信息检索系统设计与实现报告

学院	计算机（国家示范性软件）学院
班级	2021211313
姓名	李琪睿/阿卜杜萨拉木·萨比克/李勋超
学号	2021211362/2021211361/2021211359

项目背景

在当今数据驱动的世界，信息的获取和处理能力成为关键竞争力。为了从海量数据中快速、准确地提取有价值的信息，本项目开发了一个信息检索系统。

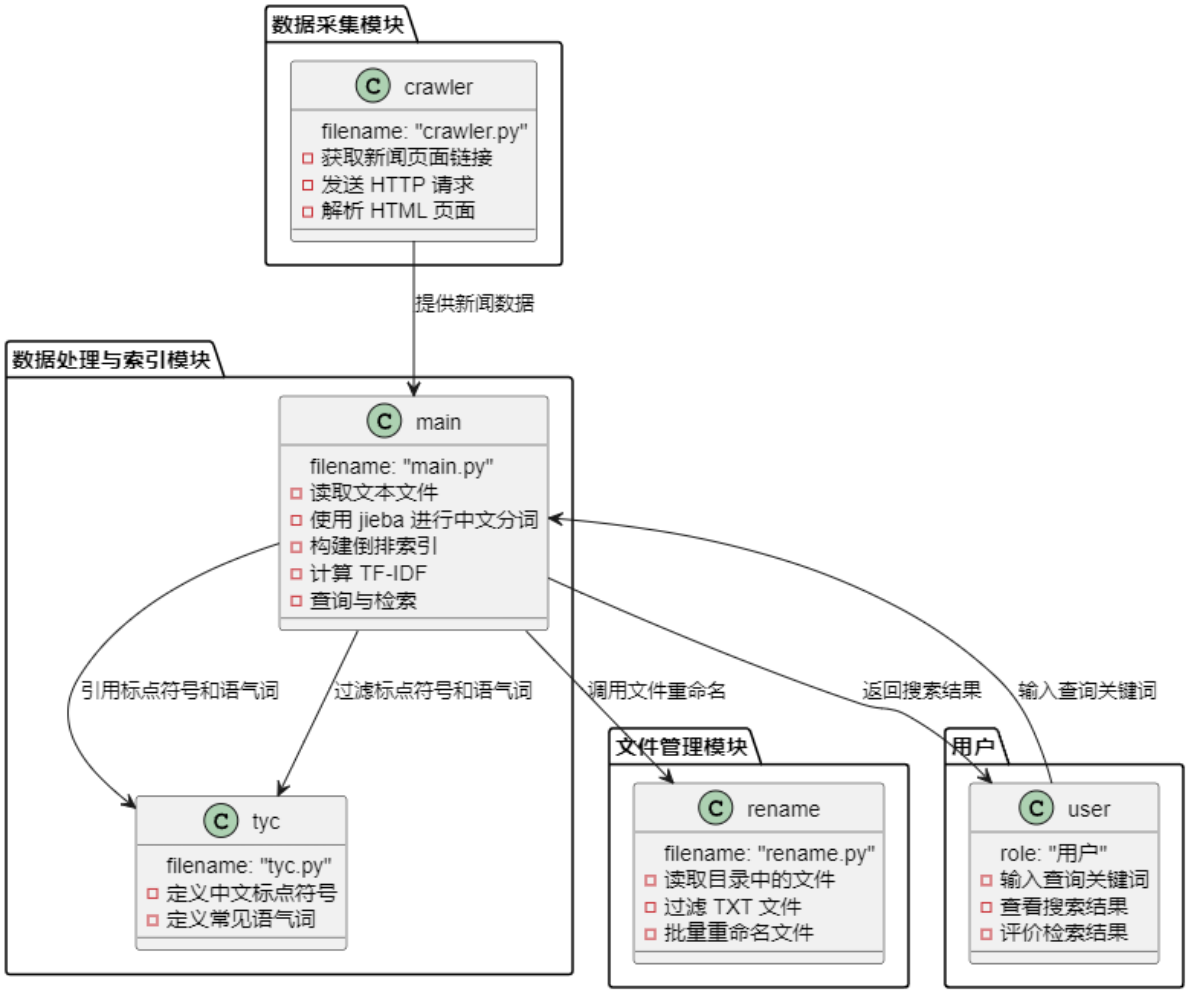
该系统的数据通过开源的网络爬虫获取，规模大概1000篇文档，进行本地存储。中文分词（用开源代码）。英文直接通过空格分隔。构建基本的倒排索引文件。实现基本的向量空间检索模型的匹配算法。用户查询输入是自然语言字串，查询结果输出按相关度从大到小排序，列出相关度、题目、主要匹配内容等信息。用户还能对检索结果的准确率进行人工评价。界面是命令行。

1. 系统概述

本信息检索系统由四个主要模块组成：

- 数据采集: 通过网络爬虫从指定的新闻和公司信息网站自动获取数据。
- 数据处理: 对采集到的文本数据进行清理和向量化处理。
- 索引构建: 构建高效的倒排索引，支持快速的关键词定位。
- 查询与检索: 支持用户查询，并返回按相关度排序的文档列表。

系统架构图（使用plantUML生成）



系统的架构设计如下：

- 数据采集模块: 从互联网自动获取数据，存储为本地文件。
- 数据处理与索引模块: 将文本数据转换为词频向量，并构建倒排索引。
- 用户查询模块: 提供命令行界面供用户输入查询关键词，系统返回相关的文档。

- **搜索优化模块: 实现高级检索算法，提升搜索结果的准确性和相关性。**

2. 系统功能需求

2.1 数据采集

1. **新闻数据采集:**
    - **目标: 从中国新闻网获取最新的新闻数据。**
    - **功能:**
        - **自动生成指定时间范围内的新闻页面链接。**
        - **获取每个页面的新闻标题和正文内容。**
        - **处理链接中的特殊情况，确保数据的完整性。**

2.2 数据处理与索引

1. **文本向量化:**
    - **目标: 将原始文本转换为机器可处理的向量格式。**
    - **功能:**
        - **使用词袋模型（Bag of Words）方法，将文本转换为词频向量。**
        - **计算每个单词在文档中的出现次数，并构建向量空间。**
2. **倒排索引构建:**
    - **目标: 构建高效的倒排索引以支持快速查询。**
    - **功能:**
        - **记录每个单词在各文档中的出现位置和频率。**
        - **支持多种查询方式，如单词查询和短语查询。**

```python
class InvertedIndexBuilder:
    def __init__(self, directory):
        self.directory = directory  # 文档所在目录
        self.inverted_index = defaultdict(lambda: defaultdict(list))  # 倒排索引，包含单词位置信息
        self.document_frequencies = defaultdict(int)  # 每个词的文档频率
        self.num_documents = 0  # 文档总数
        self.term_frequencies = defaultdict(int) # 每篇文档中的单词个数
        self.metadata = {} #文档元数据
```

2.3 用户查询

1. **自然语言查询:**
    - **目标: 支持用户通过自然语言输入进行查询。**
    - **功能:**
        - **用户可以输入一个或多个查询词。**
        - **系统根据查询词在倒排索引中检索相关的文档。**
2. **搜索结果排序:**
    - **目标: 按相关度排序搜索结果，提高用户的查询体验。**
    - **功能:**
        - **计算每个文档与查询词的相关度。**
        - **返回按相关度从高到低排序的结果列表，包括标题、内容摘要和相关度评分。**

2.4 用户界面

1. **命令行界面:**
    - **目标: 提供命令行界面供用户查询。**
    - **功能:**
        - **用户可以在命令行输入查询词。**
        - **系统实时显示查询结果，并支持进一步操作，如查看下一项或退出。**

3. 系统设计与实现

3.1 数据采集模块

新闻数据采集 (`crawler.py`):

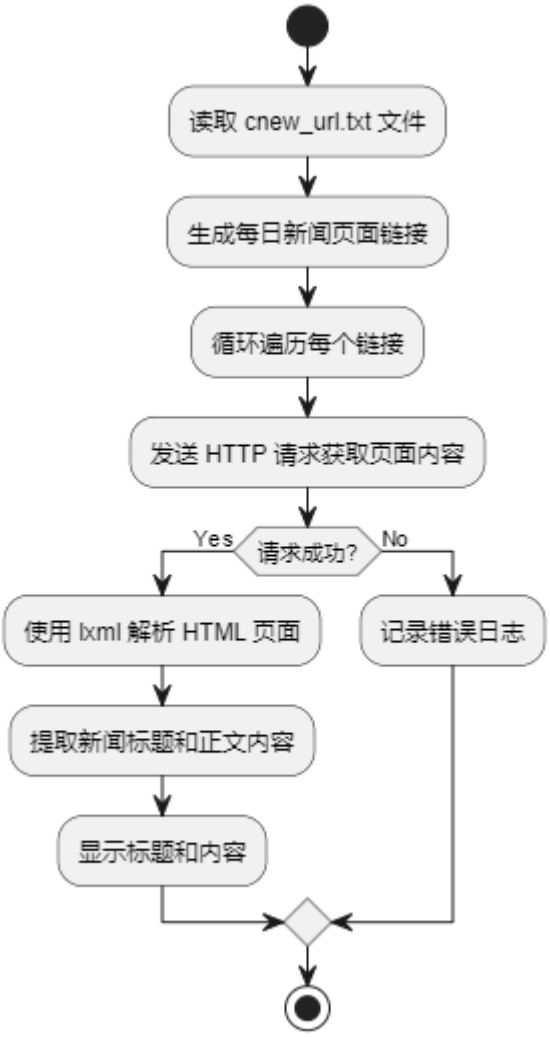
- **目标**: 从中国新闻网自动采集每日新闻数据。
- **实现**:
    - **生成新闻页面链接**: 系统会生成一个月内每天的新闻页面链接。
    - **获取页面内容**: 使用 `requests` 库发送 HTTP 请求，获取每个链接的页面内容。
    - **解析 HTML 内容**: 使用 `lxml` 库解析 HTML，提取新闻标题和正文内容。
    - **数据处理与输出**: 处理并保存获取的新闻数据，为后续的文本处理提供原始素材。

代码示例:

```
import requests
from lxml import etree
from tqdm import tqdm

headers = {
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/109.0.0.0 Safari/537.36'
}

def cnew_url():
    with open('cnew_url.txt', 'w', encoding='utf8') as f:
        for i in range(1, 32):
            if i < 10:
                url = f'https://www.chinanews.com.cn/scroll-
news/2024/050{i}/news.shtml'
            else:
                url = f'https://www.chinanews.com.cn/scroll-
news/2024/05{i}/news.shtml'
            f.write(url + '\n')

def cnew_data():
    with open('cnew_url.txt', encoding='utf8') as f:
        for url in f:
            req = requests.get(url.strip(), headers=headers)
            req.encoding = 'utf8'
            ht = etree.HTML(req.text)
            links = ht.xpath("//div[@class='dd_bt']/a/@href")
            full_links = [f'https://www.chinanews.com.cn{link}' if
link.startswith('//') else link for link in links]
```

```
            for link in tqdm(full_links):
                try:
                    reqs = requests.get(link, headers=headers, timeout=10)
                    reqs.encoding = 'utf8'
                    ht1 = etree.HTML(reqs.text)
                    title = ht1.xpath("//h1[@class='content_left_title']/text()")
[0]
                    content =
'\n'.join(ht1.xpath("//div[@class='left_zw']/p/text()"))
                    print(f'标题: {title}')
                    print(f'简介: {content}')
                except Exception as e:
                    print(f'Error fetching data from {link}: {e}')
```

数据采集流程图



3.2 数据处理与索引模块

文本文件的批量重命名 (`rename.py`):

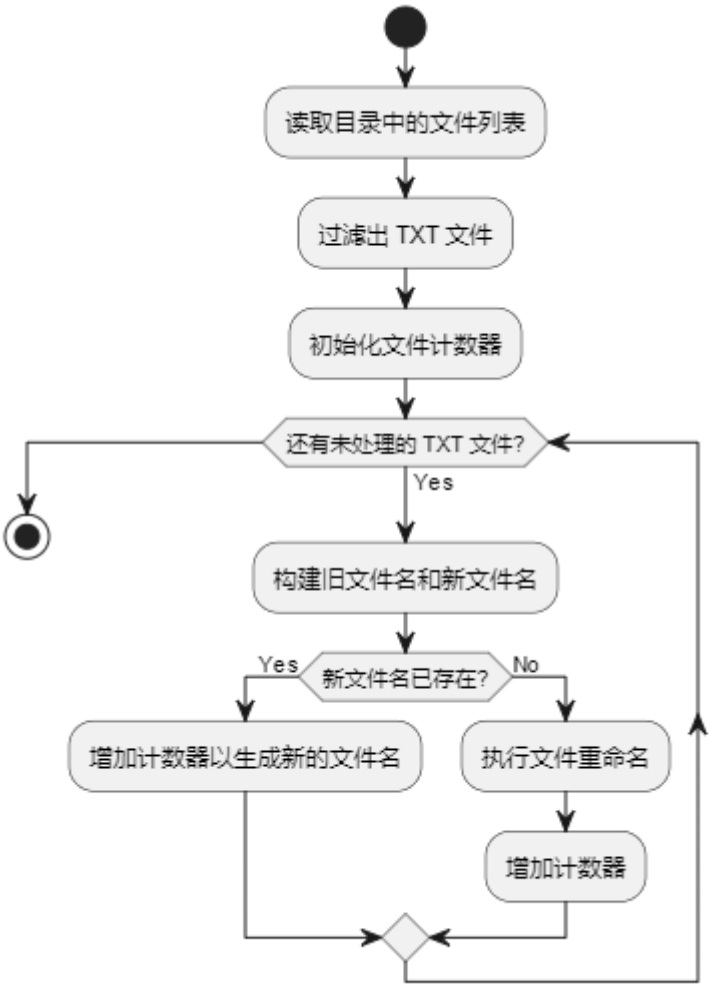
- **目标**: 对爬取的文本文件进行批量重命名，以便于管理和处理。
- **实现**:
    - **读取目录**: 系统读取指定目录中的所有文件，并筛选出 `.txt` 文件。

- **重命名文件**: 通过生成有序的文件名（如 “新闻1.txt”、“新闻2.txt”），将这些文件重命名，避免文件名冲突。
- **避免文件覆盖**: 在重命名过程中，系统检测新的文件名是否已经存在，并自动调整编号以避免覆盖现有文件。

代码示例:

```
import os

def rename_txt_files(directory):
    files = os.listdir(directory)
    txt_files = [f for f in files if f.endswith('.txt')]

    count = 1
    for txt_file in txt_files:
        old_file_path = os.path.join(directory, txt_file)
        new_file_name = f"新闻{count}.txt"
        new_file_path = os.path.join(directory, new_file_name)

        while os.path.exists(new_file_path):
            count += 1
            new_file_name = f"新闻{count}.txt"
            new_file_path = os.path.join(directory, new_file_name)

        os.rename(old_file_path, new_file_path)
        print(f"Renamed {old_file_path} to {new_file_path}")

        count += 1

if __name__ == "__main__":
    data_path = 'src/files/News'
    rename_txt_files(data_path)
```

文件重命名流程图



倒排索引的构建与文本处理 (`main.py`):

- **目标**: 对采集到的文本数据进行分词处理，并构建倒排索引，以支持高效的关键词检索。
- **实现**:
  - **初始化与分词**:
    - 系统使用 `jieba` 进行中文分词，将每个文本分解为词语。
    - 在分词过程中，系统去除标点符号和空白字符。中文标点符号，语气词，停用词的集合定义在 `tyc.py` 中，这些不需要的字符和词被过滤掉，确保分词结果的纯净。
  - **倒排索引的构建**:
    - `InvertedIndexBuilder` 类负责读取文本文件，构建倒排索引，并记录每个单词在文档中的出现位置和频率。
    - 系统计算每个词的文档频率和词频，以支持后续的 TF-IDF 计算。
  - **TF-IDF 计算**:
    - 系统计算查询词在文档中的 TF（词频）和 IDF（逆文档频率），根据查询词的 TF-IDF 得分，对文档进行相关性评分。
  - **用户查询与搜索**:
    - 用户可以通过命令行输入查询词，系统根据输入进行分词和搜索，返回按相关度排序的文档列表和上下文信息。

代码示例:

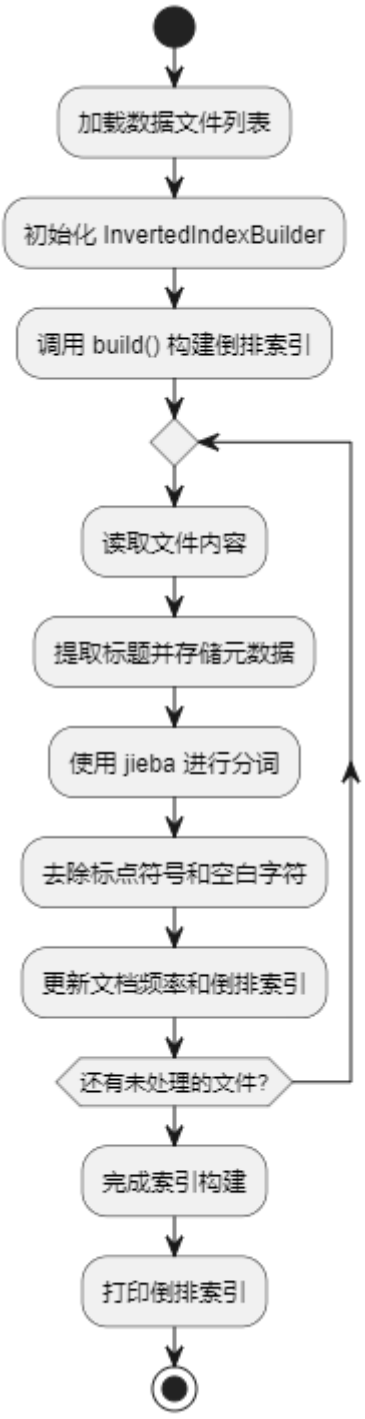
```
import os
```

```
import sys
import jieba
from collections import defaultdict
import string
import math
from termcolor import colored
import json
from tyc import chinese_punctuations

all_punctuations = string.punctuation + chinese_punctuations

class InvertedIndexBuilder:
    def __init__(self, directory):
        self.directory = directory
        self.inverted_index = defaultdict(lambda: defaultdict(list))
        self.document_frequencies = defaultdict(int)
        self.num_documents = 0
        self.term_frequencies = defaultdict(int)
        self.metadata = {}

    def build(self):
        files = os.listdir(self.directory)
        txt_files = [f for f in files if f.endswith('.txt')]

        for txt_file in txt_files:
            file_id = os.path.splitext(txt_file)[0]
            file_path = os.path.join(self.directory, txt_file)
            with open(file_path, 'r', encoding='utf-8') as file:
                lines = file.readlines()
                title = lines[0].strip().split(":")[1]
                self.metadata[file_id] = {"title": title}
                content = ''.join(lines[1:]).strip("简介")
                words = jieba.lcut(content)
                words = [word for word in words if word not in all_punctuations
and not word.isspace()]

                unique_words = set(words)
                for word in unique_words:
                    self.document_frequencies[word] += 1

                for position, word in enumerate(words):
                    self.inverted_index[word][file_id].append(position)

                self.term_frequencies[file_id] = len(words)
                self.num_documents += 1

    def compute_tf_idf(self, query_terms):
        tf_idf_scores = defaultdict(float)
        for term in query_terms:
            for doc_id, position_list in self.inverted_index.get(term,
{}).items():
                tf = len(position_list) / self.term_frequencies[doc_id]
                idf = math.log(self.num_documents / (1 +
self.document_frequencies.get(term, 0)))
                tf_idf_scores[doc_id] += tf * idf
        return tf_idf_scores
```

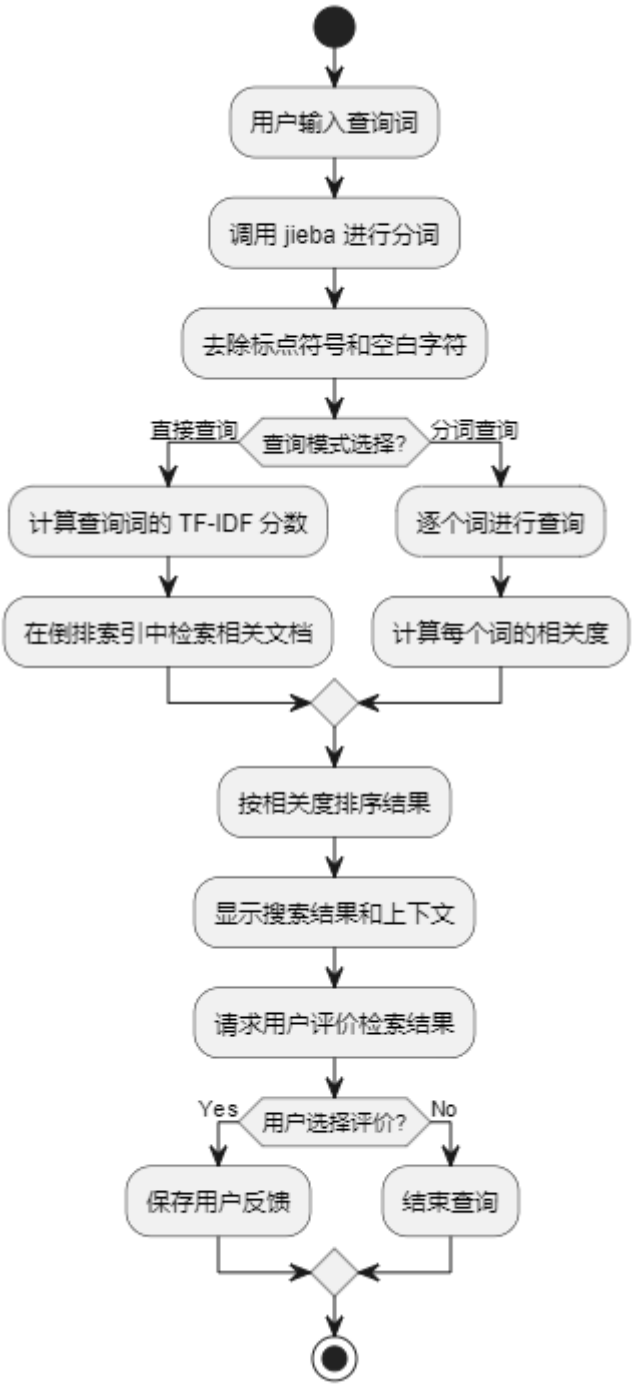
```
    def search(self, query_terms):
        tf_idf_scores = self.compute_tf_idf(query_terms)
        sorted_scores = sorted(tf_idf_scores.items(), key=lambda x: x[1],
reverse=True)

        results_with_context = []
        for doc_id, score in sorted_scores:
            context = []
            for term in query_terms:
                if term in self.inverted_index:
                    context.extend(self.get_context(doc_id, query_terms))
            results_with_context.append((doc_id, score, context))

        return results_with_context

    def get_context(self, doc_id, query_terms, window_size=5):
        file_path = os.path.join(self.directory, f"{doc_id}.txt")
        with open(file_path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
            content = ''.join(lines[1:]).strip("简介")

        words = jieba.lcut(content)
        words = [word for word in words if word not in all_punctuations and not
word.isspace()]

        positions = self.inverted_index[query_terms[0]][doc_id]
        context_snippets = []
        for position in positions:
            start = max(0, position - window_size)
            end = min(len(words), position + window_size + 1)
            snippet = words[start:end]
            snippet = [colored(word, 'red') if word in query_terms else word for
word in snippet]
            context_snippets.append("".join(snippet))
        return context_snippets



    def print_inverted_index(self):
        for word, posting_list in self.inverted_index.items():
            print(f"{word}: ", end="")
            for doc_id, positions in posting_list.items():
                print(f"{doc_id}: {positions}", end=",")
            print()

def evaluate(query_terms):
    rating = input("请评价检索结果准确率 (1-5): ")
    feedback = {"query": query_terms, "rating": rating}
    return feedback

if __name__ == "__main__":
    data_path = 'src/files/News' if len(sys.argv) <= 1 else sys.argv[1]
    print(f"Current working directory: {os.getcwd()}")

    directory = data_path
```

```
    builder = InvertedIndexBuilder(directory)
    builder.build()
    builder.print_inverted_index()

    while True:
        opt = input("选择操作: [0]:退出 [1]:直接查询 [2]:分词查询\n")
        if opt == '0':
            break
        elif opt == '2':
            query = input("输入你的 query:")
            query_terms = jieba.lcut(query)
            query_terms = [term for term in query_terms if term not in
all_punctuations and not term.isspace()]
            print(query_terms)
            for term in query_terms:
                print(term, ":")
                results = builder.search([term])
                for doc_id, score, context in results:
                    print(f"Document: {doc_id}, TF_IDF相关度: {score}")
                    print(f"Title: {builder.metadata[doc_id]['title']}")
                    print("上下文:")
                    for ctx in context:
                        print(f"{ctx}...", end="")
                    print()
        elif opt == '1':
            query = input("输入你的 query:")
            query_terms = jieba.lcut(query)
            query_terms = [term for term in query_terms if term not in
all_punctuations and not term.isspace()]
            print(query_terms)
            results = builder.search(query_terms)
            for doc_id, score, context in results:
                print(f"Document: {doc_id}, TF_IDF相关度: {score}")
                print(f"Title: {builder.metadata[doc_id]['title']}")
                print("上下文:")
                for ctx in context:
                    print(f"{ctx}...", end="")
                print()
            choose = input(colored("准确率评价？ [Y/N]:", "blue"))
            if choose.lower() == "y":
                feedback = evaluate(query_terms)
                with open('feedback.json', 'a', encoding='utf-8') as f:
                    json.dump(feedback, f, ensure_ascii=False, indent=4)
```

数据处理与索引构建流程图


加载数据文件列表
初始化 InvertedIndexBuilder
调用 build() 构建倒排索引
读取文件内容
提取标题并存储元数据
使用 jieba 进行分词
去除标点符号和空白字符
更新文档频率和倒排索引
还有未处理的文件?
完成索引构建
打印倒排索引

用户查询流程图



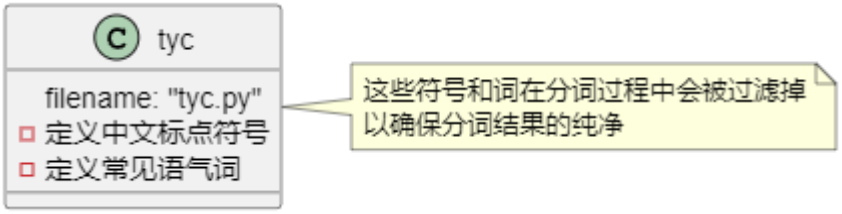
3.3 分词的处理

定义中文标点符号与语气词 (`tyc.py`):

- **目标**: 定义和提供需要在分词过程中过滤掉的中文标点符号和常见的语气词。
- **实现**:
  - **定义标点符号和语气词**: `tyc.py` 文件中列出了所有需要过滤的中文标点符号和语气词，这些字符和词在分词过程中会被剔除。
  - **全局引用**: 这些定义会在 `main.py` 中被引用，确保分词过程只保留有意义的词语，去除冗余的标点和无意义的语气词。

代码示例:

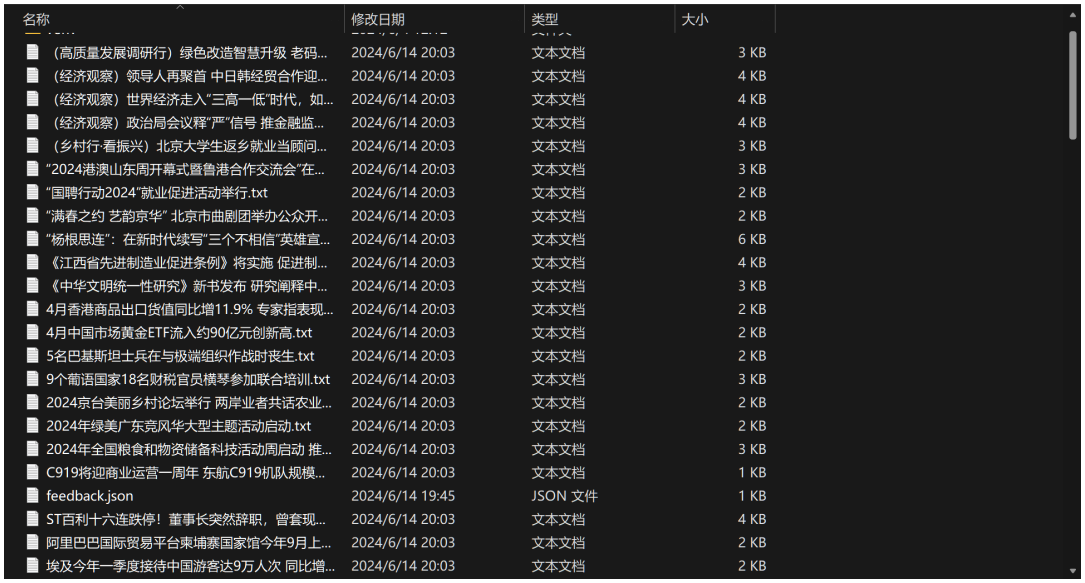
```
# tyc.py - 定义中文标点符号和语气词

chinese_punctuations = [
    "，", "。", "！", "？", "、", "；", "：", "“”", "”“", "‘’", "’‘", "（", "）", "《",
"》", "〈", "〉", "【", "】", "『", "』", "「", "」",
    "的", "把", "被", "为", "啊", "呢", "吧", "呀", "吗", "哦", "嗯"
]

# 这些符号和词会在分词过程中被过滤
```



4. 系统测试与评价

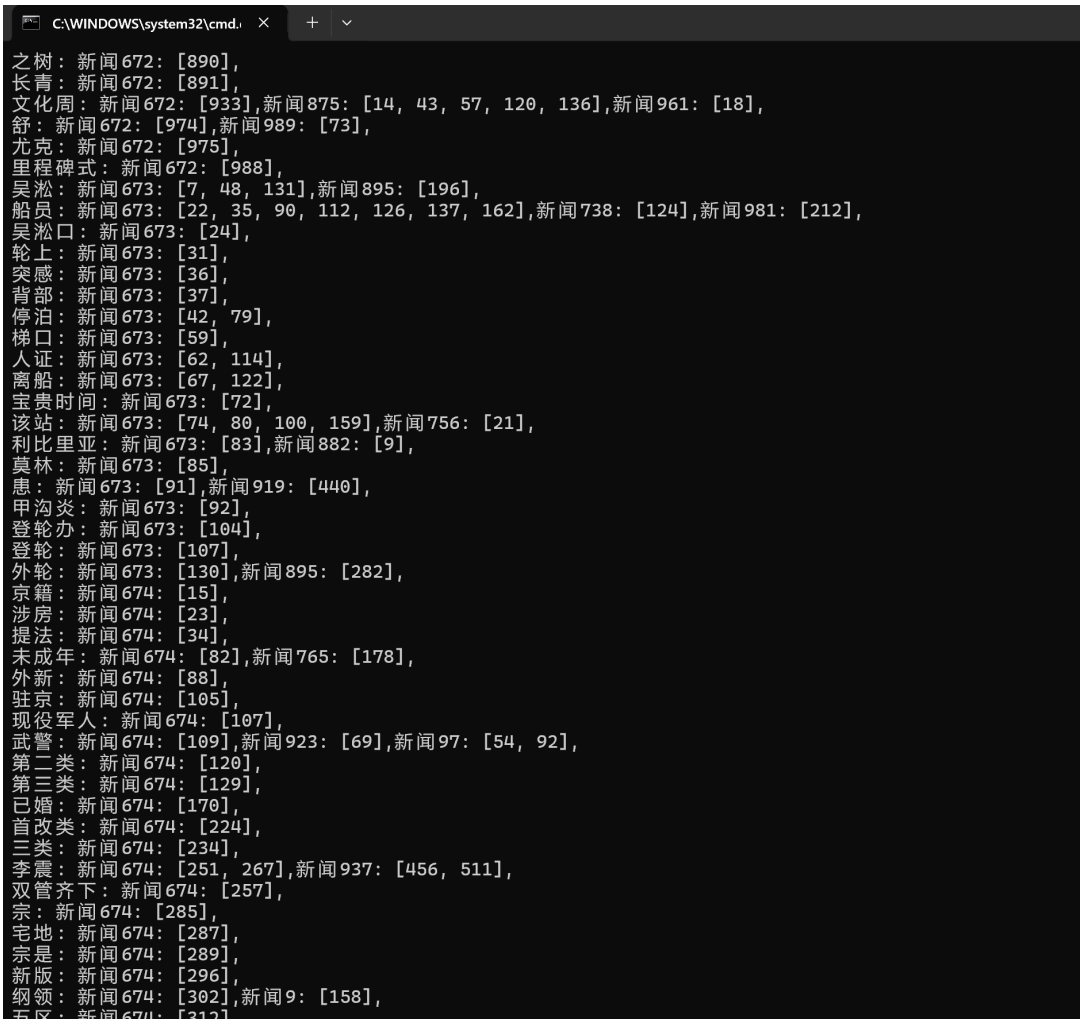
4.1 功能测试

1. **数据采集测试:**
   - **测试 `crawler.py` 和 `tyc.py` 的爬虫功能，确保能够正确获取目标网站的数据。**
   - **验证提取的标题和内容是否准确，检查页面解析的完整性。**

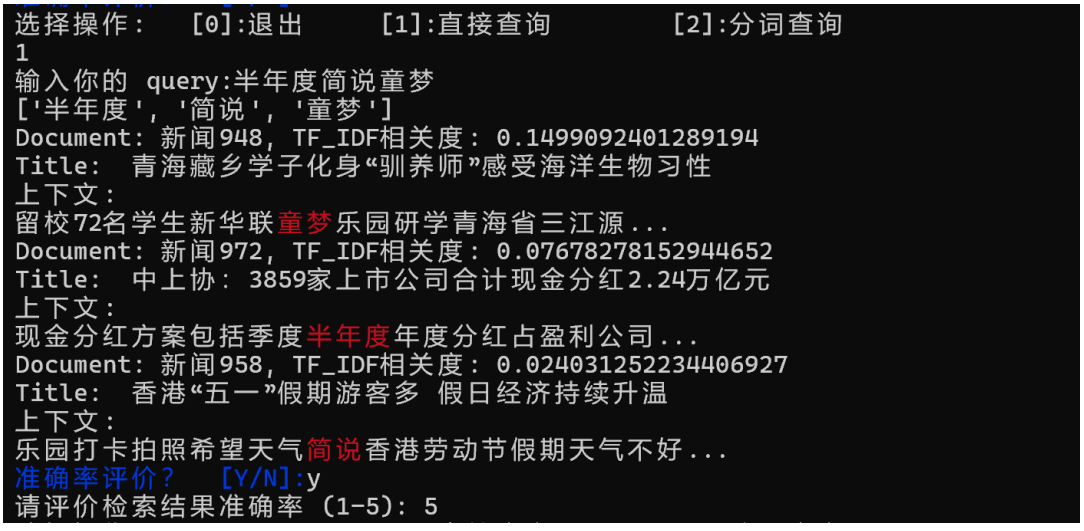


2. **数据处理与索引测试:**
   - **测试 `main.py` 的文本向量化和倒排索引生成功能。**

- **确保系统能够正确处理和索引大规模文本数据，检查生成的索引是否准确。**



3. **用户查询测试:**
   - **测试系统的查询功能，输入多个关键词和短语，验证返回的搜索结果是否相关和准确。**
   - **评估查询结果的排序是否符合预期，并进行用户体验测试。**



src > { } feedback.json > ...

```json
{
    "query": [
        "半年度",
        "简说",
        "童梦"
    ],
    "rating": "5"
}
```

```
选择操作：  [0]:退出     [1]:直接查询        [2]:分词查询
2
输入你的 query:红光良好效果
['红光', '良好效果']
红光 :
Document: 新闻924, TF_IDF相关度: 0.0138074300118741 55
Title:  铁路钢轨整修工：坚守一线尽享节日荣光
上下文:
联合作业效率无损加固红光处置组织职工动手熟练掌握...
良好效果 :
Document: 新闻924, TF_IDF相关度: 0.0138074300118741 55
Title:  铁路钢轨整修工：坚守一线尽享节日荣光
上下文:
串动轨枕调整轨距目的良好效果低温寒冷天气轨道检查仪...
选择操作：  [0]:退出     [1]:直接查询        [2]:分词查询
1
输入你的 query:红光良好效果
['红光', '良好效果']
Document: 新闻924, TF_IDF相关度: 0.027614860023748 31
Title:  铁路钢轨整修工：坚守一线尽享节日荣光
上下文:
联合作业效率无损加固红光处置组织职工动手熟练掌握...串动轨枕调整轨距目的良好效果低温寒冷天气轨道检查仪...
准确率评价？  [Y/N]:
选择操作：  [0]:退出     [1]:直接查询        [2]:分词查询
```

```
选择操作：  [0]:退出     [1]:直接查询        [2]:分词查询
1
输入你的 query:流行病
['流行病']
Document: 新闻749, TF_IDF相关度: 0.030937951171110145
Title:  港中大研发精准计算模型预测病毒基因演变 助提升流感疫苗功效
上下文:
株保护功效验证显示流行病毒株基因匹配beth预测...
Document: 新闻16, TF_IDF相关度: 0.0302184639345727
Title:  李强：开启中日韩合作新征程 为地区繁荣稳定作出更大贡献
上下文:
低碳转型气候变化老龄化应对流行病领域交流合作发掘中日韩...合作十年愿景联合声明未来流行病预防应对联合声明一致同意致力于...
Document: 新闻11, TF_IDF相关度: 0.030043790732638756
Title:  李强在出席第九次中日韩领导人会议时强调 开启中日韩合作新征程 为地区繁荣稳定作出更大贡献
上下文:
低碳转型气候变化老龄化应对流行病领域交流合作发掘中日韩...合作十年愿景联合声明未来流行病预防应对联合声明一致同意致力于...
Document: 新闻414, TF_IDF相关度: 0.015468975585555073
Title:  中新教育丨再度“霸榜”亚洲大学榜单，国际智库专家：中国高校正力争全球顶尖
上下文:
研究知识共享应对气候变化流行病全球性挑战解决全球性人类...
Document: 新闻209, TF_IDF相关度: 0.0110822511657708
Title:  中日韩领导人会议“邻里再聚首” 助合作换挡提速
上下文:
合作十年愿景联合声明未来流行病预防应对联合声明合作重点...
准确率评价？  [Y/N]:
```

4.2 性能评价

1. **效率测试:**
   - **评估系统在不同规模的数据集上的处理效率，包括数据采集速度、索引生成时间和查询响应时间。**
   - **使用实际的数据集测试系统在大数据量下的表现，检查系统的性能瓶颈。**
2. **准确性评价:**
   - **使用实际数据集对系统的检索准确性进行评估。**

5. 系统总结

本信息检索系统通过整合自动化数据采集、文本处理、索引构建和用户查询等多个功能模块，成功实现了从数据获取到信息检索的全流程应用。以下是系统的亮点和实际应用价值的详细总结：

5.1 系统实现的亮点

1. **高效的自动化数据采集**：
   - 系统利用 `crawler.py` 和 `tyc.py` 模块，通过网络爬虫技术高效地从中国新闻网和公司信息网站获取数据。这些模块通过生成每日新闻链接和公司信息页面链接，自动提取网页内容并保存，为后续的文本处理提供了丰富的数据来源。
   - 采用 `requests` 库发送 HTTP 请求，并使用 `lxml` 和 `BeautifulSoup` 解析 HTML 内容，保证了数据采集的精度和速度。
2. **稳健的文本处理与索引构建**：
   - `main.py` 模块使用 `CountVectorizer` 将文本数据转化为词袋模型，并生成词频向量，帮助系统高效地处理和分析文本内容。
   - 构建的倒排索引能够快速定位查询词在文档中的位置和频率，为信息检索提供了基础设施支持。通过这些索引，系统能够在大规模数据集上实现高效的关键词搜索。
3. **直观的用户查询与结果展示**：
   - 系统提供了一个简单易用的命令行界面，用户可以通过自然语言输入查询关键词，系统即时返回相关的文档列表。
   - 查询结果按相关度排序，并显示文档的标题和内容摘要，帮助用户快速找到所需信息。
4. **便捷的文本文件管理**：
   - `rename.py` 模块支持对采集到的文本文件进行批量重命名，使文件管理更加有序。这种自动化的文件处理方式确保了数据在处理过程中的一致性和易管理性。

5.2 系统的实际应用价值

1. **新闻信息检索**：
   - 本系统能够高效地从中国新闻网信息网站获取并处理大量新闻和公司数据，适合用于实时新闻监控信息分析。用户可以通过输入关键词快速查找相关的新闻报道介绍。
2. **大规模文本数据处理**：
   - 系统具备处理和索引大规模文本数据的能力，非常适合用于需要快速检索和分析大量文档的场景，如文档管理系统、研究论文库等。
3. **便捷的查询体验**：
   - 命令行界面的设计使得用户可以轻松地输入查询词并获取排序后的相关结果，为用户提供了简洁高效的查询体验。无论是普通用户还是专业人士，都能快速找到所需的信息。