

# peterlitsdoc文档类

Peterlits Zo

2020 年 6 月 6 日

## 目录

<b>1</b>	<b>前言</b>	<b>2</b>
<b>2</b>	<b>peterlitsdoc 所提供命令</b>	<b>2</b>
2.1	pltpara 命令和它的快捷命令 - 带标注的段落 . . . . .	2
2.2	pltendant 命令 - 文末的作者信息 . . . . .	3
2.3	pltrun 环境 - 显示 L <sup>A</sup> T <sub>E</sub> X 代码 . . . . .	3
2.4	pltpic 命令 - 显示图片 . . . . .	4
2.5	plttodo 命令 - todo 格子 . . . . .	4
2.6	pltbox 环境 - 表格, 或者选项 . . . . .	5
2.7	plttodoenv 环境 - 简单的 todo 列表环境 . . . . .	5
2.8	pltplan 环境 - 时间表 . . . . .	6
2.9	plttimeline 环境 - 时间线 . . . . .	6
2.10	颜色 . . . . .	7
<b>3</b>	<b>peterlitsdoc 的更改</b>	<b>7</b>
3.1	边距 . . . . .	7
3.2	代码摘录 . . . . .	7
3.3	列表 . . . . .	8
3.4	代码环境 . . . . .	8
3.5	默认宏包 . . . . .	8
3.6	编译环境 . . . . .	9

## SECTION 1

## 前言

出于一些原因，我开始使用 markdown 来写文档，然后使用 pandoc 来转换为 PDF 文档。

但是转换成的 PDF 文档格式有点不尽人意。后来为了更棒的样式，我开始使用 Python 写了一个 filter 来操控中间层 json 数据。

后来 python 的 filter 的数据越来越多，比如我在 meta data 区中写下了很多我平时会用到的短命令（比如，添加图片，带标注的段落等等），pandoc 会把 code block 设置为单独的一个段，我还递归修改让它依附上一个段，来让它有着合理的前后间距。

那为什么不直接一开始就写 L<sup>A</sup>T<sub>E</sub>X 文档呢？因为我一开始觉得 L<sup>A</sup>T<sub>E</sub>X 的语法好傻哦，后来又看了看 Lisp 才慢慢看到 L<sup>A</sup>T<sub>E</sub>X 的美感，于是之后我把写的 filter 的程序中可能会用到的命令和繁琐的设置全部都放到这个文档类中。

这个文档类的目的是写一个面向中文使用的漂亮的小文章。

这个文档类的设计哲学是：命令具体，尽可能覆盖住需要的部分，在它的 view 层，需要做到简洁具体。不应该太过于花里胡哨。

欢迎在 <https://github.com/PeterlitsZo/peterlitsdoc> 中提交 issue 来让我添加一些有用的功能，或者下载相应的 peterlitsdoc.cls，或者提交 pull request 来改进（欢迎～）。

基于 GPL 发布。

## SECTION 2

## peterlitsdoc 所提供命令

## 2.1. pltpara 命令和它的快捷命令 - 带标注的段落

使用 `\pltpara{name}{content}` 来做一个 peterlitsdoc 风格的带注释自然段。

```
\pltpara{测试}{测试}
```

**测试** 测试

还定义了 `\plttrit` 和 `\pltnte` 来作为订正和批注命令。

```
\plttrit{测试}
```

```
\pltnte{测试}
```

**改错** 测试

**批注** 测试

命令是可以叠加的。

```
\pltnte{\plttrit{\pltnte{测试}改错}批注}
```

批注 改错 批注 测试  
改错  
批注

## 2.2. pltendant 命令 - 文末的作者信息

使用类似于`\pltendant{yyyy}{mm}{dd}{name}`的格式来定义结束的名字。

这就是《瓦尔登湖》所解释的哲学。

and the winner is: la vie.

```
\pltendant{2020}{5}{25}{peterlits zo}
```

这就是《瓦尔登湖》所解释的哲学。  
and the winner is: la vie.

2020 - 5 - 25  
peterlits zo

## 2.3. pltrun 环境 - 显示 L<sup>A</sup>T<sub>E</sub>X 代码

有的时候，可能需要显示 latex 的代码和它运行的结果，这里提供了两个一模一样的环境，分别为 `pltrun` 和 `pltRun`，就像这样：

```
\begin{pltrun}  
\pltnte{测试}  
\end{pltrun}
```

```
\pltnte{测试}
```

批注 测试

或者

```
\begin{pltRun}  
\plttrit{测试}  
\end{pltRun}
```

```
\plttrit{测试}
```

改错 测试

使用两个不一样的命令的原因是试图伪嵌套使用这个环境，而出于 L<sup>A</sup>T<sub>E</sub>X 的原因，它会在它匹配的第一个 `\end` 命令处停止，从而报错。所以两个一样的命令（除了大小写不一样之外）的目的就是为了嵌套使用。实际使用的话其实用哪个的效果都是一模一样的。

不过，伪嵌套也最多能够嵌套两层，如果想嵌套多层的话，还是应该在拷贝下来的 `peterlitsdoc.cls` 中更改来拷贝出更多的命令。（不过应该没有人这么干吧）

## 2.4. pltpic 命令 - 显示图片

使用`\pltpic`来显示图片，接受的参数是文件名，标题和引用名。需要注意的是，浮动体内不能放在盒子里。

图 1 的相应代码是：

图`\ref{head}`的相应代码是：

```
\pltpic[0.4]{./usage.jpg}{头像}{head}
```



图 1: 头像

其中命令`\pltpic[width]{path}{title}{refname}`有一个可选参数，默认为 0.85，表示关于整个文本区的比例宽度。`path`则是它的文件路径。`title`是它的小标题，而 `refname`是引用名字，可以被`\ref` 命令调用得到对应的序号。

## 2.5. plttodo 命令 - todo 格子

使用 `plttodo`命令会显示一个 to-do 框框。应该会比较棒吧。格式应该是`\plttodo[<char>]`，如果 `char`是 `v`的话，就是已经完成的框框，如果没有完成的话，就应该把框框搞成空格。如果处于叠加态的话，就应该把 `char`设置为 `x`，这个时候，它是一个半完成没有特别完成的状态中。如果是其他情况下的话，那它就是一种不合法的状态。显示为有问号的框框。

因为可能会经常用到，它有一个别名，是`\pltt`。

使用`\plttodo[v]`来标注已经完成的对象!  
使用`\plttodo[x]`来标注快完成的对象。

`\plttodo[v]`喂猫咪

`\plttodo[x]`做数学作业

`\plttodo[ ]`跑步

它会比较第一个字符，然后根据第一个字符来显示不同的内容。

`\pltt[ ]`   `\pltt[x]`   `\pltt[ ]`   `\pltt[ ]`  
`\pltt[aa]`   `\pltt[ a]`   `\pltt[v]`   `\pltt[vv]`  
`\pltt[ v]`

使用 ☒ 来标注已经完成的对象! 使用 ☒ 来标注快完成的对象。

☒ 喂猫咪

☒ 做数学作业

☐ 跑步

它会比较第一个字符，然后根据第一个字符来显示不同的内容。

☐ ☒ ☐ ☐ ☐ ☐ ☒ ☒ ☐

## 2.6. pltbox 环境 - 表格，或者选项

定义了 `pltbox` 环境，可以用来更好的定义制表符环境。因为是在 `tabbing` 的外面新添加了一个命令 `\col` 所以说基本上是一样的。可以自己谷歌一下 `tabbing`。

```
\begin{pltbox}
\col{3}{1}\=\col{3}{1}\=\col{3}{1}\kill
A. this   \>B. that   \>C. help   \\\
\plttodo[ ]apple   \>
\plttodo[x]water   \>
\plttodo[ ]kiss   \\\
\end{pltbox}
```

A. this	B. that	C. help
<input type="checkbox"/> apple	<input checked="" type="checkbox"/> water	<input type="checkbox"/> kiss

## 2.7. plttodoenv 环境 - 简单的 todo 列表环境

有时候想要联合使用 `plttodo` 和 `pltbox` 来制作一份漂亮的 `ToDo` 盒子是经常有的事情，于是我定义了一个简单的环境，`plttodoenv`。

它在不仅环境中定义了 `\plttodo` 的别名 — `\t`，使用这个环境，它还会自动添加 `ToDo` 表格应该有的附注，让你能够更加简单随心的的使用。

这个环境需要一个参数，来表示 `todo` 列表的列数。

```
\begin{plttodoenv}{2}
\t[ ]喝奶   \t[x]跑步   \t[v]耍
\t[ ]数学作业   \t[ ]英语作业   \t[x]走
\end{plttodoenv}
```

<input type="checkbox"/> 喝奶	<input checked="" type="checkbox"/> 跑步
<input checked="" type="checkbox"/> 耍	<input type="checkbox"/> 数学作业
<input type="checkbox"/> 英语作业	<input checked="" type="checkbox"/> 走

注: ☐ 未完成, ☒ 正在完成中, ☒ 已经完成。

## 2.8. pltplan 环境 - 时间表

没有想到 pltplan 太大了。我的 pltrun 根本放不下。

总的来说, pltplan 会生成一个好看的时间表! 使用 \item 来简单的添加计划项。如: \item[<OK>]{<time>}{<name>}{<thinking>}

```
\begin{pltplan}
\item[x]{12:33}{跑步}{呼呼$\sim$}
\item[ ]{1:00}{做英语作业}{}
\end{pltplan}
```

总之上面的 L<sup>A</sup>T<sub>E</sub>X 代码会生成下面的这份时间表, 好看吧, 嘻嘻。

⊖	≡	☑	...
12:33	跑步	☑× 呼呼 ~	
1:00	做英语作业	☐	

注: ☐ 未完成, ☑× 正在完成中, ☑ 已经完成。

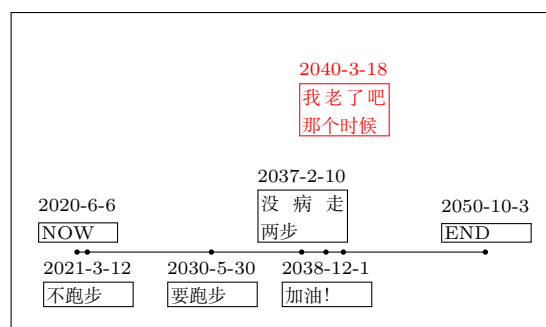
## 2.9. plttimeline 环境 - 时间线

还有 plttimeline 环境, 用来生成时间线。我使用 tikzpicture 来生成一个漂亮的图, 所以说 \D 选项能够接受一个 node 的可选参数。(默认为 below)

其中 \higher 是 4em, 只能用于这个环境。

如果不是节点会重合的话, 感觉还是尽量少用可选命令会比较好看一点。(如果放在上面的话, 有点点丑>\_<)

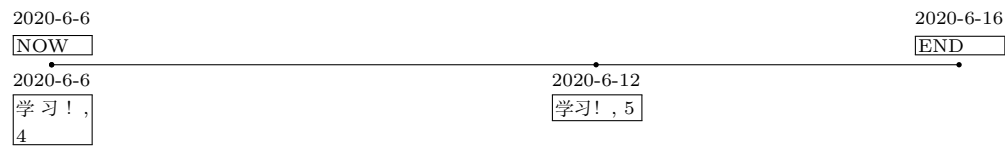
```
\begin{plttimeline}{2050}{10}{3}
\D{2011}{3}{12}{看不到我}
\D{2021}{3}{12}{不跑步}
\D{2030}{5}{30}{要跑步}
\D{2038}{12}{1}{加油! }
\D[above]{2037}{2}{10}{没病走两步}
\D[above=\higher, red]{2040}{3}{18}
  {我老了吧那个时候}
\end{plttimeline}
```



除此之外, 还提供了一个超级棒的时间线 \pltstudyline, 它需要 4 个参数, 分别是年份、月份和日期, 以及一个描述。

因为特别紧凑的原因, 所以说:

```
\pltstudyline{2020}{6}{2}{学习! }
```



它采用了一个独特的时间线结构，可以用来辅助复习。

## 2.10. 颜色

提供了一系列简单的颜色：

```
\pltred   红色\plrulerule
\pltblack 黑色\plrulerule
\pltblue  蓝色\plrulerule
\pltgray  灰色\plrulerule
```

红色 ■ 黑色 ■ 蓝色 ■ 灰色 ■

SECTION 3

peterlitsdoc 的更改

所有的更改都是基于文档类 `ctexart` 之上。

## 3.1. 边距

把原来的窄边距更改的稍微大了一些，默认值为 10pt 和 a4paper。

## 3.2. 代码摘录

原来的代码摘录环境不打算换行，尤其是行内的代码环境。为了更棒的排版选择，我认为让它能够换行更加合理一些。

我使用了宏包 `lstlisting` 来代替默认的 `\verb` 命令，现在使用 `\verb` 命令的话，底层其实是 `lst-inline`，会显得更加美观一些。

### 3.3. 列表

设置了列表环境 `enumerate` 的一些长度。

### 3.4. 代码环境

和 `\verb` 一样，环境 `lstlisting` 也是属于宏包 `listings`，所以说代码环境推荐用 `lstlisting` 来表示代码：

如果代码过长的话会换行哦。

```
\begin{lstlisting}
from math import pi

fn main() {
    using namespace std;
    cout << "this is a code" << endl;

    0 # return code 0 meaning ok
}
\end{lstlisting}
```

行距，字号都会有一点点变化。

如果代码过长的话会换行哦。

```
from math import pi

fn main() {
    using namespace std;
    cout << "this is a code" <<
        endl;

    0 # return code 0 meaning ok
}
```

行距，字号都会有一点点变化。

字号可以刚好在正文中包含 80 个字符。

```
-----[ 80 characters ]
```

### 3.5. 默认宏包

提供了一些常用的宏包，这样或许能够摆脱长长的 `\usepackage` 命令。

- |                          |                                   |
|--------------------------|-----------------------------------|
| 1. <code>calc</code>     | 在命令中使用数学表达式                       |
| 2. <code>xcolor</code>   | 使用颜色                              |
| 3. <code>mdwlist</code>  | 更好看的列表，支持命令 <code>\pltpara</code> |
| 4. <code>verbatim</code> | 摘录环境，支持命令 <code>\pltrun</code>    |
| 5. <code>listings</code> | 摘录环境，重定义命令 <code>\verb</code>     |
| 6. <code>enumitem</code> | 列表环境                              |
| 7. <code>hyperref</code> | 更好的引用                             |
| 8. <code>tikz</code>     | 流行的画图包                            |



- |                           |                 |
|---------------------------|-----------------|
| 9. <code>graphicx</code>  | 可以简单的用图片了       |
| 10. <code>titlesec</code> | 自定义的 section 样式 |
| 11. <code>url</code>      | 输入可以打开的 url     |

### **3.6. 编译环境**

作为一个面向中文环境的包，规定了使用 XeLaTeX 来作为默认的编译环境，如果不使用，可能会报错哦。