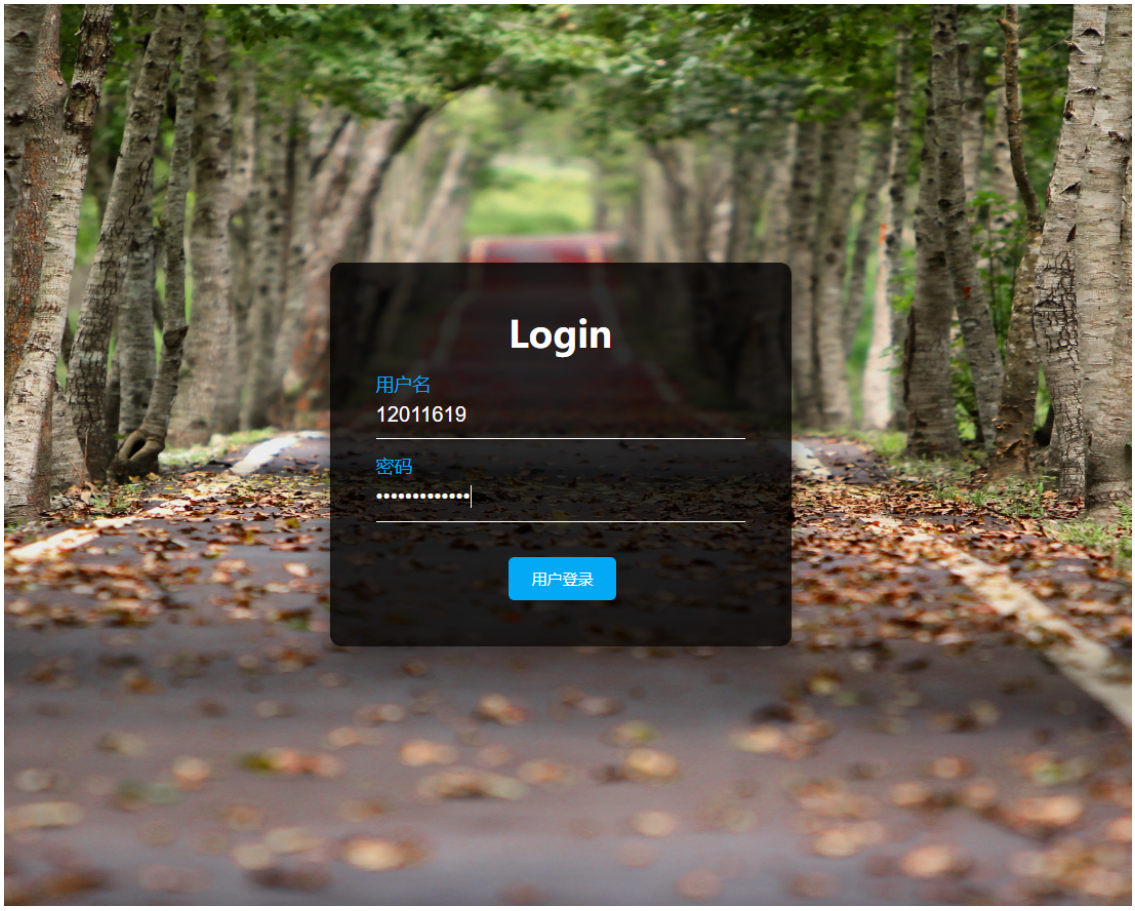
CS103智能图书馆项目报告

组长：王天飚
组员：石轩宇、李达辉、王立全、黄德赐

概述

本项目结合南方科技大学图书馆学生及教职工有关借阅书籍，入出图书馆和使用讨论间等数据，通过应用在“人工智能导论”课程中所学到的分析方法，实现以下功能：

1. 接入CAS系统，实现在校生和教职工的登录功能



2. 完成对读者借阅书籍特征的个性化分析，并以雷达图的方式直观地呈现特征

3. 实现基于对读者借阅书籍特征的雷达图数据和读者本人专业情况，为读者提供个性化的推荐书友服务
4. 基于读者借阅数据的分析和为其推荐的书友，为读者提供个性化的推荐书籍服务

数据处理

本次智能图书馆项目的数据处理部分分为对书籍数据和借阅数据的预处理和相似度计算三部分。

书籍数据预处理

1. 利用爬虫文件可以从数据库中爬取书籍数据，共有22种已有书籍类型，如

- 数理科学和化学
- 环境科学、安全科学
- 生物科学
- 天文学、地球科学

1. 根据该22种类型再人为分成五大类，自然，工程，文艺，社科，军事。再对22种书籍类型与五大类的关联程度，将总权重5分配给各个类型（数字顺序对应相应类别），如

- 数理科学和化学-50000
- 环境科学、安全科学-41000
- 历史、地理-00230

借阅数据预处理

python的内置容器类型列表和字典与json结构十分相近，因而处理json文件时具有独特的优势。

利用python将json文件转为每条记录一行，各个数据以下划线分割的形式，方便接下来的处理。

处理形成两个数据文件，第一个(tt.txt)包含人名*学号*借阅书的类型，第二个包含学号*书的isbn*书名。

相似人数计算

对第一个文件（tt.txt）进行处理,使用hashmap建立学号到五个权重的映射，然后进行五个权重的标准化，至此每个人拥有一个标准化的五维向量。

设置差异参数error，计算相似的人数，根据相似人数反馈调节error变量，直到在某一error下相似人数几乎不变。相似度可以转换为两个五维向量之间的距离进行量化。

最后相似的人可以作为书友进行推荐，书友阅读的书可以作为推荐书籍。由于人数限制，选择的书友的人数小于等于5。书友在重新运行后端或者重新运行项目中一个方法后可以进行更新。

图片爬取

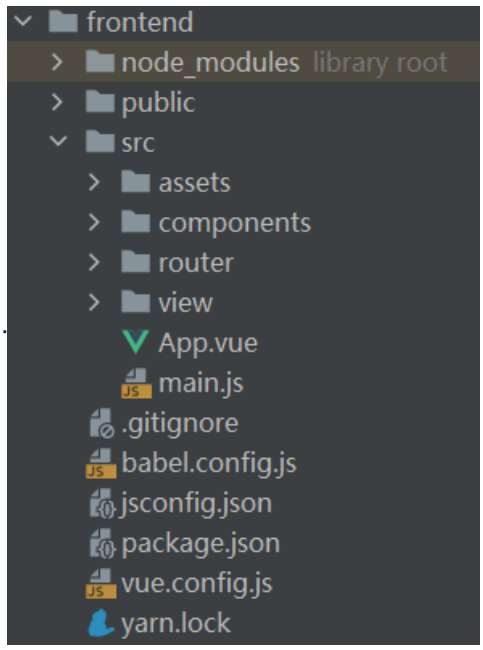
1. 目的
    - 利用爬虫将书籍图片提前保存至本地，减少服务器响应时间。
2. 分析
    - 其中爬取的目标网址是南科大的图书馆荐购网址。
    - 分析网络通信得知需要的图片采用动态加载，该网站用了另一个网站的数据，因此需要参考荐购网址的请求头仿造请求。
3. 过程与问题
    - 由于文件量很大，爬的过程十分漫长
    - 第一次尝试的时候爬了快10000张后被拒绝访问网址
    - 推测其检查User-Agent字段来检测用户，同一个User-Agent访问过多便拒绝访问
    - 弄个User-Agent的列表，被拒绝就换下一个访问
    - 第二天尝试的时候又发现拒绝访问
    - 发现荐购网址向图片网址的请求头中有个authorization字段
    - 推测其一天一换，更新一下就又能跑了
    - 图书馆每本书给了2-4个isbn号，而且有些数据不规范，如9781118927489q(epub)等，需要用正则表达式过滤
4. 反思

    事后得知爬虫存在一定的法律问题，可能需要提前和图书馆/信息中心确认

前端

前端采用Vue框架，使用yarn作为包管理器。使用echarts绘制前端数据图表，使用axios向后端发起http请求然后接受数据。前端分为component，router，view, assets 四个目录，分别管理前端组件，前端页面跳转路由，前端页面展示，前端静态资源四个部分，结构清晰。

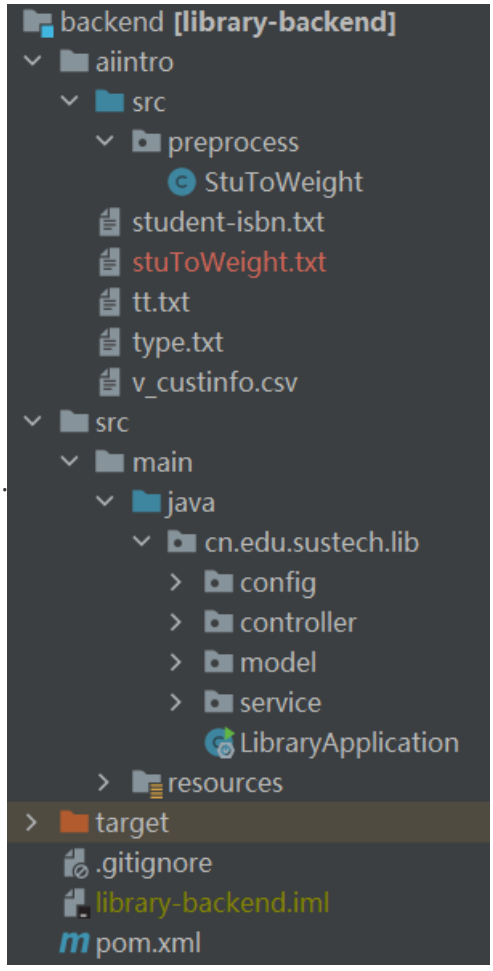
目录结构如图



后端

后端采用SpringBoot框架，用Maven进行依赖管理、项目打包和编译构建

后端使用了controller、service、dao的三层结构，逻辑清晰，降低了代码耦合性



为了便于前后端进行数据交互，我们采用Json作为数据格式，后端定义了通用的响应类`ApiResponse`使响应格式统一，便于前端处理

```
@Data
public class ApiResponse<T> {
    private int code;
    private String message;
    private T data;

    public ApiResponse(int code, String message) {
        this.code = code;
        this.message = message;
    }
}
```

为了前后端对接简便，我们采用了开源项目Knife4j作为文档生成器，这样后端免去了撰写接口文档的麻烦

只需添加`Configuration`并且注入`Docket`的Bean即可

```
@Configuration
@EnableSwagger2WebMvc
public class Knife4jConfig {

    @Bean
    public Docket docket() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(new ApiInfoBuilder()
                        .description("# RESTful APIs")
                        .version("1.0")
                        .build())
                .select()

.apis(RequestHandlerSelectors.basePackage("cn.edu.sustech.lib.controller"))
                .paths(PathSelectors.any())
                .build();
    }
}
```

生成的文档效果如图👇

.

引入`spring-boot-starter-web`依赖可以方便快捷地搭建Web服务，只需编写`Controller`类

```
@RestController
public class ExampleController{

    @GetMapping("/example")
    public ApiResponse<String> example(@RequestParam Integer id){
        return new ApiResponse(200,"ok");
    }
}
```

在对接CAS时，后端使用了okhttp库，便于发送向CAS发送http请求，具体逻辑如下

```
@SneakyThrows(IOException.class)
public boolean login(Integer userId, String password) {
    Request getRequest = new Request.Builder()
                                    .url(LOGIN_URL)
                                    .get()
                                    .build();
    @Cleanup Response response = CLIENT.newCall(getRequest).execute();
    if(response.code()!=200){
        return false;
    }
    //`execution` is the form information sent by CAS's frontend
    @SuppressWarnings("ConstantConditions")
    String execution = response.body().string().split("name=\"execution\" value=\"")
[1].split("\"")[0];
    FormBody body = new FormBody.Builder()
                                .add("username", userId+"")
                                .add("password", password)
                                .add("execution", execution)
                                .add("_eventId", "submit")
                                .build();
    Request postRequest =new Request.Builder()
                                    .url(LOGIN_URL)
                                    .post(body)
                                    .build();
    response = CLIENT.newCall(postRequest).execute();
    return response.code()==200;
}
```

这里做的时候踩了坑，Okhttp的Client创建时默认会采取重定向策略，所以得到的Response不是原始响应，所以在创建Client时应该把重定向策略关闭

```
private static final OkHttpClient CLIENT = new
OkHttpClient.Builder().followRedirects(false).build();
```

总结及改进

在本次智能图书馆的项目中，我们使用了提供的数据库搭建了一个智能推荐系统，这个系统采用了前后端分离的架构，实现了推荐书籍书友，分析读者借阅个性化特征的功能。

在改进方面，本系统仍有许多功能可以扩展，在项目前中期，组内曾多次尝试运用更先进的算法进行相似度分析和样本筛选，也曾考虑过用统计学中心极限定理处理总体数据，但因大部分样本数量太少，大部分同学借书并不能满足大基数的数据运算方法而搁置。选择需求样本数少但误差小的相似度方法是一个好的改进方向。在项目后期，组内也曾多次讨论推荐书友的机制。本系统可以从借阅相似度，不太专业需求，同年级阅读倾向，书籍借阅热度等方面进行综合推书，以进一步提升借书质量。

本项目后续也可以采用Docker-Compose进行容器化部署，便于校内访问。

首先前端使用`yarn build`命令将源代码编译成静态文件，生成在`/dist`目录下，然后利用Docker的卷挂载功能，将其关联到Nginx的静态文件夹`/nginx/html`。Nginx同样可以加入证书实现https服务，这要求需要有响应的域名指向服务器。前端的容器开放http的默认80端口以及https的默认443端口

关于后端的部署，首先在根目录下使用如下命令跳过测试打包

```
mvn package -D"maven.test.skip=true"
```

然后将生成的jar包利用卷挂载直接关联到容器内，这样做可以避免撰写`Dockerfile`利用其中的`ADD`操作重新制作新的镜像。开放8080端口，最后将entrypoint设定为jar包启动命令即可

```
version: '3'
services:
  nginx:
    image: nginx:latest
    container_name: library-nginx
    environment:
      TZ: Asia/Shanghai
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/html:/etc/nginx/html
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/cert:/etc/nginx/cert
    privileged: true #容器内的root有真正的root权限，否则只是一个普通用户，没有.conf文件权限

  main:
    image: openjdk:17
    container_name: library-main
    environment:
      TZ: Asia/Shanghai
    ports:
      - "8080:8080"
    volumes:
      - ./library-main-0.0.1-SNAPSHOT.jar:/main.jar
    entrypoint: ["java","-jar","/main.jar","--spring.profiles.active=prod"]
```

最终部署的目录树如下

```
deploy
├─.env
├─docker-compose.yml
├─library-main-0.0.1-SNAPSHOT.jar
```

```
└─nginx
    ├─nginx.conf
    ├─cert
    │    ├─7998226_example.com.key
    │    └─7998226_example.com.pem
    └─html
        ├─favicon.ico
        ├─index.html
        ├─css
        │  ├─app.e91ad99c.css
        │  └─chunk-vendors.84b717c5.css
        ├─img
        │  ├─background1.c0704357.jpg
        │  ├─img.eeed2fa6.png
        │  ├─img_1.95baee9e.png
        │  ├─img_2.568571b8.png
        │  ├─img_3.77932214.png
        │  └─img_4.637f0cc9.png
        └─js
           ├─app.4ac23e3c.js
           ├─app.4ac23e3c.js.map
           ├─chunk-vendors.0ff34ea2.js
           └─chunk-vendors.0ff34ea2.js.map
```