

CS209A-Spring Java 2

Stackoverflow Web Application Project Report

- 伦天乐（学号：12113019）贡献比：50%
- 罗嘉诚（学号：12112910）贡献比：50%

1 项目简介与部署

1.1 项目简介

[Stack Overflow](#)网站是一个非常受欢迎的IT技术在线问答社区，专注于编程和软件开发领域。它允许开发人员在网站上提出技术问题，并接收来自全球开发者社区的答案和解决方案。



本网络应用项目对Stack Overflow网站进行相关数据抽样统计，并进行数据可视化。项目主要采用 Spring Boot、Mybatis Plus 作为后端框架，Vue 3.0 作为前端框架，使用PostgreSQL 数据库作为数据存储媒介。

用户可通过网络应用，查看相应的可视化数据图，直观地了解相应日期之间的Stack Overflow 网站数据统计；用户也可以通过项目提供的 REST 服务，通过相应 API 获取处理后的数据。

GitHub开源链接：<https://github.com/XiaoLeGG/stackoverflow-web-application>

The screenshot shows the GitHub repository page for 'XiaoLeGG/stackoverflow-web-application'. The repository has 17 commits, 2 branches, and 0 tags. The README file contains the following content:

```

CS209A-Spring Java 2
Stackoverflow Web Application
Project Report

项目简介与部署

项目简介
Stack Overflow 是一个非常著名的 IT 技术在线问答社区，专注于编程和软件开发。它允许专业人员在网站上

```

1.2 项目本地部署

1. 使用下列命令克隆项目仓库到本地: `git clone`

<https://github.com/XiaoLeGG/stackoverflow-web-application.git>

2. 如需爬取数据, 请进入 `src/main/java/cn/edu/sustech/crawler` 包中, 运行

`CrawlerMain.java`, 此操作需要配置需要存储爬取数据的数据库, 即在代码中修改: 主机 (`SQL_HOST`)、端口 (`SQL_PORT`)、数据库名称 (`SQL_DATABASE`)、数据库用户名 (`SQL_USER`)、数据库用户密码 (`SQL_PASSWORD`)。

```

private static final String SQL_HOST = "localhost";
private static final String SQL_USER = "username";
private static final String SQL_PASSWORD = "password";
private static final String SQL_DATABASE = "database";
private static final int SQL_PORT = 5432;

```

3. 启用项目后端, 请进入 `src/main/java/cn/edu/sustech` 包中, 运行

`StackofWebApplication.java`, 此操作需要首先配置 `src/resources` 中的 `application.properties`。

```
spring.devtools.restart.enabled=true
spring.devtools.restart.additional-paths=src/main/java
spring.datasource.url=jdbc:postgresql://localhost:5432/database
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver
mybatis-plus.configuration.log-
impl=org.apache.ibatis.logging.stdout.StdoutImpl
```

4. 启用项目前端，请进入 `frontend` 目录，在终端中执行 `npm install` 构建，再使用 `npm run dev` 命令获取项目URL链接。

```
Terminal Local x + v
VITE v4.3.5 ready in 464 ms

→ Local: http://127.0.0.1:5173/
→ Network: use --host to expose
→ press h to show help
```

5. 确保项目前端、后端都正常运行后，可使用浏览器访问对应的项目URL链接（即<http://127.0.0.1:5173>），访问网络应用。

Stack Overflow Web 127.0.0.1:5173

Stack Overflow web application Update Time 2023/5/22 14:50:45

Introduction

Stack Overflow Web Application is a web application that collect and analyze statistics of Stack Overflow such as the most popular tags, the most popular questions, the most popular users, etc.



Statistics Content

Question With No Answers

This diagram show the distribution of questions with no answer.

From End Query

Average / Maximum Answers

This diagram show the average and maximum numbers of answers of questions.

From End Query

Stack Overflow Web 127.0.0.1:5173

Stack Overflow web application Update Time 2023/5/22 14:50:45

Statistics Content

Question With No Answers

This diagram show the distribution of questions with no answer.

From End Query

Average / Maximum Answers

This diagram show the average and maximum numbers of answers of questions.

From End Query

Answers Distribution

This diagram show the distribution of answers of questions.

From End Query

Question With Accepted Answers

This diagram show the distribution of questions with accepted answer.

From End Query

Resolution Time Distribution

This diagram show the distribution of resolution time of distribution.

From End Query

Question With Answer Better Than Accepted

This diagram show the distribution of questions with not accepted answers better than accepted answer.

From End Query

Frequent Tags

This diagram show the top 10 tags or tag combinations that frequently appeared.

From End - 1 + Query

Tags Upvotes

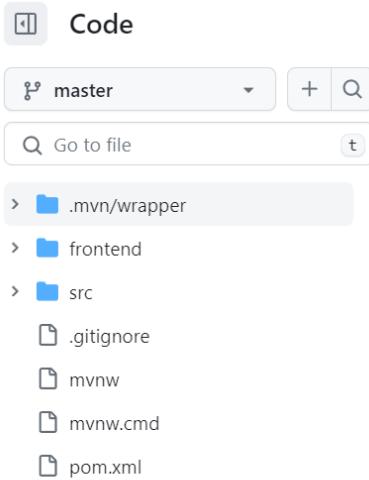
This diagram show the top 10 tags or tag combinations with most upvotes.

From End - 1 + Query

2 项目架构

2.1 总体架构

本项目主要使用 Apache Maven 进行项目管理和自动构建，分为前端和后端两部分。前端框架采用 Vue 3.0，后端框架采用 SpringBoot 和 Mybatis-Plus，使用 Java 爬取数据到 PostgreSQL 数据库。



上图的代码结构中，前端相关的代码部分保存在 `frontend` 目录下，后端相关的代码部分保存在 `src` 目录下，其他文件是 `Apache Maven` 进行项目管理和自动构建时产生的文件。

2.2 前端架构

前端相关代码保存在 `frontend` 目录下，其树形结构图如下：

```
frontend
├─public
└─src
    ├─assets
    │   └─bg
    └─components
        ├─charts
        ├─data-containers
        └─parameters
```

前端主要使用了 `Vue 3.0`、`Vite`、`Element-UI` 和 `Vue-echarts` 包来设计前端，`axios`、`dayjs` 和 `sass` 等依赖包用于获取和处理数据以及处理样式。同时，我们采用 `npm` 来处理依赖和包管理。

`frontend` 包下，`index.html` 为网页的主体html，`vite.config.js` 配置 `Vue` 相关参数，如服务器代理，`package.json` 包含了本次项目相关配置信息。

`src` 包下主要是 `Vue` 构建的基本内容，其中 `App.vue` 是主体组件，`main.js` 负责创建和导入包括 `css` 样式文件在内的全局的一些控件。

`assets` 包下包含了一些 `css` 样式文件和 `icon` 等用到的图像文件。

`components` 包下主要是 `Vue` 的其它子组件，其中包含了整体网页的顶部选项卡 (`HeaderComponent.vue`)，项目介绍组件 (`IntroductionComponent.vue`) 和数据内容展示组件 (`ContentComponent.vue`)，还有一个 `StatisticsContainer.vue` 作为数据展示容器模板。

`charts` 包下主要为实现各类图表的基本组件，是根据本项目定制后设计的组件，方便后续使用。

`data-containers` 主要包含了各类需要查询的数据图表容器组件。

`parameters` 为各类数据查询需要的参数的输入组件。

另外，前端的数据采集主要使用的是后端的 REST API，前后端可以说是完全分离的。

2.3 后端架构

后端相关代码保存在 `src` 目录下，其树形结构图如下：

```
src
├── main
│   ├── java
│   │   └── cn
│   │       └── edu
│   │           └── sustech
│   │               ├── controller
│   │               ├── crawler
│   │               ├── entity
│   │               ├── mapper
│   │               │   └── api
│   │               └── service
│   └── resources
└── test
    └── java
        └── cn
            └── edu
                └── sustech
```

后端主要使用了Spring和Mybatis-Plus来提供Web服务。

`controller` 包中存储形如 `XXXController.java` 若干与相应实体（问题、答案、评论、用户等）的控制方法，既能提供相关的 REST 服务，即通过相应 API 提供处理后的数据；又能为前端可视化提供所需要的数据。

`crawler` 包中存储有 `CrawlerMain.java` 是主方法，运行后将刷新数据库中的数据，使用到同一包下的工具类 `DatabaseService.java`（提供数据库的服务）、`DataCollector.java`（提供爬虫的服务）、`StandfordCoreNLPService.java`（提供 NLP 分词服务）。

`entity` 包中是数据表中各表的实体，在 `Mybatis-Plus` 框架中使用到。

`mapper` 包中是数据表中各表的映射，在 `Mybatis-Plus` 框架中使用到。

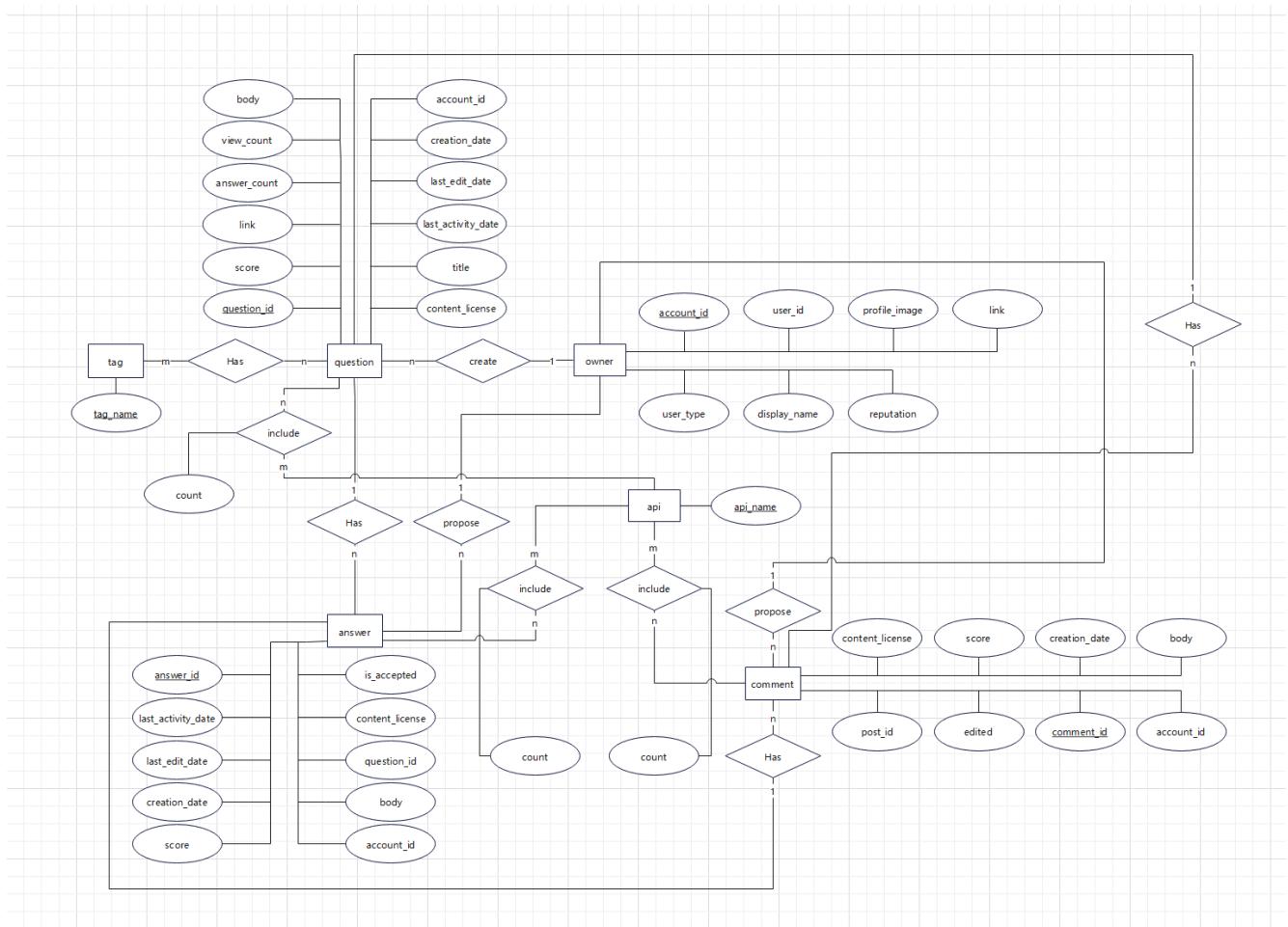
`service` 包中是使用 `Mybatis` 在各表中进行数据查询的服务类。

2.4 数据库架构

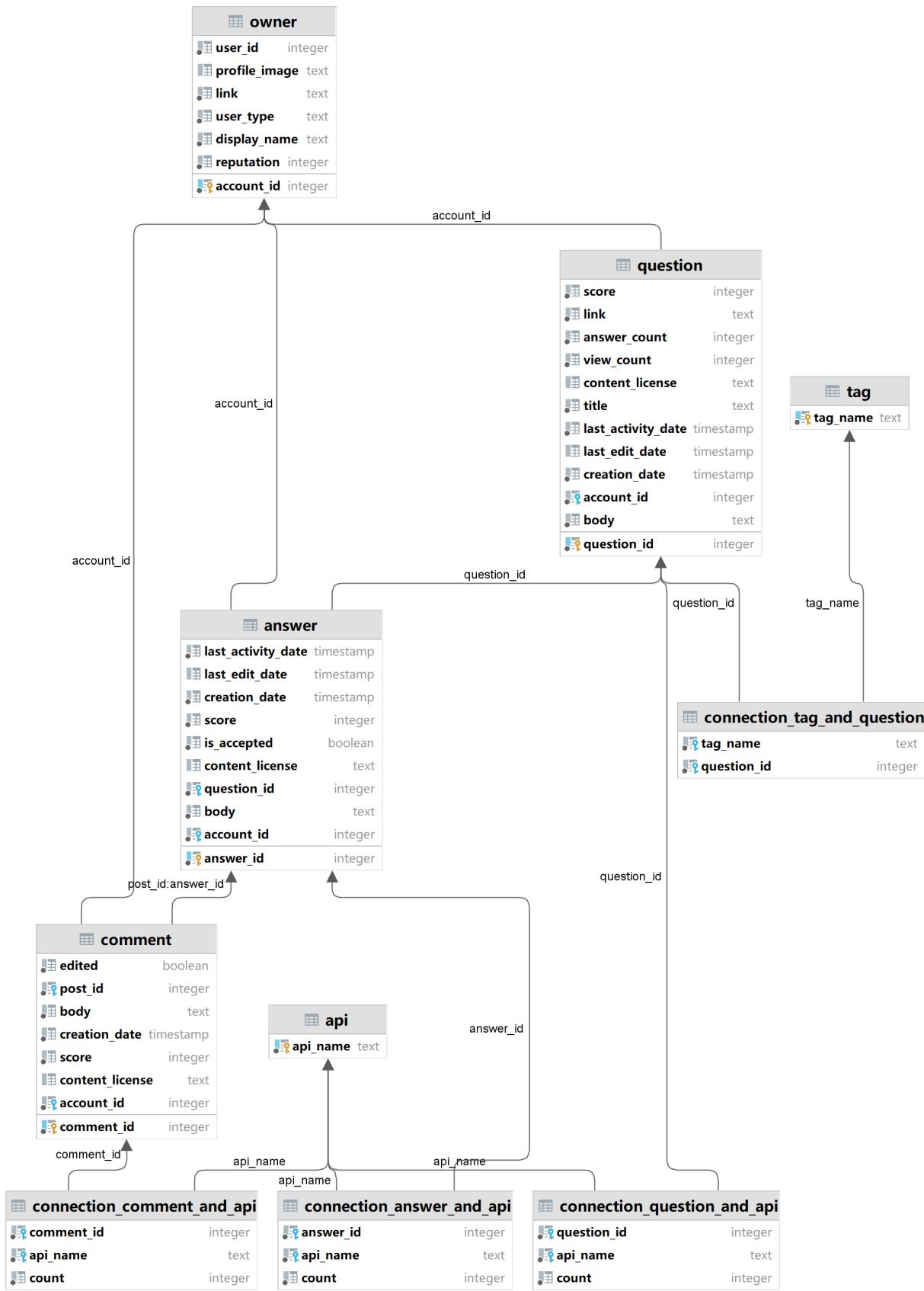
本项目采用 PostgreSQL 数据库存储数据。PostgreSQL是一个功能强大的关系型数据库管理系统（RDBMS），它的设计目标是提供可靠性、可扩展性和数据完整性。

- 可靠性：PostgreSQL是一个可靠的数据库系统，具有良好的数据完整性和持久性。它支持事务处理，可以确保在数据库操作过程中的错误或故障时数据的一致性和可靠性。
- 可扩展性：PostgreSQL具有良好的可扩展性，可以处理大量数据和高并发访问。它支持水平扩展和垂直扩展，可以通过添加更多的服务器节点或增加硬件资源来提高数据库的性能和容量。
- 强大的功能：PostgreSQL提供了广泛的功能和灵活的查询语言，支持复杂的查询操作和高级特性，如联接、子查询、触发器、视图等。它还支持多种数据类型，包括文本、数字、日期/时间、地理空间数据等。

2.4.1 ER 图



2.4.2 可视化图



3 爬虫

3.1 实现方式

使用官方 StackOverflow REST API 文档，通过相应接口随机爬取若干问题、这些问题对应的所有回答、评论。

爬虫在 `src/main/java/cn/edu/sustech/crawler` 包中得到实现，需要使用如下三个服务：

`DatabaseService.java` (提供数据库的服务)、`DataCollector.java` (提供爬虫的服务)、

`StandfordCoreNLPService.java` (提供 NLP 分词服务)。

3.1.1 数据库服务

`DatabaseService` 提供如下重要方法：

1. `connect()/close()` 连接/关闭数据库
2. `createTable()` 数据表创建
3. `disableForeignKeyCheck()/enableForeignKeyCheck()` 关闭打开外键约束
4. `insertQuestion()/insertTag()/...` 向某数据表中插入数据
5. `insertQuestionRecord()/insertAnswerRecord()/insertCommentRecord()` 将问题/答案/评论的 JSON 格式的数据插入数据库

3.1.2 爬虫服务

由两个参数控制爬取的数据量：pageSize 和 pageStep。前者表示每次分页查询的数据条数（最多是 pageSize 条，最多 100，通过也设为 100），后者表示按照 activity 排序后，每隔 pageStep 页获取一页数据，插入到数据库中。

`DataCollector` 提供如下重要方法：

1. `refresh()` 爬取最新数据，并插入到数据库中。
2. `collectData()` 数据爬取。
3. `getCommentsFromAnswer(String ids)` 获取每个回答的评论，`ids` 为 `answer_id` 的字符串，以分号分隔。
4. `getCommentsFromQuestion(String ids)` 获取每个问题的评论，`ids` 为 `question_id` 的字符串，以分号分隔。

5. `getAnswers(String ids)` 获取每个问题的回答，`ids` 为 `question_id` 的字符串，以分号分隔。

3.1.3 NLP分词服务

Stanford NLP是由斯坦福大学开发的强大工具包，专注于自然语言处理领域。它提供了丰富的功能，其中包括英文分词。英文分词是将英文文本切分成独立的词语或标记的过程。Stanford NLP中的英文分词模块采用了现代的机器学习和统计技术，以及大规模的语料库和语言资源。它能够准确地切分出英文文本中的单词、短语和标点符号。

基于此，在爬取数据写入数据库时，我们使用 Stanford NLP 工具包对所有 `Question`、`Answer`、`Comment` 的 `body` 中的内容进行 NLP 分词，获得句子中出现的 `Jave API` 纳入统计。

`StanfordCoreNLPService` 类的 `getAllJavaAPI()` 方法就是对于输入的字符串，返回一个 `Jave API` 的词频 `Map`。

```
public class StanfordCoreNLPService {  
    private final Properties props;  
    private final StanfordCoreNLP pipeline;  
    public StanfordCoreNLPService() {  
        props = new Properties();  
        props.setProperty("annotators", "tokenize,ssplit,pos,lemma,ner");  
        props.setProperty("ner.useSUTime", "false");  
        pipeline = new StanfordCoreNLP(props);  
    }  
  
    public Map<String, Integer> getAllJavaAPI(String htmlText) {  
        Map<String, Integer> javaAPIs = new HashMap<>();  
        Annotation annotation = new Annotation(htmlText);  
        pipeline.annotate(annotation);  
        List<CoreMap> sentences =  
annotation.get(CoreAnnotations.SentencesAnnotation.class);  
        for (CoreMap sentence : sentences) {  
            List<CoreLabel> tokens =  
sentence.get(CoreAnnotations.TokensAnnotation.class);  
            for (CoreLabel token : tokens) {  
                String word = token.get(CoreAnnotations.TextAnnotation.class);  
                if (word.startsWith("java.")) {  
                    if (javaAPIs.containsKey(word)) {  
                        javaAPIs.put(word, javaAPIs.get(word) + 1);  
                    } else {  
                        javaAPIs.put(word, 1);  
                    }  
                }  
            }  
        }  
        return javaAPIs;  
    }  
}
```

```

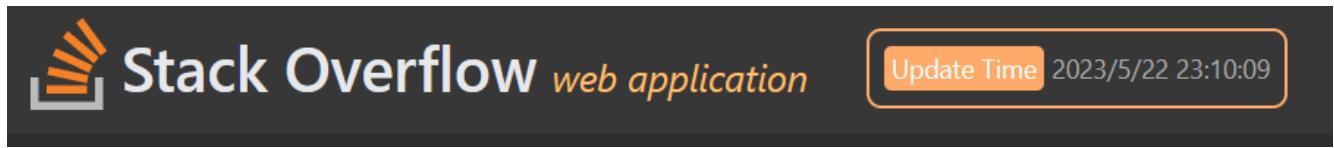
        }
    }
}

return javaAPIs;
}
}

```

3.2 数据情况

截止2023年05月22日晚23:10:09，最后一次以 pageSize = 100 和 pageStep = 100 进行数据爬取（即完成StackOverflow中 1% 问答数据的收集）。



共计收集 19,000 个 Thread 的数据。包括：

- 19,000 个问题

question		
	WHERE	ORDER BY
1	question_id : 76305054	-1 https://stackoverflow.com/questions/76305054/mysql-conn
2	76306187	0 https://stackoverflow.com/questions/76306187/couldnt-ac
3	76306192	0 https://stackoverflow.com/questions/76306192/http-reqe
4	76288087	0 https://stackoverflow.com/questions/76288087/debezium-c
5	68726096	0 https://stackoverflow.com/questions/68726096/debug-keys
6	76305111	0 https://stackoverflow.com/questions/76305111/what-is-th
7	76298050	1 https://stackoverflow.com/questions/76298050/i-cant-inv
8	50003906	18 https://stackoverflow.com/questions/50003906/storina-u

- 30,592 个答案

answer		
	WHERE	ORDER BY
1	answer_id : 76306149	2023-05-22 20:21:24.000000 <null>
2	76306147	2023-05-22 20:21:17.000000 <null>
3	76306145	2023-05-22 20:20:58.000000 <null>
4	76306119	2023-05-22 20:19:09.000000 2023-05-22 20:19:09.000000
5	76306114	2023-05-22 20:17:44.000000 <null>
6	76304794	2023-05-22 20:16:12.000000 2023-05-22 20:16:12.000000
7	76306055	2023-05-22 20:09:46.000000 <null>
8	76302572	2023-05-22 20:06:17.000000 2023-05-22 20:06:17.000000
9	76306007	2023-05-22 20:03:54.000000 <null>
10	76305982	2023-05-22 20:02:53.000000 <null>
11	76305909	2023-05-22 19:54:09.000000 <null>

- 80,372条评论

comment

	comment_id	edited	post_id	body
1	134559271	false	76305166	If your x86 jpack
2	134559177	false	76305842	If the value is i
3	134559150	false	76305166	No confusion: I c
4	134559115	false	76305983	What else could j
5	134559082	false	76305903	What evidence do
6	134559062	false	76305897	That is an hint t
7	134559035	false	76305903	@aled According
8	134559034	false	76306187	Files do not tech
9	134559032	false	76305842	You already have

- 42,407 个用户

owner

	account_id	user_id	profile_image
1	7227100	5515287	https://i.stack.imgur.com/SofKG.png?
2	22063671	16325891	https://lh3.googleusercontent.com/a-
3	28588451	21888219	https://www.gravatar.com/avatar/8eac
4	3568344	2979325	https://www.gravatar.com/avatar/ac1e
5	22013166	16283108	https://lh3.googleusercontent.com/a/
6	3163862	2674303	https://www.gravatar.com/avatar/411c
7	28172505	21530803	https://www.gravatar.com/avatar/cd52
8	6937979	5325309	https://i.stack.imgur.com/bodqN.jpg?

- 1,769 个不同的 Java API

api

	api_name
1	java.sql.Connection
2	java.sql.ResultSet
3	java.sql.Statement
4	java.io.IOException
5	java.io.EOFException
6	java.lang.reflect.Method.invoke
7	java.base
8	java.lang.Thread.run

- 5,516 种不同的问题标签

tag

	tag_name
1	java
2	maven
3	mysql-connector
4	swing
5	file
6	object
7	netbeans
8	android
9	sql-server

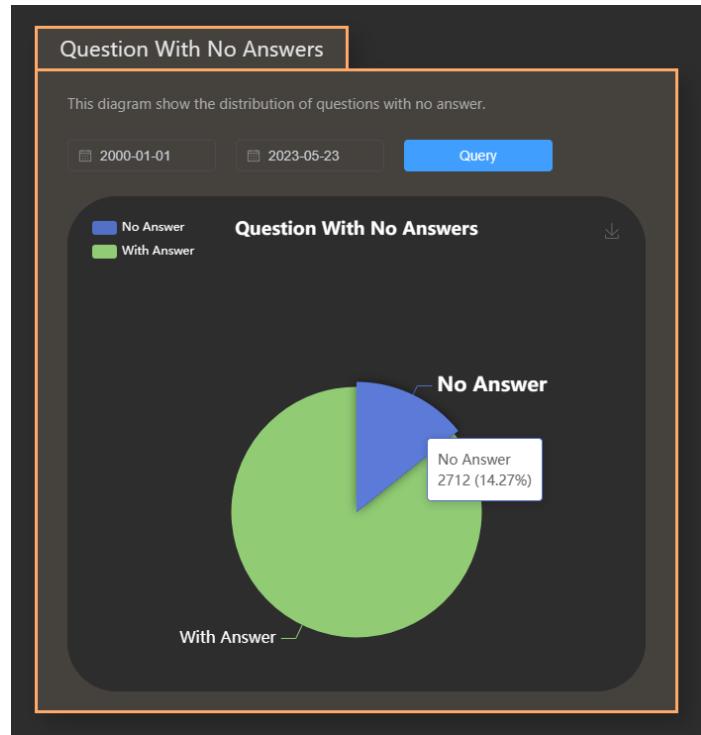
4 数据分析结果

我们选取 2000年01月01日到2023年5月23日的数据进行分析。

4.1 答案的个数

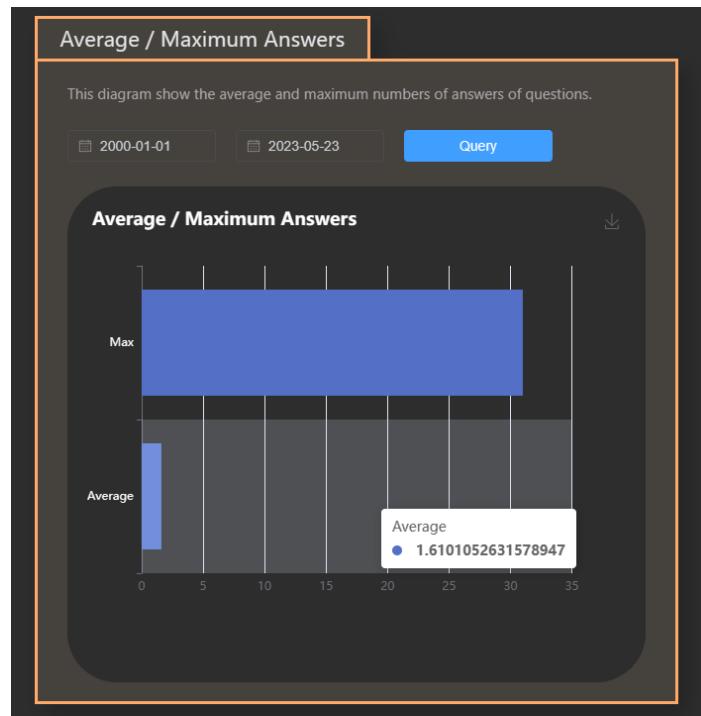
4.1.1 没有答案的问题的百分比

根据下图可以得出，没有答案的问题百分比约为 14.27%。



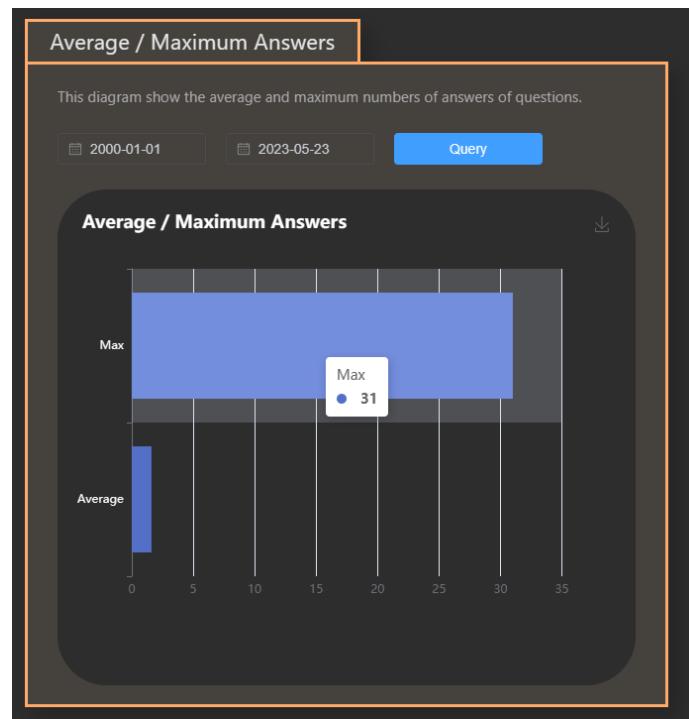
4.1.2 问题答案个数的平均值

根据下图可以得出，问题答案个数的平均值约为 1.61个。



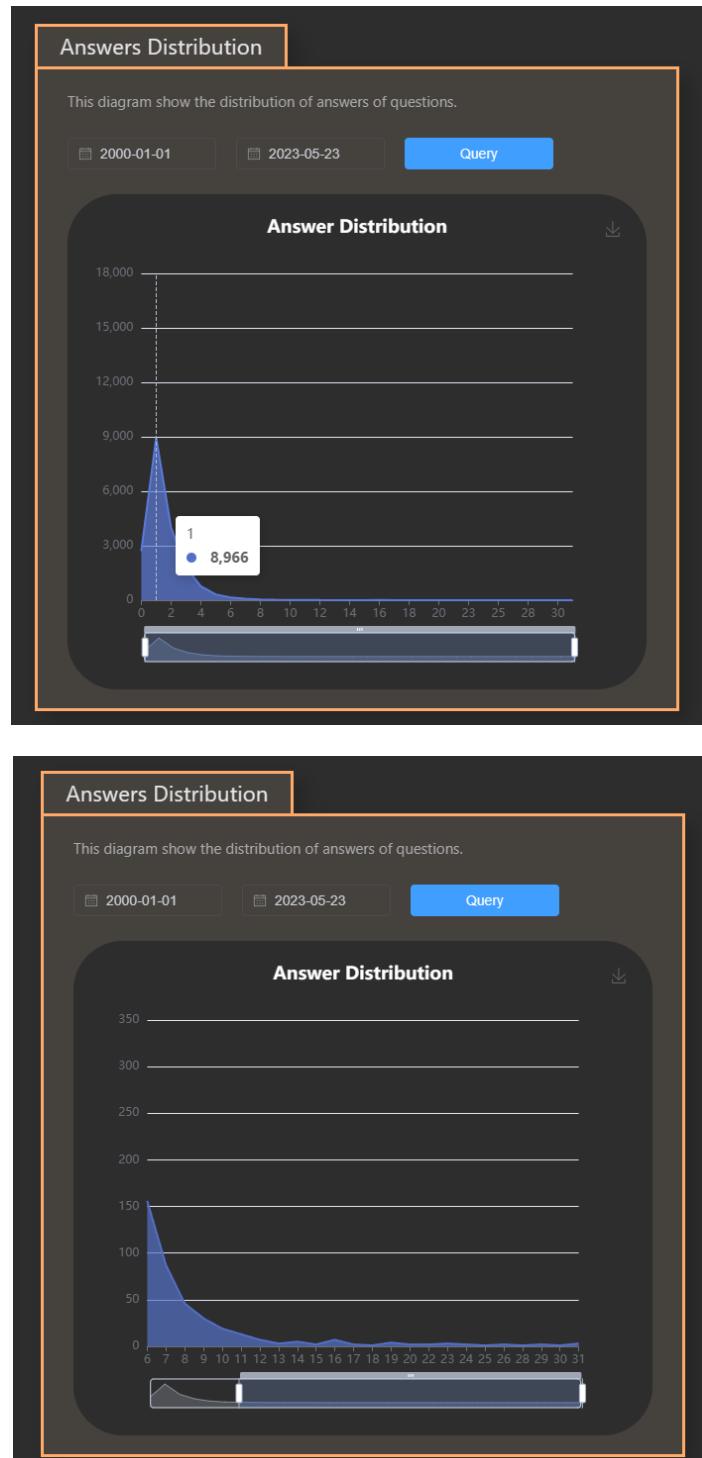
4.1.3 问题答案个数的最大值

根据下图可以得出，问题答案个数的最大值为 31 个。



4.1.4 问题答案个数的分布

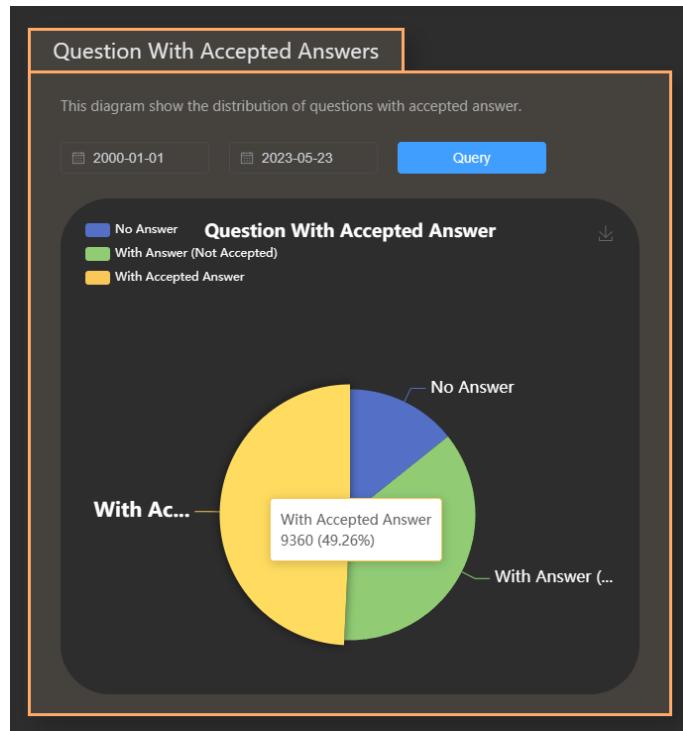
根据下图可以得出，问题答案个数总体趋势是集中在 1 个答案，随着问题答案数目的增加，拥有相应个数答案的问题数量波动下降。



4.2 被接受的答案

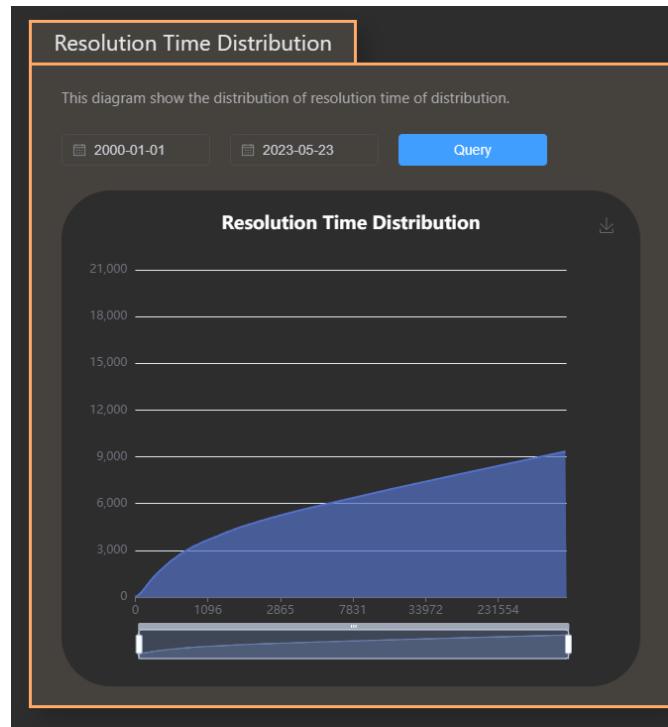
4.2.1 有被接受答案问题的百分比

根据下图可以得出，有接受答案的问题的百分比约为 49.26%。



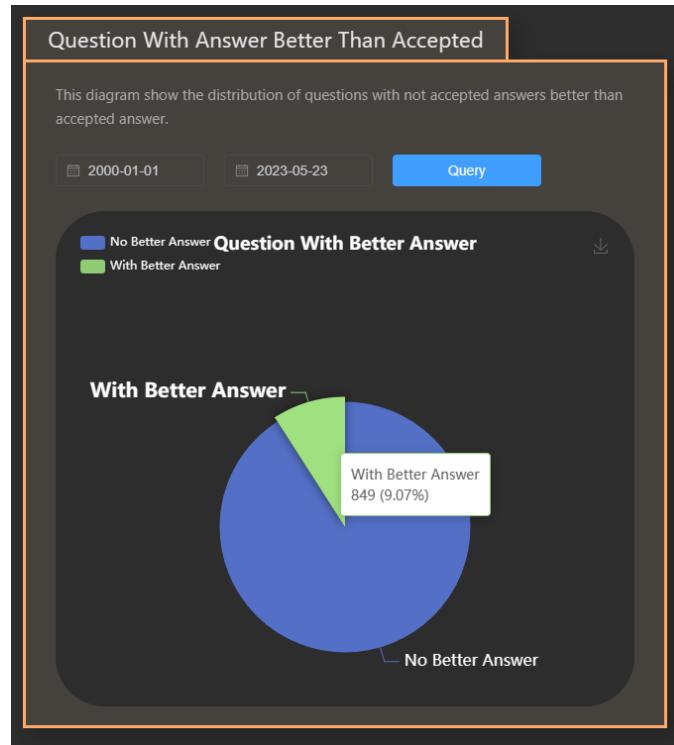
4.2.2 问题从提出到解决的时间间隔分布

根据下图可以得出，问题从提出到解决的时间间隔分布的累积分布图如下。



4.2.3 含有非接受答案的赞同数高于被接受答案的问题的百分比

根据下图可以得出，含有非接受答案的赞同数高于被接受答案的问题（拥有更好答案的问题）的百分比约为 9.07%。



4.3 标签

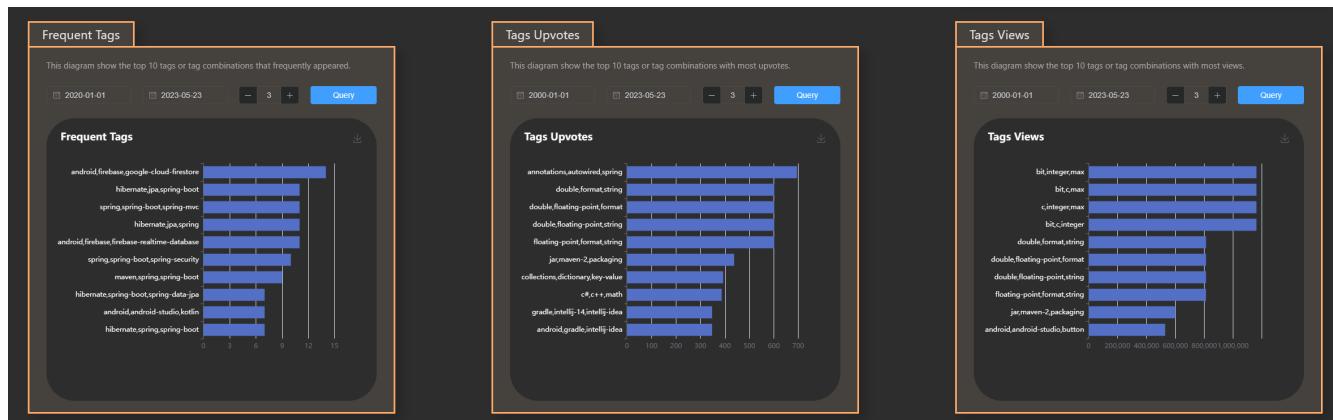
4.3.1 单标签经常出现、得到最多的点赞、得到最多的浏览量

由于我们数据收集时默认筛选 `java`，所以我们此时默认不显示 `java` 标签。可以发现单标签经常出现、得到最多的点赞、得到最多的浏览量都是 `android` 标签。



4.3.2 标签组经常出现、得到最多的点赞、得到最多的浏览量

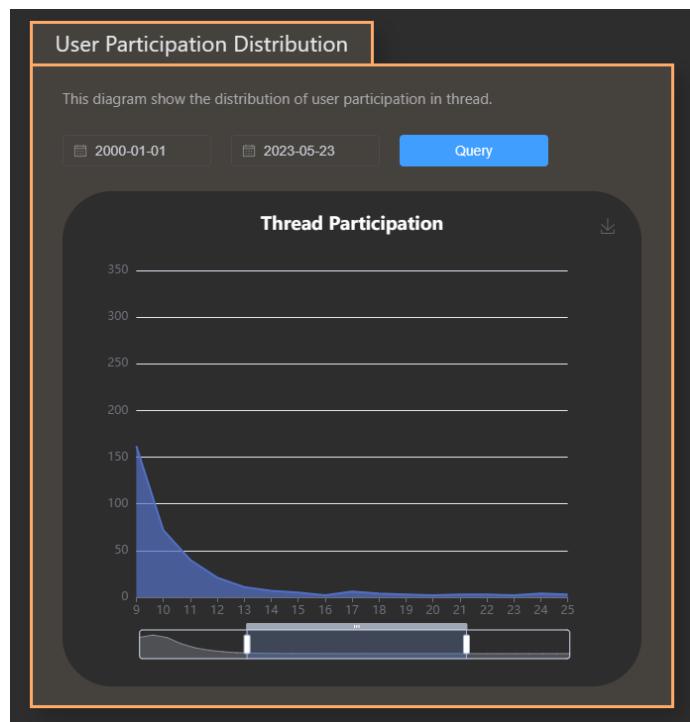
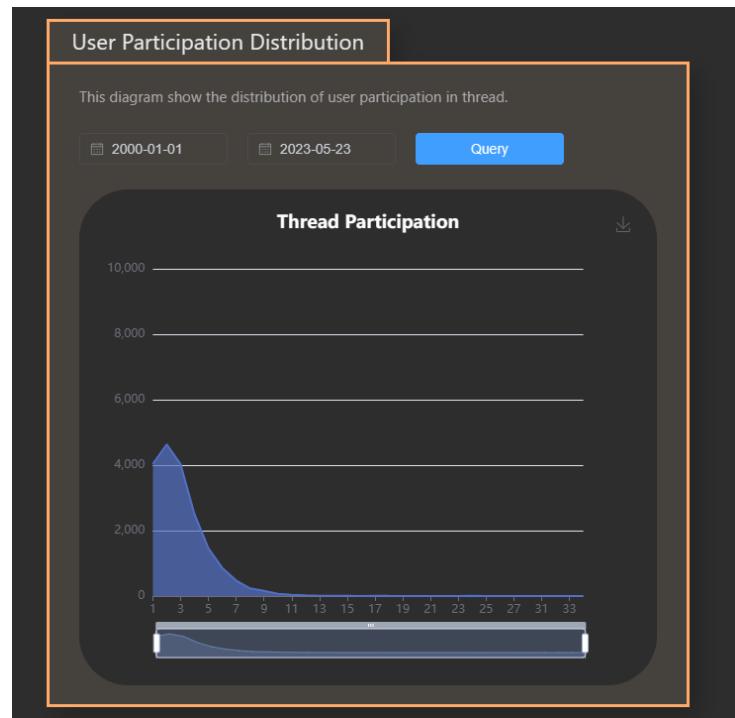
由于在StackOverflow中，标签组最多能设置 5 个标签，由于我们数据收集时默认筛选 `java`，所以此处可选择标签组标签个数范围为 1 到 4，此处以标签组标签个数为 3 举例，发现最经常出现的标签组合是 `android, firebase, google-cloud-firebase`、点赞数最高的标签组合是 `annotations, autowired, spring`、拥有最多浏览量的标签组是 `bit, integer, x`。



4.4 用户

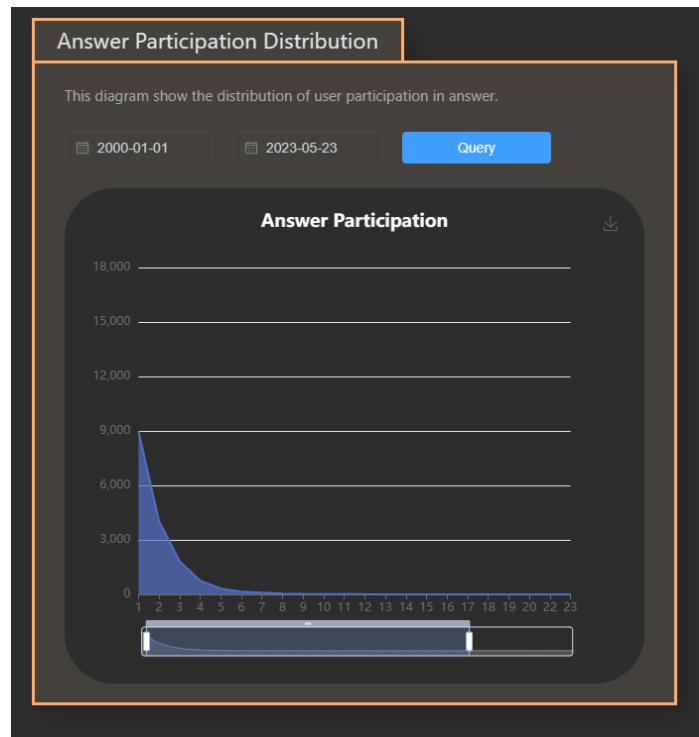
4.4.1 参与 Thread 讨论的用户数量的分布

根据下图可以得出，参与 Thread 讨论的用户数量的总体趋势是集中在 1 到 10 位用户讨论，随着参与讨论用户数目的增加，拥有相应参与讨论用户数目的问题数量波动下降。



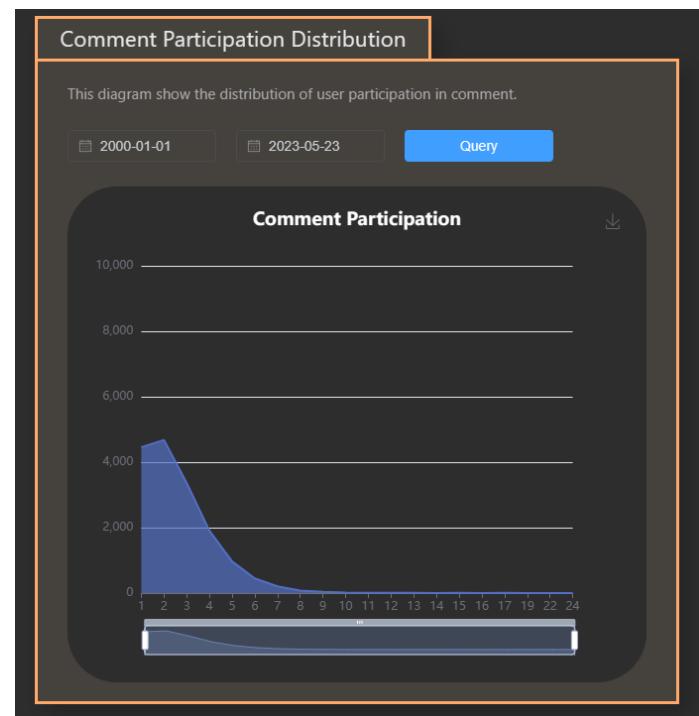
4.4.2 对于问题提出 Answer 的用户数量的分布

根据下图可以得出，参与 Answer 的用户数量的总体趋势是集中在 1 到 5 位用户讨论，随着参与讨论用户数目的增加，拥有相应参与用户数目的问题数量波动下降。



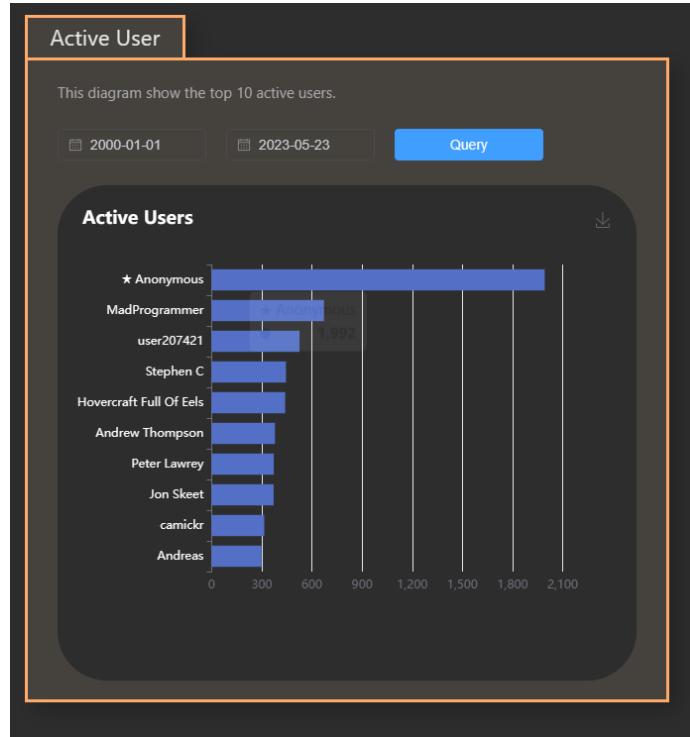
4.4.3 针对问题和对应答案参与 Comment 讨论的用户数量的分布

根据下图可以得出，参与 Comment 的用户数量的总体趋势是集中在 1 到 7 位用户讨论，随着参与讨论用户数目的增加，拥有相应参与用户数目的问题数量波动下降。



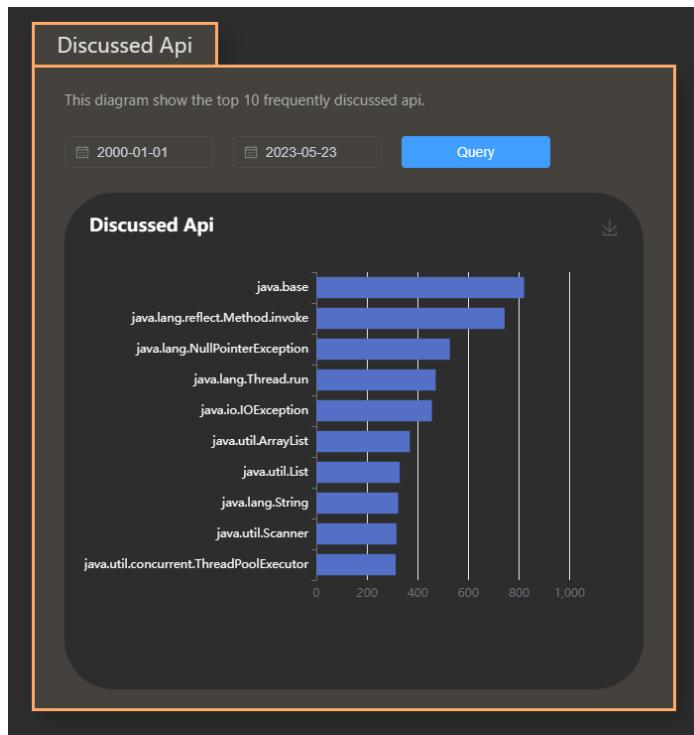
4.4.4 活跃用户

用户的活跃度定义为其提问数、回答数、评论数之和，我们统计了活跃度前10名的用户，以柱形图方式呈现。其中匿名（用五角星标出）表示用户以某种方式（注销、匿名参与）没有 `account_id` 可供识别账户的用户的活跃度。



4.5 最频繁讨论的 Java API

采用 Stanford NLP 分词统计所有收集到的 `body` 长文本内容，提取其中出现的 Java API 种类以及出现的次数进行统计，得到最频繁讨论的 Java API 的前10名。



5 RESTful API 接口文档

为了满足开发者使用本项目提供的 RESTful API，下面是本项目提供相关接口。

请求参数

字段	类型	描述
from	日期型	查询数据起始时间（采用yyyy-MM-ddTHH:mm:ss格式）
end	日期型	查询数据结束时间（采用yyyy-MM-ddTHH:mm:ss格式）
size	整数型	查询标签组中标签的个数

5.1 Number of Answers

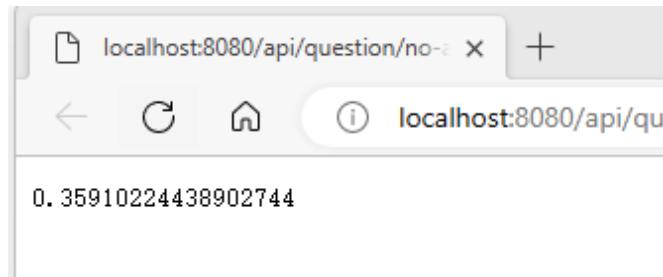
5.1.1 没有答案问题的百分比

```
GET /api/question/no-answer/percentage
```

请求参数： from (必填) 、 end (必填) 。

响应参数： 浮点数，没有答案问题的百分比。

示例： <http://localhost:8080/api/question/no-answer/percentage?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



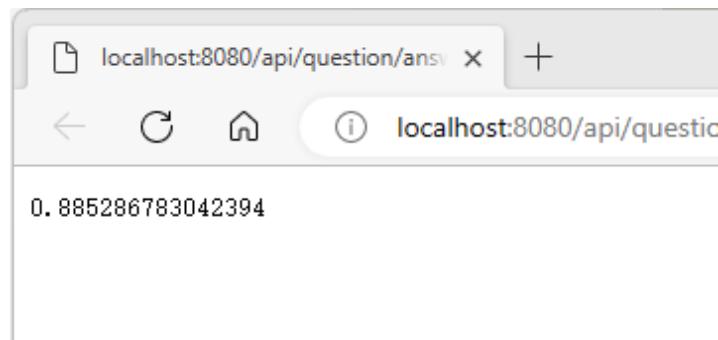
5.1.2 问题回答数量的平均值

`GET /api/question/answer/average`

请求参数: from (必填) 、 end (必填) 。

响应参数: 浮点数， 问题回答数量的平均值。

示例: <http://localhost:8080/api/question/answer/average?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



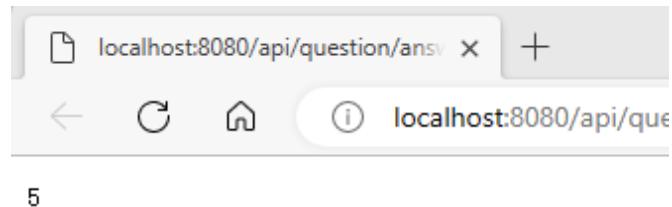
5.1.3 问题答案数量的最大值

`GET /api/question/answer/max`

请求参数: from (必填) 、 end (必填) 。

响应参数: 整数， 问题答案数量的最大值。

示例: <http://localhost:8080/api/question/answer/max?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



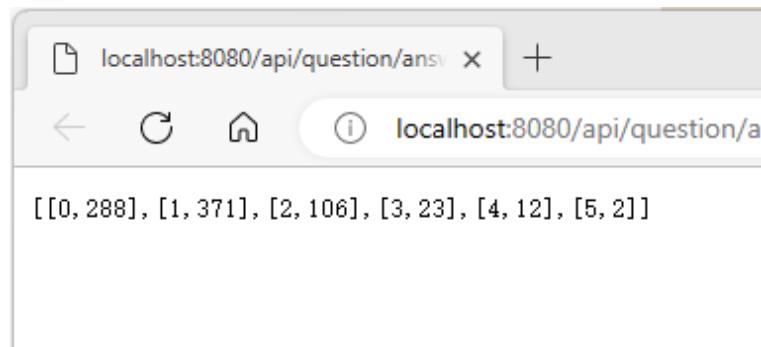
5.1.4 问题回答数量分布

GET /api/question/answer/distribution

请求参数: from (必填) 、 end (必填) 。

响应参数: 二元组集合，表示的是每一个特定回答数量的问题数量。如[1, 371]表示的是回答数只有1个问题数有371个。

示例: <http://localhost:8080/api/question/answer/distribution?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



5.2 Accepted Answers

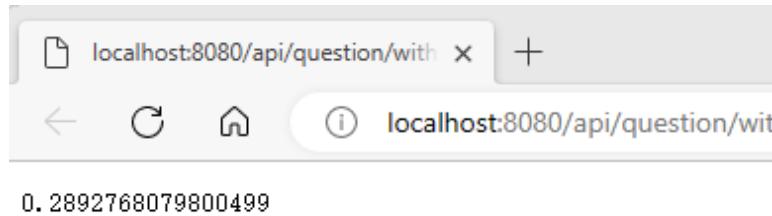
5.2.1 有被接受答案的问题（占问题总数）的百分比

GET /api/question/with-accepted-answer/percentage

请求参数: from (必填) 、 end (必填) 。

响应参数: 浮点数，有被接受答案的问题（占问题总数）的百分比。

示例: <http://localhost:8080/api/question/with-accepted-answer/percentage?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



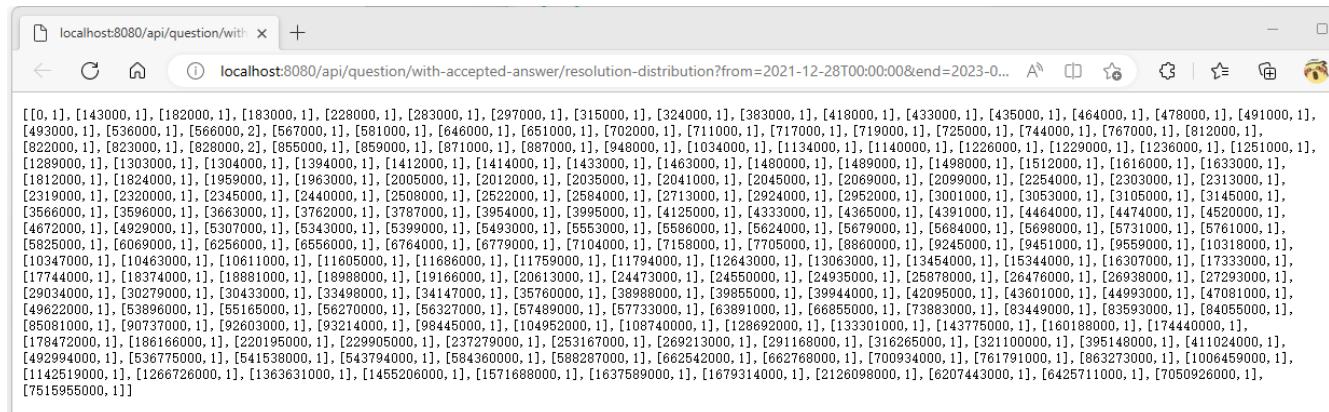
5.2.2 问题从提出到解决的时间间隔分布

`GET /api/question/with-accepted-answer/resolution-distribution`

请求参数： from (必填) 、 end (必填) 。

响应参数： 二元组集合，表示的是每一个解决时间间隔的问题数量，时间间隔单位为毫秒。如 [1289000, 1] 表示的是花了 1289000 毫秒才得到解决的问题数有一个。

示例： <http://localhost:8080/api/question/with-accepted-answer/resolution-distribution?from=2021-12-28T00:00:00&end=2023-02-01T00:00:00>。



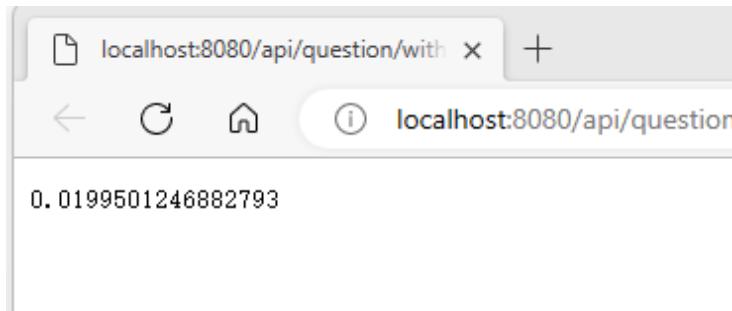
5.2.3 含有非接受答案的点赞数高于被接受答案问题的百分比

`GET /api/question/with-accepted-answer/better-answer/percentage`

请求参数： from (必填) 、 end (必填) 。

响应参数： 浮点数，没有答案问题的百分比。

示例： <http://localhost:8080/api/question/with-accepted-answer/better-answer/percentage?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



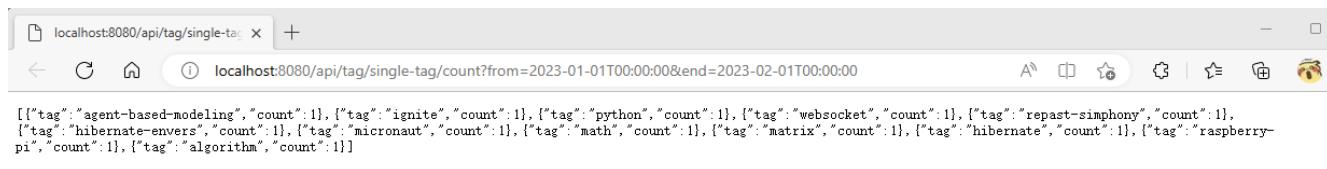
5.2.4 单 tag 出现的次数

GET /api/tag/single-tag/count

请求参数：from（必填）、end（必填）。

响应参数：Json格式数据，tag是tag的名称，count是这个tag的出现次数。

示例：<http://localhost:8080/api/tag/single-tag/count?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00>。



5.2.5 单 tag 的点赞数

GET /api/tag/single-tag/upvote

请求参数：from（必填）、end（必填）。

响应参数：Json格式数据，tag是tag的名称，count是这个tag的点赞数。

示例：<http://localhost:8080/api/tag/single-tag/upvote?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00>。



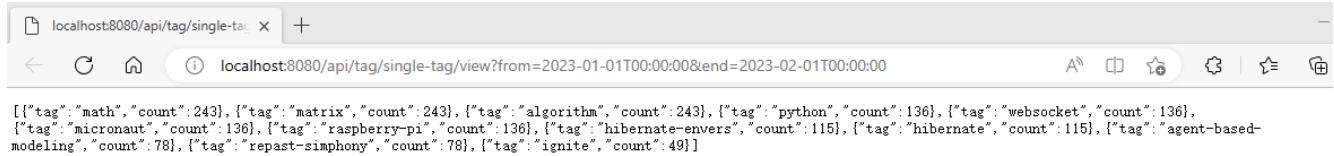
5.2.6 单 tag 的浏览量

GET /api/tag/single-tag/view

请求参数：from（必填）、end（必填）。

响应参数：Json格式数据，`tag`是tag的名称，`count`是这个tag的浏览量。

示例：<http://localhost:8080/api/tag/single-tag/view?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00>。



5.3 Tags

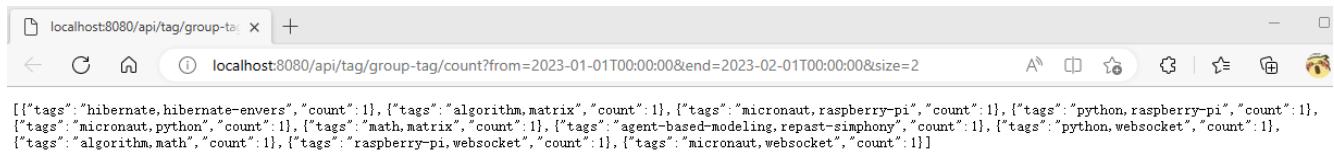
5.3.1 一定规模的tag组出现的次数

GET /api/tag/group-tag/count

请求参数：from（必填）、end（必填）、size（必填）。

响应参数：Json格式数据，`tags`是tag组的所有tag字典序排序后通过逗号分割拼起来的字符串，`count`是这个tag组的出现次数。

示例：<http://localhost:8080/api/tag/group-tag/count?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00&size=2>。



5.3.2 一定规模的tag组的点赞数

GET /api/tag/group-tag/upvote

请求参数：from（必填）、end（必填）、size（必填）。

响应参数：Json格式数据，`tags`是tag组的所有tag字典序排序后通过逗号分割拼起来的字符串，`count`是这个tag组的点赞数。

示例：<http://localhost:8080/api/tag/group-tag/upvote?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00&size=2>。

```
[{"tags": "hibernate,hibernate-envers", "count": 1}, {"tags": "algorithm,matrix", "count": 1}, {"tags": "math,matrix", "count": 1}, {"tags": "agent-based-modeling,repast-simphony", "count": 1}, {"tags": "algorithm,math", "count": 1}, {"tags": "micronaut,raspberry-pi", "count": 0}, {"tags": "python,raspberry-pi", "count": 0}, {"tags": "micronaut,python", "count": 0}, {"tags": "python,websocket", "count": 0}, {"tags": "raspberry-pi,websocket", "count": 0}, {"tags": "micronaut,websocket", "count": 0}]
```

5.3.3 一定规模的tag组的浏览量

GET /api/tag/group-tag/view

请求参数：from（必填）、end（必填）、size（必填）。

响应参数：Json格式数据，`tags` 是tag组的所有tag字典序排序后通过逗号分割拼起来的字符串，`count` 是这个tag组的浏览量。

示例：<http://localhost:8080/api/tag/group-tag/view?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00&size=2>。

```
[{"tags": "algorithm,matrix", "count": 243}, {"tags": "math,matrix", "count": 243}, {"tags": "algorithm,math", "count": 243}, {"tags": "micronaut,raspberry-pi", "count": 136}, {"tags": "python,raspberry-pi", "count": 136}, {"tags": "micronaut,python", "count": 136}, {"tags": "python,websocket", "count": 136}, {"tags": "raspberry-pi,websocket", "count": 136}, {"tags": "micronaut,websocket", "count": 136}, {"tags": "hibernate,hibernate-envers", "count": 115}, {"tags": "agent-based-modeling,repast-simphony", "count": 78}]
```

5.4 Users

5.4.1 问题讨论涉及到的用户数量分布

GET /api/user/participation-distribution

请求参数：from（必填）、end（必填）。

响应参数：二元组集合，表示的是每一个特定用户参与数的问题数量。如[1, 32]表示的是仅有一个用户参与的问题数量有32个。值得注意的是，这里不统计没有其它用户参与的问题。

示例：<http://localhost:8080/api/user/participation-distribution?from=2023-01-01T00:00:00&end=2023-02-01T00:00:00>。

```
[[1, 32], [4, 2], [5, 1]]
```

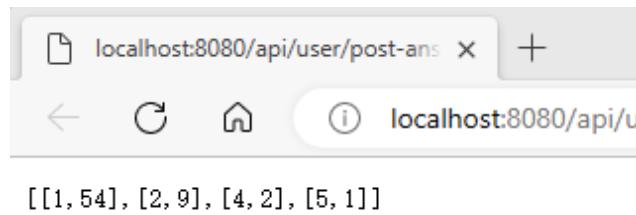
5.4.2 问题讨论涉及到的回答者数量的分布

GET /api/user/post-answer-distribution

请求参数：from（必填）、end（必填）。

响应参数：二元组集合，表示的是每一个特定回答者参与数的问题数量。如[1, 54]表示的是仅有一个回答者的问题数量有54个。值得注意的是，这里不统计没有回答的问题。

示例：<http://localhost:8080/api/user/post-answer-distribution?from=2022-12-01T00:00:00&end=2023-02-01T00:00:00>。



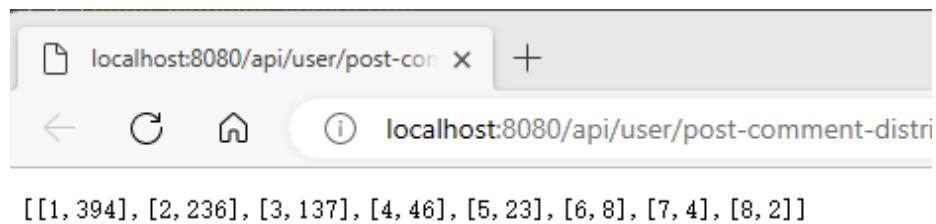
5.4.3 问题讨论涉及到的评论者数量的分布

GET /api/user/post-comment-distribution

请求参数：from（必填）、end（必填）。

响应参数：二元组集合，表示的是每一个特定评论者参与数的问题数量。如[1, 394]表示的是仅有一个评论者的问题数量有394个。值得注意的是，这里不统计没有评论的问题。

示例：<http://localhost:8080/api/user/post-comment-distribution?from=2022-01-01T00:00:00&end=2023-02-01T00:00:00>。



5.4.4 参与问题讨论的用户的活跃值

GET /api/user/activity

请求参数：from（必填）、end（必填）。

响应参数：Json格式数据，`activity`是用户的参与活跃数，`user`下是用户在`Stack Overflow`上的相关数据，包括头像、`accountId`、`userId`等数据信息。

示例：<http://localhost:8080/api/user/activity?from=2023-01-31T00:00:00&end=2023-02-01T00:00:00>。



```
[{"activity": 1, "user": {"accountId": 2139538, "userId": 1898563, "imageUrl": "https://www.gravatar.com/avatar/d99204bcce923f40f5c72b7f80121644?s=256&d=identicon&r=PG", "link": "https://stackoverflow.com/users/1898563/michael", "userType": "registered", "displayName": "Michael", "reputation": 41434}}, {"activity": 1, "user": {"accountId": 14035647, "userId": 10138416, "imageUrl": "https://i.stack.imgur.com/MivKj.png?s=256&g=1", "link": "https://stackoverflow.com/users/10138416/linker", "userType": "registered", "displayName": "linker", "reputation": 781}}, {"activity": 1, "user": {"accountId": 9571720, "userId": 7109162, "imageUrl": "https://www.gravatar.com/avatar/aef53d9fbec6c5d4c16a045de77c4e04?s=256&d=identicon&r=PG&t=y&so-version=2", "link": "https://stackoverflow.com/users/7109162/xtremebaumer", "userType": "registered", "displayName": "XtremeBaumer", "reputation": 6246}}]
```

5.5 Java API

5.5.1 查询不同的Java API出现次数

`GET /api/api/count`

请求参数：from（必填）、end（必填）。

响应参数：Json格式数据，`api`是Java API的完整路径，`count`是统计出现的次数。

示例：<http://localhost:8080/api/api/count?from=2023-01-01T00:00:00&end=2023-02-01T00:00>。



```
[{"count": 2, "api": "java.util.ArrayList"}, {"count": 1, "api": "java.util.Arrays"}, {"count": 1, "api": "java.util.regex.Matcher"}]
```