WordMarker

chensixiang

2021 年 06 月 20 日

Contents

CHAPTER 1

概述

WordMarker 是一个文档生成器，他提供了以下主要的功能。

1. 读写 `.csv` 文件。
    - 你可以使用 `CsvTemplate` 批量地读取 `.csv` 文件或者读取单个 `.csv` 文件转换为 `DataFrame` 类型的数据。
    - 你可以使用 `CsvTemplate` 将 `DataFrame` 类型的数据转换为 `.csv` 文件。
2. 读写数据库。
    - 你可以使用 `PdbcTemplate` 将 `DataFrame` 类型的数据写入到数据库中。
    - 你可以使用 `PdbcTemplate` 将数据库中的数据转换为 `DataFrame` 类型的数据。
3. 生成 Word 文档。
    - 你可以使用 `WordTemplate` 获取文本、图片、模板，通过模板生成固定格式的 Word 文档。
4. 其他。
    - 提供了一些实用的工具类，可以简化开发，它们位于 `wordmarker.utils` 模块中，使用方法可以参考 API 文档。

参见:

DataFrame 是 pandas 中的数据类型，请访问 pandas 官网 ，了解更多信息。

CHAPTER 2

快速入门

2.1 快速入门

在快速入门中，你将使用 `wordmarker` 构建一个示例项目，它包含了 `wordmarker` 的基本使用。你可以在 GitHub 上找到该示例。

2.1.1 环境搭建

1. Python 的版本 `>= 3.7`
2. 下载并安装 最新版的 `wordmarker`
3. 安装 `postgresql` 数据库
4. 最好使用开发工具 Pycharm 、VsCode 等

2.1.2 初始化项目

1. 打开终端，进入项目的根目录，输入 `wordmarker-quickstart` 命令，按照提示完成项目的初始化

```
欢迎使用 WordMarker 的脚手架，请按照提示完成项目的构建。
-------------------------------------------------
配置数据库
-------------------------------------------------
1. PostgreSQL
```

(下页继续)

(续上页)

```
2. MySql
3. Oracle
4. Microsoft
数据库的类型，默认为 [1]: 1
用户名，默认为 [postgres]:
密码，默认为 [123456]: 123456
主机，默认为 [localhost]:
端口，默认为 [5432]:
要连接的数据库: example
------------------------------------------------
配置输入输出目录
------------------------------------------------
是否启用默认的目录配置，默认为 [Y] [Y/n]: y
成功创建 data/in 目录
成功创建 data/out 目录
成功创建 template/in 目录
成功创建 template/out 目录
成功创建 img 目录
成功创建 text 目录

成功构建项目 ^_^ !
```

2. 生成的目录结构如下

```
../wordmarker-example/
                      /data/in
                      /data/out
                      /img
                      /template/in
                      /template/out
                      /text
                      /config.yaml
```

3. config.yaml 文件中包含相关的配置

```
#␣
↪==========================================================================================================

# ----------------------配置数据库信息
#
# 详细信息请查看 https://docs.sqlalchemy.org/en/14/core/engines.html
# url:
#    postgresql: "postgresql://username:password@localhost:5432/database"
#    mysql: mysql+pymysql://username:password@localhost:3306/database
```

(下页继续)

(续上页)

```yaml
# echo: 是否开启日志
# encoding: 编码，默认 utf-8
# pool_size: 连接池大小
# max_overflow: 连接的数量，以允许在连接池" 溢出"
# pool_recycle: 在给定的秒数过去之后，此设置将导致池回收连接。默认为-1。例如，设置为
3600 表示一小时后将回收连接
# pool_timeout: 放弃从池中获得连接之前要等待的秒数
# echo_pool: 是否开启日志
pdbc:
  engine:
    url: postgresql://postgres:123456@localhost:5432/example
    echo: false
    encoding: utf-8
    pool_size: 5
    max_overflow: 10
    pool_recycle: -1
    pool_timeout: 30.0
    echo_pool: false

#␣
↪==========================================================================================

# ----------------------配置 csv 文件信息
#
# path: 文件的输入路径，可以是文件，也可以是目录
#       路径分隔符为 '/' 或 '\'
#       path 的值为相对于当前 yaml 文件的路径
# dir: 文件的输出目录
#      路径分隔符为 '/' 或 '\'
#      dir 的值为相对于当前 yaml 文件的路径
data:
  csv:
    input:
      path: data/in
    output:
      dir: data/out
  docx:
    input:
      path: template/in
    output:
      dir: template/out
  img:
    output:
```

(下页继续)

(续上页)

```
      dir: img
  text:
    input:
      path: text

#␣
↪==========================================================================================
```

4. 创建 main.py 文件，最终生成的目录如下

```
../wordmarker-example/
                      /data/in
                      /data/out
                      /img
                      /template/in
                      /template/out
                      /text
                      /config.yaml
                      /main.py
```

2.1.3 编写脚本

1. 初始化 WordMarker 上下文和相关模板

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()
```

2. 下载 景气指数 _ 加盐.csv 文件，将文件放入 ../wordmarker-example/data/in 目录中，编写代码从配置中获取 .csv 文件，类型为 DataFrame

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate

```

(下页继续)

(续上页)

```
if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)
```

```
输出结果

     起飞年  起飞星期       景气指数
0    2015     1  89.57518
1    2015    10  92.69366
2    2015    11  92.43514
3    2015    12  92.51584
4    2015    13  92.45800
..    ...   ...       ...
207  2018    53  85.67820
208  2018     6  94.36436
209  2018     7  94.22866
210  2018     8  94.80134
211  2018     9  94.37548

[212 rows x 3 columns]
```

3. 创建数据库 `example`， 将 `DataFrame` 类型的数据写入数据库

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
```

(下页继续)

(续上页)

```
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")
```

4. 从 example 数据库中获取数据，类型为 DataFrame

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")

    # 从数据库中获取数据，类型为 DataFrame
    prosperity_index_database = pdbc_tpl.query_table("t_prosperity_index")
    print(prosperity_index_database)
```

```
输出结果

     起飞年  起飞星期       景气指数
0    2015     1  89.57518
```

(下页继续)

(续上页)

```
1    2015    10  92.69366
2    2015    11  92.43514
3    2015    12  92.51584
4    2015    13  92.45800
..    ...   ...       ...
207  2018    53  85.67820
208  2018     6  94.36436
209  2018     7  94.22866
210  2018     8  94.80134
211  2018     9  94.37548

[212 rows x 3 columns]

     起飞年  起飞星期       景气指数
0    2015     1  89.57518
1    2015    10  92.69366
2    2015    11  92.43514
3    2015    12  92.51584
4    2015    13  92.45800
..    ...   ...       ...
207  2018    53  85.67820
208  2018     6  94.36436
209  2018     7  94.22866
210  2018     8  94.80134
211  2018     9  94.37548

[212 rows x 3 columns]
```

5. 将 DataFrame 类型的数据转换为 .csv 文件

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
```

(下页继续)

(续上页)

```
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")

    # 从数据库中获取数据，类型为 DataFrame
    prosperity_index_database = pdbc_tpl.query_table("t_prosperity_index")
    print(prosperity_index_database)

    # 将 DataFrame 类型的数据转换为.csv 文件
    csv_tpl.df_to_csv({'景气指数_ 数据库.csv': prosperity_index_database})
```

```
输出目录

../wordmarker-example/data/out/景气指数_ 数据库.csv
```

6. 编写 example.yaml 和 meta.yaml 文件，将这两个文件放在 ../wordmarker-example/text/ 目录下

   example.yaml

```
example:
  title: '景气指数'
  img_title: '图 1 景气指数'
  explanation:
    - '2017 年上半年民航全市场景气指数上涨：国内航线 154，国际航线 125，港澳台航线
171。'
    - '2017 年国内航线景气指数同比增幅放缓至 0.40%，🄵场稳步上升。'
    - '2017 年国际航线景气指数增速上升，同比增幅与 2016 年放缓至 0.40%，但春节峰值周
景气指数超越 2016 年峰值，达到 94.24，再创新高。'
```

   meta.yaml

```
meta:
  author: '{{ author }}'
  email: 'chensixiang1234@gamil.com'
```

7. 在 meta.yaml 文件中 meta.author 对应的值包含插值表达式。要对表达式赋值，需要创建 AbstractConverter 类的实现类，并且创建实现类的对象

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate,␣
↪AbstractConverter
```

(下页继续)

(续上页)

```
class TextConverter(AbstractConverter):
    def __init__(self, word_tpl_: WordTemplate):
        super().__init__(word_tpl_)

    @staticmethod
    def author():
        return 'chensixiang'

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")

    # 从数据库中获取数据，类型为 DataFrame
    prosperity_index_database = pdbc_tpl.query_table("t_prosperity_index")
    print(prosperity_index_database)

    # 将 DataFrame 类型的数据转换为.csv 文件
    csv_tpl.df_to_csv({'景气指数_ 数据库.csv': prosperity_index_database})

    # 创建 TextConverter 对象，它继承了 AbstractConverter，可以将 yaml 模板中的
插值表达式进行转换
    text = TextConverter(word_tpl)
```

8. 下载 default_tpl.docx 文件，将文件放入 ../wordmarker-example/template/in 目录中，下载 景气指数.png 文件，将文件放入 ../wordmarker-example/img 目录中，并修改 config.yaml 中的 data.docx.input.path 的配置，以及编写 content 字典

   config.yaml

```yaml
#␣
↪==========================================================================================

# ----------------------配置数据库信息
#
# 详细信息请查看 https://docs.sqlalchemy.org/en/14/core/engines.html
# url:
#    postgresql: "postgresql://username:password@localhost:5432/database"
#    mysql: mysql+pymysql://username:password@localhost:3306/database
# echo: 是否开启日志
# encoding: 编码，默认 utf-8
# pool_size: 连接池大小
# max_overflow: 连接的数量，以允许在连接池" 溢出"
# pool_recycle: 在给定的秒数过去之后，此设置将导致池回收连接。默认为-1。例如，设置为
3600 表示一小时后将回收连接
# pool_timeout: 放弃从池中获得连接之前要等待的秒数
# echo_pool: 是否开启日志
pdbc:
  engine:
    url: postgresql://postgres:123456@localhost:5432/example
    echo: false
    encoding: utf-8
    pool_size: 5
    max_overflow: 10
    pool_recycle: -1
    pool_timeout: 30.0
    echo_pool: false

#␣
↪==========================================================================================

# ----------------------配置 csv 文件信息
#
# path: 文件的输入路径，可以是文件，也可以是目录
#       路径分隔符为 '/' 或 '\'
#       path 的值为相对于当前 yaml 文件的路径
# dir: 文件的输出目录
#      路径分隔符为 '/' 或 '\'
#      dir 的值为相对于当前 yaml 文件的路径
data:
  csv:
    input:
      path: data/in
    output:
```

(下页继续)

(续上页)

```
      dir: data/out
  docx:
    input:
      path: template/in/default_tpl.docx
    output:
      dir: template/out
  img:
    output:
      dir: img
  text:
    input:
      path: text

#␣
↪==========================================================================================
```

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate,␣
↪AbstractConverter
from docxtpl import InlineImage
from docx.shared import Mm


class TextConverter(AbstractConverter):
    def __init__(self, word_tpl_: WordTemplate):
        super().__init__(word_tpl_)

    @staticmethod
    def author():
        return 'chensixiang'

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
```

(下页继续)

(续上页)

```
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")

    # 从数据库中获取数据，类型为 DataFrame
    prosperity_index_database = pdbc_tpl.query_table("t_prosperity_index")
    print(prosperity_index_database)

    # 将 DataFrame 类型的数据转换为.csv 文件
    csv_tpl.df_to_csv({'景气指数_ 数据库.csv': prosperity_index_database})

    # 创建 TextConverter 对象，它继承了 AbstractConverter，可以将 yaml 模板中的
插值表达式进行转换
    text = TextConverter(word_tpl)

    content = {
        # 直接赋值
        'title': '景气指数',
        # 从 example.yaml 模板中获取值
        'img_title': word_tpl.get_value("example.img_title"),
        # 图片
        'img': InlineImage(word_tpl.tpl, word_tpl.get_img_file('景气指数.
↪png'),
                           width=Mm(100)),
        # 从 example.yaml 模板中获取值
        'explanation': word_tpl.get_value("example.explanation"),
        # 从 meta.yaml 模板中获取将插值表达式进行转换后的值
        'author': text.get_value("meta.author"),
        # 从 meta.yaml 模板中获取值
        'email': word_tpl.get_value("meta.email"),
    }
```

9. 添加 content 到总的上下文中，并输出 word 文档

```
from wordmarker.contexts import WordMarkerContext
from wordmarker.templates import CsvTemplate, PdbcTemplate, WordTemplate,␣
↪AbstractConverter
from docxtpl import InlineImage
from docx.shared import Mm


class TextConverter(AbstractConverter):
```

(下页继续)

(续上页)

```
    def __init__(self, word_tpl_: WordTemplate):
        super().__init__(word_tpl_)

    @staticmethod
    def author():
        return 'chensixiang'

if __name__ == '__main__':
    # 初始化上下文
    WordMarkerContext("config.yaml")
    # 读写 csv 文件的模板
    csv_tpl = CsvTemplate()
    # 读写数据库的模板
    pdbc_tpl = PdbcTemplate()
    # 读写 word 文档的模板
    word_tpl = WordTemplate()

    # 从配置中获取.csv 文件，类型为 DataFrame
    csv_dict = csv_tpl.csv_to_df()
    prosperity_index_file = csv_dict['景气指数_ 加盐.csv']
    print(prosperity_index_file)

    # 将 DataFrame 类型的数据写入数据库
    pdbc_tpl.update_table(prosperity_index_file, "t_prosperity_index")

    # 从数据库中获取数据，类型为 DataFrame
    prosperity_index_database = pdbc_tpl.query_table("t_prosperity_index")
    print(prosperity_index_database)

    # 将 DataFrame 类型的数据转换为.csv 文件
    csv_tpl.df_to_csv({'景气指数_ 数据库.csv': prosperity_index_database})

    # 创建 TextConverter 对象，它继承了 AbstractConverter，可以将 yaml 模板中的
插值表达式进行转换
    text = TextConverter(word_tpl)

    content = {
        # 直接赋值
        'title': '景气指数',
        # 从 example.yaml 模板中获取值
        'img_title': word_tpl.get_value("example.img_title"),
        # 图片
        'img': InlineImage(word_tpl.tpl, word_tpl.get_img_file('景气指数.
↪png'),
```

(下页继续)

(续上页)

```
                            width=Mm(100)),
        # 从 example.yaml 模板中获取值
        'explanation': word_tpl.get_value("example.explanation"),
        # 从 meta.yaml 模板中获取将插值表达式进行转换后的值
        'author': text.get_value("meta.author"),
        # 从 meta.yaml 模板中获取值
        'email': word_tpl.get_value("meta.email"),
    }

    # 添加 content 到总的上下文中
    word_tpl.append(content)
    # 输出 word 文档
    word_tpl.build("example.docx")
```

```
输出目录

../wordmarker-example/template/out/
                                    /example/
                                            /img/景气指数.png
                                            /example.docx
```

10. 下载 example.docx 文件

CHAPTER 3

参考文档

- 敬请期待

CHAPTER 4

API 文档

- genindex
- modindex
- search

4.1 wordmarker.contexts package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块包含整个应用的上下文。

1. `wordmarker.contexts.context`

   上下文的抽象类，所有的上下文都要继承它。

2. `wordmarker.contexts.system_context`

   系统上下文，获取和系统有关的属性。例如，路径分隔符，文件分隔符等。

3. `wordmarker.contexts.yaml_context`

   yaml 文件的上下文，解析 yaml 文件。根据 key 值返回 yaml 文件中对应的 value 值。

4. wordmarker.contexts.application_context

应用上下文，用来初始化工厂和其他的上下文。

一般初始化，应用上下文 WordMarkerContext 来初始化整个应用。

4.1.1 Submodules

4.1.2 wordmarker.contexts.application_context module

class WordMarkerContext(**args*, ***kwargs*)

基类：*wordmarker.contexts.context.Context*

应用上下文，你可以从上下文中获取到：

bean_factory：工厂里存放的 bean 实例的相关信息

yaml_context：加载的 yaml 文件的相关信息

property bean_factory

注解： 获取 bean 工厂

返回

- bean 工厂

property yaml_context

注解： 获取 yaml 文件的上下文

返回

- yaml 文件的上下文

4.1.3 wordmarker.contexts.context module

class Context

基类：object

抽象类：所有上下文的公共父类

4.1.4 wordmarker.contexts.system_context module

class SystemContext

基类：*wordmarker.contexts.context.Context*

系统上下文，获取和系统有关的属性

file_separator = ':'

path_separator = '/'

4.1.5 wordmarker.contexts.yaml_context module

class YamlContext(*path*)

基　类：*wordmarker.contexts.context.Context*, *wordmarker.loaders.yaml_resource_loader.YamlResourceLoader*

yaml 文件的上下文

get_value(*prop*)

注解： 从 yaml 字典中，根据属性获取对应的值

加载多个 yaml 文件，排在后面的文件里的值，会覆盖前面的文件里的值

参数 **prop** –属性，用 . 分隔，例如，pdbc.engine.url

返回

- yaml 字典中对应的值

get_yaml()

注解： 获取从 yaml 文件中读取的数据，类型为 dict

返回

- path 为文件，返回一个字典，内容为 yaml 文件的内容
- path 为目录，返回一个嵌套的字典
  - key 为 yaml 文件的绝对路径
  - value 为 yaml 文件的内容，是一个字典

property path

注解： 存放 yaml 文件的路径，可以是文件，也可以是目录

返回

- yaml 文件的路径

4.2 wordmarker.creatives package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块使用工厂和生成器模式来管理对象的创建。

1. `wordmarker.creatives.factory`

   工厂模块。

2. `wordmarker.creatives.builder`

   生成器模块。

4.2.1 Submodules

4.2.2 wordmarker.creatives.builder module

class AbstractBuilder

基类：object

用于构造复杂的对象

abstract append(**args*, ***kwargs*)

注解： 添加对象的部件

返回

- 当前 builder 对象

abstract build()

注解： 构建复杂对象的实例

返回

- 复杂对象的实例

4.2.3 wordmarker.creatives.factory module

class AbstractBeanFactory

基类：object

BeanFactory 的抽象类

abstract contain_bean(*name: str*)

注解： 判断工厂中是否包含某个 bean 实例

参数 name –名字

返回

- 包含，返回 True
- 不包含，返回 False

abstract get_bean(*name: str*)

注解： 根据 bean 的名字获取 bean 实例

参数 name –名字

返回

- bean 实例

abstract get_bean_names()

注解： 获取工厂中所有 bean 实例对应的名字

返回

- 所有的 bean 实例对应的名字

abstract get_type(*name: str*)

注解： 获取 bean 实例的类型

参数 name –名字

返回

- 工厂中存在 bean 实例，返回 bean 实例的类型
- 不存在，返回 None 对应的类型 NoneType

class BeanFactory(**args*, ***kwargs*)

基类：*wordmarker.creatives.factory.AbstractBeanFactory*

```
AbstractBeanFactory 的实现类
```

contain_bean(*name: str*)

注解： 判断工厂中是否包含某个 bean 实例

参数 **name** –名字

返回

- 包含，返回 True
- 不包含，返回 False

get_bean(*name: str*)

注解： 根据 bean 的名字获取 bean 实例

参数 **name** –名字

返回

- bean 实例

get_bean_names() → collections.abc.KeysView

注解： 获取工厂中所有 bean 实例对应的名字

返回

- 所有的 bean 实例对应的名字

get_type(*name: str*)

注解： 获取 bean 实例的类型

参数 **name** –名字

返回

- 工厂中存在 bean 实例，返回 bean 实例的类型
- 不存在，返回 None 对应的类型 NoneType

class FactoryBean(**args*, ***kwargs*)

基类：*wordmarker.creatives.factory.BeanFactory*

```
BeanFactory 的子类

通过 add_bean 方法，将 bean 实例添加到工厂
```

add_bean(*name: str*, *bean*)

注解： 将 bean 实例添加到工厂

参数

- **name** –名字
- **bean** –bean 实例

4.3 wordmarker.data package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块用来处理数据和资源。

1. `wordmarker.data.resource`

   ```
   加载的资源，包含和资源相关的属性和判断。
   ```

2. `wordmarker.data.formatter`

   ```
   格式化数据，对某些数据进行处理。
   ```

4.3.1 Submodules

4.3.2 wordmarker.data.formatter module

class Formatter

基类：object

格式化的抽象类

abstract format(**args*)

注解： 格式化数据

参数 args –数据

返回

- 格式化后的数据

class SqlFormatter

基类：*wordmarker.data.formatter.Formatter*

格式化 sql 语句

format(*sql*)

注解： 格式化用户输入的 sql 语句

例如：

```
-- 输入 --
select * from t_user where username=? and password=?
-- 输出 --
select * from t_user where username=:a and password=:b
```

小心： 拼接的 :a :b 使用的是 26 个字母，也就是说一次查询的 ?，不能超过 26 个。

参数 sql –sql 语句

返回

- 格式化后的 sql 语句

4.3.3 wordmarker.data.resource module

class Resource(**args*)

基类：*wordmarker.contexts.system_context.SystemContext*

资源类，包含与资源相关的属性和判断的方法

exists()

注解： 判断文件或目录是否存在

返回

- 存在，返回 True
- 不存在，返回 False

get_dir()

注解： 获取目录，返回目录的绝对路径

返回

- 是目录，返回目录的绝对路径
- 是文件，返回文件所在的目录

get_file()

注解： 获取文件，返回文件的绝对路径

返回

- 是文件，返回文件的绝对路径
- 是目录，返回当前目录下所有文件的绝对路径

get_file_encoding()

注解： 获取文件的编码

返回

- 是文件，获取文件的编码，返回一个文件编码的字符串
- 是目录，获取当前目录下所有文件的编码，返回一个字典
  - key 为文件的绝对路径
  - value 为文件的编码

get_file_name()

注解： 获取文件名

返回

- 是文件，返回文件的名字
- 是目录，返回当前目录下的所有文件的文件名

get_file_name_prefix_suffix()

注解： 获取文件的前缀和后缀

返回

- 是文件，获取文件的前缀和后缀，返回一个元组
- 是目录，获取目录下所有文件的前缀和后缀，返回一个字典
  - key 为文件名
  - value 为由文件的前缀和后缀组成的元组

get_loader()

注解： 获取加载当前资源的加载器

返回

- 加载器

is_file()

注解： 判断是否为文件

返回

- 是文件，返回 True
- 不是文件，返回 False

4.4 wordmarker.loaders package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块用来加载资源。

1. wordmarker.loaders.default_resource_loader

   默认的资源加载器。

2. wordmarker.loaders.yaml_resource_loader

   yaml 文件的资源加载器。

3. wordmarker.loaders.resource_loader

   资源加载器的抽象类。

4.4.1 Submodules

4.4.2 wordmarker.loaders.default_resource_loader module

class DefaultResourceLoader(**args*)

基类：*wordmarker.loaders.resource_loader.ResourceLoader*

默认的资源加载器

get_resource(*path=None*) → *wordmarker.data.resource.Resource*

注解： 获取资源

参数 path –路径

返回

- 资源实例

load(*resource:* wordmarker.data.resource.Resource)

注解： 加载资源

参数 resource –资源实例

返回

- 资源为文件，获取文件中的数据
- 资源为目录，获取目录下所有文件中的数据，返回一个字典
  - key 为文件的绝对路径
  - value 为文件的数据

4.4.3 wordmarker.loaders.resource_loader module

class ResourceLoader

基类：object

资源加载器的抽象类

abstract get_resource(*path*)

注解： 获取资源

参数 path –路径

返回

- 资源

abstract load(*resource*)

注解： 加载资源

参数 resource –资源

返回

- 资源中的数据

4.4.4 wordmarker.loaders.yaml_resource_loader module

class YamlResourceLoader(**args*)

基类：*wordmarker.loaders.default_resource_loader.DefaultResourceLoader*

yaml 文件的资源加载器

load(*resource:* wordmarker.data.resource.Resource)

注解： 加载 yaml 资源

参数 **resource** –yaml 资源实例

返回

- yaml 资源为文件，获取文件中的数据
- yaml 文件资源为目录，获取目录下所有 yaml 文件中的数据，返回一个字典
  - key 为文件的绝对路径
  - value 为文件的数据

4.5 wordmarker.templates package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块是 WordMarker 的核心模块，可以读写 csv 文件，操纵数据库，读写 docx 文件等。

1. `wordmarker.templates.pdbc_template`

   操纵数据库的模板。

2. `wordmarker.templates.csv_template`

   读写 csv 文件的模板。

3. `wordmarker.templates.word_template`

   读写 docx 文件的模板。

4.5.1 Submodules

4.5.2 wordmarker.templates.csv_template module

class CsvHelper(**args*, ***kwargs*)

基　　类: *wordmarker.loaders.default_resource_loader.DefaultResourceLoader*, *wordmarker.contexts.system_context.SystemContext*

通过读取配置文件，获取 csv 文件的相关信息

get_csv()

注解： 获取 csv 文件的绝对路径

返回

- yaml 文件中 `data.csv.input.path` 是目录，返回当前目录下的所有 csv 文件的绝对路径
- yaml 文件中 `data.csv.input.path` 是文件，返回当前文件的绝对路径

get_csv_file_name()

注解： 获取 csv 文件的文件名

返回

- 是文件，返回文件的名字
- 是目录，返回当前目录下的所有文件的文件名

get_csv_in_path()

注解： 通过读取 yaml 文件的 `data.csv.in.path` 属性，获取输入的 csv 文件或目录的绝对路径

返回

- csv 文件或目录的绝对路径

get_csv_out_path()

注解： 通过读取 yaml 文件的 `data.csv.output.dir` 属性，获取输出的 csv 的目录的绝对路径

返回

- csv 目录的绝对路径

class CsvOperations

基类：`object`

csv 文件的相关操作的抽象类

abstract csv_to_df()

注解： 将 csv 文件转换为 DataFrame

返回

- 数据框

abstract df_to_csv(**args*, ***kwargs*)

注解： 将 DataFrame 转换为 csv 文件

class CsvTemplate(**args*, ***kwargs*)

基类：*wordmarker.templates.csv_template.CsvOperations*, *wordmarker.templates.csv_template.CsvHelper*

操作 csv 文件的模板

csv_to_df(*sep=<object object>*, *delimiter=None*, *header='infer'*, *names=None*, *index_col=0*, *usecols=None*, *squeeze=False*, *prefix=None*, *mangle_dupe_cols=True*, *dtype=None*, *engine=None*, *converters=None*, *true_values=None*, *false_values=None*, *skipinitialspace=False*, *skiprows=None*, *skipfooter=0*, *nrows=None*, *na_values=None*, *keep_default_na=True*, *na_filter=True*, *verbose=False*, *skip_blank_lines=True*, *parse_dates=False*, *infer_datetime_format=False*, *keep_date_col=False*, *date_parser=None*, *dayfirst=False*, *cache_dates=True*, *iterator=False*, *chunksize=None*, *compression='infer'*, *thousands=None*, *decimal: str = '.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *doublequote=True*, *escapechar=None*, *comment=None*, *encoding=None*, *dialect=None*, *error_bad_lines=True*, *warn_bad_lines=True*, *delim_whitespace=False*, *low_memory=True*, *memory_map=False*, *float_precision=None*, *storage_options: Optional[Dict[str, Any]] = None*) → Union[pandas.io.parsers.TextFileReader, pandas.core.series.Series, pandas.core.frame.DataFrame, None, Dict[str, Optional[Union[pandas.io.parsers.TextFileReader, pandas.core.series.Series, pandas.core.frame.DataFrame]]]]

注解： 从 yaml 配置中读取 csv 文件，转换为 pandas 的 DataFrame

小技巧： csv_to_df 的参数详情，请访问 pandas 官网，了解更多信息

返回

- 多个 csv 文件，返回一个 `DataFrame` 的字典
  - key 为文件名
  - value 为 `DataFrame`
- 一个 csv 文件，返回一个 `DataFrame`

df_to_csv(*data_dict: Dict[str, pandas.core.frame.DataFrame]*, *sep: str = ','*, *na_rep: str = ''*, *float_format: Optional[str] = None*, *columns: Optional[Sequence[Optional[Hashable]]] = None*, *header: Union[bool, List[str]] = True*, *index: bool = True*, *index_label: Union[Hashable, None, Sequence[Optional[Hashable]]] = None*, *mode: str = 'w'*, *encoding: Optional[str] = None*, *compression: Optional[Union[str, Dict[str, Any]]] = 'infer'*, *quoting: Optional[int] = None*, *quotechar: str = '"'*, *line_terminator: Optional[str] = None*, *chunksize: Optional[int] = None*, *date_format: Optional[str] = None*, *doublequote: bool = True*, *escapechar: Optional[str] = None*, *decimal: str = '.'*, *errors: str = 'strict'*, *storage_options: Optional[Dict[str, Any]] = None*)

注解： 将一个或多个 `DataFrame` 转换成 csv 文件，输出到指定文件夹

小技巧： df_to_csv 的参数详情，请访问 pandas 官网，了解更多信息

参数 data_dict – 由 `DataFrame` 组成的字典，key 为输出的文件名，value 为 `DataFrame`

4.5.3 wordmarker.templates.pdbc_template module

class PdbcHelper(**args*, ***kwargs*)

基类：`object`

通过读取配置文件获取数据库的信息，进而建立连接

property engine: sqlalchemy.engine.base.Engine

注解： 获取引擎对象

返回

- 引擎对象 engine

property engine_dict

注解： 获取引擎对象中设置的值

返回

- 引擎对象中设置的值
- key 为属性
- value 为对应的值

set_engine(***kwargs*)

注解： 设置引擎

小技巧： 你可以在调用 PdbcTemplate 内的方法之前，设置引擎需要的其他参数（不包括配置文件内的参数）

参数 kwargs –除去配置文件中其他的值，采用 key=value 的形式

返回

- engine 对象

class PdbcOperations

基类：object

数据库的相关操作的抽象类

abstract execute(*sql*)

注解: 执行 sql 语句，建议 sql 类型为 DDL（数据定义语言）时候使用

参数 sql –sql 语句

abstract query(*sql*)

注解: 查询数据库

参数 sql –sql 语句

返回

- 查询结果

abstract update(*sql*)

注解: 更新数据库

参数 sql –sql 语句

class PdbcTemplate(**args*, ***kwargs*)

基 类: *wordmarker.templates.pdbc_template.PdbcOperations*, *wordmarker.templates.pdbc_template.PdbcHelper*, *wordmarker.data.formatter.SqlFormatter*

操作数据库的模板

execute(*sql*, **args*, ***kwargs*)

注解: 使用 sqlalchemy 中的方法执行 sql，建议 sql 类型为 DDL（数据定义语言）时候使用

小技巧: 如果 sql 为 `select`，建议使用 query 方法

如果 sql 为 `update` ， `delete` ， `insert` ， 建议使用 update 方法

参数 sql –sql 语句

query(*sql*, **args*) → Union[pandas.core.frame.DataFrame, Iterator[pandas.core.frame.DataFrame]]

注解： 查询数据库

参数

- **sql** –sql 语句
- **args** –问号对应的值

返回

- 查询的数据

query_table(*table_name*, *schema=None*, *index_col=None*, *coerce_float=True*, *parse_dates=None*, *columns=None*, *chunksize: Optional[int] = None*) → Union[pandas.core.frame.DataFrame, Iterator[pandas.core.frame.DataFrame]]

注解： 使用 `pandas.read_sql_table` 方法，读取整张表的数据

小技巧： query_table 的参数详情 ， 请访问 pandas 官网 ， 了解更多信息

返回

- 查询的数据

update(*sql*, **args*)

注解： 更新数据库

参数

- **sql** –sql 语句
- **args** –问号对应的值

update_table(*data: pandas.core.frame.DataFrame*, *name: str*, *schema=None*, *if_exists: str = 'replace'*, *index: bool = False*, *index_label=None*, *chunksize=None*, *dtype=None*, *method=None*)

注解： 使用 pandas.to_sql 方法，将数据写入数据库中的一张表中

小技巧： update_table 的参数详情，请访问 pandas 官网，了解更多信息

参数 data –数据框

4.5.4 wordmarker.templates.word_template module

class AbstractConverter(*word_tpl:* wordmarker.templates.word_template.WordTemplate)

基类：object

此类是用来实现的，可以将 yaml 模板中的插值进行转换

定义的方法可以为 @staticmethod 修饰的方法，不能有任何参数，也可以为由 self 一个参数构成的方法

convert_to_dict() → dict

注解： 将读取的 yaml 模板中的内容转换为字典

返回

- yaml 模板中的内容，类型为字典

convert_to_str() → str

注解： 将读取的 yaml 模板中的内容转换为字符串

返回

- yaml 模板中的内容，类型为字符串

get_value(*prop*)

注解： 从转换后的 yaml 字典中，根据属性获取对应的值

加载多个 yaml 文件，排在后面的文件里的值，会覆盖前面的文件里的值

参数 prop –属性，用 . 分隔，例如，pdbc.engine.url

返回

- 转换后的 yaml 字典中对应的值

class DocxHelper(**args*, ***kwargs*)

基类：*wordmarker.loaders.default_resource_loader.DefaultResourceLoader*

通过读取配置文件，获取 docx 文件模板的相关信息

property docx

注解： 获取 docx 文件的绝对路径

返回

- yaml 文件中 data.docx.input.path 是目录，返回当前目录下 docx 文件的绝对路径
- yaml 文件中 data.docx.input.path 是文件，返回 docx 文件的绝对路径

property docx_in_path

注解： 通过读取 yaml 文件的 data.docx.input.path 属性，获取输入的 docx 文件或目录的绝对路径

返回

- docx 文件或目录的绝对路径

property docx_out_path

注解： 通过读取 yaml 文件的 data.docx.output.dir 属性，获取输出的 docx 目录的绝对路径

返回

- docx 目录的绝对路径

get_docx_file_name()

注解： 获取 docx 文件的文件名

返回

- 是 docx 文件，返回文件的名字
- 是目录，返回当前目录下的所有 docx 文件的文件名

class ImgHelper(**args*, ***kwargs*)

基类：*wordmarker.contexts.system_context.SystemContext*

通过读取配置文件，获取 img 文件的相关信息

clear_img()

注解： 清除 data.img.output.dir 属性对应的 img 的输出目录下的所有文件和目录

---

get_img_file(*img_name*)

注解： 根据图片的名字，获取图片的绝对路径

警告： 必须先将图片输出到输出目录下，才能获取到

参数 img_name –图片的名字

返回

- 图片的绝对路径

property img_out_path

注解： 通过读取 yaml 文件的 data.img.output.dir 属性，获取输出的 img 目录的绝对路径

返回

- img 目录的绝对路径

class TextHelper(**args*, ***kwargs*)

基类：object

通过读取配置文件，获取文本 yaml 文件的相关信息

get_value(*prop*)

注解： 从 yaml 字典中，根据属性获取对应的值

加载多个 yaml 文件，排在后面的文件里的值，会覆盖前面的文件里的值

参数 prop –属性，用 . 分隔，例如，pdbc.engine.url

返回

- yaml 字典中对应的值

get_yaml() → dict

注解： 获取从 yaml 文件中读取的数据，类型为 dict

返回

- path 为文件，返回一个字典，内容为 yaml 文件的内容
- path 为目录，返回一个嵌套的字典
  - key 为 yaml 文件的绝对路径
  - value 为 yaml 文件的内容，是一个字典

get_yaml_singleton()

注解： 获取从 data.text.input.path 属性中对应的路径下所有 yaml 文件中的数据，类型为 dict

加载多个 yaml 文件，排在后面的文件里的值，会覆盖前面的文件里的值

返回

- 返回一个字典，内容为所有 yaml 文件的内容

get_yaml_singleton_str()

注解： 获取从 `data.text.input.path` 属性中对应的路径下所有 yaml 文件中的数据，类型为 str，内容为一个字典

加载多个 yaml 文件，排在后面的文件里的值，会覆盖前面的文件里的值

返回

- 返回一个字符串，内容为一个字典，内容为所有 yaml 文件的内容

property text_in_path

注解： 通过读取 yaml 文件的 `data.text.input.path` 属性，获取输入的文本 yaml 文件的绝对路径

返回

- 文本 yaml 文件的绝对路径

class WordTemplate(**args*, ***kwargs*)

基 类：*wordmarker.creatives.builder.AbstractBuilder*, *wordmarker.templates.word_template.TextHelper*, *wordmarker.templates.word_template.ImgHelper*, *wordmarker.templates.word_template.DocxHelper*

操作 docx 文件的模板

append(*content*)

注解： 添加其他的 content 到全局的 content 中

参数 content –其他的 content，类型是 `dict`

返回

- self

build(*file_name=None*)

注解： 创建 docx 文件

参数 file_name –docx 文件的文件名

property tpl

注解： 获取 `DocxTemplate` 对象，

小技巧： `DocxTemplate` 对象的详细信息，请访问 python-docx-template 的文档

返回

- `DocxTemplate` 对象

4.6 wordmarker.utils package

作者 陈思祥

时间 2021 年 4 月

概述 当前模块是工具模块，有操作日志、文件等的工具类

1. `wordmarker.utils.logger`

   日志工具类。

2. `wordmarker.utils.yaml`

   yaml 文件工具类。

3. `wordmarker.utils.file`

   文件、路径工具类。

4.6.1 Submodules

4.6.2 wordmarker.utils.file module

class PathUtils(*src*, *tgt*)

基类：*wordmarker.contexts.system_context.SystemContext*

路径工具类

static filter_file(*file_list: Union[list, str]*, *suffix_list: Union[list, str]*)

注解： 通过后缀名，过滤文件

参数

- **file_list** –文件或文件列表
- **suffix_list** –后缀或后缀列表

返回

- 过滤后的文件或文件列表

get_relative_path()

注解： 将两个路径拼接起来

例子:

```
src = c:/a/b/c

tgt = ../../d/c.txt

final = c:/a/d/c.txt

--------------------------------

src = c:/a/b/c/foo.txt

tgt = ../../d/c.txt

final = c:/a/d/c.txt
```

返回

- 最终路径

4.6.3 wordmarker.utils.logger module

class LoggerFactory

基类：object

```
logger 工厂，获取 logger 对象
```

static get_logger(*logger_name=None*)

注解： 获取日志对象 logger

参数 logger_name –logger 名字

返回

- 日志对象 logger

log(*fun*)

注解： 装饰器，为类注入 logger 对象

小技巧： 在类的 __init__ 方法上加上 @log ，可以通过类的 self._logger 获取到 logger 对象，来打印日志

你也可以直接使用 LoggerFactory 获取 logger 对象，来打印日志

错误： 只能放在类的 __init__ 方法上，不能放在类上，放在类上会修改类的元类

如果当前类是被继承或继承了某个类，会导致元类冲突的异常，所以目前只支持放在 __init__ 方法上

4.6.4 wordmarker.utils.yaml module

class YamlUtils(**args*)

基类：object

yaml 文件工具类

get_value(*yaml_dict*, *prop_list*, *temp_prop*, *file_name*)

注解： 递归调用获取 yaml 文件中属性对应的值

参数

- **yaml_dict** –yaml 文件中的数据，以字典形式存放
- **prop_list** –属性列表，例如：pdbc.engine.url -> [pdbc, engine, url]
- **temp_prop** –临时属性，在日志中进行提示，例如：pdbc.engine.url
- **file_name** –yaml 文件名

返回

- yaml 文件中属性对应的值

Python 模块索引

c

d

l

t

u

索引

G

I

L

P

Q

R

S

T

U

W

Y

模块