

海量图数据管理与挖掘大作业报告

graphweibo

0. 队伍信息

人员	学号
陈俊达	2001213077
时旻	2001213082
冯贤兵	2001213073

1. 需求和架构设计

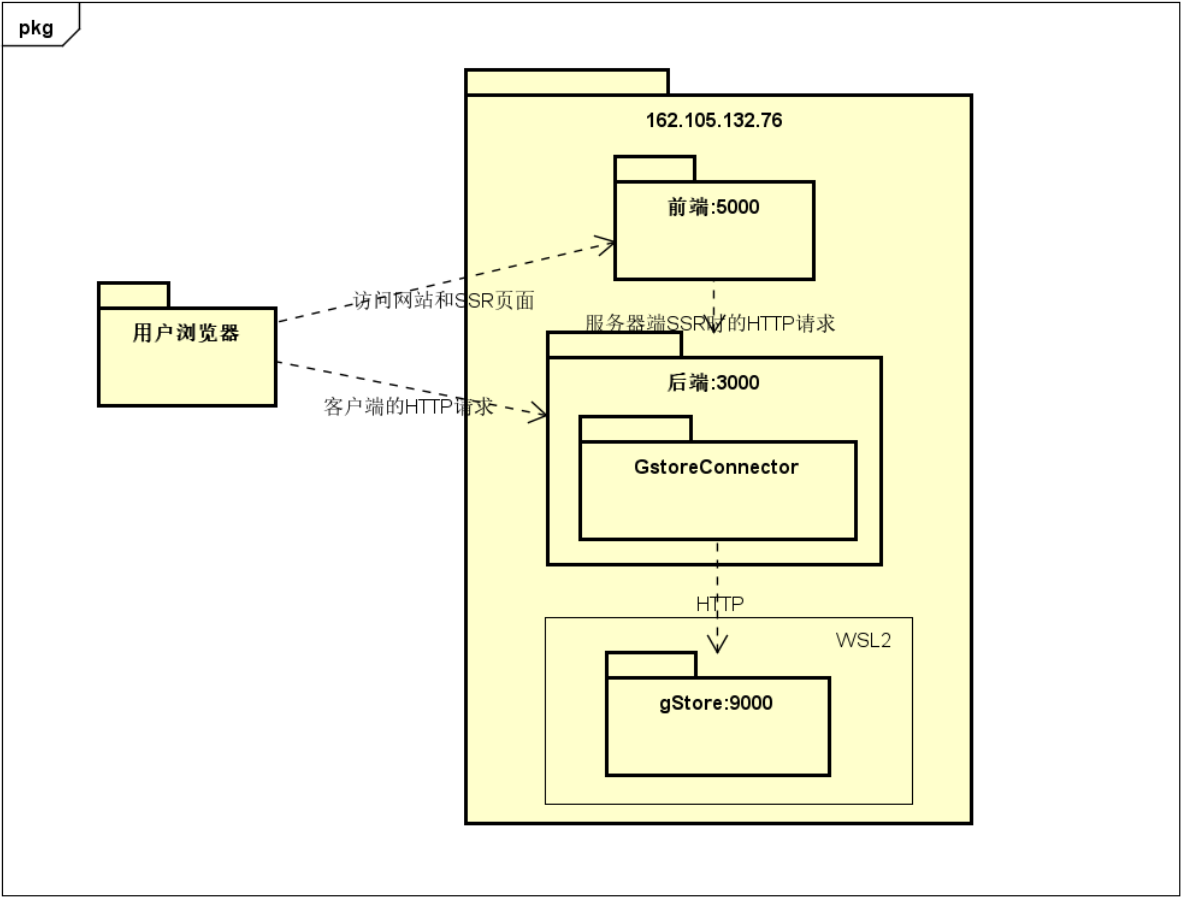
1.1 需求列表

系统各个需求的编号及描述如下表：

需求编号	描述
FR1	未登录用户查看系统中最新的微博
FR2	已登录用户查看自己关注的人发送的微博
FR3	搜索系统中的用户
FR4	查看系统中的用户的微博列表
FR5	查看系统中的用户的已关注的用户
FR6	查看关注系统中的用户的用户

FR7	已登录用户关注系统中的用户
FR8	已登录用户取消关注已经关注的用户
FR9	查看系统中的用户的用户名和 ID
FR10	已登录用户发送微博
FR11	查看两个用户之间四跳之内的关注关系
FR12	切换中文或者英文界面

1.2 架构设计及分工



系统架构图如上。系统分为前端、后端和 gStore 三个部分，分别运行在 5000、3000 和 9000 端口。每个部分的主要任务和人员如下表：

部分	主要工作	人员
前端	<ul style="list-style-type: none"> ● 实现前端 ● 定义前端所需的接口 	陈俊达
后端	<ul style="list-style-type: none"> ● 实现前端需要的接口和业务逻辑 ● 定义所需要的 gStore 数据的接口 	时旻
数据库	<ul style="list-style-type: none"> ● 部署和维护 gStore ● 编写 sparql, 实现后端所需要的接口 	冯贤兵

1.3 技术选型

本系统技术选型如下：

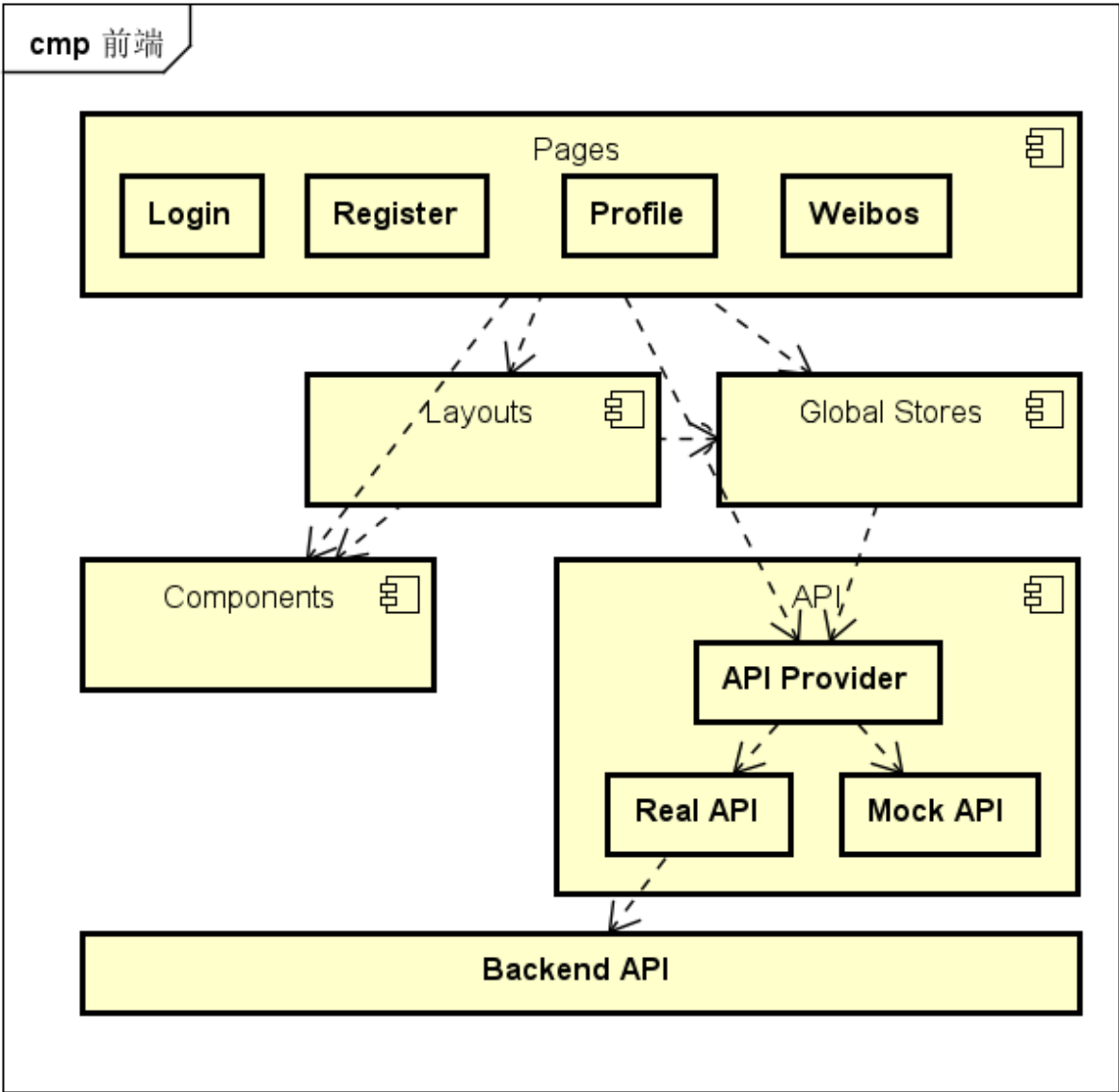
部分	技术	解释
	Next.js	服务器端渲染（SSR）
	React	前端框架
	TypeScript	编程语言
	grommet	UI 库
后端	Python	编程语言
	Flask	Web 框架
鉴权	JWT	实现鉴权
数据库	Python	编程语言
	gStore	图数据库

	SparQL	图数据库查询语言
--	--------	----------

2. 前端设计和实现

2.1 前端架构设计

前端架构如下图。



各个模块的解释如下表：

部分	作用
----	----

Pages	页面
Layouts	公共布局组件和代码
Global Stores	全局状态库，如用户登录状态
Components	公共组件，不依赖其他状态和网络
API	API 抽象层。所有网络请求必须经过 API 层。 API 层可以在真实 API 和提供假数据的 mock API 之间切换

2.2 前端分页

一些功能点（FR4, FR5, FR6 等）的数据为一个列表，当数据过多时，每次获取所有数据存在速度较慢，加大后端压力，延长网络请求时间等问题。所以，需要对这些数据的获取进行分页处理，即每次只取某一页的部分数据而非全部数据。

下图为获取一个用户发送过的微博列表功能(FR4)。此用户发送了 13 条微博，若每页 10 个微博，那么总共有 2 页，第二页有 3 条数据。



前端分页功能的实现需要三个数据：**第几页**，**每页多少项元素**以及**列表总共有多少项元**

素。其中，前端在获取这些数据时，需要指定获取第几个页的数据（querystring 中 page 参数）；整个系统中每页多少项元素是固定的，都是 10 项，可以在后端修改。后端在返回响应时，不仅要返回这一页的数据（response 中的 results），还需要返回总共有多少项元素（response 中的 totalCount）。前端通过总共有多少项元素以及每页有多少个元素可以计算出总共有多少页，并在用户点击对应页数时，请求对应页数的数据。

下图获得用户发送过的微博（FR4）的前端 API 定义，从中可以看到上文中提到的 page 参数和 results 和 totalCount 响应。

```
export interface WeiboGetByUserSchema {  
  querystring: {  
    /** 用户ID */  
    userId: string;  
    /**  
     * 页数。每页10项，从1开始，不设置的话，默认值为1  
     * @default 1  
     */  
    page?: number;  
  }  
  responses: {  
    200: {  
      totalCount: number;  
      results: WeiboResult[];  
    };  
    /** 用户不存在。 */  
    404: {}  
  }  
}
```

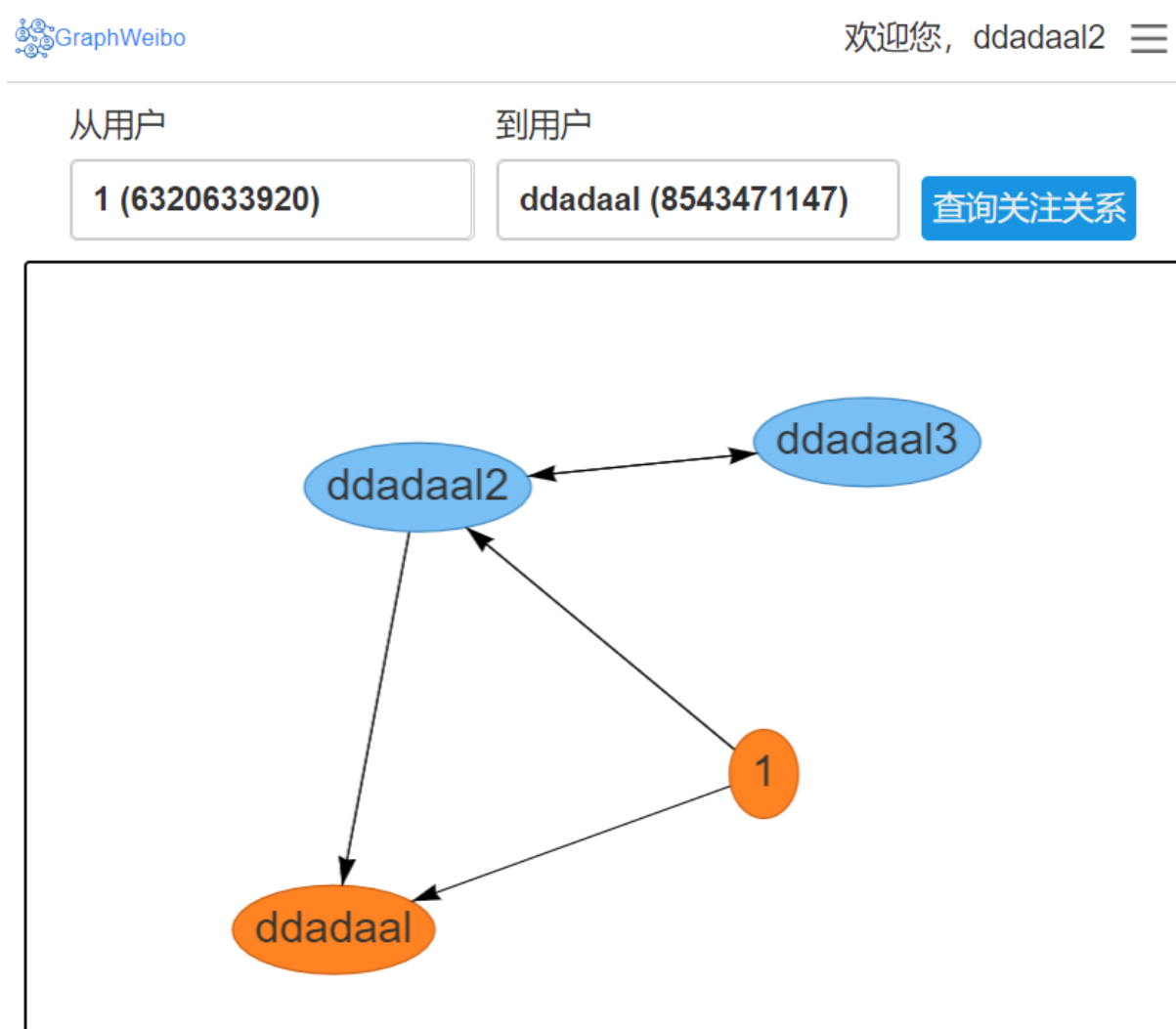
除此之外，刷微博的功能点（FR1，FR2）的数据是流式的，不需要获取总共有多少页数数据，只需要每次刷新都拿下一页数据就可以。这两个需求对应的响应中不存在 totalCount 参数，前端每次请求当前下一页的数据即可。

本系统还实现了刷到底部时自动获取下一页的功能，提高了用户体验。在这个功能的前端实现中，每当下拉到底部时，系统直接请求当前下一页的数据，并将数据拼接到已有的数据后面。

2.3 前端用户关注关系

前端实现了查看两个用户四跳之内的关注关系的功能。用户可以在界面中搜索关键词并选择用户，并点击查询关注关系查看两个用户之间的关注关系。系统获得关注关系后，将会将关注关系做图，开始和结束用户用其他颜色来表示以提高辨识度。系统会查看环形的关注关系（即 1、2 互相关注的情况），故查询结果的数据量以及相应做出来的图片可能较大。

下图为用户 1 到用户 ddadaal 的关注关系，其中箭头指向关系为关注关系，如 1 关注了 ddadaal 和 ddadaal2，ddadaal2 关注了 ddadaal 和 ddadaal3，ddadaal3 关注了 ddadaal2。



下图为前端获取这个功能的数据的 API 定义。前端将会提供需要查询的两个用户的 ID，需要获取两个数据做图：usernames 为用户 ID 到用户名的映射关系；paths 为所有源用户到目标用户之间的所有路径。例如[a, b, c]表示 a->b->c 这样一条路径。

```

export interface UserConnectionSchema {
  querystring: {
    fromUserId: string;
    toUserId: string;
  };
  responses: {
    /** 关系 */
    200: {
      /** 图里所有用户的ID到用户名的映射 */
      usernames: Record<string, string>;

      /**
       * 所有源用户和目标用户之间的路径。
       * 每个节点用ID表示，每个小数组为一条路径。
       * 路径中需要包含源节点和目标节点
       */
      paths: string[][];
    };
    /** 某个用户不存在 */
    404: {
      /** 来源用户不存在 */
      fromUserNotExists: boolean;
      /** 目标用户不存在 */
      toUserNotExists: boolean;
    };
  };
}

```

系统将会将 usernames 中的每个用户作为一个节点，paths 中的每条路径的每相邻两个节点之间有一条路径，之后把边和节点信息通过 vis.js 库进行做图。


```

const data = useMemo(() => {
  const edges = [] as Record<string, string>[];

  paths.forEach((path) => {
    range(0, path.length-1).forEach((i) => {
      edges.push({
        from: path[i],
        to: path[i+1],
        color: themeStore.theme === "dark" ? colors.white : colors.black,
      });
    });
  });

  const nodes = Object.entries(usernames).map(([userId, username]) => ({
    id: userId,
    label: username,
    color: userId === fromId ? colors.from
      : userId === toId ? colors.to : colors.intermediate,
  }));

  return { edges: new DataSet(edges), nodes: new DataSet(nodes) };
}, [usernames, paths, themeStore.theme]);

```

2.4 其他前端用户体验相关

系统还做了一些用户体验相关的工作，主要包含：

1. 在查看用户之间的关系时，用户可以在文本框中输入用户名的关键词，系统会根据用户输入的关键词搜索对应的用户，用户可以在给出的建议中选择自己要搜索的用户。

从用户	到用户
123	
<div> <div>迎接新的一年123 (1002294350)</div> <div>苏州老黄-123 (1010536397)</div> <div>东现12345 (1031154335)</div> <div>bihailantian2012 (1047944662)</div> <div>bowie312 (1053089350)</div> <div>指上弹兵123 (1065355394)</div> </div>	

2. 实现了中文和英文两种语言，可以通过底部热切换显示语言
3. 支持了黑暗模式
4. 进行了响应式设计，可以支持从手机到电脑端多种屏幕大小的设备的正常显示和使用



3. 后端设计与实现

后端部分主要起到一个承前启后的作用，接收前端的请求，调用数据库相关函数，并将查询结果返回至前端。

3.1 Flask 框架

后端主要使用 Flask 框架完成。Flask 是一个使用 Python 编写的轻量级 Web 应用框架，它被称为微框架(microframework)，在保持代码简洁的同时还做到了易于扩展，其主要特征是核心构成比较简单，但具有很强的扩展性和兼容性。

Flask 的基本模式为在程序里将一个视图函数分配给一个 URL，每当用户访问这个 URL 时，系统就会执行给该 URL 分配好的视图函数，获取函数的返回值并发送相应的 Response。

例如在如下的函数中，对于'/weibo'的后缀，如果请求的方法是 GET，就进行发微博的操作，如果请求的方法是 POST，就进行获取微博的操作，并根据数据库查询状态，返回成功或

否的响应。

```
# 发微博
@weibo.route('', methods=['GET', 'POST'])
def getWeibo():
    if request.method == 'POST':
        # 获取微博内容
        data = json.loads(request.get_data())
        contents = data['content']
        # 获取当前用户
        identity = identify(request)
        if identity['state']:
            userID = identity['msg']
        else:
            return Response(status=401)
        result = postWeibo(userID, contents)
        if result['state']:
            return Response(status=200)
        else:
            return Response(status=404)

    if request.method == 'GET':
        # 获取查询ID
        data = request.args
        querystr = data['userId']
        page = get_page(data)
        # 获取该用户的所有微博
        result = getUserWeibo(querystr, page)
        if result['state']:
            ans, count = result['result']
            return Response(json.dumps({'results': ans, 'totalCount': count}), status=200)
        else:
            return Response(status=404)
```

此外，我们还使用 Blueprint 进行模块化设计，便于管理与后续扩展。使用 Blueprint 时，首先需要注册相关模块，如下图所示，对每个模块的前缀等进行整体定义，然后在每个模块的文件中进行后续路由以及函数定义即可。

在本项目具体实现中，后端分为三个模块，分别是 user、profile 和 weibo，其中 user 模块负责处理用户关注相关操作，profile 模块进行用户个人信息获取，weibo 模块则与用户发微博、刷微博等行为相关。

```

from flask import Flask
from flask_cors import CORS
from flask import Flask

from main.profile.profile import profile
from main.weibo.weibo import weibo
from main.user.user import user

app = Flask(__name__)
CORS(app, supports_credentials=True)
app.register_blueprint(user, url_prefix='/user')
app.register_blueprint(profile, url_prefix='/profile')
app.register_blueprint(weibo, url_prefix='/weibo')

app.config['SECRET_KEY'] = '...selfgenerated'
app.debug = True

```

3.2 JWT 用户鉴权

在用户的登录过程中，我们使用 JWT (Json Web Token) 实现基于 Token 的用户认证授权。JWT 是基于 Token 认证方式的一种机制，用于在用户和服务器间传递认证用户身份信息以便获取资源，是实现单点登录认证的一种有效方法。相比以往的 session 认证，JWT 是无状态的，当用户登入登出时无需在服务器端保留用户认证与会话信息，从而减少了服务器压力。

JWT 需要前端与后端配合进行用户认证与身份管理，其运行流程如下所示：

- 用户注册/登录时后端生成包含用户 ID 的 Token，返回至前端
- 前端将得到的当前用户的 Token 保存在 cookie 中
- 前端在后续的请求的 header 中加入 Token（如下图所示）
- 后端读取 header 中的 Token，获得当前的用户 ID
- 进行后续处理，获取与用户身份相关的信息

▼ General

Request URL: http://162.105.132.76:3000/weibo

Request Method: POST

Status Code: 200 OK

Remote Address: 127.0.0.1:1080

Referrer Policy: strict-origin-when-cross-origin

▶ Response Headers (10)

▼ Request Headersview source

Accept: */*

Accept-Encoding: gzip, deflate

Accept-Language: zh,en-US;q=0.9,en;q=0.8

authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaWMyMDYzMzk5MCIsImV4cCI6MTYxMTE0NTA5OXB0.WzI3U7nfLkszvMvdKcaExiExFqAmZ05Wh96J_QwebBo

Cache-Control: no-cache

Content-Length: 23

content-type: application/json

Host: 162.105.132.76:3000

Origin: http://localhost:5000

Pragma: no-cache

Proxy-Connection: keep-alive

Referer: http://localhost:5000/

User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4386.0 Mobile Safari/537.36 Edg/89.0.767.0

▼ Request Payloadview source

▼ {content: "哈哈哈"}

content: "哈哈哈"

在后端，定义了以下三个函数进行用户的身份管理：

- encodeToken：实现对用户 ID 的加密，返回一个与用户身份相关的 Token，也可以在 Payload 中添加其他个人信息，以及该身份过期的时间等。
- decodeToken：与上述操作相反，将 Token 解码为用户 ID。
- identify：实现从 request 的 header 中读取相关用户 Token，若读取成功则解密获取当前用户 ID，若读取失败则返回相关错误信息。

```

6 def encodeToken(userID) -> str:
7     # 生成认证Token
8     payload = {
9         'user_id': userID,
10        'exp': datetime.utcnow() + timedelta(seconds=600)
11    }
12    return jwt.encode(payload, key=scret_key, algorithm='HS256').decode()
13
14 def decodeToken(token: str):
15     # 验证Token
16     try:
17         payload = jwt.decode(token, scret_key, algorithm='HS256', options={'verify_exp': False})
18         if 'user_id' in payload:
19             return payload
20         else:
21             raise jwt.InvalidTokenError
22     except jwt.ExpiredSignatureError:
23         return 'Token过期'
24     except jwt.InvalidTokenError:
25         return '无效Token'
26
27
28 def identify(request):
29     # 用户鉴权
30     auth_header = request.headers.get('Authorization')
31     if auth_header:
32         auth_tokenArr = auth_header.split(" ")
33         if (not auth_tokenArr or auth_tokenArr[0] != 'Bearer' or len(auth_tokenArr) != 2):
34             result = {'state': False, 'msg': '请传递正确的验证头信息' }
35         else:
36             auth_token = auth_tokenArr[1]
37             payload = decodeToken(auth_token)
38             userID = payload['user_id']

```

4. 数据库

4.1 环境搭建

数据库基于 gStore 实现，首先下载 gStore 并编译，然后通过 d2rq 将 sql 文件转化为 RDF 文件，以原 sql 文件中的表结构作为谓词，表内容分别为主语和宾语。然后通过 gbulid 命令生成基于 nt 文件的数据库。接下来采用 ghttp 通信的方式对数据库进行增删查改的操作，即使用 ghttp 挂载数据库并在 9000 端口监听消息。前期环境搭建工作结束。

接下来与后端工程师协商好 API 后，通过 GitHub 共享需求函数列表 sql_func_desp,相关链接如下，

https://github.com/ddadaal/graphweibo/blob/master/backend/sql_func_desp.md，同时选择调用官方文档中的 GstoreConnector.py。

4.2 设计函数

1. 注册用户，首先判断用户 uname 是否存在，如果已经存在则返回 false。接着给该用户分

配一个独一无二的 uid 用于做之后的标识。接着向数据库中插入三元组，包括 uid 对应的密码，用户名，创建时间等。。

2. 用户登录，用传入的用户名和密码生成 sparql 语句查询数据库，若存在对应的三元组，则返回成功，否则失败
3. 获取用户信息，传入 uid，查询该 uid 对应的用户名，注册时间，微博数等并返回。
4. 判断是否关注，传入 uid1 和 uid2，判断 uid1 是否关注了 uid2
5. 关注用户，传入 uid1 和 uid2，实现 uid1 对应的用户关注 uid2 对应的用户。首先判断 uid1 和 uid2 是否相等，若相等则返回 false，否则判断 uid2 是否存在，若否则返回 false，接着判断 uid1 是否已经关注了 uid2，若是则返回 false，接着插入 2 个三元组，即 `userrelation/{uid1}/{uid2}> vocab:userrelation_tuid '{uid2}'`，`userrelation/{uid1}/{uid2}> vocab:userrelation_suid '{uid1}'`。接着更新用户信息，获取 uid1 的关注数目，并+1 后写回，获取 uid2 的粉丝数目然后+1 并写回。
6. 取关用户，传入 uid1 和 uid2，实现 uid1 取关 uid2 的操作，首先判断 uid1 是否等于 uid2，如果相等则返回 false，否则删除两个三元组，即 `userrelation/{uid1}/{uid2}> vocab:userrelation_tuid '{uid2}'`，`userrelation/{uid1}/{uid2}> vocab:userrelation_suid '{uid1}'`，接着更新用户信息，获取 uid1 的关注数目，并-1 后写回，获取 uid2 的粉丝数目然后-1 并写回。
7. 分页查询，由于查询的数据量过大，如果一次将所有的查询数据返回会导致速度很慢，所以设计了分页查询。其中设置了一个 query_count 的 bool 值，如果这个值为 true，则需要返回一个值告知前端此时有多少页的数据，比如查询某个用户。而当刷最新的微博时，则传入的值为 false。同时会传入一个 page 值告知当前要查询第几页的数据。接着编写 sparql 语句使用 order by, limit, offset 关键词查询分页结果。
8. 获取粉丝信息，传入一个 uid，进行分页查询，对查询的结果打包返回。
9. 获取关注的人信息，传入一个 uid，进行分页查询，对查询的结果打包返回。
10. 搜索用户，进行带正则的分页查询得到备选用户列表，接着对列表中的每个用户查询微博数，粉丝数，互相关注关系等，返回一个用户信息列表。
11. 发微博，首先随机生成一个 16 位独一无二的微博 id，接着将 weiboid 与 uid，以及 weibotex 构成多个三元组插入到数据库中。接着更新用户信息，即发微博数目自增。
12. 获取某个用户的微博，传入一个 uid 和 page，即要查该用户的第几页的微博，进行分页

查询后，将返回的数据提取出 weiboid，发送时间，内容等打包返回。

13. 获取正在关注的用户微博，首先判断传入的 uid 是否存在，如果不合法则返回 false，接着查询 uid 关注的用户列表，接着按照发送时间进行分页查询，获取 weiboid，username，sendtime，content 等打包好返回。
14. 获取最新微博，该功能在未登录时可以直接使用，即按时间获取最新的微博。直接按照发送时间进行分页查询，获取 weiboid，username，sendtime，content 等打包好返回。
15. 查询跳数，给定 uid1 和 uid2 以及跳数 depth，返回长度为 depth 的关注链。首先判断跳数是否等于 1，若是则直接查询关注关系即可，若存在关注关系，则返回[uid1, uid2]的列表。否则编写以下 sparql 语句对查询结果进行分割，得到查询长度为 depth 的关注链（列表），返回这个关注链。

```
q = prefix + f"""
select * where {{
    {os.linesep.join((f'
        ?r{i} vocab:userrelation_suid ?mid_{i} .
        ?r{i} vocab:userrelation_tuid ?mid_{i+1} .
    ' for i in range(depth)))
    .replace("?mid_0", f"'{uid1}'")
    .replace(f"?mid_{depth}", f"'{uid2}'")
}}
"""
```

16. 获取用户之间的关系，给定 uid1 和 uid2，判断 uid1 是否在一定长度内存在关注 uid2 的关系。首先判断 uid1 和 uid2 是否都存在，若否则返回 false，接着对跳数开始一个长度为 4 的循环，每个循环内进行一次查询跳数，若在当前跳数内存在关注链，则记录下来，同时查询关注链内的 uid 对应的 uname，最终返回结果

5. 运行截图

5.1 功能实现

未登录的用户可以看到系统中最新的微博（FR1）。往下滑动可以刷新更多的微博。





123



猪_Sir

2014-05-12 09:10

#同桌的你#怀着一种心情看完了《同桌的你》，映像回到了那些年，再致青春！好久没有这种怦然心动的感觉，很投入地看完了，追忆着曾经经历的一切，找到了共鸣的节拍。。。感叹时间都去哪儿了的同时，内心却又那么一丝小忧伤！！或许时间定格了一切，未来更显憧憬。。。



细雨小雨

2014-05-12 09:09

看了同桌的你，感触：趁着年轻想做什么就去做，同一件事现在的我会疯狂，会追逐，十年后的我已经没有了勇气，或者说现实太锋利了，只有在最单纯的时候，爱就去追，想就去做，不要就走，要做不简单的单纯



欢乐girl1989

2014-05-12 09:09

和大学归途中火车上遇到的投缘小妹初次逛街，话唠的我们扫荡完肯德基，进影院观影《同桌的你》，此影片感觉不错奥，随后抱着大桶爆米花，继续榆林街小吃



学习是终生的修炼

2014-05-12 09:08

忽然想起同桌的你里面的一个场景。林一去国外公司面试，拿着一张英语四级成绩单。。。。老外面试官问what is cet[晕][吃惊][挖鼻屎][挖鼻屎][挖鼻屎][挖鼻屎]一种默默

用户可以通过登录->注册入口注册自己的账号。注册成功后将会自动登录。

注册

用户名

密码

☒ 记住我

注册

去登录

已经注册的用户名已经被占用，系统将会报错。

用户名已经被占用，请重新换一个!

注册

用户名

密码

☒ 记住我

注册

去登录

已经有账号的用户也可以直接登录自己的账号。

登录

用户名

密码

☒ 记住我

[登录](#)

[去注册](#)

登录后，系统将会自动进入自己关注的人的微博列表界面（FR2）。由于新用户没有关注用户，所以这里的列表是空的。

有什么新鲜事想告诉大家?

发送

加载更多微博



点击搜索用户链接 (FR3), 可以搜索系统中的所有用户。



123
微博数 0

粉丝 2 已关注 3

关注关系

关注



screenshot
微博数 0

粉丝 0 已关注 0

关注关系

自己



肥羊羊咩咩咩
微博数 1123

粉丝 362 已关注 282

关注关系

关注



菠萝蚕宝宝
微博数 1825

粉丝 233 已关注 106

关注关系

关注




硬爆料
微博数 20745

粉丝 9311 已关注 665


关注关系

关注

没有登录的用户也可以进行搜索（以及后续的查看用户信息的功能），但是不能关注一个用户。



主页 搜索用户 用户关系 登录




123

微博数 0

粉丝 2

已关注 3




screenshot

微博数 1

粉丝 0

已关注 1




肥羊羊咩咩咩

微博数 1123

粉丝 362

已关注 282




菠萝蚕宝宝

微博数 1825

粉丝 233

已关注 106




硬爆料

微博数 20745

粉丝 9311

已关注 665



邹毅的邹

微博数 714

粉丝 501

已关注 699

在搜索框中输入搜索关键词，系统将会搜索用户名中包含有这个关键词的用户并返回。

ddadaal

**ddadaal5**
微博数 1

粉丝 1 已关注 1

关注关系

关注

**ddadaal**
微博数 0

粉丝 3 已关注 1

关注关系

关注

« < 1 > »



用户可以在这里点击关注 (FR7)，关注一个用户。

关注成功!



ddadaal

**ddadaal5**
微博数 1

粉丝 2 已关注 1

关注关系

已关注

**ddadaal**
微博数 0

粉丝 3 已关注 1

关注关系

关注

« < 1 > »



也可以对已经关注的用户点击取消关注按钮，来取消关注一个用户（FR8）。



关注用户后，可以在主页看到自己关注的用户发送的最新微博（FR2）。

有什么新鲜事想告诉大家?

发送



ddadaal5

2021-01-21 09:16

哈哈!

加载更多微博



在微博或者搜索框中点击用户名，可以显示对应用户的个人信息和微博列表（FR9、FR4）。



ddadaal5

用户ID 3790976491

微博(1)

Ta关注的用户(1)

关注Ta的用户(2)



ddadaal5

2021-01-21 09:16

哈哈!

« < **1** > »



若微博太多，系统将会分页显示。关注用户的、搜索用户等数据量较大的界面也都实现了分页显示。



1

用户ID 1129990945

微博(11)

Ta关注的用户(3)

关注Ta的用户(2)



1

2021-01-20 20:22

123

« ‹ 1 2 › »



点击 Ta 关注的用户链接可以查看这个用户所关注的用户 (FR5)。

**ddadaal5**

用户ID 3790976491

[微博\(1\)](#)[Ta关注的用户\(1\)](#)[关注Ta的用户\(2\)](#)**123**

微博数 0

粉丝

2

已关注

3

[关注关系](#)[关注](#)

« < 1 > »



点击关注 Ta 的用户链接可以显示关注这个用户的用户 (FR6)。

**ddadaal5**

用户ID 3790976491

[微博\(1\)](#)[Ta关注的用户\(1\)](#)[关注Ta的用户\(2\)](#)**123**

微博数 0

粉丝

2

已关注

3

[关注关系](#)[关注](#)**screenshot**

微博数 0

粉丝

0

已关注

1

[关注关系](#)[自己](#)

« < 1 > »



在首页可以输入微博并点击发送来发送微博（FR9）。发送后的微博可以显示在自己以及关注自己的用户的首页。



在搜索用户界面、关注自己的用户、自己关注的用户等有“关注关系”按钮的界面点击关注关系按钮，可以查询自己和这个用户之间的关注关系（FR11）。

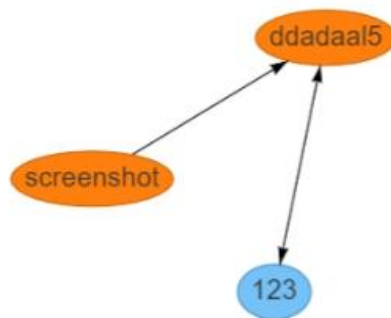
用户也可以点击导航栏上的用户关系链接，在文本框中输入用户名关键词，在弹出的用户中选择用户，并点击查询关系按钮查询两个用户之间的关系。

从用户

screenshot (0739719691)

到用户

ddadaal5 (3790976491)

[查询关注关系](#)

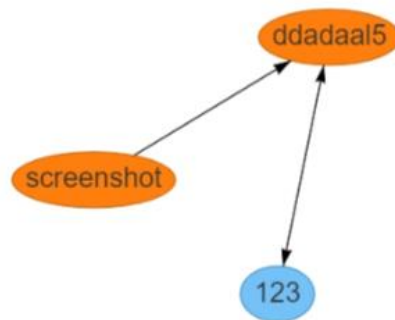
点击底部 English 和简体中文链接可以快速切换界面中各个文本显示的语言 (FR12)。

from User

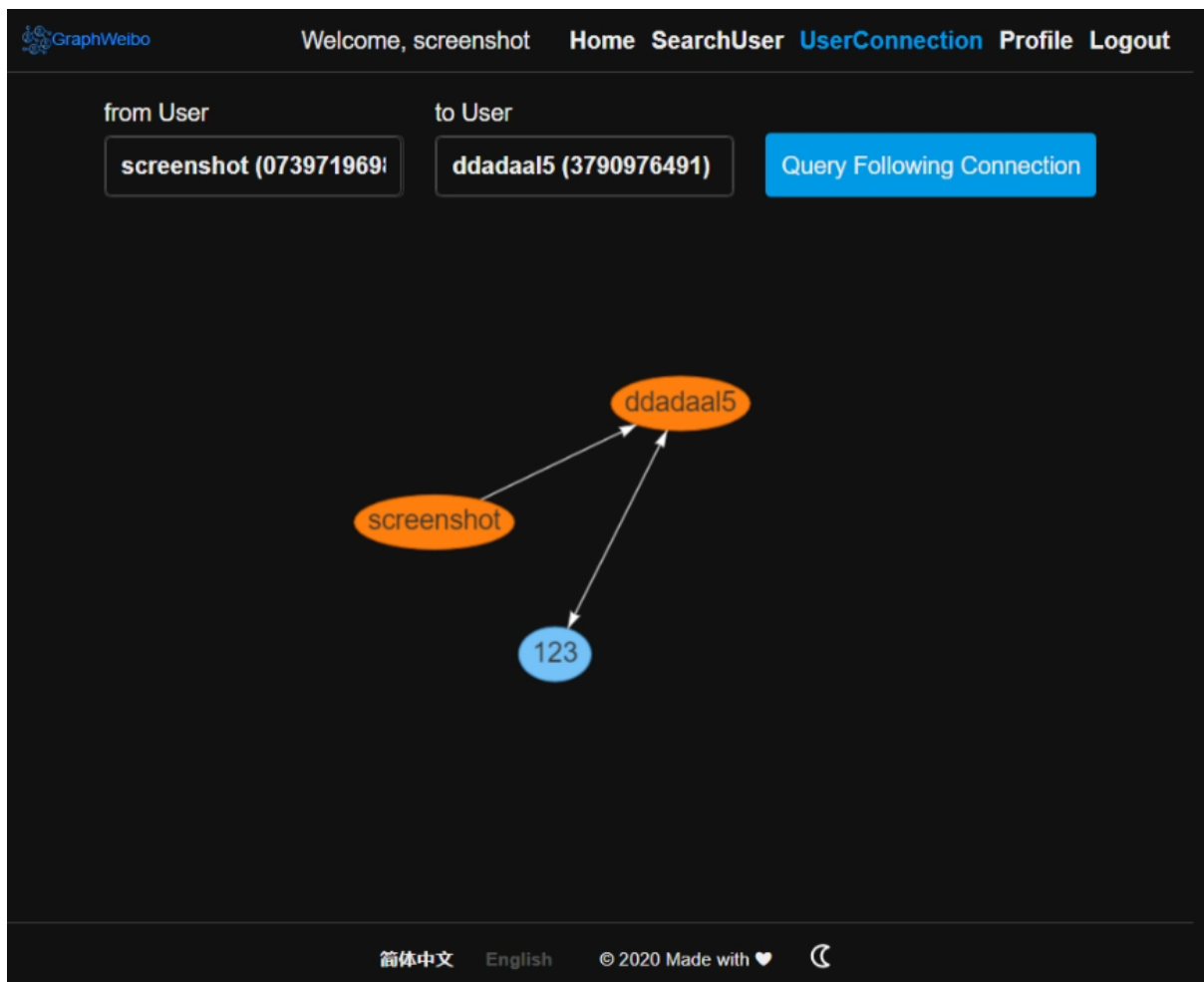
screenshot (0739719691

to User

ddadaal5 (3790976491)

[Query Following Connection](#)

点击右边的阳光图标可以切换黑暗模式，也在黑暗模式下点击同样的位置切换回来。



5.2 后端日志

后端 Flask 框架日志截图。

```
162.105.132.76 - - [21/Jan/2021 14:46:14] "GET /weibo/new?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:14] "OPTIONS /weibo/new?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:14] "GET /weibo/new?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "OPTIONS /user/token?username=1&password=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "GET /user/token?username=1&password=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "OPTIONS /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "GET /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "OPTIONS /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:17] "GET /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:27] "OPTIONS /weibo HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:27] "POST /weibo HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:27] "OPTIONS /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:27] "GET /weibo/followings?page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:28] "OPTIONS /user/search?query= HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:28] "GET /user/search?query= HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:30] "OPTIONS /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:30] "OPTIONS /weibo?userId=1000080335&page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:30] "GET /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:30] "GET /weibo?userId=1000080335&page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:32] "OPTIONS /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:32] "OPTIONS /user/followings?userId=1000080335&page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:32] "GET /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:32] "GET /user/followings?userId=1000080335&page=1 HTTP/1.1" 200 -162.105.1
32.76 - - [21/Jan/2021 14:46:33] "OPTIONS /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:33] "OPTIONS /user/followers?userId=1000080335&page=1 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:33] "GET /profile?userId=1000080335 HTTP/1.1" 200 -
162.105.132.76 - - [21/Jan/2021 14:46:33] "GET /user/followers?userId=1000080335&page=1 HTTP/1.1" 200 -
```