基于 Hbase 的 USTC 官方文件查询系统

殷腾　PB20030785
李新涛　PB21151754

2024 年 12 月 27 日

摘要

本项目基于 Hbase 数据库，实现了一个 USTC 官方文件查询系统。通过预先爬虫爬取 USTC 官方网站的文件，将文件存储在 Hbase 数据库中，并提供了一个简单的界面供用户查询。用户可以通过关键词检索文件，系统将返回相关文件的信息。系统支持对文件信息的排序，用户可以根据搜索词与文件关联性对文件进行排序。系统还支持对文件的下载，用户可以通过界面下载文件到本地进行查看。系统的实现主要分为数据库与界面等部分。系统的实现过程中，我们使用了 Python 语言，Hbase 数据库，PyQt5 等工具。最终系统的实现效果良好，用户可以通过界面方便地查询、下载 USTC 官方文件。

目录

§1 项目简介

本项目基于 Hbase 数据库，实现了一个 USTC 官方文件查询系统。通过按照一定格式爬取文件后，系统会将其存储在 Hbase 数据库中。用户可以通过系统提供的图形化界面输入关键词，系统将返回相关文件的信息。用户可以通过界面查看文件的信息，并下载文件到本地进行查看，避免了用户在 USTC 官方网站上进行繁琐的查找。

§2 项目框架

2.1 界面

界面使用 PyQt5 实现，主要包括一个检索输入框和结果展示列表。当用户输入关键词后，系统会在 Hbase 数据库中检索相关文件，并将相关文件的各项信息结果展示在列表中。同时列表中的结果支持直接下载，用户可以通过点击列表中的文件名下载文件到本地进行查看。除此之外，若用户输入的关键词为空或者无相关文件，系统会给出提示信息。

2.2 检索

检索功能主要通过 Hbase 数据库实现，使用 Python 第三方库 happybase 连接 Hbase 数据库。系统会在 Hbase 数据库中检索相关文件，将检索到的文件信息返回给用户。检索功能支持对文件信息的排序，用户可以根据搜索词与文件关联性对文件进行排序。

§3 细节实现

3.1 爬虫

爬虫主要使用 Python 第三方库 requests、BeautifulSoup 实现，爬取 USTC 官方网站的文件。本次实验中主要选取了人工智能与数据科学学院、计算机科学与技术学院、数学科学学院、网络空间安全学院、信息科学技术学院、软件学院、财务处、出版社、学位与研究生教育、财务处这十个学院的官方文件进行爬取。

由于一些网站本身的文件数量就很少，使用爬虫收益太低，所以我们选择手动下载这些学院的文件。此外，我们还尝试使用了 IDM 中的站点爬取功能，获取了未被列举在文档中心的 doc、docx、xls、xlsx 类型文件，总计 894 个文件。

3.2 建表

考虑到在使用校内文件搜索时，不同学院的学生会有不同的需求，即对于某一项规定，不同学院学生可能会有不同的要求。因此，在建表时，除了必要的文件名、数据这两项属性外，还增加了来源机构这一属性。而考虑到不同学院可能会存在同名的文件，选择 (文件名 _ 来源机构) 作为 row_key。

```python
def store_docs():
    with open('config.json', 'r') as f:
        config = json.load(f)

    table_name = config['table_name']
    host = config['hbase_host']

    connection = happybase.Connection(host)
    connection.open()
    # get table object
    table = connection.table(table_name)

    # traverse directory in "documents" folder, each subdirectory's name is the college name
    # each file in the subdirectory is a document
    # "documents" folder is in the same directory as this script
    curdir = os.path.dirname(__file__)
    for college in os.listdir(curdir+'/documents'):
        if os.path.isfile(college):
            continue
        for filename in os.listdir(curdir+f'/documents/{college}'):
            # read file
            abspath = curdir+f'/documents/{college}/{filename}'
            data = doc_to_bytes(abspath)

            row_key = f'{college}_{filename}'.encode('utf-8')
            content = {b'info:filename': filename.encode('utf-8'), b'info:college': college.encode('utf-8'), b'data:data': data}
            # create put object
            put = table.put(row_key, content)

    connection.close()
```

图 1: 存储文件

3.3 排序

考虑到需要按照相似性进行排序，因此选择 TfidfVectorizer 进行文本特征提取以计算余弦相似度。主要使用 habbybase 的 scan 函数遍历表选择符合条件的文件，随后通过 tfidf 值计算余弦相似度来的到相似性排序。在这个过程中，主要耗时在于遍历和计算 tfidf 值。后续优化的思路，一是在遍历时使用多线程，二则是选择更高效的文本特征提取方法。

```python
class HbaseConnection:
    def search_ambigous(self, table_name, columns, query, sort=True):
        # import jieba
        table = self.get_table(table_name)
        documents = []
        result = {}
        # import time
        # t = time.time()
        for key, data in table.scan():
            college = data[b'info:college'].decode('utf-8')
            filename = data[b'info:filename'].decode('utf-8')
            if query in college or query in filename:
                documents.append(key)
                result[key] = [filename, college, data[b'data:data']]

        # print('search time:', time.time()-t)
        if sort:
            from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
            from sklearn.metrics.pairwise import cosine_similarity
            # for each column, get an similarity score, using tf-idf with scikit-learn
            # finally sort the result by score
            tfidf = TfidfVectorizer()
            # tfidf = CountVectorizer()
            all_docs = [query]+documents
            tfidf_matrix = tfidf.fit_transform(all_docs)
            # print('trans time:', time.time()-t)
            similarity_matrix = cosine_similarity(tfidf_matrix)
            sort_result = {}
            for i in range(1, len(similarity_matrix)):
                sort_result[all_docs[i]] = similarity_matrix[0][i]
            sort_result = dict(sorted(sort_result.items(), key=lambda x: x[1], reverse=True))

            result = [(k, *result[k]) for k in sort_result.keys()]
        return result
```

图 2: 搜索

§4 使用说明

4.1 依赖

项目开发环境为 Python 3.9，需要安装 happybase、PyQt5 等第三方库。此外若需要使用检索结果相关性排序功能，需要安装 scikit-learn 等库。

4.2 文件

本部分介绍如何存储爬取的文件，以及从数据库下载到本地的文件，请参照附录??中的文件结构。

1. 爬取的文件需要按照学院分类存储，每个学院的文件存储在一个以学院名命名的文件夹中。然后分别将这些文件夹存储在 `initialization/documents` 目录下。
2. 系统运行时检索并下载到本地的文件会存储在 `document_output` 目录下。用户可以自行决定是否保留这些文件。

4.3 方法

完成环境安装后，先运行 start_hbase.py 以启动软件，在将导入文件按要求放置完毕后允许 initialize.py 文件完成建表和数据导入，最后运行 main.py 文件, 在显示的搜索界面进行搜索。使用完毕后，运行 stop_hbase.py 文件以终止运行。

§5 效果展示

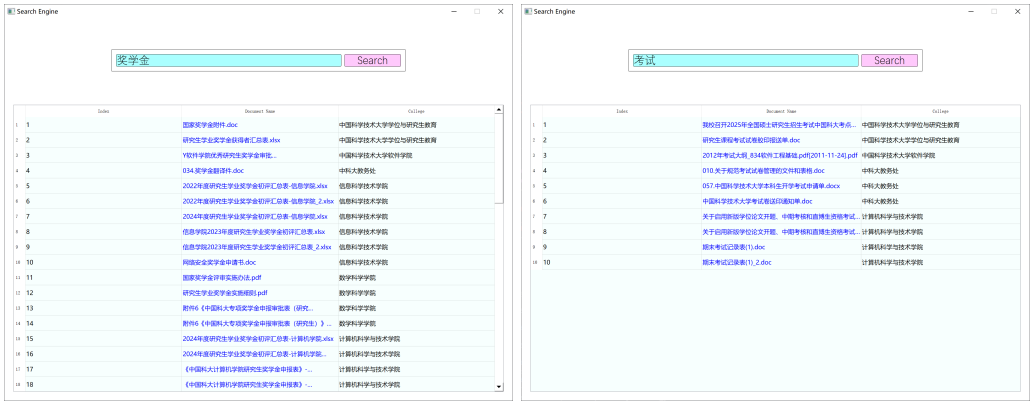

图 3: 搜索效果演示

§6 成员分工

- 殷腾 PB20030785：项目整体框架搭建，前端图形化界面设计。
- 李新涛 PB21151754：文件爬取与环境准备、数据库表设计及排序搜索。

附录

```
/
├── crawl
│   ├── _init_
│   ├── crawler.py
│   ├── grad.py
│   ├── mathematics.py
│   ├── press.py
│   ├── scs.py
│   ├── sist.py
│   ├── sse.py
│   └── teach.py
├── document_output
├── initialization
│   ├── documents
│   ├── create_table.py
│   ├── store_docs.py
│   └── initializa.py
├── report
│   ├── report.pdf
│   └── report.tex
├── scripts_hbase
│   ├── config_hbase.json
│   ├── start_hbase.py
│   └── stop_hbase.py
├── uis
│   ├── __init__.py
│   ├── mainwindow.py
│   ├── ui_mainwindow.py
│   ├── ui_warningwindow.py
│   └── warningwindow.py
├── utils
│   ├── __init__.py
│   ├── docs.py
│   └── hbase.py
├── config.json
├── crawl_data
├── main.py
└── README.md
```

表 1: 项目文件结构