时候，看到一个美元 **\$** 符号，如果我们输入 **su** 命令，然后进入 **root** 账户，看到开头有一个 **#** 符号，其中 **\$** 表示普通用户，**#** 表示现在是超级用

^① 如果有很多软件需要升级的时候推荐使用命令：**sudo apt-get dist-upgrade** 这样不容易出错些。

户。然后我们看到一个波浪号 ~，这个波浪号的意思就是当前用户的个人主文件夹，比如现在我这里 ~ 的意思就表示目录/home/wanze。

cd 一般 **dir** 或者 **ls** 之后列出该目录所包含的文件夹或者文件，然后执行如下命令：

cd 某个文件夹名

就进入这个文件夹了。如果 **cd** 一个文件就会报错。然后特殊的有：

cd ~ 回到个人主文件夹

cd . 其中点表示当前目录

cd .. 表示返回上一级目录

cd 某个目录，比如 **cd / etc** 那么就直接跳到系统的 / etc 目录下了。

cp 复制文件命令：

cp 要复制的文件 目标文件目录

which 查看系统某个命令的位置

touch 创建某个文件或者对已存在的文件更新时间戳

rm 删除某个文件，一些具体选项我就不罗嗦了，如果有不明白的请自己用 **-- help** 来查看帮助信息。比如这里的 **-r** 选项就可以强制删除一个目录下所有的文件。

mkdir 创建文件夹

rmdir 删除文件夹

mv 移动某个文件或者重命名某个文件

通过 ppa 安装软件

ppa 也就是软件源，一般推荐 **ppa** 安装软件即该软件还处于活跃开发期中，所以推荐尽快更新到官方最新的版本。过程就是添加 **ppa** 源，然后用前面讲的 **apt -get upgrade** 来更新源下的软件。添加 **ppa** 源的格式如下：

```
sudo add-apt-repository ppa: 后面就是什么官方源地址了
```

然后更新源下的软件列表：

```
sudo apt-get update
```

接下来就是通过 `apt-get install` 来安装该软件了。`upgrade` 是更新源下已经安装了软件，所以第一次装还是要用 `install` 命令。^①

软件推荐

以下编号：Ⓐ 表示可以从 `ubutnu` 软件中下载到，Ⓑ 表示可以从终端 `apt-get` 命令中下载到，Ⓒ 表示推荐从 `ppa` 安装。

新立得软件包管理器 这个有时安装一套软件组合很有用的。Ⓐ

chromium 不清楚和 `google-chrome` 的区别。它的插件很方便，`firefox` 也很好，对于配置低的机器更适合一些。Ⓐ

`chromium` 插件推荐：

Daum Equation Editor 本文暂时不会讨论数学公式的输入问题，这里有个简单的解决方案，就是在这个插件里面编辑公式，下面就有对应的 \LaTeX 的代码。

代理助手 好吧，我讲这个完全是因为天朝的防火墙，不学会翻墙严重干扰我们搜索信息。具体设置是 `http` 代理填写相关地址，`china list` 这个选项也勾上吧。下面还有详细说明。

翻墙软件 要搜东西真的要用 `google`，在天朝你懂的，因为要跑防火墙慢死了。轮子的自由门还可以。^②翻完墙之后到弹出来的那个网站或者 `voa` 网站哪里多下几个类似的翻墙软件吧，有备无患。`freegate` 在 `ubuntu` 下需要使用 `wine` 模拟，在模拟前先用 `winetricks` 装好 `mfc42.dll`，然后就行了。`freegate` 进去之后在通道哪里选择经典模式。`google-chrome` 那边的设置在代理助手哪里，就是使用 `freegate` 提供的那个 `http` 地址加上端口号即可——在内容那一页。一般代理助手就点系统自动连接吧，毕竟是免费代理，别想质量有多好。

texmaker 写 \LaTeX 非常好的界面环境，和 `windows` 下的 `texstudio` 类似，但是感觉界面更加的漂亮。Ⓐ

① [how-install-any-software-ubuntu-ppa](#)

② [下载 freegate 的链接](#)

shutter 一个非常好的截图软件。**ubuntu** 系统自带的截图软件截完图不能拖动改变大小，这个可以。④

wine 虽然 **wine** 软件一直在更新，但是老实说总感觉不是十分令人满意。这里提到 **wine** 是因为前面谈到的翻墙软件需要用 **wine** 来模拟。**wine** 的安装不同于其他软件，尽量下载最新版本的吧。到官网上查看最新的稳定版本号，比如 **wine1.6**。然后在终端上通过 **ppa** 来安装，**ppa** 地址见脚注。©①

workrave 人们对久坐的危害总是低估了，这个软件帮助提醒人们做一段时间之后最好起来活动活动。④

Gcolor2 这个小软件取色，RGB 转换等很有用。④

wps for linux 有时不得已还是要用 **doc** 格式，而在 **ubuntu** 下我试过很多解决方案，比如 **wine**，**libreoffice**，或者网上的在线编辑。结果都不尽如人意。因为我需要的是最好完完全全兼容 **microoffice2003** 的格式。金山公司的这个 **wps for linux** 目前还在开发中，但是效果还是可以了。下载地址见脚注。①②

其他还有很多有用的软件看官自己探索吧。

rime 输入法

更多信息请参考 [rime 官方 wiki](#)。还有一种 **ppa** 安装方法，**ppa** 地址是：[**ppa:lotem/rime**]

。

rime 输入法也可以在 **fcitx** 输入法框架下进行，就目前来说（2013-12-04）感觉 **fcitx** 输入法框架要比 **ibus** 稍微好一点点。

1. 安装 **ibus-rime**:

```
sudo apt-get install ibus-rime
```

2. **ibus** 输入框不见了:

```
ibus-daemon -drx
```

① **ppa:ubuntu-wine/ppa**

② **wps for linux**

3. 重启 ibus: 有时 ibus 会出现一些奇怪的问题, 在终端运行下面命令重启 ibus 也许会解决:

```
killall ibus-daemon
ibus-daemon -d
```

4. Rime 输入法的其他设置: 在输入框调到 Rime 的状态的时候, 按一下 F4, 就弹出一些设置选项。
5. 用 Rime 输入常见的特殊符号: 比如我要输入 *, 按下 shift 和数字 8 按键, 就出来一些选项供你选择。其他很多符号类似, 具体请参考官网上的说明。
6. Rime 输入法是架构在 ibus 框架之上的, 所以在系统输入法首选项哪里, 可以设置 Rime 候选项是横着显示或者竖着显示, 还有下面可以自定义显示的字体和字体大小。推荐用 sans-serif 字体。然后字体大小要比你输出之后的效果稍微大一点, 这样醒目些。

备份和还原

输入法用久了, 你的自造词还有你输入相关词的词频这些信息是非常有用的, 能够极大地提高你的工作效率。rime 备份的核心词是命令:

rime_dict_manager。你需要在 rime 的用户配置文件夹里面打开终端, 然后输入这个命令, 你就可以看到提示信息了。

首先输入 -l 选项, 你可以看到现在已经有的词典, 比如 luna_pinyin。然后用 -s 同步, 这样你就会看到一个 sync 文件夹, 里面放着快照文件。-b 是选择性是备份某一个词典。-r 是还原快照文件。-e, -i 是 txt 格式操作, txt 保存数据会有点损失。将用户 rime 配置文件夹整个复制到同步盘即可。

把 QQ 拼音上你的词库导入 rime

1. 在 windows 下在 QQ 拼音设置里面找到那个导出词库的命令。就是一个 txt 文件。
2. 将这个 txt 文件另存为 utf-8 编码
3. 进入 linux (以下 windows 用户原理类似) 找个在线的简繁转换网站, 如果你使用的是简体略过这步。然后转换成为繁体, 由于词库比较大, 建议你耐心点, 推荐笨笨

网站简繁转换，不容易卡死。实在不行就只好下载软件了。我的一万多条还是能够转换出来。

4. 打开 `libreoffice`，将前面的 `txt` 文件，复制，粘贴。然后会弹出一个选项，分隔符注意勾选空格。
5. 前三列有用，后面的意义不大，统统删除。
6. 按 `ctrl+f`，下面弹出查找于替换。输入，后面输入空格全部替换。输入' 后面输入空格全部替换。
7. 将第一列第二列位置互换，就是剪切粘贴操作。
8. 全部选择前三列，复制，粘贴到文本。
9. `rime_dict_manager -i luna_pinyin` 上面谈及的你创建的文本名字。
10. 注意上面要在 `rime` 用户目录下操作即 `.config/ibus/rime`。
11. 备份 `dict_manager -b luna_pinyin`

定制 rime 输出特殊符号

我希望通过 `rime` 输入法能够快速地将 `Unicode` 中的符号打出来。本文主要关注这个问题，其他定制请参阅官网 `wiki`。

`symbols.yaml` 文件的说明

为了文章精简，这一块内容大多删除了，详细信息请参见 `github` 上的那个网站。我在 `github` 网站上新建了一个项目。里面有我编辑的 `symbols.yaml` 文件。现在将我所做的修改工作简单说明如下：

前面是全角和半角部分，如果你有需要可以自己定制下，这里我略过。然后看到 `symbols:` 哪里，开始自己定制命令。前面那个代码我们看到了匹配是以 `/` 开始的，所以下面所有的命令都要以符号 `/` 开始，如果你想有其他模式修改正则表达式就可以了，比如说可以换成命令以符号 `\` 开始。

yaml 的缩进表达我不太清楚，实际上我不太喜欢那种缩进的语法。**yaml** 还提供了另外一种语法形式。就是一个系列可以用一个方括号明确标识出来。^①在这里格式很简单，就是建立了很多符号调出命令，对应的后选词选项用方括号包围起来。

然后还需要一提的就是 **yaml** 语言支持锚点，意思就是某一个变量，跟着 **&** 任意的名字，后面就可以用 ***** 那个名字来表示那个词条了，这个有点类似于取别名的意思。**anyway**，用这样的方法可以给你建立的命令取几个别名。

重新部署 rime

最后将上面两个文件放入用户 **rime** 配置文件夹哪里，然后删除 **default.yaml** 文件，然后运行

```
ibus-daemon -drx
```

等着 **rime** 重新部署完毕即可。

具体输入符号的命令

现在我在 **rime** 下输入 **/gcs**，就会弹出很多符号。类似的还有很多，就不一一介绍了。

ubuntu 美化

1. 不要 **unity** 环境，我一直都用 **unity** 环境，觉得自己还是有权发言了，真的不好。推荐安装 **gnome** 环境。我不喜欢什么桌面特效，用无特效环境，毕竟系统流畅才是最重要的东西。
2. 安装 **vareity** 软件自动换桌面，推荐在加点其他的网站壁纸资源。然后登录界面也选上^②。

① 请参看[这个网站](#)

② [vareity](#) 官网

3. 按住 **alt** 键右键点击 **gnome** 面板，然后优化。推荐把上面的设置到底部，因为上面的视角很重要。
4. 顺便谈谈快捷键，在系统设置 → 键盘 → 快捷键那里。你可以设置各种程序的快捷键。比如 **ctrl+alt+t** 是打开终端设置 **ctrl+alt+w** 打开默认网页浏览器等等。

去除登录界面的白点

如果你放一张非常漂亮的图片，比如 **variety** 可以设置，那么这些白点将会让你很不舒服。^①

通过运行下面的 **bash** 脚本，就是一行行输入进去即可，重启之后确实去除了。

```
sudo xhost +SI:localuser:lightdm
sudo su lightdm -s /bin/bash
gsettings set com.canonical.unity-greeter draw-grid false
exit
```

配置 L^AT_EX 编写环境

1. `sudo apt-get install texlive` (下载安装有名的 **texlive**)
2. `sudo apt-get install texlive-full` (下载安装 **texlive** 的各个包)
3. 在 **ubuntu** 软件中心中下载安装 **texmaker** 软件。
4. **ibus** 似乎在这里面有点小问题，有点不稳定。在终端中运行
`sudo apt-get install ibus-qt4`
 则问题解决。
5. 在 **texmaker** 的选项和配置 **texmaker** 里面^②，设置快速构建，点上用户自定义那一栏，然后输入如下命令：

① 参考了这个网站

② 现在 **texmaker** 已经加入 **X_YL^AT_EX** 编译命令。


```
xelatex -interaction=nonstopmode %.tex|
```

这是使用 **xelatex** 来对目标 **tex** 文件进行编译，而不是传统的 **latex** 或者 **pdflatex** 方式，之所以这样是因为多方对比之后，觉得其在字体处理方面是未来的趋势。

新宏包的安装

安装 **texlive-full** 之后，如果还遇到没有的宏包，可以先到 **CTAN** 官网上下载到这个宏包之后，然后将这个宏包解压到系统目录：

/usr/share/texmf/tex/latex 里面即可。

当然你也可以在另外一个文件夹里面，这里必须是你的主文件夹下，新建一个文件夹 **texmf**，然后里面新建一个 **tex**，然后再新建一个目录 **latex**，然后在这个 **latex** 里面放着你下载下来的宏包。具体比如说有一个宏包名字叫做 **config**，那么 **latex** 下面就是 **config** 文件夹，然后里面就是 **config.sty** 文件。你自己写的宏包扔进去一样有效。唯一要额外做的操作就是在 **texmf** 目录之下运行命令：

```
sudo texhash
```

让 **texlive** 把这个目录也加入搜索范围。

阅读宏包代码

你所接触到的 **latex** 宏包源码大多在目录：

```
/usr/share/texlive/texmf-dist/tex/latex
```

如果有需要的可以阅读这些代码来学习你正在的 **sty** 或者 **cls** 文件都是怎么编写的。其中 **base** 文件夹里面有 **book**，**article**，**report** 之类文档类型的定义，也许里面有些命令的原始代码是你需要的。

texmaker 技巧

自动补全命令

在菜单里找到用户自定义的 **customize completion** 也就是自动完成，里面加入你想要的命令。比如：

`\textbackslash` 然后点击 **add**，这样以后想输入显示命令前面的那个斜线的时候会方便点。如果括号里面加入 `@` 符号，那么就会出现类似系统自带命令 `\section{•}` 的那个黑点 `•`。

自定义命令

在菜单哪里用户自定义，你看到可以用户自定义命令，填好之后就是快速构建下面那些备用的 1:2:3: 哪里将成为有意义的命令。

保存你的设置文件

在选项哪里有保存设置文件的功能，主要是自定义命令，自动补全命令等可以保存下来。

用 rime 快速输出 L^AT_EX 命令

具体效果就是我按下 `/tex` 等命令，就会弹出很多 L^AT_EX 命令，这个也是修改 `symbols.yaml` 文件来达到的，也是在我新建的那个 [github 项目](#) 里面，我做了很多优化工作，有兴趣的可以研究下。^①

其他 T_EX 编辑器评测^①

emacs 安装 emacs24 之后，用 package manager 安装 auctex 宏包。然后一些基本的 latex 编辑器功能都有了。稍微熟悉下就行了。唯一遗憾的是没有那种左侧很方便的目录导航功能。然后 emacs 的各种快捷键让我压力好大。

winefish latex 太高端了，刚进去一片空白？

gedit gedit 安装好 latex 的插件之后也很好。因为 latexila 编辑器在编译大文件的时候速度好慢，而 gedit 是直接调用终端模式所以没有损伤的。值得一提的是 latexila

^① 虽然 texmaker 软件也有些小缺陷，但综合起来我觉得是最棒的。

编辑器的一些功能在 **gedit** 里面可以通过外部工具这个插件自己编写终端命令来实现。

texworks 没怎么评测。

kile 没怎么评测。

texstudio 没怎么评测

gummi 小的文档小的测试或者 **tikz** 制图是极好极好的。。但是不支持你自己定义的宏包。是我心目中的 **number two**。

latexila 这个软件刚进去我就感觉设计理念非常的简洁，和 **gedit** 一样。然后我特别在意的文档结构图显示，语法染色都有。⁽²⁾

gedit 的一些技巧简要说明

在编辑 → 首选项那里请安装好 **latex** 插件和外部工具插件，恩，代码注释插件和文件浏览器插件推荐。

我在这里重点介绍一下外部工具插件的用法。看到工具 → 外部工具。请选择最下面的那个管理外部工具，看到在此处打开终端，**gnome-terminal** 后面去除掉，然后就能正常工作了。⁽³⁾

现在我们新建一个命令，名字叫做 **xelatex**，具体内容如下：

```
#!/bin/sh
filename=$GEDIT_CURRENT_DOCUMENT_NAME
shortname='echo $filename | sed 's/\(.*\)\\.tex$/\1/'
xelatex -interaction batchmode -src $filename
```

那么这个小工具就实现 **xelatex** 编译功能了。

然后我们也可以再新建一个小工具，名字叫清理：

```
#!/bin/sh
filename=$GEDIT_CURRENT_DOCUMENT_NAME
```

```

shortname='echo $filename | sed 's/\(.*\)\.tex$/\1/'
rm -f $shortname.aux $shortname.ent $shortname.out
$shortname.lot $shortname.idx $shortname.lof
$shortname.ilg $shortname.ind $shortname.log
$shortname.toc $shortname.bbl $shortname.blg

```

这样运行它就可以清理临时文件了。

安装字体

找字体文件

如果你装了 windows 系统，那么你可以到 windows 下 copy 这些字体文件。比如 windows 常用的宋体，times new roman 等，在 C 盘的 windows 的 fonts 文件夹里面。本文用的就是 adobe 中文系列：adobe 宋体 std，adobe 黑体 std，adobe 楷体 std。很奇怪，在 pdf 上我觉得这几个字体感觉很好，但是在屏幕上就觉得不太好了。

放置字体文件

推荐都放在 ubuntu 的主目录的 .fonts 文件夹里面^①，如果没有请新建一个。这是通常默认用户新加字体放置的目录。当然你也可以放在其他目录里面，比如你的同步盘里面，然后用 font-manager 安装字体也是可以的。^②

命令行安装字体

运行命令：

```
fc-cache -f -v
```

字体就安装好了，如果你要看现在你的系统上有那些可用的中文字体，在终端运行命

① 这是一个隐藏目录

② 这是本文推荐的方式，安装卸载字体都方便些。

令^①:

```
fc-list :lang=zh | sort >ziti.txt
```

打开 `ziti.txt`, 里面就是你的可用中文字体的信息, 比如:

```
/home/wanze/.fonts/simsun.ttc:
```

```
宋体,SimSun:style=Regular
```

其中第一个是字体文件所在的目录, 第二列信息是可以调用的名字, 有宋体和 `SimSun`。

用 fontmanager 安装字体

你可以安装其他软件来安装和管理字体, 比如 `fontmanager`:

```
sudo apt-get install font-manager
```

这个软件查看安装卸载或者禁用某些字体都很方便的, 需要提醒的是这个软件占用了默认的用户配置文件 `fonts.conf`。然后你的字体 DIY 需要到 `~/.config/font-manager/local.conf` 哪里去设置。这个下面会讲到。

ubuntu 系统字体的配置

如果你不喜欢 `fontmanager`, 那么在 `Ubuntu14.04` 以后, 在 `.config/` `fontconfig` 文件夹里面, 有一个 `fonts.conf` 文件, 进行设置也是一样的。这里是文泉驿给出的网页自动生成脚本方案:

http://wenq.org/cloud/fcdesigner_local.html

不是特别难看的情况就没必要改动系统字体, 因为我们不要低估了人眼的适应能力。这里的配置主要是指由于系统升级带来的字体的改变, 特别是中文字体的改变, 手动配置将其固定下来。⁽⁴⁾

code:B-1

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<fontconfig>
```

^① | 表示 linux 命令中的通道, 第一个命令的输出信息流会流向 `sort` 命令, 排序之后重定向到 `ziti.txt` 文件里面。然后终端的数据就保存在这个文件里面了。

```

<match target="font">
<edit name="rgba" mode="assign"><const>rgb</const>
</edit></match>

<match>
<test name="lang" compare="contains"><string>zh</string></test>
<test name="family"><string>serif</string></test>
<edit name="family" mode="prepend"><string>微软雅黑</string>
</edit></match>

<match>
<test name="lang" compare="contains"><string>zh</string></test>
<test name="family"><string>sans-serif</string></test>
<edit name="family" mode="prepend"><string>微软雅黑</string>
</edit></match>

<match>
<test name="lang" compare="contains"><string>zh</string></test>
<test name="family"><string>monospace</string></test>
<edit name="family" mode="prepend"><string>微软雅黑</string>
</edit></match>

</fontconfig>

```

有很多内容没有深究，第一个是打开 **rgba** 模式，优化液晶显示的。然后下面就是对三大字族设置，如果是 **zh** 中文的话那么就 **prepend** 也就是插入微软雅黑，也就是在搜索队列中微软雅黑优先级最高。这个可以通过 **fc-match -s serif** 等来查看。然后其他字体设置不想涉及了，只希望他们能够稳定下来。感觉设置雅黑字体了，系统的主题换为 **Radiance** 更好看些。然后用 **font-manager** 针对微软雅黑高级设置加上 **AA** 和 **AH**。一个是反锯齿一个是自动粗细设置吧。就这样了。⁽⁵⁾

或者更简单的方法？

通过 `sudo apt-get install gnome-tweak-tool` 安装 `gnome-tweak-tool` 或者其他软件快速设置字体，不太清楚这种方法是不是中英文一起改动的。

ubuntu 的备份和还原^①

首先是利用系统自带的备份软件将 `home` 文件夹里面的一些内容备份好，注意配置文件排除法则。

备份步骤简介如下：

1. 备份你的 PPA，也就是你通过 PPA 装的软件。有的可能都没有通过 PPA 装过软件，那么这一步和下面的备份 PPA 的 `key` 都可以省略。命令如下：

```
sudo cp -r /etc/apt/sources.list.d ~/Nutstore/ubuntu-config
```

2. 备份你的 PPA 的 `key`:

```
sudo apt-key exportall > ~/Nutstore/ubuntu-config/myppakey
```

3. 通过新力得软件包管理器生成你安装的所有软件包的列表：选择文件 **F**→ 将标记的项目另存为 **A**→ 然后下面保存全部状态，不仅仅是变更选项勾上 → 然后选择一个地址取一个有意义的名字。

如果重装系统了，还原步骤相当于前面备份步骤的反向操作吧：

1. 还原你的 PPA: 这里就不写命令了，和上面类似也是 `cp -r`，不同的是现在将你保存的 PPA 还原到新系统的 `etc` 的 `apt` 目录下。
2. 还原你的 PPA `key`，假设对照上面的情况，我们有命令：

```
sudo apt-key add ~/Nutstore/ubuntu-config/myppakey
```

^① 以下主要参考了[这个网站](#)

3. 通过新力得软件包全部安装你之前保存的软件包列表：选择文件 **F**→ 读取标记的项目 **R**→ 然后类似正常的通过新力得安装软件包的步骤，点击应用即可。
4. 通过上面的步骤，你的新系统的所有软件全新安装好了，至于其他配置文件前面说了一些你在意的配置保存，然后选择相应的位置，一般在`.config` 文件夹里面，复制进去就行了。

更简单的备份？

当然你可以用 `luckybackup` 软件（ubuntu 软件中心里有）或者其他云盘将你觉得重要的资料重点备份即可，这取决于你的喜好了。

更多 ubuntu 知识

更多 ubuntu 知识将被放在 [指尖上的 Ubuntu] 一书中。

我又扫了一遍。这一部分进入冷处理状态，即除非特别的需要，不再新加入内容了。

注释

- (1) 在 `rime` 中文模式下，直接按 `enter` 键输出的也是英文，所以一些简单地命令直接输入也是很快捷的。
- (2) `latexila` 编辑器使用 `xelatex` 引擎生成文档，一样要自己编写一个生成命令，在创建 → 首选项那里 → 标签就写 `xelatex`，写其他的也可以。扩展名写`.tex`，然后命令和 `texmaker` 一样写上这个：`xelatex -interaction=nonstopmode $filename`。
- (3) 本信息来自 `ubuntu13.04`，`gedit` 版本号为 `3.6.2`。
- (4) 这个代码主要参考了[这个网站](#)
- (5) `chromium` 字体也都设置成为微软雅黑吧，然后我感觉页面稍微放大点更好看。

git 基础

本章参考资料有：1.[git - 简明指南](#)

2.[git howto](#)

3.[图解 git](#)

4.[git community book 中文版](#)

结合 github 进行项目管理

以下主要参考[这个网站](#)⁽¹⁾

到 **github** 上注册，创建新的项目，还有安装 **git** 软件（**ubuntu12.10** 自带的有）都很简单的，我就不多说了。下面就 **git** 命令使用的基本流程说明如下：

基本命令

远程仓库文件到本地

网上创建项目之后，你需要将网上的存档下载到本地，在你希望下载的地点，打开终端：

```
git clone https://github.com/a358003542/xelatex-guide-book.git
```

后面的这个链接地址在你创建的项目的右下角哪里，写着 **HTTPS clone URL**。

git init 命令用于本地创建的文件夹上传到远程仓库，**git clone** 下来的仓库文件已经索引了。⁽²⁾

本地仓库文件进入索引

下载下来的本地仓库文件进入 **git** 的索引，该文件夹内的所有文件都进入索引则在终端中输入如下命令：

```
git add .
```

因为我们在 **github** 创建项目的时候已经创建了一个配置文件，比如我选择的是 **latex** 语言，然后它会自动会处理将某些文件不上传。

如果你本地删除了文件，你希望远程仓库也删除这些文件，那么使用命令：

```
git add --all .
```

将索引中改动的文件提交到 head

不太清楚这个索引，**head** 具体是什么意思，**anyway**，过程就是这样的。

```
git commit -m '2013-08-25:19:00'
```

后面的文字串等下在 **github** 网站中会看到的，表示这个文件的标示符吧，你也可以取其他的名字，比如 **version0.1** 之类的。

将 head 中改动的文件更新到远程仓库

第一次你需要给你的远程服务器取个简单点的名字：

```
git remote add origin  
https://github.com/a358003542/xelatex-guide-book.git
```

然后以后都可以用这个简单的命令来更新了：

```
git push origin master
```

这里的 **origin** 的意思是远程服务器的简称，按理来说这个名字是可以随便取的，不过大家似乎都是取得 **origin**，然后你从 **github** 上 **clone** 下来的内容大多也是 **origin**，**master** 是远程仓库默认的一个分支，后面会讲到你可以创建其他的分支。

远程仓库的改动更新到本地

下面这个命令 `git` 对文件的操作是合并式的，也就是只是替换最新改动的文件。如果你希望远程仓库所有改动包括删除也更新到本地，使用可选项 `--all`。

```
git pull origin master
```

日常改动提交流程

一般情况下就用前面讲的的三步，`add` . `commit -m` 然后 `push`。这是基本的日常维护提交流程。

如果你在网站上对远程仓库做了一些修改，记得先用 `pull` 命令将远程仓库的改动更新到本地。

本地仓库管理文件丢失

如果你把本地仓库隐藏的 `.git` 文件夹删除了，但是本地的更改你又想上传到远程仓库，你首先需要 `git init`，然后添加远程服务器名字，`git remote add origin` 地址。然后建立本地索引，`git add --all`。然后 `commit` 和 `push`。这里可能远程服务器会拒绝，`push` 的时候加入 `-f` 选项会强制 `push`，但是 `github` 网页里面所有之前 `commit` 的记录都没有了，如何无缝对接我还不清楚。

安装

Ubuntu 用 `apt-get` 命令安装 `git` 即可：

```
sudo apt-get install git
```

还有些文档说要安装 `git-core`，Bergman 说这是一个 `dummy package`（仿制品）

①，删掉就用 `git` 也没事。

① [参考网页](#)

第一次使用的配置

设置你的名字和 email:

```
git config --global user.name "Your Name"
git config --global user.email "your_email@whatever.com"
```

这个是 **git** 的全局设置，和具体项目无关。你可以打开主文件夹的 **.gitconfig** 文件看一下。

初始化仓库

运行:

```
git init
```

即在当前工作目录下初始化 **git** 的管理仓库，如果你打开查看隐藏文件，会看到一个 **.git** 文件夹，**git** 用于管理当前项目的一些文件就存放在这里面。

如果你是在 **github** 上创建的项目，然后将这个项目克隆下来，那么就不需要再执行 **init** 命令，远程仓库已经执行了。

github 上删除项目

在项目网站右下角 **settings** 哪里进去有很多项目管理内容，其中最下面有删除项目的功能，请慎重使用。

远程仓库文件到本地

网上创建项目之后，你需要将网上的存档下载到本地，在你希望下载的地点，打开终端:

```
git clone theURL
```

上面代码的“theURL”就是你的项目的网页地址，在地址栏复制即可。此外还有什么 SSH 链接在 **github** 网页项目的右下角那里，一般使用就用 **https** 链接吧。

本地文件夹进入索引

下载下来的本地仓库文件进入 **git** 的索引，该文件夹内的所有文件都进入索引则在终端中输入如下命令：

```
git add theFilename
```

我们在 **github** 创建项目的时候已经创建了一个 **.gitignore** 配置文件，然后 **git** 会不索引某些后缀名的文件。这里推荐[这个网站](#)，你可以输入多个关键词，比如 **python** 和 **latex** 等，然后生成一个更加专业的 **.gitignore** 文件。

如果你本地删除了文件，你希望远程仓库也删除这些文件，那么使用加上 **--all** 选项，这样我们一般日常更新本地文件夹的索引常使用如下命令：

```
git add --all .
```

将索引跟踪的文件提交本地仓库

这里的 **commit** 命令提交是指提交给本地 **git** 的管理仓库，就在 **.git** 文件夹里面的一些内容。

```
git commit -m '0.01'
```

后面的文字串等下在 **github** 网站中会看到的，跟在文件名后面的，所以建议取简短一点。

本地仓库提交到远程仓库

第一次你需要给你的远程服务器取个简单点的名字：

```
git remote add origin theURL
```

上面代码的“theURL”就是你的项目的网页地址。

然后以后都可以用这个简单的命令来更新了：

```
git push origin master
```

这里的 **origin** 的意思是远程服务器的简称，按理来说这个名字是可以随便取的，不过大家似乎都是取得 **origin**，然后你从 **github** 上 **clone** 下来的内容大多也是 **origin**^①，**master** 是远程仓库默认的一个分支，后面会讲到你可以创建其他的分支。

远程仓库的改动更新到本地

下面这个命令 **git pull** 对文件的操作是合并式的，也就是只是替换最新改动的文件。如果你希望远程仓库所有改动包括删除也更新到本地，使用请可选项 **--all**。

```
git pull origin master
```

日常改动提交流程

一般情况下就用前面讲的的三步，**add**，**commit** 然后 **push** 即可，这是基本的日常维护提交流程。

如果你在网站上对远程仓库做了一些修改，记得先用 **pull** 命令将远程仓库的改动更新到本地。

本地仓库管理文件丢失

如果你把本地仓库隐藏的 **.git** 文件夹删除了，但是本地的更改你又想上传到远程仓库，你首先需要 **git init**，然后添加远程服务器名字，**git remote add origin** 地址。然

^① 也就是你从远程仓库 **clone** 下面的内容不需要在 **add origin** 这个属性了，直接 **push** 就可以了。

后建立本地索引，`git add -- all`。然后 `commit` 和 `push`。这里可能远程服务器会拒绝，`push` 的时候加入 `-f` 选项会强制 `push`，但是 `github` 网页里面所有之前 `commit` 的记录都没有了，如何无缝对接我还不大清楚。

git 进阶

查看本地仓库更改记录

```
git log
```

使用 **git log** 命令就可以查看本地仓库的更改记录。**git log** 有很多设置，这里有个最简单的例子（参考了 [githowtu](#) 网页）：

```
git log --pretty="%h | %s%d[%an]" --graph
```

--pretty=，控制输出格式：其中commit hash值的缩写；**%s**是 **commit** 的内容；**%d**是 **commit** 的其它信息备注；**%an**是 **commit** 的作者。然后**--graph**选项显示 **commit** 的分支树结构。

查看本地仓库目前的状态

```
git status
```

使用 **git status** 命令是查看目前仓库的状态，

设置快捷命令

git 的 **alias** 也提供了一定的命令别名功能，具体例子如下所示：参考了 githowto 的^①网页

```
git config --global alias.hist  
    'log --pretty="%h | %s%d[%an]" --graph'
```

这个命令具体是修改了主文件夹的 `.gitconfig` 文件，你可以打开看一下。

值得一提的是 **bash** 的 **alias** 命令来进行某些日常工作常用命令的封装也是很便捷的。比如把下面这行命令加入 `.bashrc` 文件中，以后就可以快捷地输入命令 “**gitpush**” 即可，其中 **commit** 的信息是当前的日期和时间。

一般日常提交就用这样简单的日期时间信息，如果有特别重大的版本推荐加上标签。

```
alias gitpush="git add --all . ;git commit -m '$(date +%F %R)'; git push origin master"
```

加上标签

git tag 命令是给某个重大的版本加上标签用的。加上的 **tag** 值可以作为 **checkout** 命令的索引值。

返回功能

版本控制软件最重要的一个功能就是能够返回到之前的文本编辑状态。

返回某个特定的版本

```
git checkout 3fbdf2c
```

^① <http://githowto.com/aliases>

具体 `commit` 的 `hash` 值可用 `git log` 查看之，这个命令将会将本地文件夹返回到某个特定的版本状态。

丢弃本地操作

```
git checkout .
```

如果你修改了本地的文件还没有 `add` 和 `commit`，然后觉得不满意想要将修改全部丢弃，那么可以使用如上命令，这样本地文件夹将会返回到最后 `master` 未修改时候的样子。

如果你修改了本地的文件，已经 `add` 了，但是还没有 `commit`，然后你想丢弃本地全部的操作，那么可以先：

```
git reset HEAD .
```

然后再如上 `checkout` 之。

以上例子都用了当前目录下所有的文件`.`的表示，具体操作某个单个文件你可以直接引用那个文件的文件名。如：

```
git checkout test.txt
```

然后就算已经 `add` 了，你可以直接 `checkout` 返回 `HEAD`。

```
git checkout HEAD .
```

然后如果已经 `commit` 了，那么返回到那个 `commit` 的那个版本即可，`git` 当然可以取消某个 `commit`，这里略过了。

分支功能

`tex` 文档管理并不太适合分支功能，分支功能某些大型项目编程更加适合。

emacs 入门

其实在你打开 **emacs** 开始输入文本的那一刻起，你就已经入门了。**emacs** 说到底只是一个功能特别强大的文本编辑器，它主要的角色还是文本编辑器，一切都是围绕这个来展开的。

最基本的快捷键操作

快捷键用的最多的是 **C** 和 **M** 这两个字母，其中 **C** 是 **Control** 键，而 **M** 一般是 **Alt** 键吧。然后“**C-w**”表示同时按下 **Control** 键和 **w** 字母。然后“**C-x C-c**”中间有个空格，表示先按什么，再停顿一下，然后再按什么。

S 表示 **Shift** 键，小写 **s** 表示 **Super** 键这个了解下，然后“**C-x 1**”表示只留当前一个窗口，“**C-x 2**”表示下面新开一个窗口，“**C-x 3**”表示右边新开一个窗口，“**M+x**”执行 **emacs lisp** 的内部命令，“**C-g**”取消你在 **minibuff** 中一般的输入，这些都是很基本的了，读者请记住。

lisp 语法简介

在进入接下来的讨论之前，先简单介绍下 **lisp** 的语法，这里不是要让读者学会 **lisp** 语言了，而是为了让你对接下来要接触的最基本的 **emacs lisp** 语法不会感到恐惧。

以下例子读者如果有兴趣可以进入 **elisp** 的交互环境试验一下，具体按键是：

M+x ielm

lisp 语言的一大特色就是里面的程序和数据都是列表，如

```
'(1, 2, 3)
```

这样的形式。

这个引号很关键，因为你输入给 **lisp** 的字符串都会被 **eval** 一次，然后前面加个引号，那么里面的东西 **eval** 一次之后就成 **(1, 2, 3)** 这样的形式了（即去掉引号），也就是列表数据了。

那么下面这个是什么呢？

```
x
'x
```

上面的 **x** 如果被送入 **eval** 的话，将会发生寻址操作，也就是成变量了；而第二个 **x** 加上引号，**eval** 之后，去掉引号，就是 **x** 这样的形式了，我们可以把这个 **x** 看作字符串，当然如果我们要将其送入 **eval**，那么你可以认为这个 **x** 字符串又变成变量了。

然后 **lisp** 中程序的结构就是一系列和数学结构类似的括号，学过数学括号结构的都知道，先算里面的再算外面的.....，直到最终算成这样的形式

```
(+ 1 1)
```

lisp 实际上也是类似的这样运算的，上面的 **(+ 1 1)** 送入 **lisp** 运算就会得到 **2**。**lisp** 处理这样的结构有一个原则，总认为这样的括号的第一个元素是一个函数，如果不是，将会发生错误。然后后面的东西都是一系列管他什么的参数。

然后再来讲讲设置变量和定义函数。

设置变量

设置变量有两个方法，一般为了方便使用的是 **setq** 吧。

```
(set 'x 1)
(setq x 1)
```

`set` 和 `setq` 的区别就在那个引号，`setq` 不需要加上那个引号，第一个元素的字符形式总被视作变量。

定义函数

最简单的形式如下所示：

```
(defun add (x y) (+ x y))
```

这样你就定义了一个 `add` 函数，然后 `(add x y)` 实际上就是执行的是 `(+ x y)`。

最简单的 `lisp` 语法简介就是这么多了，后面还会接触一些，当然要深入起来里面内容还很丰富的，但目前已经够我们继续探索下去了。

最基本的配置

首先是利用 `emacs` 的菜单来选择主题和其他相关配置，然后把你的设置保存好。这个我需要吐槽一下了，为什么 `emacs24` 了都还没有一个汉化方案，几个菜单汉化工作有那么难吗？就不能照顾一下新手啊。

好了进入主文件夹的 `.emacs` 文件^①，如果不出意外的读者已经会看到一个关于 **custom-set-variables** 的一些配置，读者用 `Enter` 键将其推到最下面吧，这些配置就放在那里，其内记录着你刚才的主题配置还有开始不要打开向导界面等等，最好还是留着。

加入 smex 插件

本文档不会讨论其他的插件了，但这个 `smex` 插件真的非常有用，就在这里作为 `emacs` 插件安装的一般例子在这里说明了。你以后自己编写的插件其实就是一个 `el` 文件，其内其实就是一些 `elisp` 语言的代码，等你熟悉 `elisp` 语法了，编写自己的插件还是很简单的事了。

^① 一个隐藏配置文件，每次运行 `emacs` 之前都会先加载它。

smex 的官方 github 站在[这里](#)，不过看了我下面的介绍大致也差不多了。

首先当然是 `git clone` 下来，很小的一个文件，就单独下载那个 `smex.el` 文件也是可以的，然后你将这个文件放在一个你喜欢的地方上，我放在 `~/工作空间/emacs/` 里面的。

然后在 `.emacs` 文件中加入如下代码：

```
; smex 插件设置
; 加入新的搜索路径
(add-to-list 'load-path "~/工作空间/emacs/smex")
(require 'smex); 加载某个模块
(global-set-key (kbd "M-x") 'smex)
(global-set-key (kbd "M-X") 'smex-major-mode-commands)
;; This is your old M-x.
(global-set-key (kbd "C-c C-c M-x") 'execute-extended-command)
```

下面详细讲解之。**add-to-list** 是一个函数，其接受一个列表变量，然后将后面的一个元素加入该列表中。请看下面的例子，其中 **nil** 是一个空列表的意思：

```
ELISP> (set 'x nil)
nil
ELISP> x
nil
ELISP> (add-to-list 'x 1)
(1)

ELISP> (add-to-list 'x 2)
(2 1)

ELISP> (add-to-list 'x 3)
(3 2 1)
```

这里的 **load-path** 变量放着的就是 `emacs` 加载插件时的搜索路径，这里把一个新的 `smex` 插件所在的路径加进来了，注意路径的下面就应该放着目标 `el` 文件。

以后你要加载新的插件“**what**”，大致也是类似的配置语句：

```
(add-to-list 'load-path "~/工作空间/emacs/what")
```

把 **what.el** 放入 **emacs** 文件夹里的 **what** 文件夹里面即可。

然后接下来就是实际的加载这个插件了：

```
(require 'smex); 加载某个模块
```

require 函数后面跟着插件的名字，对应的就是那个 **smex.el** 文件。

自定义按键配置

接下来是进一步的按键配置：

```
(global-set-key (kbd "M-x") 'smex)
(global-set-key (kbd "M-X") 'smex-major-mode-commands)
;; This is your old M-x.
(global-set-key (kbd "C-c C-c M-x") 'execute-extended-command)
```

这种自定义按键配置方式以后我们可以用来随意的配置后面学到的命令。

global-set-key 函数后面跟着 **kbd** 函数对按键字符的封装，可以把第一个元素看作具体的按键，第二个元素看作具体调用的命令。

这里第一个命令的意思是，如果你按下了 **Alt+x** 按键，那么将会触发 **smex** 命令。然后之前谈及 **Alt+x** 对应的命令实际上就是 **execute-extended-command**，而经过这样的配置之后，以后就需要通过这么复杂的按键才能激活了，等于打入冷宫了。

基本使用说明

那么 **smex** 插件具体有什么用处了，现在我们重新打开 **emacs**，然后按下 **M-x**，这个时候我们会看到很多选项，这正是 **smex** 插件的功能，跟输入法一样基本，所以很有用的。

基本的使用就是输入你想要的命令前面几个字母，**smex** 会给出提示，然后你按下 **Tab** 键即可自动补全，然后 **smex** 跟输入法一样会记录你的操作习惯。然后就是 **enter** 直接输入第一个候选项，方向键向右将第二个候选项变为第一个候选项等等，方向键向左反之，方向键上下使用来切换历史记录的。

基本使用说明就这么多，**smex** 使用久了，会跟输入法一样使得你输入某些命令速度会非常的快，当然你也可以选择自定义按键配置来绑定新的按键。

一些命令一览

set-default-font 设置默认字体

set-face-attribute 设置默认字体大小

以上两个命令一般放入`.emacs` 文件中，如下配置成为全局配置。

```
(set-default-font "DejaVu Sans Mono"); 设置默认字体
(set-face-attribute 'default nil :height 110); 设置默认字体大小 11pt
```

toggle-frame-maximized 窗口最大化你可以将其加入`.emacs` 文件让你的 `emacs` 一打开就窗口最大化。

```
(toggle-frame-maximized)
```

global-linum-mode 左侧显示行号

如下加入`.emacs` 文件中你可以让你的 `emacs` 左侧永远显示行号。

```
(global-linum-mode t)
```

global-visual-line-mode 显示的段落自动换行

如下加入`.emacs` 文件中你可以让你的 `emacs` 显示的文件段落会自动换行。

```
(global-visual-line-mode t)
```

help-with-tutorial 默认按键“`C-h t`”，打开新手教程

describe-key 默认按键是“C-h k”，描述某个按键

describe-function 默认按键“C-h f”，描述某个函数

describe-variable 默认按键“C-h v”，描述某个变量

通过以上三个函数你能够学习到 **emacs lisp** 的很多内部知识，是一条不错的获取信息的来源 [[redguardtoo](#)]。

比如之前的“C-x 1”等等按键对应的命令如下：

delete-other-windows 默认按键“C-x 1”，关闭其他窗口

split-window-below 默认按键“C-x 2”，下面新开一个窗口

split-window-right 默认按键“C-x 3”，右边新开一个窗口

delete-window 默认按键“C-x 0”，删除当前窗口

emacs lisp 还有很多很多内容，学习到后面当然是参看手册了。

DIY 之门

下面这几个命令也很常用，我们可以将其定义为更加简短一点命令

kill-buffer 默认按键“C-x k”，关闭当前 **buffer**。

switch-to-buffer 默认按键“C-x b”，切换 **buffer**。

find-file 默认按键“C-x C-f”，打开文件

save-buffer 默认按键“C-x C-s”，保存某个 **buffer**，或者说通常意义上的保存文件。

save-some-buffers 默认按键“C-x s”，保存所有 **buffer**，或者说保存所有文件。

save-buffers-kill-terminal 默认按键“C-x C-c”，退出 **emacs**。

你可以通过 **defun** 来定义新的函数，这里采用一种简单的机制 **defalias** 来给原函数取一个新的名字：

```
(defalias 'open 'find-file)
(defalias 'save 'save-buffer)
(defalias 'saveall 'save-some-buffers)
(defalias 'kill 'kill-buffer)
(defalias 'buffer 'switch-to-buffer)
```

然后我们再进入 **emacs**，我们发现 **smex** 里面多了一个 **open** 命令，之前没有的，然后原有的 **find-file** 命令也还在。

初步的试验已经有那么一点意思了，当然接下来所谓的 **DIY** 之门，就是编写自己的插件，这其中很大一部分就是编写自己的函数。**elisp** 语言还有很多内容，不过本章节就此结束了。

其他问题的讨论

TeX 一些常用命令简介

relax 命令

类似 python 的 pass 命令，什么也不做的意思。

TeX 中执行系统的命令

[参考了这个网站](#)

TeX 的 `writel8` 命令将一连串文本命令送入 `shell` 中执行，一般语句的格式如下（`immediate` 命令是立即执行的意思。这里表示将命令立即送入 `shell` 现在就执行。）：

```
\immediate\writel8{your command put it here}
```

要成功运行这个命令记得你的编译选项上还需要加上：

```
--enable-writel8
```

比如目前我的 `texmaker` 的编译命令如下：

```
xelatex -synctex=1 -interaction=nonstopmode  
--enable-writel8 %.tex
```

TeX 中写文件和读文件

[参考了这个网站](#)

TeX 中写文件的流程如下：

1. 首先新建一个待写的文件：

```
\newwrite\tempfile
```

2. 然后打开这个文件并给这个文件一个实际的文件名（支持带文件夹的相对路径名前缀）：

```
\immediate\openout\tempfile=test.txt
```

3. 然后用 `write` 命令给这个文件写入信息，只要文件没有关闭，可以一直附加的写：

```
\immediate\write\tempfile{this is interesting}
```

4. 写完之后记得关闭文件，这样写入的内容才实际从缓存区进入真实的文件区：

```
\immediate\closeout\tempfile
```

TeX 中读入文件一般用 `input` 命令即可，此外 `verbatim` 宏包还提供了原文照列的 `verbatiminput` 命令，还有 `fancyvrb` 宏包有 `Verbatiminput` 命令。

TeX 中的程序结构

条件控制语句

[参考了这个网站](#)

条件控制语句的首先要新建一个条件控制变量：

`\newif\iflogvar` 默认这个变量是设置为假，然后通过以下模式来建立条件语句：

```
\iflogvar
  aaaa
\else
  bbbb
\fi
```

条件语句可以嵌套，然后你可以通过这样的命令来调整条件控制变量的真假：

```
\logvartrue
\logvarfalse
```

ifnum 命令

`ifnum` 命令用于判断数字。

```
\ifnum number = 0
```

`ifnum` 控制结构和 `if` 命令类似，这里数字判断还可以是大于 `>` 或者小于 `<`。

TeX 生成前面有零的数字序列

[参考了这个网站](#)

TeX 中的计数器用 `arabic` 命令之后是 `1 2 3...` 这样的形式，如何产生 `001 002... 010 011 ... 100 ...` 这样的形式呢？之所以有这样的需求是我在试着编写 `endnotes` 宏包的时候用了一种错误的思路，然后希望使用 `linux` 系统的 `cat` 命令来讲一条条的尾注汇总，`cat` 命令对于计数器产生的文件名，如果按照原来 `1 2 3...` 这样的形式顺序是错误的，为了纠正就需要变成 `001 002 ...` 这样的一致规则形式。

```
\def\@entname{\jobname
\ifnum\value{@ShowEndnoteCounter}<10 0\fi
\ifnum\value{@ShowEndnoteCounter}<100 0\fi
\arabic{@ShowEndnoteCounter}}%
```

比如上面这个命令，就是我目前使用的 `endnotes` 宏包产生的临时 `ent` 文件，文件名如下：`xelatex 指南 001.ent xelatex 指南 002.ent` 等等。

这里的逻辑如下，首先假定这个计数器从 `0` 开始吧，小于 `0` 的情况不予考虑，不要视图做出什么情况都考虑到的完美程序，那只是浪费脑力罢了。如果计数器范围在 `0` 到 `9`，那么经过第二行将会生成一个 `0`，经过第三行将会生成另外一个 `0`，然后后面生成个位数的计数器 `1` 之类的。如果计数器范围 `10` 到 `99`，那么经过第二行不会生成 `0` 什么也没有，如是格式就有了 `010 011` 之类的。这种构造考虑程序自上而下的顺序，很是简单，比如你如果觉得一千之类的范围太小，那就再加上类似的一行。

文本中的上标还有下标

上标使用命令 `textsuperscript` 命令，下标感觉自己缩小点就差不多了吧。示例如下：

上标^{上标} 下标_{下标}

更常见的是化学分子式里面的上标和下标，推荐使用 `mhchem` 宏包，至于数学模式里的上标和下标自不必说了。

多张图片并列显示

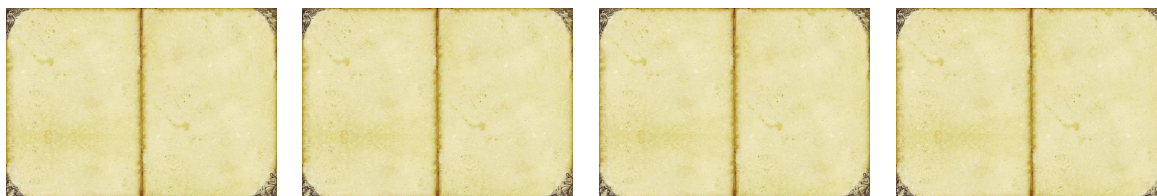


图 6-1: denosie fig

具体代码如下：

```
\begin{figure}[h]

\label{fig: 四栏图片}
\begin{multicols}{4}
\includegraphics[width=\linewidth ,totalheight=\textheight ,
    keepaspectratio]{temp.jpg}
i want some test to show there is a text and not to column break.\\
\centerline{test}
\columnbreak
\includegraphics[width=\linewidth ,totalheight=\textheight ,
    keepaspectratio]{temp.jpg}
\centerline{test}
```

```

\columnbreak
\includegraphics[width=\linewidth ,totalheight=\textheight ,
               keepaspectratio]{temp.jpg}
\centerline{test}
\columnbreak
\includegraphics[width=\linewidth ,totalheight=\textheight ,
               keepaspectratio]{temp.jpg}
\centerline{test}
\end{multicols}
\caption{denosie fig}

\end{figure}

```

这段代码的分栏还有插入图片知识都已经介绍了，值得一提的就是 **widetext** 环境，也就是有 **changepage** 宏包而来的临时改变页面布局环境和浮动体环境 **figure** 以及 **table** 不兼容。比如放入浮动体环境内才能起作用。然后 **caption** 命令似乎只是默认的 **linewidth** 居中。所以如果想要图片和表格在扩大的文本布局中居中对齐，那么需要在浮动体环境内部使用改变页面布局命令，然后使用居中命令。

生成字体所有已有的字形⁽³⁾

```

\documentclass[landscape]{article}
\usepackage{geometry}
\usepackage{fontspec}
\setmainfont{DejaVu Sans}
\usepackage{multicol}
\setlength{\columnseprule}{0.4pt}
\usepackage{multido}
\setlength{\parindent}{0pt}
\begin{document}
\begin{LARGE}
\begin{multicols}{5}

```

```
\multido{\i=0+1}{196607}{\% from U+0000 to U+2FFFF
```

% 后面的还会继续扩展，目前一般还没使用。

```
\iffontchar\font\i
\makebox[4em][l]{\i}%
\symbol{\i}\endgraf
\fi
}
\end{multicols}
\end{LARGE}
\end{document}
```

方便手机上观看的 pdf

原文档内容都不需要修改，只需要修改 **geometry** 的设置就可以满足要求，然后在手机上看的时候选择 **adobe pdf** 软件的连续观看功能会有更好的体验。具体配置如下：(4)

```
\ifphone
%for phone
\RequirePackage[
  paperwidth=105mm, % 除去旁註其他沒變,115, 再稍微小點
  paperheight=190mm,% 太長了縮短點,
  bindingoffset=0mm,% 裝訂線
  top=15mm, % 上邊距 包括頁眉
  bottom=15mm,% 下邊距 包括頁腳
  left=5mm, % 左邊距 or inner
  right=5mm, % 右邊距 or outer
  headheight=10mm,% 頁眉
  footskip=10mm,% 頁腳
  includemp=true,% 旁註寬度計入 width
  marginparsep=0mm, % 沒有旁註
  marginparwidth=0mm, % 沒有旁註
]{geometry}
```



```
\else
```

```
\RequirePackage[a4paper, %a4paper size 297:210 mm
```

```
bindingoffset=0mm,% 裝訂線
```

```
top=45mm, % 上邊距 包括頁眉
```

```
bottom=40mm,% 下邊距 包括頁腳
```

```
left=35mm, % 左邊距 or inner
```

```
right=40mm, % 右邊距 or outer
```

```
headheight=25mm,% 頁眉
```

```
footskip=25mm,% 頁腳
```

```
includemp=true,% 旁註寬度計入 width
```

```
marginparsep=0mm, % 旁註與正文間距
```

```
marginparwidth=0mm, % 旁註寬度
```

```
{geometry}
```

```
\fi
```

这里使用了一个条件命令。在加载 `myconfig.sty` 宏包的时候已经新建了一个条件变量：

```
\newif\ifphone
```

`\phonefalse` `ifphone` 就是一个条件判断命令，这里涉及到的命令我也不大懂，总之，如果我改成：

```
\phonetrue
```

就会自动生成适合在手机上观看的 pdf。

临时改变页面布局

临时改变页面布局前面讲的 `geometry` 宏包也有一种实现机制，但是不太好用而且会把后面的段落格式打乱。这里推荐使用 `changepage` 宏包。

进入 `adjustwidth` 环境就可以调整旁注宽度了。比如这里我新建一个 `widetext` 环境代码如下：

```
\newenvironment{widetext}
```

```
{\begin{adjustwidth}{-70mm}}%marginparwidth+marginparsep
```

`{\end{adjustwidth}}`

新建环境请看30.2。我们在这里建立了一个 `widetext` 环境，主要是 `adjustwidth` 环境操作，我们看到后面有两个花括号，第一个指左 `margin`，第二个指右 `margin`。注意这里不要和前面 `geometry` 宏包里面设置的 `includemp` 弄混了。^①这里左 `margin` 就是前面设置的 `left`，右 `margin` 是前面设置的：

`right+marginparwidth+marginparsep`。所以可以考虑 `geometry` 宏包不要设置 `includemp`，我前面只是为了理解简单才如此设置的，当然这里理解这个概念了就没有什么问题了。

然后正数表示 `margin` 宽度增大，负数表示 `margin` 部分宽度减小。这里设置 `-70mm` 即右边 `marginparwidth+marginparsep` 的宽度。然后自己注意进入这个环境了就不要使用 `marginpar` 或者其他旁注的命令了，这是显而易见的。不设置数值即表示不改变，出了 `adjustwidth` 环境，一切复原。

还有一个 `adjustwidth*` 环境，意思表示 `margin` 改变随着页面奇偶数变化而定，这个宏包的页面奇偶数设定及相关讨论后面都略过了，因为本文只关注于 `oneside` 模式，毕竟电子书籍设置成为 `oneside` 更好一些。

`changepage` 宏包还有 `changetext` 和 `changepage` 命令，有兴趣的可以看下，感觉并不太好用。

xverbatim 宏包说明

我新建了一个 `xverbatim` 宏包，提供了一个 `xverbatim` 环境，能够对代码进行染色显示输出，并对某些编程语言具有执行和显示输出结果的功能。这个环境还提供集中可选模式让实际操作更加灵活。由于这个宏包包括很多行 `pygments` (`python` 的一个宏包) 的输出配置信息，这里就不将宏包全部列出来了。完整的宏包内容在本 `github` 的 `texmf` 目录之下。使用 `pygments` 的灵感来自[这个网站](#)。

虽然整个宏包有两百多行，但主干结构也就一百行左右。具体实现结构和前面谈及的 `endnotes` 宏包有点类似，在编写的时候也参考了 James Brochie 的 `python` 宏包。

^① `geometry` 宏包里面设置 `includemp=true` 让 `marginpar` 部分进入 `width`，也就是 `right margin` 并不包含旁注部分了，但是只适用于 `geometry` 宏包。

正常运行这个宏包你需要安装 python 的 pygments 宏包，在 ubuntu 下运行：

```
sudo apt-get install python-pygments
```

或者 python3 的版本：

```
sudo apt-get install python3-pygments
```

首先我们看到程序的入口，定义 xverbatim 环境的这一段程序，xverbatim 宏包的其他部分都是围绕着这个环境命令展开的。

```
\newenvironment{xverbatim}[2][1]
{
\immediate\stepcounter{@xverbatimCounter}
\def\@modestatus{#1}
\def\@codefilename{#2}
\StrBefore{\@codefilename}{.}\@codefile]
\StrBehind{\@codefilename}{.}\@codemode]

\immediate\write18{
if [ ! -d codeexport/\arabic{@xverbatimCounter} ];
then mkdir -p codeexport/\arabic{@xverbatimCounter} ; fi}

\newcommand{\@outname}
{codeexport/\arabic{@xverbatimCounter}/\@codefile }

\immediate\openout\@codeout=\@outname.\@codemode
\newlinechar='15
\begingroup \catcode'\^M=12 %
\let\do\@makeother\dospecials\obeyspaces%
\@xverbatim}
{\endgroup
\immediate\closeout\@codeout
\ifnum \@modestatus = 0
\else \ifnum \@modestatus = 1
\@showcode
\else \ifnum \@modestatus = 12
```

```

\@showcode \@executecode \@writefeedback
\else \ifnum \@modestatus = 129
\@showcode \@executecode \@showresult \@writefeedback
\else \ifnum \@modestatus = 19
\@showcode \@showresult
\else
\errmessage{you typed wrong optional parameter,
the choice is 0 1 12 129 19 .}
\fi\fi\fi \fi \fi
}

```

首先新建 **xverbatim** 环境，然后接下来那么一大段都是 **xverbatim** 环境的定义。**xverbatim** 接受两个参数，一个可选参数 **#1**，只能接受 0, 1, 12, 129, 19 这几个数字，其他将会报错。然后还要接受一个必填参数 **#2**，（这里出于简单的美学考虑），这个参数接受等下要创建的文件的完整文件名。比如 **test.py**。这个文件将会放在 **codeexport** 目录的第几个例子计数文件夹里面。

这里先将 **@xverbatimCounter** 这个计数器加一，为的是它的第一次出现是 1。然后每次遇到一个 **xverbatim** 环境都会加一。

接下来定义了 **@modestatus** 命令接受参数 **#1**，为了将这个参数传递到环境的结尾部分。然后定义了 **codefilename** 接受第二个参数。

接下来的两行需要详细说明一下，这里使用的 **StrBefore** 和 **StrBehind** 命令来自 **xstring** 宏包，用处是从某个字符串中提取出某个字符前面的字符串或者后面的字符串。这里是处理 **@codefilename** 命令也就是接受的文件名，字符分隔符为 ‘.’，在 ‘.’ 的前面（默认是第一次出现，所以为了防止出错文件名只能出现一次 ‘.’）的字符串将会赋值给 **\@codefile**，下面的 **StrBehind** 命令类似，不同的时候将后面出现的字符串赋值给 **\@codemode**，这个后面方便 **pygments** 处理代码用的。

接下来的 **write18** 命令是直接向终端输送某个命令：这个命令首先是 **if** 开头表示进入条件语句，然后方括号表示某个条件判断语句，**-d** 表示判断某个文件夹是不是存在，**!** 表示逻辑否。分号表示换行，**bash** 命令是有一定的格式的，如果你需要一行表示某一个命令，可以用分号表示换行。**then** 表示条件语句进入如果判断真那么执行下面的语句，然后 **mkdir -p** 表示创建某个文件夹包括父目录。然后分号然后 **fi** 表示条件语句结束。整个

命令就是确保 `codeexport` 中的 `@xverbatimCounter` 命令的那个目录存在。

接下来新建一个 `@outname` 命令，这个命令等下要控制写入文件的所在目录和文件名的。

然后打开那个文件，其中文件名已经包含了目录，文件名和文件后缀。

接下来的几行可能有点多余，这是 `python` 宏包里面的，可能和写入文档有关，然后接下来的 `@xverbatim` 命令由前面那一系列符号转码规则确定，太难懂了，我也没弄明白，只知道它的作用就是将环境内的所有内容写入文件中，改动的话就是 `xverbatim` 这几个字对应环境。

然后我们看到 `xverbatim` 环境结尾部分，那个 `\endgroup` 命令让我不知所以，但是少了它程序就会出错。

接下来就是关闭文件，这个没什么问题。然后接下来就是有点稍显花哨的东西，其实就是前面 `@modestatus` 接受的那个参数，在这里构建了一个类似 `switch` 语句的结构。

首先如果 `@modestatus` 等于 0，那么什么也不做，在这里的意思就是既不显示代码也不做其他的。如果 `@modestatus` 等于 1，那么执行 `@showcode` 命令。我们再看到 `@showcode` 命令。

这里有个 `ifdefstring` 命令是 `etoolbox` 宏包提供的，意思是如果 `\@codemode` 等于 `tex` 的话，那么执行命令，否则执行下一个花括号里面的命令。这里主要是 `tex` 代码后缀和 `pygmentize` 命令生成的 `tex` 代码有点容易混淆，就单独开出来，区别只是文件名。

`pygmentize` 命令就是 `pygments` 提供的，它有很多选项，最常用的有：

表 6-1: pygmentize 常用选项

选项	说明
-o	输出到某个文件
-f	告知要输出的文件类型，比如 <code>tex</code> ，如果 <code>-o</code> 后面跟着的文件名带着后缀，命令会自动推测的。
-l	告知要染色的方案，比如 <code>-l python</code> ，就是 <code>python</code> 的染色方案，如果处理的文件带着后缀，命令会自动推测。
-O	全部信息，这里主要 <code>tex</code> 下 <code>Verbatim</code> 环境中的颜色支持命令。
--help	帮助信息

然后我们再往下看，如果选项是 12，那么将会执行

```
\@showcode \@executecode \@writefeedback
```

这三个命令。其中 `@writefeedback` 和程序如果发生错误并把错误信息写入文档有关，这里略过了。

而 `@executecode` 命令就是根据不同的 `codemode` 来选择不同的终端命令，这部分 `tex` 和 `c` 我有点考虑删除掉，最多支持 `python3` 和 `common-lisp` 和 `scheme` 和 `bash` 这几个脚本高级语言吧。

然后就剩下 `@showresult` 命令，值得一提的是我之前就是简单使用 `verbatiminput` 命令导入生成的结果文件，而在这里我则是使用的 `input` 命令，这是有深意的。因为我在探索 `sympy` 宏包的时候发现，这个 `python` 宏包使用 `latex` 命令可以直接生成 `tex` 文档下的数学模式下的代码，也就是我直接 `input` 将会显示很漂亮的数学公式。这里加了个 `group` 方便调整下 `input` 之后的一些格式，比如使用 `obeylines` 命令 `input` 的时候空格换行都将保留，然后取消了行缩进。

简要的说明就到此为止吧。哦，对了，你看到这个宏包前面有很大一段定义命令，这个是 `pygmentize` 对于 `Verbatim` 环境的颜色支持，通过选项 `-O` 获得的，复制到这里以后就不用引入了。

xverbatim 宏包的使用

xverbatim 宏包的使用主要是围绕著 xverbatim 环境展开的。首先我们来看一个例子：

```
\begin{xverbatim}[129]{py}
from sympy import *
a,b,c,x=symbols('a b c x')
ep1=diff(a*x**2+b*x+c,x)
ep2=integrate(a*x**2+b*x+c,x)
print('$'$ + latex(ep1) + '$$', '$'$ + latex(ep2) + '$$')
\end{xverbatim}
```

code:F-1

```
from sympy import *
a,b,c,x=symbols('a b c x')
ep1=diff(a*x**2+b*x+c,x)
ep2=integrate(a*x**2+b*x+c,x)
print('$'$ + latex(ep1) + '$$', '$'$ + latex(ep2) + '$$')
```

Traceback (most recent call last):

File "codeexport/code:F-1/code:F-1.py", line 2, in <module>

from sympy import *

ImportError: No module named 'sympy'

基本用法就是这样的，其中可选项 1 表示显示代码，2 表示执行代码，9 表示显示结果。除了 129 之外，还有的行为模式有：0，1，12 和 19。其中较常用的是 1，就是染色的显示代码。12 表示只显示代码和执行代码，19 表示只显示代码和显示结果（这个在之前已经执行了有用。）总的来说常用的模式有 0 什么都不做，1 显示代码和 129 模式。

后面跟一个可选项表示目标程序语言的后缀，可执行代码的目前只有：py，lisp 和 scm。如果只是想显示代码的话，后面可以跟的代码后缀对应的染色方案就有很多了，比

如: `c`, `cpp`, `tex`, `html`, `php`, `java`, `pl` 等等, 更多具体细节请参看文件夹 [自建宏包的演示和说明] 里面的内容。

注释

- (1) 在项目网站右下角 **settings** 哪里进去有很多项目管理内容, 其中最下面有删除项目的功能, 请慎重使用。
- (2) 参考了[这个网站](#)
- (3) 主要参照了[这个网站](#)
- (4) 主要参考了[这个网站](#)。

参考文献

参 考 文 献

[lshort] 有名的《一份不太简短的 \LaTeX 介绍》，原版作者：Tobias Oetiker, Hubert Partl, Irene Hyna 等，版本：2001-08-09，中文翻译：CTeX 用户小组，版本：2002-05，pdf 下载链接：

<http://www.ctan.org/tex-archive/info/lshort/chinese>。

[latex123] 《大家来学 \LaTeX 》，原版作者：李果正，版本：2004-03-08，网站链接：

<http://edt1023.sayya.org/tex/latex123/>。

[wikibook-latex] 维基图书 latex 篇英文版，版本：2013-08，网站链接

： <http://en.wikibooks.org/wiki/LaTeX>。这本书甚至都没有 cite 它一下，因为不知道插在哪里，几乎到处都有对它的引用，谢谢所有 wikibook 的编写者。

[boxes] using boxes and glue in \TeX and \LaTeX ，原版作者：Nelson H. F. Beebe，版本：2009-03-28，pdf 下载链接：链接地址不好复制，请 google 搜索书名之。

[latex-companion] The Latex Companion

[org-guide] 中文翻译网站，org 模式官方文档链接。

[redguardtoo] [mastering-emacs-in-one-year-guide](#)