CS209 Project Documentation

一、数据获取方法

在此次的project中，我们主要使用了Stack Overflow提供的API实现了爬虫，接下来将对此进行简要介绍。

请求数据网址的构建。在该project中，我们采用了tags/java/faq/GET方法来获取数据：以”<https://api.stackexchage.com/2.3/tags/java/faq>”作为基础URL，随后附加页数，每页大小，站点，过滤器【1】以及API密钥作为变量。

文件解析方面。由于Stack Overflow上公告称通过API返回的文件为压缩包格式，因此我们按照公告提供的信息，设置了请求方法GET和头信息”Accept-Encoding”，“gzip“以及”Accept“，”application/json”，

接着是读取API响应。我们设置了一个BufferedReader用于读取传输回的信息，并将相应转化为JSONObject对象。

我们注意到在Stack Overflow的公告中显示，如果请求速率过快，或是短时间内重复请求，会被自动判定为不良信息请求。因此，我们在每读取数据若干次后都会让进程休眠1s，并且会适当调整读取的起始页数，使得请求不会过于频繁地被Stack Overflow阻止。

爬虫部分的代码实现如下：

public class StackExchangeAPIExample {  
 public static void main(String[] args) {  
 String baseUrl = "https://api.stackexchange.com/2.3/tags/java/faq";   
 String filter = "!1563jJ9RK7\_nVIsWtcNpRl3)pK\_2Q7et4nJDmQ().6Dx8N";  
 String apiKey="nVBaFWRs25i0Z3j80Q9OVA((";  
 String site = "stackoverflow";  
 int pageSize = 25;  
 int pageCount = 20;  
 String questionFilename = "java\_questions.json";  
 String answerFilename = "java\_answers1.json";  
 try {  
 FileWriter questionFile = new FileWriter(questionFilename);  
 FileWriter answerFile = new FileWriter(answerFilename);  
 for (int page = 1; page <= pageCount; page++) {  
 String urlString = String.*format*("%s?page=%d&pagesize=%d&site=%s&filter=%s&key=%s", baseUrl, page, pageSize, site, filter, apiKey);  
 URL url = new URL(urlString);  
 HttpURLConnection conn = (HttpURLConnection) url.openConnection();   
 conn.setRequestMethod("GET");  
 conn.setRequestProperty("Accept-Encoding", "gzip");  
 conn.setRequestProperty("Accept", "application/json");  
 BufferedReader in;  
 if ("gzip".equals(conn.getContentEncoding())) {  
 in = new BufferedReader(new InputStreamReader(new GZIPInputStream(conn.getInputStream())));  
 } else {  
 in = new BufferedReader(new InputStreamReader(conn.getInputStream()));  
 }  
 String inputLine;  
 StringBuilder response = new StringBuilder();  
 while ((inputLine = in.readLine()) != null) {  
 response.append(inputLine);  
 }  
 in.close();  
 JSONObject jsonObject = new JSONObject(response.toString());  
 JSONArray items = jsonObject.getJSONArray("items");  
 for (int i = 0; i < items.length(); i++) {  
 JSONObject question = items.getJSONObject(i);  
 questionFile.write(question.toString() + "\n");  
 }  
 TimeUnit.*MILLISECONDS*.sleep(1000);  
 }  
 questionFile.close();  
 answerFile.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

二、项目架构

本项目使用Spring Boot，Thymeleaf等后端架构，Chart.js，ECharts等前端框架。

后端架构中，项目使用Application类对项目进行启动。项目后端的具体功能在Controller类中进行。Controller类中，先用@PostConstruct注释方法，在程序启动时读取数据文件。对于项目的每一个部分，使用@GetMapping注释方法，先读取数据，进行要求的数据分析，随后将分析得出的数据使用Thymeleaf添加model以便前端使用，最后return对应的前端页面。下面是第三部分的方法代码作为示例：

@GetMapping({"/Part3"})  
public String Part3(Model model){  
 Map<String,Integer> tagsMap = new HashMap<>();  
  
 for(JSONObject object: jsonList){  
 JSONArray tags = object.getJSONArray("tags");  
  
 for(int i = 0;i<tags.length();i++) {  
 String q = tags.getString(i);  
 if (!q.equals("java")) {  
 if (tagsMap.containsKey(q)) {  
 int countTags = tagsMap.get(q);  
 tagsMap.put(q, countTags + 1);  
  
 } else {  
 tagsMap.put(q, 1);  
 }  
  
 }  
 }  
 }  
 model.addAttribute("dataMap",tagsMap);  
  
 Map<String,Integer> upVoteMap = new HashMap<>();  
 Map<String,Integer> viewMap = new HashMap<>();  
 for(JSONObject object: jsonList){  
 int upVoteCount =object.getInt("up\_vote\_count");  
 int viewCount = object.getInt("view\_count");  
 JSONArray tags = object.getJSONArray("tags");  
 for(int j =1;j<tags.length();j++){  
 String key = "java, "+tags.getString(j);  
 if(upVoteMap.containsKey(key)){  
 int countUpVotes = upVoteMap.get(key);  
 int countViews = viewMap.get(key);  
 upVoteMap.put(key, countUpVotes+upVoteCount);  
 viewMap.put(key,countViews+viewCount);  
 }  
 else{  
 upVoteMap.put(key,upVoteCount);  
 viewMap.put(key,viewCount);  
 }  
  
 }  
 }  
 Map<String,Integer> upVoteResMap = new LinkedHashMap<>();  
 Map<String,Integer> viewResMap = new LinkedHashMap<>();  
 upVoteMap.entrySet()  
 .stream()  
 .sorted(Map.Entry.<String,Integer>*comparingByValue*().reversed())  
 .limit(10)  
 .forEachOrdered(e ->{upVoteResMap.put(e.getKey(),e.getValue());});  
 viewMap.entrySet()  
 .stream()  
 .sorted(Map.Entry.<String,Integer>*comparingByValue*().reversed())  
 .limit(10)  
 .forEachOrdered(e ->{viewResMap.put(e.getKey(),e.getValue());});  
 model.addAttribute("dataMap2",upVoteResMap);  
 model.addAttribute("dataMap3",viewResMap);  
 return "Tags";  
}

Controller类中包含4个如上的方法，构成了项目的后端结构。

对于前端，在程序开始运行时，先显示Home.html页面，其中包含4个按钮，对应项目的四个部分。当按下对应按钮时，会在后端运行对应方法，方法会将数据处理完毕后，返还前端页面。在前端文件中，使用Thymeleaf获取后端数据，并且对于数据进行可视化绘图处理。如下为Tags部分的前端代码，包括数据获取和图像绘制：

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Tags</title>  
 <script src='https://cdn.jsdelivr.net/npm/echarts@5/dist/echarts.min.js'></script>  
 <script src="https://cdn.jsdelivr.net/npm/echarts-wordcloud@2/dist/echarts-wordcloud.min.js"></script>  
</head>  
<body>  
 <h1>Part3</h1>  
 <h3>The tags with a lot of upvote: -1</h3>  
 <h3>The tags with a lot of views: -1</h3>  
<div style="width: 100%;height: 60vh;display: flex">  
 <div style="width: 50%;height: 100%">  
 <div id="wordcloud" style="height: 100%;width: 100%"></div>  
 </div>  
 <div style="width: 50%;height: 100%">  
 <div id="barchart" style="height: 100%;width: 100%"></div>  
 </div>  
 <div style="width: 50%;height: 100%">  
 <div id="barchart2" style="height: 100%;width: 100%"></div>  
 </div>  
</div>  
</body>  
<script th:inline="javascript">  
 var ***dataMap*** = /\*[[${dataMap}]]\*/ {};  
 var ***dataMap2*** = /\*[[${dataMap2}]]\*/ {};  
 var ***dataMap3*** = /\*[[${dataMap3}]]\*/ {};  
 var ***dataArray*** = [];  
 for (var ***key*** in ***dataMap***) {  
 if (***dataMap***.hasOwnProperty(***key***)) {  
 var ***value*** = ***dataMap***[***key***];  
 ***dataArray***.push({ name: ***key***, value: ***value*** });  
 }  
 }  
 var ***myChart*** = echarts.init(***document***.getElementById('wordcloud'));  
 var ***option*** = {  
 title: {  
 text: 'tags frequently appeared together with java',   
 left: 'center'   
 },  
 series: [{  
 type: 'wordCloud',  
 shape: 'circle',  
 textStyle: {  
 fontFamily: 'Arial',  
 fontWeight: 'bold',  
 color: function () {  
 return 'rgb(' + [  
 ***Math***.round(***Math***.random() \* 255),  
 ***Math***.round(***Math***.random() \* 255),  
 ***Math***.round(***Math***.random() \* 255)  
 ].join(',') + ')';  
 }  
 },  
 data: ***dataArray*** }]  
 };  
 ***option***.tooltip = {  
 formatter: function (params) {  
 var name = params.name;  
 var value = ***dataMap***[name];  
 return '词汇: ' + name + '<br>数量: ' + value;  
 }  
 };  
 ***myChart***.setOption(***option***)  
  
 var ***barChart1*** = echarts.init(***document***.getElementById('barchart'));  
 var ***barChartOption1*** = {  
 title: {  
 text: 'tag combinations with most upvotes',   
 left: 'center'   
 },  
 xAxis: {  
 type: 'category',  
 data: ***Object***.keys(***dataMap2***),  
 axisLabel: {  
 show: true,   
 interval: 0,   
 rotate: 20  
 }  
  
 },  
 yAxis: {  
 type: 'value'  
 },  
 series: [{  
 type: 'bar',  
 data: ***Object***.values(***dataMap2***)  
 }]  
 };  
 ***barChartOption1***.tooltip = {  
 formatter: function (params) {  
 var name = params.name;  
 var value = params.value;  
 return 'tags: ' + name + '<br>up votes: ' + value;  
 }  
 };  
 ***barChart1***.setOption(***barChartOption1***);  
  
 var ***barChart2*** = echarts.init(***document***.getElementById('barchart2'));  
 // 配置柱状图参数  
 var ***barChartOption2*** = {  
 title: {  
 text: 'tag combinations with most views',   
 left: 'center'   
 },  
  
  
 xAxis: {  
 type: 'category',  
 data: ***Object***.keys(***dataMap3***),  
 axisLabel: {  
 show: true,   
 interval: 0,   
 rotate: 20  
  
 }  
  
 },  
 yAxis: {  
 type: 'value'  
 },  
 series: [{  
 type: 'bar',  
 data: ***Object***.values(***dataMap3***)  
 }]  
 };  
 ***barChartOption2***.tooltip = {  
 formatter: function (params) {  
 var name = params.name;  
 var value = params.value;  
 return 'tags: ' + name + '<br>views: ' + value;  
 }  
 };  
 ***barChart2***.setOption(***barChartOption2***);  
</script>  
</html>

主页面、四个部分以及原始数据的6个html文件存储于templates文件夹中，构成了项目的前端部分。

三、数据分析

在对500条数据进行分析后，发现数据的情况如下：

500个问题中，只有1个问题答案数为0，占比0.2%，且该问题的“is\_answered”属性为true，是回答后答案被删除。问题的平均答案数为17，最大答案数为90。答案数为1至20个的问题分布最多，为314个。

500个问题中，有87.4%的问题都有accepted answer。其中，仅有2.5%的问题中有未被接受的答案获得upvote数比被接受的答案高。在有accepted answer的437个答案中，我们使用accepted answer的创建时间与问题的创建时间的差值作为问题解决时间，其中有220个问题在10分钟之内就被解决。另外还有100个答案在10分钟至1小时内被解决。证明**问题解决非常高效，甚至有点不正常的可能**。

对于tags，我们去除一定会出现的”java”，然后对于每一个问题的tags数组的每一项进行计数。**最终，最高频率出现的tags依次为“android”，40次；”string”，35次；”swing”，24次；”arrays”，23次，随后是”generics”，“collections”，都为15次**。

对于tags组合，项目只统计了java与另一个tag的组合。因为如果统计3个tags的组合，会有多个仅出现1次却得到非常高view和upvotes的组合占据排行的前几位。从upvotes来看，java与string的组合得到最高的数量，为32916。java与performance组合排名第二，为30642。java与c++的排名组合第三，为27281。java与branch-prediction以及cpu-architecture的组合并列排名第四，为27090。经过对数据的检查，发现performance，c++，branch-prediction以及cpu-architecture出现在同一个问题中，其中branch-prediction和cpu-architecture两个tag在所有的问题中只出现了一次。如果统计3个tags的组合，排名前10的组合中至少会有6个是这4个tags的组合，并且数值会十分接近，效果非常不理想。同时也可以得出，真正高质量的问题在该网站中影响力很大。从views来看，最高的组合是java和string的组合，有4351万次的浏览。排名第二的是java和arrays的组合，数量为2548万次。第三与第四为file-io和android，分别为1442万次与1431万次，之后的tags组合浏览量变化较为平稳。可以看出，**string是网站最受关注的tag，有着第二高的出现次数，最高的浏览量与获赞数。Android虽然出现次数最多，但是相关问题的质量并没有特别高**。

对于users，我们发现最高回答数的用户回答了38个问题，最高评论数的用户回答了108个问题。**然而，回答数排名前10的用户和评论数排名前10的用户没有任何重叠。也就是说，高频回答和高频评论的用户很可能是两批人**。

注释：

[1] 我们仅请求了问题所在Thread的答案和评论，其中每个问题、答案、评论都包含了对应的ID和必要的owner信息。与此同时，我们也让每个问题附带对于问题本身的答案数和评论数，每个答案附带对于答案本身的评论数。最终的filter为：“nVBaFWRs25i0z3j80Q90VA((“。