

peterlitsdoc文档类

Peterlits Zo

2020 年 5 月 27 日

目录

1 前言	2
2 peterlitsdoc 所提供命令	2
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 ^A T _E 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 颜色	5
3 peterlitsdoc 的更改	5
3.1 边距	5
3.2 代码摘录	5
3.3 列表	6
3.4 代码环境	6
3.5 默认宏包	6
3.6 编译环境	7

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^AT_EX 文档呢？因为我一开始觉得 L^AT_EX 的语法好傻哦，后来又看了看 Lisp 才慢慢看到 L^AT_EX 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^AT_EX 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^AT_EX 的原因,它会在它匹配的第一个`\end`命令出停止,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]{./temp.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`是非空格的话，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。

使用`\plttodo[x]`来标注已经完成的对象！

`\plttodo[x]` 喂猫咪

`\plttodo[]` 跑步

可以看到，是否是已完成计划，取决于是否有至少一个字符和是否第一个为空格。

`\plttodo[]` `\plttodo[x]` `\plttodo[]`
`\plttodo[]` `\plttodo[aa]` `\plttodo[a]`

使用☑ 来标注已经完成的对象！

☑ 喂猫咪

☐ 跑步

可以看到，是否是已完成计划，取决于是否有至少一个字符和是否第一个为空格。

☑ ☑ ☐ ☐ ☑ ☐

2.6. 颜色

提供了一系列简单的颜色：

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

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. `calc` 在命令中使用数学表达式
2. `xcolor` 使用颜色
3. `mdwlist` 更好看的列表，支持命令`\pltpara`
4. `verbatim` 摘录环境，支持命令`\pltrun`
5. `listings` 摘录环境，重定义命令`\verb`
6. `enumitem` 列表环境

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。