Text Processing Library 说明文档

概述

Text Processing Library 是一个自主开发的可以相对灵活使用的Python库，提供全面的文本处理、文件操作和矩阵计算功能。该库采用模块化设计，支持灵活扩展，适用于各种文本数据处理场景。

额外说明：*此项目调用的三个第三方库：**pytest，setuptools以及coverage均是用于接口测试和类库打包，未应用于类库实际开发***。

安装

由于未将类库的tar.gz和whl包上传pypi，因此需要通过 pip 本地安装 Text Processing Library：

```
pip install ..\text_processing_lib-1.0.0-py3-none-any.whl
```

安装依赖

```
pip install -r requirements.txt
```

核心功能

文本处理

-**文本清洗**：移除特殊字符和标点
-**分词**：将文本拆分为单词列表
-**词频统计**：计算单词出现频率
-**关键词提取**：提取文本中最重要的关键词

文件处理

-**文本文件读取**：支持多种文本格式（.txt, .md, .log等）
-**CSV文件处理**：读取、解析和提取CSV数据
-**元数据提取**：获取文件大小、修改时间等信息
-**批量处理**：处理目录中的所有文件

矩阵操作

-**行/列操作**：获取、添加、更新、删除行或列
-**元素操作**：获取和更新矩阵元素
-**矩阵转置**：行列互换
-**矩阵转换**：转换为CSV、JSON等格式

设计模式应用

本库在设计中应用了多种经典设计模式：

1. 工厂模式 (Factory Pattern)
   -**应用**： ProcessorFactory类负责创建各种处理器实例
   -**优势**：解耦客户端代码与具体处理器实现，支持动态添加新处理器
2. 装饰器模式 (Decorator Pattern)
   -**应用**： LoggingDecorator为处理器添加日志功能

-**优势**：无需修改处理器核心逻辑即可添加额外功能

3. 组合模式 (Composite Pattern)
   -**应用**：CompositeProcessor将多个处理器组合成处理管道
   -**优势**：统一处理单个处理器和处理器集合的接口
4. 策略模式 (Strategy Pattern)
   -**应用**：不同的处理器实现相同接口但提供不同算法
   -**优势**：客户端可以根据需要灵活选择处理算法
5. 模板方法模式 (Template Method Pattern)
   -**应用**：TextProcessor抽象基类定义处理流程框架
   -**优势**：子类可以重写特定步骤而不改变算法结构

扩展维度

本库在多个维度支持扩展，以满足不同场景需求：

1. 处理器扩展
   扩展方式：创建新的处理器类并注册到ProcessorFactory
   示例：

```
class EmojiRemover(TextProcessor):
    def process(self, text: str) -> str:
        return ''.join(c for c in text if c.isalnum() or c.isspace())

TextProcessingAPI.register_processor("remove_emoji", EmojiRemover)
```

2. 文件格式支持
   扩展方式：继承FileProcessor并实现新文件格式的处理
   示例：

```
class JSONFileReader(FileProcessor):
    SUPPORTED_EXTENSIONS = ['.json']

    def process(self, file_path: str) -> dict:
        import json
        with open(file_path, 'r') as f:
            return json.load(f)
```

3. 矩阵操作扩展
   扩展方式：创建新的矩阵处理器实现特定算法
   示例：

```
class MatrixStatsProcessor(TextProcessor):
    def process(self, matrix: List[List[float]]) -> dict:
        return {
            "mean": sum(sum(row) for row in matrix) / (len(matrix) *
len(matrix[0])),
```

```
            "max": max(max(row) for row in matrix),
            "min": min(min(row) for row in matrix)
        }
```

4. 管道组合扩展
   扩展方式：创建自定义处理器组合，形成新的处理流程
   示例：

```
def create_sentiment_analysis_pipeline():
    return [
        "clean",
        "tokenize",
        "remove_stopwords",
        "sentiment_analyzer"
    ]
```

5. 输出格式扩展
   扩展方法：创建新的MatrixConverter子类支持更多输出格式
   示例：

```
class MatrixToHTMLConverter(MatrixConverter):
    def process(self, matrix: List[List[Any]]) -> str:
        html = "<table>\n"
        for row in matrix:
            html += "  <tr>"
            html += "".join(f"<td>{cell}</td>" for cell in row)
            html += "</tr>\n"
        html += "</table>"
        return html
```

Text Processing Library API 使用指南

下面是 Text Processing Library 的主要 API 方法及其使用说明：

API 方法	操作描述	示例用法
`process_text(text, pipeline)`	处理文本数据	`result = TextProcessingAPI.process_text("Hello, world!", ["clean", "tokenize"])` `# 输出：["Hello", "world"]`

API 方法	操作描述	示例用法
`process_file(file_path, pipeline)`	处理文件内容	`result = TextProcessingAPI.process_file("data.txt", [("text_file", {}), "clean"])` `# 输出：清洗后的文件内容`
`process_matrix(matrix, pipeline)`	处理矩阵数据	`result = TextProcessingAPI.process_matrix([[1,2],[3,4]], ["matrix_transpose"])` `# 输出：[[1,3],[2,4]]`
`create_pipeline(operations)`	创建处理管道	`pipeline = TextProcessingAPI.create_pipeline(["clean", "tokenize"])` `result = pipeline.process("Hello, world!")`
`list_available_processors()`	列出可用处理器	`processors = TextProcessingAPI.list_available_processors()` `# 输出：["clean", "tokenize", "csv_file", ...]`
`register_processor(name, processor_class)`	注册新处理器	`TextProcessingAPI.register_processor("uppercase", UppercaseProcessor)`
`set_logging(enabled)`	启用/禁用日志	`TextProcessingAPI.set_logging(True)`

API 方法	操作描述	示例用法
`get_matrix_row(matrix, index)`	获取矩阵行	`row = TextProcessingAPI.get_matrix_row([[1,2],[3,4]], 0)` `# 输出：[1,2]`
`add_matrix_row(matrix, row)`	添加矩阵行	`new_matrix = TextProcessingAPI.add_matrix_row([[1,2]], [3,4])` `# 输出：[[1,2],[3,4]]`
`update_matrix_row(matrix, index, row)`	更新矩阵行	`new_matrix = TextProcessingAPI.update_matrix_row([[1,2],[3,4]], 0, [0,0])` `# 输出：[[0,0],[3,4]]`
`delete_matrix_row(matrix, index)`	删除矩阵行	`new_matrix = TextProcessingAPI.delete_matrix_row([[1,2],[3,4]], 0)` `# 输出：[[3,4]]`
`get_matrix_column(matrix, index)`	获取矩阵列	`col = TextProcessingAPI.get_matrix_column([[1,2],[3,4]], 1)` `# 输出：[2,4]`
`add_matrix_column(matrix, column)`	添加矩阵列	`new_matrix = TextProcessingAPI.add_matrix_column([[1],[3]], [2,4])` `# 输出：[[1,2],[3,4]]`
`update_matrix_column(matrix, index, column)`	更新矩阵列	`new_matrix = TextProcessingAPI.update_matrix_column([[1,2],[3,4]], 0, [0,0])` `# 输出：[[0,2],[0,4]]`

API 方法	操作描述	示例用法
`delete_matrix_column(matrix, index)`	删除矩阵列	`new_matrix = TextProcessingAPI.delete_matrix_column([[1,2],[3,4]], 0)` `# 输出：[[2],[4]]`
`get_matrix_element(matrix, row, column)`	获取矩阵元素	`element = TextProcessingAPI.get_matrix_element([[1,2],[3,4]], 1, 0)` `# 输出：3`
`update_matrix_element(matrix, row, column, value)`	更新矩阵元素	`new_matrix = TextProcessingAPI.update_matrix_element([[1,2],[3,4]], 0, 1, 5)` `# 输出：[[1,5],[3,4]]`
`matrix_to_csv(matrix, delimiter=',')`	矩阵转CSV	`csv_str = TextProcessingAPI.matrix_to_csv([["Name","Age"],["Alice",30]])` `# 输出："Name,Age\r\nAlice,30\r\n"`

处理器参数说明

在创建管道时，可以通过元组形式传递处理器参数：

```
pipeline = [
    ("clean", {"remove_numbers": True}),  # 清洗时移除数字
    ("tokenize", {"split_char": " "}),     # 使用空格分词
    ("keywords", {"top_k": 5})             # 提取前5个关键词
]
```

常用处理器参数

处理器名称	可用参数	默认值	描述

`clean`	`remove_numbers`
	`remove_punctuation`
	`lowercase`
		`True`
		`True`
		`False`
			文本清洗选项

处理器名称	可用参数	默认值	描述
tokenize	split_char split_words	None True	分词选项
keywords	top_k min_length	10 3	关键词提取选项
text_file	encoding errors	'utf-8' 'strict'	文本文件读取选项
csv_file	encoding delimiter has_header	'utf-8' ',' False	CSV文件读取选项
csv_extract	column_index output_format	0 'text'	CSV列提取选项
matrix_row	operation index row	'get' None None	矩阵行操作
matrix_col	operation index column	'get' None None	矩阵列操作
matrix_element	operation row column value	'get' None None None	矩阵元素操作
matrix_transpose	-	-	矩阵转置
matrix_convert	output_format row_separator col_separator	'text' '|' ','	矩阵转换选项