zLaTeX Series Introduction

Eureka

2024 年 6 月 18 日

目录

第一部分

Document

CHAPTER 1

Introduction

1.1 简介

1.1.1 为何叫 z?

也不知道为什么这个系列名称要加以 'z' 的前缀,可能是因为个人爱好,或是因为觉得这个字母对自己而言有着一些别的意味。最开始此系列中此包含一个基本的文档类，叫做 πLaTeX，但是后面自己想开发一个用于绘图的宏包，主要基于 TiKZ. 用于常见平面图形的绘制以及外部程序的交互. 也许是看到了 `tikz` 库名称中的 "z"，于是便以 'z' 为前缀，产生了 zLaTeX 系列。

1.1.2 项目地址

目前本项目已经在 GitHub, Gitlab, Gitee 上开源，地址如下:

- GitHub: https://github.com/zongpingding/ZLaTeX_ZTikZ
- Gitlab: https://gitlab.com/zongpingding/ZLaTeX_ZTikZ
- Gitee: https://gitee.com/zongpingding/ZLaTeX_ZTikZ

项目中包含 zLaTeX 文档类源码 `zlatex.cls`，zTikZ 宏包源码 `ztikz.sty`，以及二者的说明文档. 后续在开发过程中，可能会保证 Github 的同步更新，至于 Gitlab 与 Gitee 则不一定会同步本系列的最新版.

zLaTeX 系列源代码完全开放，欢迎各位对源代码的修改以及二次分发. 如果想要和我一同改进此模板，请在 Github 提 Issue 或者是 PR. 不要在 Gitee 或者是 Gitlab 上提问，本人只维护 Github 上的仓库，尽管有时可能会为了国内用户下载方便，把 Github 上的仓库同步到这两处.

作为一个完全免费 (为爱发电) 的项目，我不对任何本模板的使用者负责，如果使用者在使用后遇到任何的严重后果，我不负任何责任. 我很乐意给大家解决问题，但是在提问前请先了解 LaTeX 提问规范，一起营造一个愉快的讨论氛围.

想要体验本模板请到 Github 仓库:Release 界面下载对应的最新模板. 由于本模板现在正处于早期开发阶段，所以很多的接口并不稳定，不保证模板的向后兼容性, 请各位见谅.

1.1.3 基本组成

本系列目前包含以下的三个组成部分，一个文档类，一个 beamer 宏包以及一个绘图库:

- zLaTeX 文档类
- zTikZ 宏包
- zslide beamer 宏包

其中前者主要用于指定排版文档的基本属性，zTikZ 宏包主要用于绘图[1]，最后一个 zslide bemaer 宏包是自己设计的一套 beamer 主题。尽管 zLaTeX 本身也可以在加上 `layout/slide` 选项后称为一个演示文档，但是在严肃的场景下，还是推荐使用 beamer 文档类.

其实从这个介绍文档就可以看出，本模板是十分的朴素的，没有十分华丽的色彩和精美的页面布局，但是在折腾了许久的 LaTeX 之后，现在这个模板才是最对我胃口的；至于，是否适合你，那就不得而知了。你可以去使用更加精美的模板，比如 ElegantLaTeX, BeautyLaTeX 等优秀的模板.

注记 1.1.1 后续可能还会开发一个 zTool 宏包, 作为 zLaTeX 系列的补充.

1.1.4 使用教程

本文档中的章 二一般的使用者可以跳过，这一部分主要是我自己对本模板的设计思路和关于写 LaTeX 的个人感受, 对于使用模板和宏包而言，没有任何的帮助. 如果要了解 `zlatex.cls` 文档类的使用，请直接跳转到 “zLaTeX 文档类教程” 文档:zlatex_manual.pdf. 如果要了解 `ztikz.sty` 宏包的使用，请跳转到 zTikZ 宏包教程文档:ztikz_manual.pdf. 目前 zslie 还没有对应的文档说明，仅有一个示例: zslide_manual.pdf.

[1] 众所周知的，在 LaTeX 中绘图是一件十分痛苦的事情，于是乎你会看到很多书籍或笔记中的图形都是手绘或者是截图，并非矢量图

CHAPTER 2

模板设计

2.1 设计历程

本模板的设计经历了相当长的一个周期，从最开始的初始 LaTeX，我把自己常用的宏扔到了一个.sty 文件中，以为这就是一个宏包了；之后了解到了ElegantLaTeX系列模板，也使用这个系列中的 book 文档类写了一点自己的笔记，但是用了一段时间之后总归是不满意，很多地方都想要自己定制，不喜欢模板默认的样式；奈何自己当时的水平不够，打开模板，看到的就是一堆的乱码。但是，后来也知道了有知乎上的优秀文章，所以就去看这些文章，慢慢的积累，渐渐的对 LaTeX 熟悉了一些，于是就着手设计属于自己的模板。

第一版的 zLaTeX 其实是完全仿照 ElegantLaTeX 的 book 文档类，然后一步一步的慢慢加东西，进行一些简单的修改，比如字体，颜色等等。但是写到后面，发现这个代码的的结构太不好控制了[1]. 尤其是其中的模板语言切换，那个\ifdefstring 语句写起来是极其痛苦的。下面就是当初写的代码片段:

```
\DeclareVoidOption{cn}{\kvs{lang=cn}}
\DeclareVoidOption{en}{\kvs{lang=en}}
\DeclareStringOption[cn]{lang}
```

再加上当时的基本文档类是 article，很多 book 文档类的内部计数器和章节命令都没有，需要自己去声明；但是结果往往是自己设计的命令和别的宏包还不协调，冲突. 其中最重要的就是 hyperref 宏包了，初代模板中它的跳转功能是不正常的，由于自己定义的计数器不正确，在使用\label 命令时，激活的章节元素 (跳转位置) 根本不对。当初的目录结构也是自己设计，但是也有着同样的跳转为题. 初代 zLaTeX 文档类全部采用 LaTeX2ε 进行构建，很多的宏展开的地方都写的很繁琐，而且大部分的实现方案都是在 TeX-StackExchange 上找到的，很多时候都是处于一种能跑就行的状态，并不知道其背后的原理.

2.1.1 zLaTeX

后来自己便把 zTiKZ 从中 zLaTeX 文档类中剥离出来，同时使用 LaTeX3 对原始文档类和 zTikZ 进行重构. 其中 zLaTeX 文档类继承自 book 文档类，之后几乎所有命令几乎都自己书写，知道它们的具体作用，对其他的宏包的影响。于是 zLaTeX 系列就诞生了. 果然，在使用 LaTeX3 对原始项目进行重构之后，整个项目的代码清爽了许多，比如下面的 zLaTeX 文档类选项声明:

[1] 其实最开始这个 zTikZ 宏包和 zLaTeX 是一体的，当时的代码是极其混乱的

```
\zlatex_define_option:n {
    % language
    lang                .str_gset:N   =  \g__zlatex_lang_str,
    lang                .initial:n    =  { en },
    % page layout
    layout              .str_gset:N   =  \g__zlatex_layout_str,
    layout              .initial:n    =  { twoside },
    % margin option
    margin              .bool_gset:N  =  \g__zlatex_margin_bool,
    margin              .initial:n    =  { true },
}
\ProcessKeysOptions {zlatex / option}
```

但是后面发现这样还是不够的简洁与清晰，主要的问题就在于如果你需要加载的子文档类的选项比较多时，你需要声明许多这样的 key-value，当整个文档类的 key-value 声明的过多时，模板便会变得难以维护。于是我引入了 l3keys2e 中的**元键** (`.meta:nn`) 用于将所有的 key-value 进行一个层级的划分，方便以后的模板维护. 目前的文档类键值对接口如下:

```
\zlatex_define_option:n {
    % zlatex language
    lang            .str_gset:N   =  \g__zlatex_lang_str,
    lang            .initial:n    =  { en },
    % class and options
    class           .str_gset:N   = \g__zlatex_subclass_type_str,
    class           .initial:n    = { book },
    classOption     .clist_gset:N = \g__zlatex_subclass_option_clist,
    classOption     .initial:n    = { oneside, 10pt },
    % zlatex options meta key
    layout          .meta:nn      = {zlatex / layout}{#1},
    mathSpec        .meta:nn      = {zlatex / mathSpec}{#1},
    font            .meta:nn      = {zlatex / font}{#1},
    bib             .meta:nn      = {zlatex / bib}{#1},
    ...
}
```

同时声明的 `<classOption>` 可以轻松简单的处理子文档类的选项传递问题.

2.1.2 zTikZ

对于宏包 ztikz.sty 的开发也是经历了一个相当漫长的周期. zTikZ 也从最开始的一个大宏包变成了一个个的小的子模块: cache, python, gnuplot, wolfram 和 zdraw. 这些模块通过如下命令:

```
\ProvidesExplFile{ztikzmodule.cache.tex}{2024/06/15}{1.0.0}{cache~module~for~ztikz}
```

进行声明, 然后在主宏包 ztikz.sty 中声明如下命令来进行模块的调用:

```
\cs_new_nopar:Npn \g__ztikz_load_module:n #1 {
  \clist_map_inline:nn {#1} {
    \file_if_exist_input:nF {modules/ztikzmodule.##1.tex} {}
  }
}
\NewDocumentCommand\ztikzLoadModule{m}{
  \g__ztikz_load_module:n {#1}
}
```

在处理好上述的模块划分以及接口声明后，对应的宏包使用者便只需通过:

```
\ztikzLoadModule{cache, python}
```

进行相应的模块的单独调用. 而且, 在划分好模块后，不但可以方便宏包的使用者，更让我可以对不同的模块进行单独开发，我认为这大大的提高了开发效率.

2.2 设计参考

本系列从诞生之初便由我一个人一直开发，在开发过程中参考了诸多优秀文档类/模板，参考最多的 CTEXart 文档类,几乎是本项目的大部分代码思路来源。此文档类完全采用 LATEX3 语法写成,本文档类中的**选项配置**模块主要参见 TEX-StackExchange 上的讨论,采用了 LATEX3 的 l3keys2e 模块；这样的好处有: 选项配置简洁，符合用户习惯，模板维护方便.

2.3 设计原则

其实这个标题有一点太大了，什么是设计原则，我也不知道，但是我就只是想让我的模板看着舒服。怎么才能让自己的模板看着舒服呢? 我也不知道，但是我觉得肯定和页边距，字体大小，字体样式等的有关。并且这三者一定是相互影响的.

比如你的页边距变大之后, 压缩了你的版心大小, 那么此时你的正文字体一定得做相应的改变. 那么一行多少个字合适呢? 去查了一下 TEX.SE, 针对于英文, 一行的字母个数在 65-90 是比较合适的，并且字体尺寸一般为 `10pt,11pt,12pt`；页边距到底设置多少呢? 自己去比对了 EleganLATEX 和其它模板的页边距 (就差用尺子量了); 好歹后面发现了一个宏包，可以在生成的 PDF 中查看页面布局尺寸等信息, 这个宏包就是 `fgruler`, 使用语法也是很简单的, 如下:

```
\usepackage[hshift=0mm,vshift=0mm]{fgruler}
```

当你在导言区引入之后，便可以在你的每一个页面的看到如图 2.1的效果，这样就不用打印出来用尺子量了.

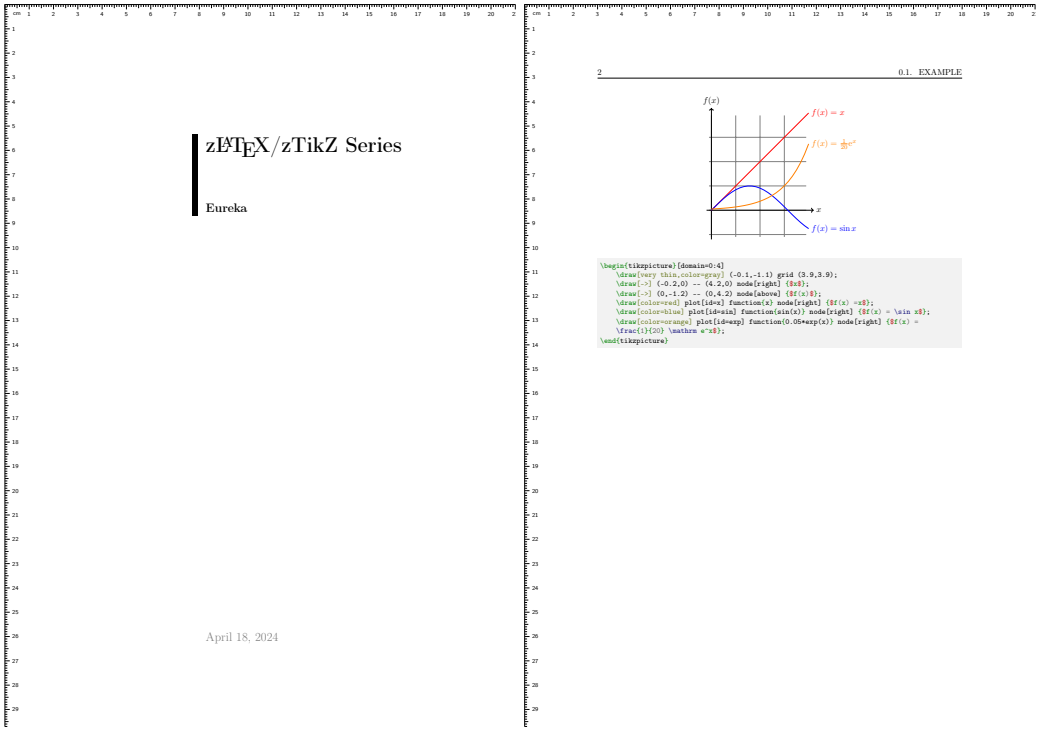

图 2.1: 页面布局示意图

在设计本模板的时，我也一直在纠结字体的问题，我应该把字体打包进入模板吗？或者是我应该在模板中给用户进行默认的字体设置吗？在这个系列的上一版中我就去找了一些免费的中文字体和西文字体，直接放在模板的文件夹下，但是这样产生的问题就很多了:

- 用户需要这个字体吗, 增加的字体会变成这个模板的负担吗?
- 这个字体真的免费吗?
- 中文字体的字形往往是不全的, 怎么解决?

于是最终的办法就是，我的模板不负责字体的设置，不添加任何和字体相关的配置，所有的字体由用户指定.

最后参考这这些标准，一步一步的调整，使得整体的页面布局稍微的合理些. 在设计这个模板时，还要考虑行距等各种元素。但是设计一个模板，你考虑的还不只这些，反正就是，如果你不会的话，那么就一切保持默认:Be simple, Be fool.

2.4 无题

时至今日，再次回头来看我的这个模板，我反而有了一些其他的感受. 一个模板到底需要给用户定制什么东西？到底需要给用户多大的自由空间 (配置选项)？如果你的配置选项过多，

像koma-script, Memoir那样处理很多的细节，提供种类繁多的接口. 还是像部分简单的模板仅提供几个必要的选项而已; 如果一个模板的说明文档都达到了上百页, 我作为一个用户为什么自己学习做模板，然后写一个有针对性的，适合自己的模板呢? 如果模板的配置选项过少，那么用户又会觉得这个模板不够灵活. 所以，到底什么样的一个模板设计才能够称得上是:简单, 灵活, 易用? 这个问题就只能留给使用者回答了...

zLaTeX 系列写到今天，它已经不再是一个简单的文档类 + 绘图库了，所以这个系列可能并不适合新手. 同时我也意识到，很多时候其实我们并没有必要写一个模板出来，你需要什么样的功能，你就去找什么样的宏包，然后去调用它，根据它提供的命令实现自己的需求. 这样会更加的方便，而且你也不用去考虑太多的细节问题, 更没有必要去花费那个你在使用别人模板时阅读模板细节的时间. 似乎，有了基础文档类 article, book+ 各种功能宏包之后的 LaTeX 就已经的足够好了，并不需要我们这些闲来无事人写的模板? 所以我反而觉得，我们这些人更应该把多余的时间花在基础宏包的开发上，就像 l3draw 宏包: 我只提供基础的几组底层绘制命令，至于上层的封装，那就交给用户去实现吧. hahaa >_<

CHAPTER 3

敬告

3.1 兼容性

目前本系列已经实现 Windows 和 Linux 下的兼容; 但是 MacOS 下: 目前仅支持 zLaTeX 文档类. zTikZ 还未进行适配 (参见下文了解具体原因), 所以不保证本系列中的 zTikZ 文档类可以在 MacOS 下正常运行. 具体的兼容情况请参见后续的兼容性章节.

3.2 说明文档

3.2.1 编译

本文档对应的源文件可以在 Github 仓库下载, 如果想要编译此文档请遵守如下步骤:

- 首先清除之前的编译文件, 比如`.aux`, `.log`, `.toc` 等文件以及 `ztikz_output/`文件夹。
- 使用 `xelatex` 编译此文档,编译两次。(如果第一次编译报错 `missing \begin {document}...`) 那么请注释掉主文件 `zlatex_ztikz_doc.tex` 中和 `indextool` 相关的两行语句:`\makeindex []{}`, `\printindex []{}`, 然后再次编译. 如果你想要生成索引, 请取消注释上一步中的两行语句, 然后再次编译.
- Want Build From Scratch? 那么需要本地环境中有配置好的: WolframScript, gnuplot, Python, Sed.

3.2.2 复制样例

本文档中给出了相当部分的样例及其对应的代码, 在书写本文当时为了读者的阅读体验, 对代码抄录环境中的部分符号进行了重写。比如你会在代码中看到换行符为:↪, 那么在复制此环境代码时, 请删掉此符号。亦或者是源代码中有行号, 那么在复制后, 请删掉多余的行号.

3.2.3 键值参数

本系列中的大部分命令均采用键值对的形式进行调用, 所以如果一个命令的可用键太多, 那么此时我并不一定会在正文中全部说明其可用键。我会在对应的命令下方插入一个源代码抄录, 用于说明此命令对应的声明原型, 其中就包含了此命令可用的键值以及不同键的默认值.

比如如下的命令:

```
% key-value setup
\keys_define:nn { ztikz / polygon }{
    radius       .fp_set:N  = \l__polygon_radius_fp,
    radius       .initial:n = { 1 },
    edgeColor    .tl_set:N  = \l__polygon_edge_color_tl,
    edgeColor    .initial:n = { black },
    fillColor    .tl_set:N  = \l__polygon_fill_color_tl,
    fillColor    .initial:n = { white },
    fillOpacity  .fp_set:N  = \l__polygon_fill_opacity_fp,
    fillOpacity  .initial:n = { 0 },
    rotate       .fp_set:N  = \l__polygon_rotate_angle,
    rotate       .initial:n = { 0 },
    shift        .tl_set:N  = \l__polygon_shift_tl,
    shift        .initial:n = { (0,0) },
    marker       .tl_set:N  = \l__polygon_marker_option_tl,
    marker       .initial:n = { },
}
% command
\NewDocumentCommand\Polygon{ O{}m }{
    \group_begin:
    \keys_set:nn { ztikz / polygon } { #1 }
    ...
}
```

上述的\Polygon 命令即表示：此命令的第一个参数为一个可选参数 (O 类型)，对应的选项设置为键值对. 可用的键有:radius, edgeColor, fillColor, fillOpacity, rotate, shift, marker 等等. 其中 <radius> 表示可以接受一个浮点数 (\fp _set:N), 默认值为 1(.initial:n = { 1 }); 再比如第二和键 <edgeColor> 表示可以接受一个 TokenList(\tl _set:N), 默认值为黑色 (.initial:n = { black }). 其余类似的键不在说明.