

README

南方小智

2020 年 8 月 2 日

目录

第一章 概述	2
1.1 处理单个文档	2
1.2 处理多个文档	3
1.3 生成图片文件	3
1.4 所有测试用例	4
第二章 Markdown 支持	6
2.1 特殊字符	6
2.2 导入代码文件	6
2.3 导入 tex 格式的图片文件	9
2.4 Markdown 转化器	9
第三章 TODO	13

第一章 概述

通过运行以下命令，可以将本 Markdown 文档生成 pdf 格式，建议阅读 pdf 版本的 README。

```
1 make readme
```

本工具用于个人辅助写作和作图等，满足各种文档编辑的需求。

- 该系统写作部分以基本的 Markdown 语法为核心，作图部分使用 Latex 和 dot 等通用工具包。
- 该系统会生成 Markdown 语法树，可通过 Latex 渲染生成 pdf 等，也可以通过其他渲染方式生成 Html, Word 等格式，方便发布文章。
- 该系统支持将批量的 Markdown 文档整理成册，比如合成一本书，并设置封面和章节目录等。
- 该系统可以通过 pgfplot, tikz 等工具，快速制作复杂的数学图片。

1.1 处理单个文档

本系统可以将一个 Markdown 文档转化为 tex 文档，并自动生成封面和目录，后续还通过 latex 工具包转化为 pdf。只需要将文档路径传入 path。

```
1 python3 create_book.py --path README.md --name README --author 南方小智
2 @echo 'xelatex cmd support Chinese'
3 xelatex -output-directory log README.tex
4 @echo 'run twice to build toc correctly'
5 xelatex -output-directory log README.tex
```

- name 代表生成的文件名和封面标题。
- author 代表生成的封面作者。

- 生成的 tex 文件在 log 文件夹中。

1.2 处理多个文档

本系统可以将多个 Markdown 文档合成一个 tex 文档，并自动生成封面和目录，后续还通过 latex 工具包转化为 pdf。只需要将所有文档放在同一个文件夹里，并将文件夹路径传入 path。

```
1 python3 create_book.py --path examples/趣题集/ --name 趣题集 --bg images/趣题集/background.jpeg --author 南方小智
2 @echo 'xelatex cmd support Chinese'
3 xelatex -output-directory log 趣题集.tex
4 @echo 'run twice to build toc correctly'
5 xelatex -output-directory log 趣题集.tex
```

- 支持多层文件夹，每层文件夹对应一个目录级别，比如文件夹内第一层目录每个文件夹名字为每章的名字，第二层目录每个文件夹名字为每节的名字。
- MAIN 文件为保留文件名，为该层目录对应章节的导言部分。
- _INDEX 文件为保留文件名，将用于对文件夹（章节）进行排序，目前章节的排列方法是较长的章节放在靠前的位置。
- 名字以 “_” 开始的文件夹或者文件会被忽略。

1.3 生成图片文件

本系统可以编译单一 tex 文档，并生成图片。只需要使用 simple 参数，就可以省略标题，目录，页码等，生成一个独立的图片形式，后续可以用 convert 命令将 pdf 转为图片。

```
1 python3 create_book.py --path images/趣题集/三角形悖论/image0.tex --name 三角形 --simple
2 xelatex -output-directory log 三角形.tex
3 @echo '需要安装brew install imagemagick'
4 convert -density 300 log/三角形.pdf -quality 90 log/三角形.png
```

1.4 所有测试用例

```
1 make readme      # 将本文档生成pdf
2 make all         # 生成一本《趣题集》
3 make image       # 生成一张png图片（需要安装brew install imagemagick）
4 make clean       # 清空log文件
5 make clean_all   # 清空所有生成文件，包括，log文件，tex文件， pdf文件等
```

具体细节请参照 makefile:

```
1 all:
2   python3 create_book.py --path examples/趣题集/ --name 趣题集 --bg images/趣
   题集/background.jpeg --author 南方小智
3   @echo 'xelatex cmd support Chinese'
4   xelatex -output-directory log 趣题集.tex
5   @echo 'run twice to build toc correctly'
6   xelatex -output-directory log 趣题集.tex
7   open log/趣题集.pdf
8
9 readme:
10  python3 create_book.py --path README.md --name README --author 南方小智
11  @echo 'xelatex cmd support Chinese'
12  xelatex -output-directory log README.tex
13  @echo 'run twice to build toc correctly'
14  xelatex -output-directory log README.tex
15  open log/README.pdf
16
17 image:
18  python3 create_book.py --path images/趣题集/三角形悖论/image0.tex --name 三
   角形 --simple
19  xelatex -output-directory log 三角形.tex
20  @echo '需要安装brew install imagemagick'
21  convert -density 300 log/三角形.pdf -quality 90 log/三角形.png
22  open log/三角形.png
23
24 test:
25  python3 test.py --path README.md
26
```

```
27 clean:
28   rm -f log/*.aux
29   rm -f log/*.toc
30   rm -f log/*.log
31   rm -f log/*.out
32
33 clean_all:
34   rm -f log/*.aux
35   rm -f log/*.toc
36   rm -f log/*.log
37   rm -f log/*.out
38   rm -f log/*.tex
39   rm -f log/*.png
40   rm -f log/*.pdf
```

第二章 Markdown 支持

基本语法请参照：[markdown wiki](#)

2.1 特殊字符

在 Markdown 文件中使用特殊字符时需要进行特殊处理，比如添加转义字符 \

```
1 _ -> \_    # _ 可用于表示斜体
2 $ -> \$    # $ 可用于开始数学表达式模式
3 \ -> \\    # \ 是特殊的Latex符号
```

2.2 导入代码文件

```
1 {{create_book.py}}[code:Python] # 导入代码文件

1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 from src.render.latex import DOCUMENT, ABSTRACT, SECTION, NEW_LINE
4 from src.utils.path import get_sections, read, write
5 from src.converter import md2tex
6 from src.constants import LOGGER_FORMAT
7 from os import listdir
8 from os.path import isfile, isdir
9 import markdown
10 import argparse
11 import logging
12 logger = logging.getLogger()
```

```

13 logging.basicConfig(level=logging.INFO, format=LOGGER_FORMAT)
14
15 MAIN_FILE = "MAIN"
16
17 def create_by_folder(args) -> str:
18     def travel(path: str, level: int = 0) -> str:
19         # TODO: refactoring into a class
20         content = ""
21         sections = get_sections(path)
22         sections.sort()
23
24         # Add main content in this section before add sub sections.
25         if MAIN_FILE in sections:
26             child_path = path + "/" + MAIN_FILE
27             if isfile(child_path):
28                 file_content = md2tex(read(child_path), level + 1)
29                 content += NEW_LINE([file_content])
30
31         str_sections = []
32         for section in sections:
33             # ignore this section for now if it's prefix with '_'
34             if len(section) > 0 and section[0] in ['_', '.']:
35                 continue
36             child_path = path + "/" + section
37             if isfile(child_path) and section != MAIN_FILE:
38                 file_content = md2tex(read(child_path), level + 1)
39                 str_sections.append(SECTION(section, file_content, level))
40             elif isdir(child_path):
41                 str_section = NEW_LINE([
42                     SECTION(section, "", level),
43                     travel(child_path, level=level+1)
44                 ])
45                 str_sections.append(str_section)
46         str_sections = sorted(str_sections, key=len, reverse=True)
47         content += NEW_LINE(str_sections)
48         return content
49     return travel(args.path)

```



```

50
51
52
53 def create_by_file(args) -> str:
54     content = md2tex(read(args.path), level=0)
55     return content
56
57
58 def get_args() -> argparse.Namespace:
59     logger.info("Parse arguments.")
60     parser = argparse.ArgumentParser()
61     parser.add_argument(
62         "--path", help="The folder/file path for the book", required=True
63     )
64     parser.add_argument(
65         "--bg", help="Background image for the title page", required=False
66     )
67     parser.add_argument(
68         "--simple", help="Output a simple standalone pdf", required=False,
69         action="store_true"
70     )
71     parser.add_argument(
72         "--name", help="The name of the book", required=True
73     )
74     parser.add_argument(
75         "--author", help="The author of the book", required=False
76     )
77     return parser.parse_args()
78
79 if __name__ == "__main__":
80     args = get_args()
81     logger.info(args)
82
83     output = f"log/{args.name}.tex"
84     output_content = ""
85     if isdir(args.path):

```

```

86     output_content = create_by_folder(args)
87     elif isfile(args.path):
88         output_content = create_by_file(args)
89
90     write(
91         output,
92         DOCUMENT(
93             args.name,
94             output_content,
95             bg_image=args.bg,
96             book_author=args.author,
97             is_simple=args.simple,
98         )
99     )

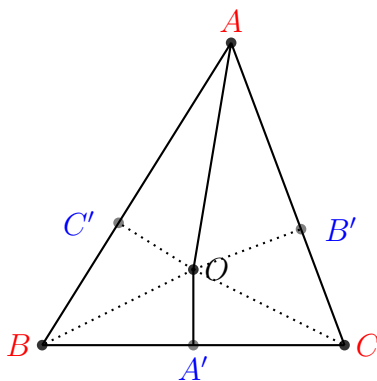
```

2.3 导入 tex 格式的图片文件

```

1 {{images/趣题集/三角形悖论/image1.tex}}[image] # 导入tex格式的图片文件

```



2.4 Markdown 转化器

该系统自带一个 Markdown 解析器，会将 Markdown 文本转化为一棵语法树。使用如下命令可以打印出本 Readme 文档的语法树：

```
1 make test # python3 test.py --path README.md
```

通过渲染器,可以将 Markdown 渲染成 tex,html,word 等格式。只需要继承 MDRender, 实现对应的抽象方法即可添加新的渲染器, 比如 src/render/md_tex_render.py。

```
1 #!/usr/bin/python
2 # -*- coding: UTF-8 -*-
3 from typing import List
4 from .constants import ListType, InLineType, IncludeType
5 import abc
6 import re
7
8
9 # use (.*?) to extract the pattern
10 # use (?:) to make the extraction optional
11 # use (.*?) to make a non-greedy match (shortest match)
12 # use (.*?)? to make the match optional
13 class MDRender():
14     def __init__(self, level: int = 0):
15         self.level = level
16         self.render_line_configs = {
17             InLineType.Link: {
18                 'pattern_str': r"\[(.*?)\]\((.*?)\)",
19                 'group_num': 2,
20             },
21             InLineType.Bold: {
22                 'pattern_str': r"\*(.*?)\*",
23                 'group_num': 1,
24             },
25             InLineType.Include: {
26                 'pattern_str': r"\{\{(.*?)\}\}\[(.*?)\]",
27                 'group_num': 2,
28             },
29         }
30
31     @abc.abstractmethod
32     def render_title(self, title, level) -> str:
33         pass
```

```

34
35     @abc.abstractmethod
36     def render_list(self, items: List[str], type_: ListType=ListType.Normal)
-> str:
37         pass
38
39     @abc.abstractmethod
40     def build_inline_link(self, title: str, link: str) -> str:
41         pass
42
43     @abc.abstractmethod
44     def build_inline_bold(self, content: str) -> str:
45         pass
46
47     @abc.abstractmethod
48     def build_inline_include(self, content: str, type_: IncludeType) -> str:
49         pass
50
51     def build_inline_new_pattern(self, match, inline_type: InLineType) -> str:
52         if inline_type == InLineType.Link:
53             title = match.group(1)
54             link = match.group(2)
55             return self.build_inline_link(title, link)
56         elif inline_type == InLineType.Bold:
57             content = match.group(1)
58             return self.build_inline_bold(content)
59         elif inline_type == InLineType.Include:
60             content = match.group(1)
61             setting = match.group(2)
62             if len(setting.split(":", 1)) == 2:
63                 type_ = setting.split(":", 1)[0]
64                 config = setting.split(":", 1)[1]
65             else:
66                 type_ = setting
67                 config = ""
68             return self.build_inline_include(content, type_, config)
69     return ""

```

```

70
71     def render_line_with(self, line, inline_type, config) -> str:
72         changed = True
73         while changed:
74             pattern = re.compile(config['pattern_str'], re.IGNORECASE)
75             match = pattern.search(line)
76             if match and len(match.groups()) == config['group_num']:
77                 old_pattern = match.group(0)
78                 new_pattern = self.build_inline_new_pattern(match, inline_type
79 )
80                 line = line.replace(old_pattern, new_pattern)
81             else:
82                 changed = False
83         return line
84
85     @abc.abstractmethod
86     def render_line(self, line: str) -> str:
87         for inline_type, config in self.render_line_configs.items():
88             line = self.render_line_with(line, inline_type, config)
89         return line
90
91     @abc.abstractmethod
92     def render_blockquote(self, content) -> str:
93         pass
94
95     @abc.abstractmethod
96     def render_math(self, content) -> str:
97         pass
98
99     @abc.abstractmethod
100    def render_code(self, content, language) -> str:
        pass

```

第三章 TODO

- 目前很多 latex 设置都是 Hard Code 的，应当将这些设置变成可配置的：
 - 目前目录最多显示两层标题结构
 - 目前最高层目录默认为 Chapter，设置 level 参数使其可配置化
- 目前 Markdown 解析器仍然有许多 Markdown 基本语法并不支持，需要：
 - 支持行内斜体，下划线等。
 - 支持行内代码。
 - 支持一般图片格式的导入
 - [Done] 支持嵌套 List
 - * 测试 3 层嵌套
 - 支持表格
- 目前 Latex 渲染器仍然有一些 Latex 用法支持不够，需要：
 - 支持更多类似 \$, <, > 等 latex 特殊字符。
- 支持更多的 Features：
 - 添加页面背景水印，这样可以防止发布的材料被盗版使用。
 - 添加程序辅助画图（特别是制作复杂的各类数学图片）。
 - 添加作者介绍。
 - 添加 html 和 word 渲染器。
- Better Engineering：
 - md_tree 的 Parser 主体可以用自动机的方式实现，从而减少 if/else 的判断和重复代码片。