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

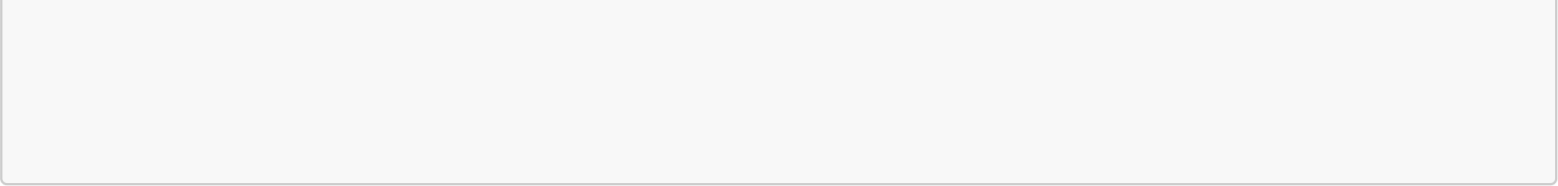
**示例**：

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：

**操 作 描 述**

**API 方法**

**示例用法**

处 理 文 本 数 据

**process\_text(text, pipeline)**

result = TextProcessingAPI.process\_text("Hello, world!", ["clean", "tokenize"])

# 输出 : ["Hello", "world"]

**操 作 描 述**

**API 方法**

**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) |

**操 作 描 述**

**API 方法**

**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]] |

**操 作 描 述**

**API 方法**

**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]] |

矩 阵 转

**matrix\_to\_csv(matrix, delimiter=',')**

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 |