

本科实验报告

实验名称：数据采集实验

学 员	<u>李 瀚 迅</u>	学 号	<u>202002723003</u>
培养类型	<u>非指挥类</u>	年 级	<u>2020 级</u>
专 业	<u>网 安</u>	所属学院	<u>理 学 院</u>
指导教员	<u>马 行 空</u>	职 称	<u>教 授</u>

中国人民解放军国防科技大学

目录

1 实验目的与内容	2
2 评分标准	2
3 项目背景	3
4 原理学习	4
4.1 微博接口学习	4
4.2 Scrapy 框架	8
4.3 Flask 框架	10
5 系统框架及实现	11
5.1 后端实现	13
5.1.1 人物信息	13
5.1.2 博文信息	19
5.1.3 评论信息	21
5.1.4 数据存储	23
5.2 反反爬措施	25
5.3 多线程加速及系统优化	31
5.4 前端实现	31
5.4.1 搜索页面	31
5.4.2 展示界面	35
5.4.2.1 导航栏	35
5.4.2.2 图表区	39
6 实验总结	41

1 实验目的与内容

社交平台中的大 V 用户对舆论导向具有重要影响力，应当对其进行重点监控跟踪，避免网络谣言、不当言论等向外扩散。为此，对于给定的任意用户 ID，本课题要求采集以下信息：

- 基本属性信息：昵称、真实姓名、所在地、性别、生日、博客地址、个性域名、简介、注册时间、邮箱、QQ、MSN、职业信息、教育信息。
- 社交关系信息：所有关注人和粉丝（如果关注人数量或者粉丝数量超过 10，则只采集前 10 个），每个人的信息包括用户昵称、链接地址、关注人数、粉丝数、微博数、所在地。
- 动态信息：所有发帖和转发贴（如果发帖和转发贴的总量超过 10，则只采集前 10 条），每个帖子的信息包括发帖时间、发帖内容、转发次数、评论次数、点赞次数、是否是转发、按照热度排序的前 10 条评论（评论人 ID、评论人昵称、评论时间、评论内容、点赞次数）。当用户更新了发帖或转发后，应在 1 分钟内监控到该变化，并及时更新本地信息。

2 评分标准

1. 自动登录：数据采集前，个人账号应通过程序自动登录，若遇到验证码可在程序中手动输入并继续登录。（10 分）
2. 热点社会微博采集（20 分）
3. 热点动态更新（15 分）
4. 热点信息关联（20 分）
5. 可视化：能够以 Web、App 等形式较美观的展示采集到的数据（15 分）
6. TA 评定：检查代码是否存在抄袭嫌疑、质询方法过程、质询代码逻辑、是否采用创新方法（多线程加速、防检测等），综合给出评判。（20 分）

3 项目背景

网络信息采集是指在互联网上收集、获取、分析、处理和利用信息的过程。网络信息采集可以应用于多个领域，例如市场调研、舆情监测、搜索引擎优化等。根据采集方式的不同，网络信息采集可以分为以下几类：爬虫技术、API 接口和模拟登录。当今主流的爬虫技术如下：

1. Scrapy

Scrapy 是一个 Python 编写的开源网络爬虫框架，用于快速、高效地爬取网站数据，支持分布式爬虫、页面解析、数据存储等功能。Scrapy 使用 Twisted 异步网络框架，可以同时处理多个请求和响应。Scrapy 还提供了丰富的中间件和扩展，可以自定义下载器、解析器等，使其更加灵活。

2. BeautifulSoup

Beautiful Soup 是一个 Python 库，可以从 HTML 和 XML 文档中提取数据。它可以解析 HTML 和 XML 标记，并将文档转换为树形结构，方便我们对网页内容进行遍历和操作。Beautiful Soup 的解析速度较快，可以定位标签、属性等，也支持 CSS 选择器。

3. PySpider

PySpider 是一个 Python 编写的分布式网络爬虫框架，可以自定义爬虫流程、调度器、解析器等。PySpider 支持 JavaScript 渲染，可以爬取动态网页，支持分布式部署和调度，可以快速抓取大量数据。

4. Selenium

Selenium 是一个自动化测试工具，也可用于爬虫。Selenium 可以模拟浏览器的行为，支持多种浏览器，可以实现模拟登录、抓取动态网页等操作。Selenium 相对于其他爬虫工具来说，需要较多的计算资源和时间，但是它对于一些需要模拟人类操作的场景来说是非常有用的。

这些爬虫技术各有特点，适用于不同的爬虫场景。例如，Scrapy 适用于大规模数据的爬取，而 BeautifulSoup 适用于小规模数据的提取，Selenium 适用于需要模拟人类操作的场景。在实际应用中，也可以结合使用这些技术，根据实际情况选择合适的爬虫方案。

微博 API 接口是微博开放平台提供了一种数据获取方式，通过 API 接口可以获取微博用户信息、微博内容、评论信息等数据。与其他数据获取方式相比，使用微博 API 接口可以获得更加准确和全面的数据，并且可以避免一些反爬虫策略的限制。

与微博 API 接口相比，使用 Scrapy 等爬虫技术进行数据采集的方式也有其优劣。Scrapy 等爬虫技术可以针对不同的网站进行数据采集，不受 API 接口的限制，可以获取更加全面的数据。此外，使用爬虫技术可以自定义数据的处理方式，可以进行数据清洗、筛选等操作，获取更加精确的数据。

然而，使用爬虫技术进行数据采集也存在一些问题。首先，由于网站反爬虫机制的存在，使用爬虫技术采集数据的效率和准确性受到了一定的限制。其次，采集数据的过程中容易被识别为恶意行为，导致 IP 被封禁等问题。最后，使用爬虫技术进行数据采集的合法性也需要考虑，需要遵守相关法律法规，保护个人隐私和数据安全。

4 原理学习

4.1 微博接口学习

新浪微博 API 的调用是需要获取用户身份认证的（用户授权）。目前微博开放平台用户身份鉴权主要采用的是 OAuth2.0。从流程图中可以看到，为调用 API 服务器内容，需要将 access token 告诉 API 服务器；而 access token 是在用户授权后由新浪（授权服务器）返回给我们创建的应用的；为完成用户授权，我们的应用首先要将授权页面给到用户（authorization request）。

用户授权后浏览器的 URL 大概长这样：<https://api.weibo.com/oauth2/default.html?code=0acebd79dc4cdd04879699e803af038a>

我们需要向新浪授权服务器提交 code 后面的字符串才能获得 access token。相当于告诉新浪服务器我们的应用已经得到用户的授权，现在可以访问用户的数据了，于是授权服务器给到我们通行证（access token），就可以从 API 服务器获取微博数据了。理解了上面的机制，我才可以在后续的代码编写中实现自动化获取 code。

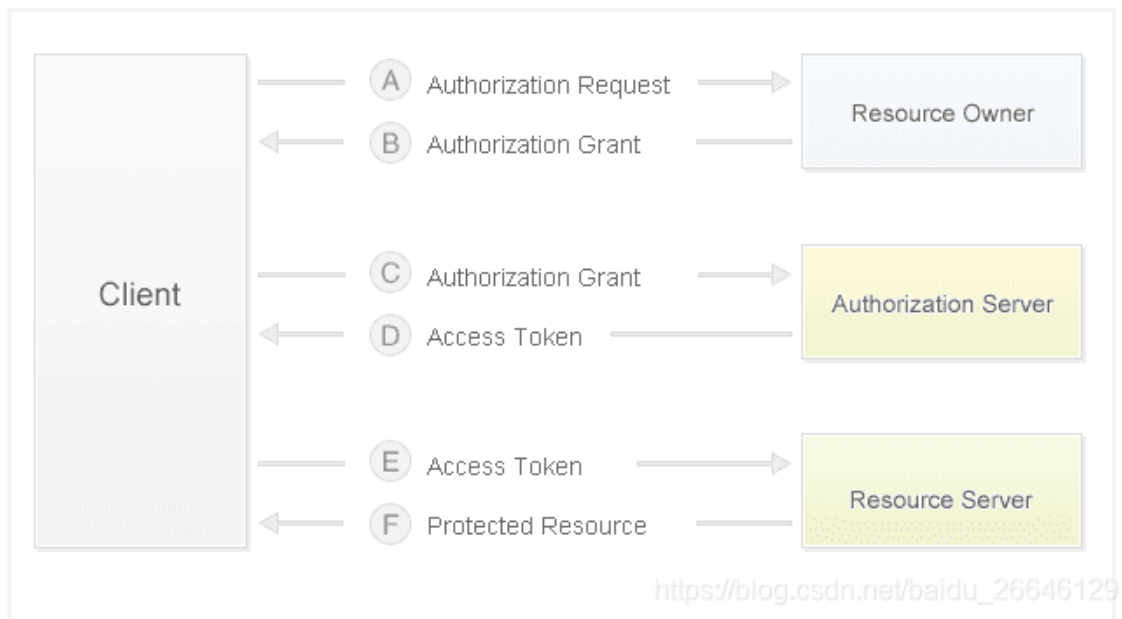


图 1: API 认证过程

新浪微博 API 为用户提供了多种操作，本项目中可能有用的主要有如下几个：

1. users/show：授权之后获取用户信息
2. statuses/user_timeline：获取授权用户发布的微博
3. comments/show：获取某条微博的评论列表
4. friendships/followers：获取用户的关注列表

但自 2013 年 7 月 2 日起，微博开放平台将对用户关系读取、微博内容读取类接口进行升级；接口升级后：uid 与 screen_name 只能为当前授权用户，第三方微博类客户端不受影响。因此上述的第二、四个功能只能用于自己的用户名和 ID，没有实用价值。

下面的代码展示了如何使用微博 API 接口。

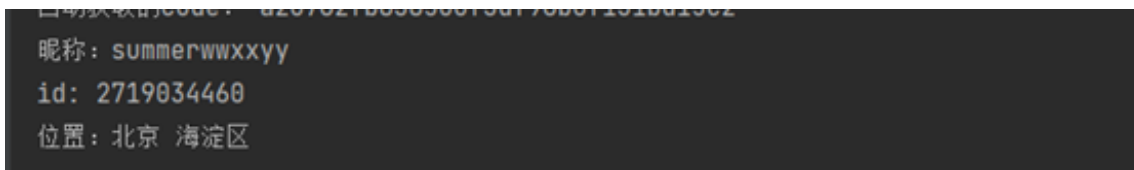
```
1 client = APIClient(app_key=app_key, app_secret=app_secret,  
2                     redirect_uri=callback_url)  
3 url=client.get_authorize_url() # 得到授权页面的 url  
4 webbrowser.open_new(url) # 打开这个 url
```

```

5 code = input('输入 url 中 code 后面的内容后按回车键: ')
6 r = self.client.request_access_token(code)
7 # 新浪返回的 token, 类似 abc123xyz456
8 access_token = r.access_token
9 expires_in = r.expires_in
10 # 设置得到的 access_token
11 client.set_access_token(access_token, expires_in)
12 # 根据用户名查询用户信息
13 result = client.users__show(screen_name=name)
14 print(u'昵称: ' + result['screen_name'])
15 print(u'id: ' + str(result['id']))
16 print(u'位置: ' + result['location'])

```

得到的输出如图 2 所示:



```

昵称: summerwwwxyy
id: 2719034460
位置: 北京 海淀区

```

图 2: API 运行结果

上述代码实际上运行起来非常麻烦, 因为 code 的获取需要让我们用 `webbrowser.open_new(url)` 手动获取, 在尝试网上的做法没有成功后, 我尝试出一种可以自动获取 code 的方法。首先需要我们提供 `app_key`, `app_secret`, `callback_url`, `username` 和 `password`, 借助 `urllib.parse` 手动构造 http 请求。

```

1 conn = http.client.HTTPSConnection('api.weibo.com')
2 postdata = urllib.parse.urlencode(
3     {'client_id': app_key, 'response_type': 'code', 'redirect_uri':
4     callback_url, 'action': 'submit', 'userId': username, 'passwd':
5     password, 'isLoginSina': 0, 'from': '', 'regCallback': '',
6     'state': '', 'ticket': '', 'withOfficalFlag': 0})
7 url = 'https://api.weibo.com/oauth2/authorize?' + postdata

```

```
8 conn.request('GET', url)
9 res = conn.getresponse()
10 location = res.getheader('Location')
11 code = location.split('=')[1]
```

上述代码根据提供的四个信息构造 url 访问 `https://api.weibo.com/oauth2/authorize?` (参数), 在浏览器中我追踪网络流, 可以看到该页面的交互过程。



图 3: 网络流

如图 4, 服务器返回的 response 中的 location 字段告诉了我 code 的值, 只需要提取该字段即可。但实际上, 上述代码没有成功返回 code, 经过对比, 我发现浏览器的请求比我的 request 请求多了 cookie 字段, 由于 cookie 在一段时间内均有效, 因此我复制浏览器的 cookie, 增添到 conn.request 的请求中, 随后可以成功自动获取 code。



图 4: cookie 字段

最终自动得到的 code 如图 5 所示。

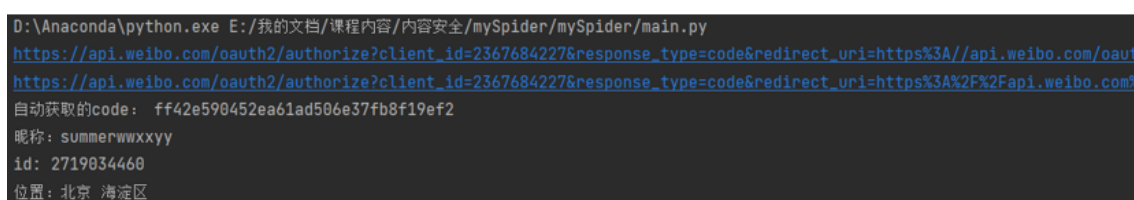


图 5: 自动获取 code

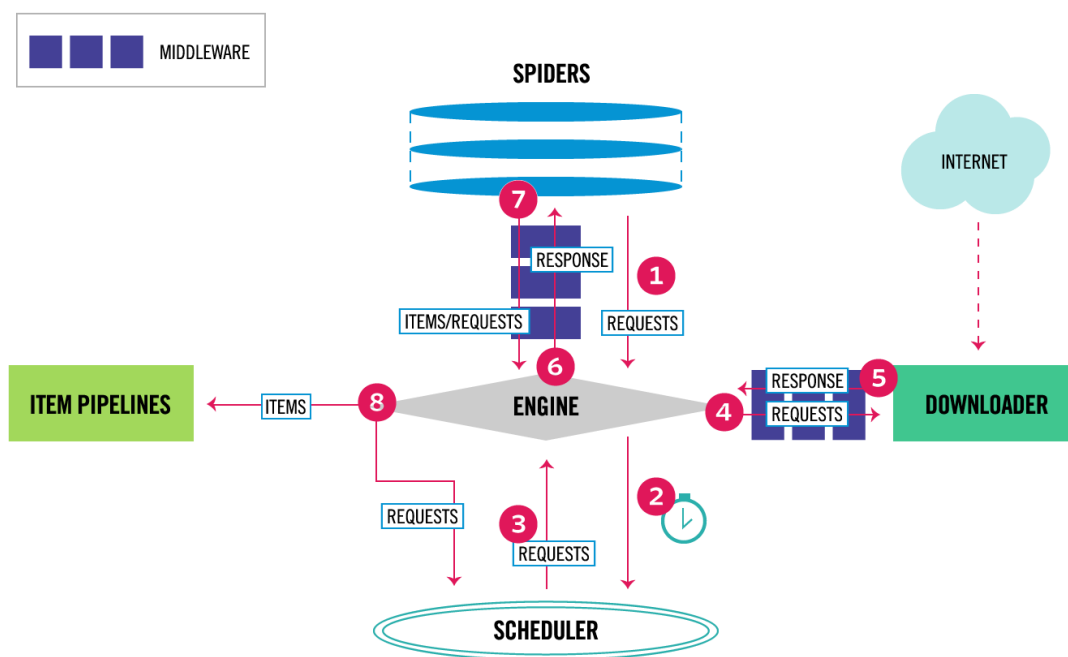
4.2 Scrapy 框架

Scrapy 是适用于 Python 的一个快速、高层次的屏幕抓取和 web 抓取框架，用于抓取 web 站点并从页面中提取结构化的数据。Scrapy 用途广泛，可以用于数据挖掘、监测和自动化测试。其系统框架如图 6 所示。

Scrapy 有如下基本模块：

1. 调度器 (Scheduler)

调度器，说白了把它假设成为一个 URL（抓取网页的网址或者说是链接）的优先队列，由它来决定下一个要抓取的网址是什么，同时去除重复的网址（不



<https://blog.csdn.net/javakkk100>

图 6: scrapy 框架

做无用功)。用户可以自己的需求定制调度器。

2. 下载器 (Downloader)

下载器，是所有组件中负担最大的，它用于高速地下载网络上的资源。Scrapy 的下载器代码不会太复杂，但效率高，主要的原因是 Scrapy 下载器是建立在 twisted 这个高效的异步模型上的（其实整个框架都在建立在这个模型上的）。

3. 爬虫 (Spider)

爬虫，是用户最关心的部份。用户定制自己的爬虫（通过定制正则表达式等语法），用于从特定的网页中提取自己需要的信息，即所谓的实体 (Item)。例如使用 Xpath 提取感兴趣的信息。用户也可以从中提取出链接，让 Scrapy 继续抓取下一个页面。

4. 实体管道 (Item Pipeline)

实体管道，用于接收网络爬虫传过来的数据，以便做进一步处理。例如验证实体的有效性、清除不需要的信息、存入数据库（持久化实体）、存入文本文件等。

5. Scrapy 引擎 (Scrapy Engine)

Scrapy 引擎是整个框架的核心，用来处理整个系统的数据流，触发各种事件。它用来控制调试器、下载器、爬虫。实际上，引擎相当于计算机的 CPU，它控制着整个流程。

6. 中间件

整个 Scrapy 框架有很多中间件，如下载器中间件、网络爬虫中间件等，这些中间件相当于过滤器，夹在不同部分之间截获数据流，并进行特殊的加工处理。

Scrapy 的具体使用将在系统框架中详细叙述。

4.3 Flask 框架

Flask 是一个 Python 编写的 Web 微框架，让我们可以使用 Python 语言快速实现一个网站或 Web 服务。下面的代码展示了一个最基本的 flask 应用。

```
1 from flask import Flask
2 app = Flask(__name__)
3 @app.route('/')
4 def hello_world():
5     return 'Hello World'
6 if __name__ == '__main__':
7     app.run()
```

首先我们导入了 Flask 类。接着我们创建一个该类的实例。第一个参数是应用模块或者包的名称。如果使用一个单一模块（就像本例），那么应当使用 name，因为名称会根据这个模块是按应用方式使用还是作为一个模块导入而发生变化（可能是 'main'，也可能是实际导入的名称）。这个参数是必需的，这样 Flask 才能知道在哪里可以找到模板和静态文件等东西。然后我们使用 route() 装饰器来告诉 Flask 触发函数的 URL。函数名称被用于生成相关联的 URL。函数最后返回需要在用户浏览器中显示的信息。

在系统框架中将其进行详细的分析。

5 系统框架及实现

本项目实现了微博用户信息采集的功能，从个人信息、关注信息、粉丝信息、微博信息和评论信息的角度进行爬取。在后端采用 Scrapy 爬取，前端使用 Flask 框架进行呈现，前后端采用 MySQL 数据库进行交互。本项目无论是前端还是后端都是纯手写代码，并未套用任何现成的项目代码。最终的前端展示界面如图 7。



sid	创建时间	文本	发送设备	阅读数	转发数	评论数
4893374819862663	Sat Apr 22 17:15:34 2023	分享图片		14741451	1000000	466803
4892329041728138	Wed Apr 19 20:00:01 2023	#蔡徐坤突破新引力#每一次进阶，是破与立，也是奇妙的引力相吸。4月21日，一同见证@汤臣倍健Yep#胶原蛋白新一代硬核升级#http://t.cn/A6NMh5An	微博视频号	16007234	1000000	911500
4891468407769470	Mon Apr 17 11:00:10 2023	任务达成@gentlemonsterofficial SECRETCODE:#CRYSTALCLEAR# 即将迎来下一个挑战，准备好了吗？ #GENTLEMONSTERXKUN#http://t.cn/A6NVXG9a	微博视频号	18753168	1000000	1000000
4889641595176790	Wed Apr 12 10:01:04 2023	这瓶百事可乐无糖我请你。@百事中国 http://t.cn/A6NbWZUQ	微博视频号	24818348	1000000	1000000
4889278952505521	Tue Apr 11 10:00:02 2023	#蔡徐坤Meco蜜谷代言人# Meco果汁茶，美味好搭档，缤纷果汁，原茶现萃，酸甜果味，百搭对味！很高兴成为@Meco蜜谷果汁茶品牌代言人，希望未来和Meco一起，解锁更多对味体验！ http://t.cn/A6N4oGaR	微博视频号	36405227	1000000	1000000
4888668772765935	Sun Apr 09 17:35:25 2023			36088555	1000000	1000000
4887897049931509	Fri Apr 07 14:28:52 2023	Makeeachdaycount		29507769	1000000	1000000
4885409286851655	Fri Mar 31 17:43:23 2023	追时间的人		39055900	1000000	1000000
4885111286534670	Thu Mar 30 21:59:14 2023	与@TAGHeuer泰格豪雅一起成为#追时间的人#，即刻开启时间新篇章，竞速不止，传奇不熄！#泰格豪雅卡莱拉系列60周年##卡莱拉冠军赛车手表#		23208281	1000000	1000000
4884723861556853	Wed Mar 29 10:00:00 2023	你那里几点？		37747718	1000000	1000000

图 7: 发文列表

前后端的架构如图 8,9 所示。

总体实现逻辑为，首先运行 route.py 在本地启动服务器，展示 index.html 的搜索界面。当用户输入搜索的用户名时，route.py 根据输入信息调用 run.py 执行 Scrapy 框架进行爬虫爬取，并在框架中实现存入数据库，待爬取完成后跳转 search.html 展示结果页面，该页面读取数据库内容并将其以表格的形式呈现。其 python 运行过程如图 10 所示。

5.1 后端实现

本项目中有三类不同的数据，分别为人物信息，博文信息和评论信息。上述三类信息的组织结构、提取字段和网址规律完全不一样，需要分开编写。不仅如此，后端还需要处理采集到的数据，进行存储，同时对微博的反爬措施进行反反爬，并对爬虫性能进行优化。

5.1.1 人物信息

人物信息包含某用户自己填写的个人信息。粉丝、用户和关注的人的信息均可套用人物信息框架。我首先在浏览器中对某用户的个人信息进行查看，如图 11 所示。



图 11: 用户信息

浏览器的 `fetch` 和 `xhr` (`XMLHttpRequest`) 都是用于发送 HTTP 请求的技术，它们可以从服务器端获取数据并将其呈现在页面上。

`fetch` 是一个比 `xhr` 更先进的技术，它支持 Promise API，可以更容易地编写异步代码。`fetch` 使用简单，只需要传入一个 URL 和一些可选的参数，就可以发送一个 HTTP 请求。当请求成功时，它返回一个 `Response` 对象，我们可以使用这个对象来操作响应数据。

`xhr` 是一个比较老的技术，但它仍然被广泛使用。`xhr` 使用回调函数来处理异

步请求，它具有更多的配置选项，例如设置请求头、发送表单数据等等。当请求成功时，xhr 会触发一个回调函数，我们可以在回调函数中处理响应数据。

因此进一步查看该页面的 Network->Fetch/XHR 获取页面内容。

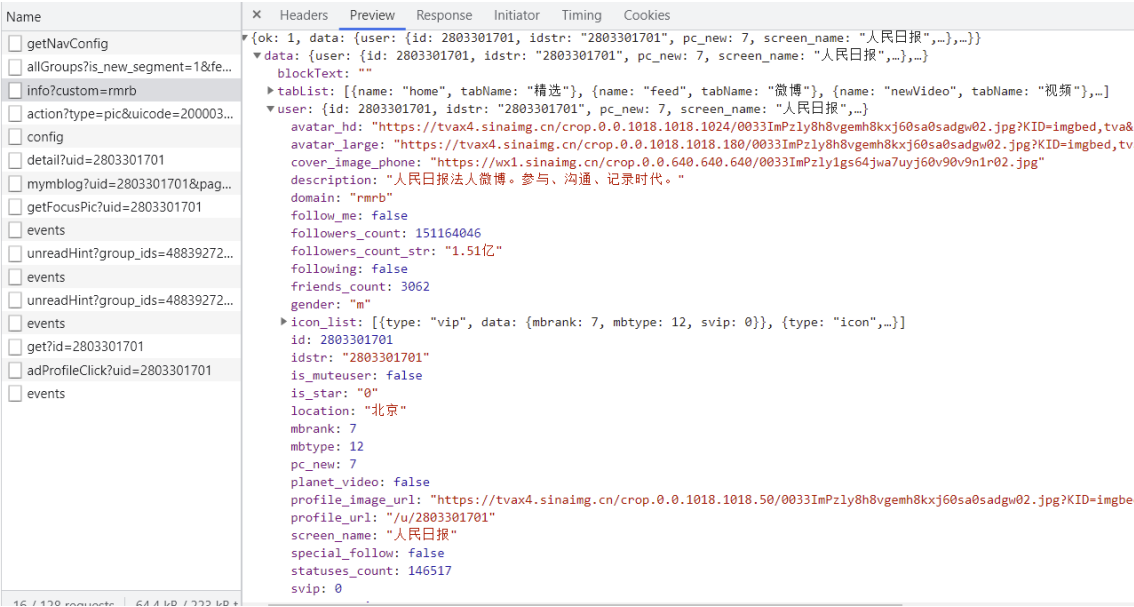


图 12: 用户信息

可以看到对 <https://weibo.com/ajax/profile/info?uid=2803301701> 的 fetch 请求返回了一个类似字典的数据，包含了人民日报的所有用户信息。而我们单独用浏览器访问该网站也可以获取相应内容。

除此之外，我还注意到另一个网址 <https://weibo.com/ajax/profile/detail?uid=2803301701> 包含了更多的用户详细信息，如图所示。

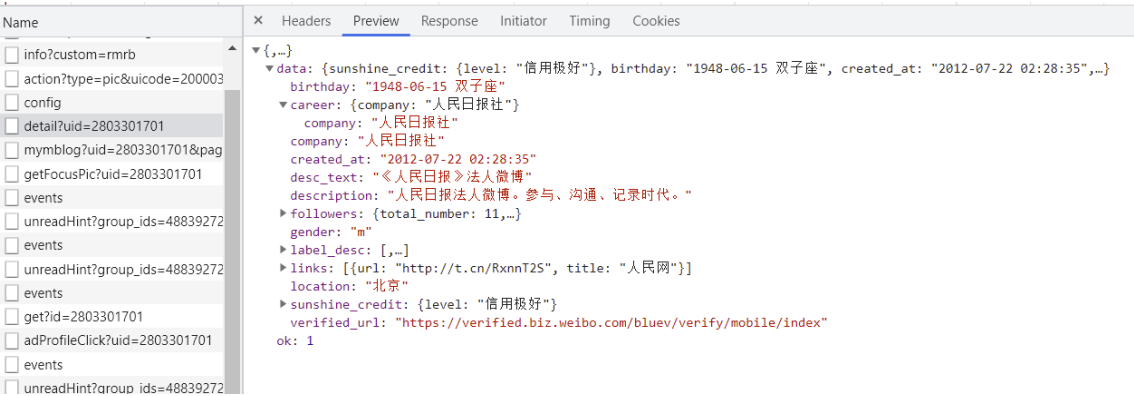


图 13: 用户详细信息

我们首先根据上述页面呈现的内容提取如下人物信息。

```
1 class UserItem(scrapy.Item):
2     """ 用户信息字段 """
3     # 告诉 pipeline 得到的是粉丝还是关注者还是自己,
4     # 0-> 自己, 1-> 粉丝, 2-> 关注的人
5     group = scrapy.Field()
6     uid = scrapy.Field() # 用户 ID
7     screen_name = scrapy.Field() # 用户昵称
8     followers_count = scrapy.Field() # 粉丝数
9     follow_count = scrapy.Field() # 关注数
10    description = scrapy.Field() # 简介
11    gender = scrapy.Field() # 性别
12    location = scrapy.Field() # 位置
13    statuses_count = scrapy.Field() # 微博数量
14    verified = scrapy.Field() # 否是微博认证用户, 即加 V 用户
15    # 用户的微博会员等级, 0: 普通用户、1-6: 微博会员等级
16    mbrank = scrapy.Field()
17    birthday = scrapy.Field() # 生日
18    regist_time = scrapy.Field() # 注册时间
19    qq = scrapy.Field() # QQ
20    real_name = scrapy.Field() # 真实姓名
21    profession = scrapy.Field() # 职业信息
22    education = scrapy.Field() # 教育信息
23    lable = scrapy.Field() # 标签
24    credit = scrapy.Field() # 信誉
25    ip_location = scrapy.Field() # ip 属地
```

接下来我们继续根据信息字典定义每个字段的提取规则, 在 `utils.py` 中实现提取规则 (较为简单不在此展示了)。在该 `py` 文件中, 我对提取出的文字用正则表达式进行了预处理, 避免无法识别的字符或表情造成错误。该函数如下:


```

1 def standardize_sentence(sentence):
2     p = re.compile(
3         u'['u'\U0001F300-\U0001F64F' u'\U0001F680-\U0001F6FF'
4         u'\u2600-\u2B55 \U00010000-\U0010ffff]+' )
5     sentence = re.sub(p, '', sentence).replace(' ', '')
6     return sentence

```

接下来我进行粉丝信息搜集, 点击进入粉丝界面, 可以发现 <https://weibo.com/ajax/friendships/friends?relate=fans&page=1&uid=2803301701> 页面返回了粉丝信息, 每一组 20 个粉丝, 如图 14 所示。

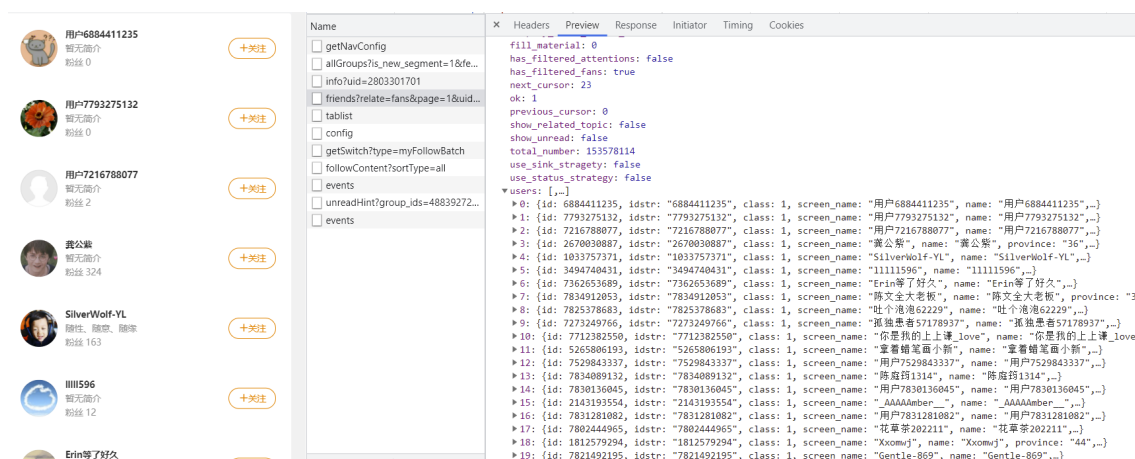


图 14: 粉丝信息

为了遍历所有粉丝信息, 我继续寻找粉丝页面的 url 变化规律, 向下翻阅粉丝清单, 右侧多了几个 fetch 请求, 其规律为 page 不断加 1。当遍历到最后一个 page 时, ok 参数会变为 0 (其他时候均为 1), 该参数作为我判断是否遍历完成的标志。

接下来是爬虫的主体部分, 主要实现的功能为向该页面发送 request 请求。在实现过程中, 微博禁止未登录用户进行查看, 因此我将我账号的 cookie 添加到 middlewares.py 文件中, 后面将详细叙述。爬虫的主要逻辑是首先用 request 访问用户基本信息 url, 然后访问用户详细信息 url, 构造包含用户所有信息的 Useritem 对象使用 yield 传给 pipeline 进行后续存入数据库的操作。

然后遍历访问粉丝信息 url (该 url 仅包含每个粉丝的基本信息), 对每个粉丝构造包含他们详细信息的 url 对基本信息进行完善, 每完善好一个粉丝信息就使用

yield 传给 pipeline 进行存储。使用循环不断增加 page 遍历所有粉丝，直到 ok 为 0，后续构造博文信息的 url 进行访问。

```
1 def start_requests(self):
2     """ 首先请求第一个 js 文件，包含有关注量，姓名等信息 """
3     info_url = 'https://weibo.com/ajax/profile/info?uid=' + settings.ID
4     yield scrapy.Request(url=info_url, callback=self.parse_user_info,
5                           headers=self.fetch['headers'], meta={})
6
7 def parse_user_info(self, response):
8     content = json.loads(response.text)
9     # 调用者是 start 则用 settings, fans 调用则用传入的 ID
10    if 'ID' in response.meta.keys():
11        ID = str(response.meta['ID'])
12    else:
13        ID = settings.ID
14    # 如果没有接受到 ok 字段，表示页面为空，报错返回
15    if not content["ok"]:
16        print("=====user info error=====")
17        return
18    # 获取 user 字典，传入 utils 中的提取字段函数
19    info = content['data']['user']
20    user_item = UserItem()
21    user_item = utils.gen_user_info(user_item, info)
22    # 下一个获取用户详细信息的 url 信息
23    detail_url = "https://weibo.com/ajax/profile/detail?uid=" + ID
24    # meta 为 parse_user_detail 函数传参，只有查询本人信息需要调用此函数
25    meta = {'user_item': user_item, 'group': 0}
26    yield scrapy.Request(url=detail_url, callback=self.parse_user_detail,
27                          headers=self.fetch['headers'], meta=meta)
28
29 def parse_user_detail(self, response):
30     """ 接受一个 meta 参数 {'user_item': None, 'group': None} """
```

```

31     user_item = response.meta['user_item']
32     group = response.meta['group']
33     content = json.loads(response.text)
34     if not content["ok"]:
35         print("=====user detail error=====")
36         return
37     detail = content['data']
38     user_item = utils.gen_user_detail(user_item, detail)
39     user_item['group'] = group
40     yield user_item
41     # flag 用来确保只搜集一次某用户的粉丝信息
42     if self.flag:
43         # 设置数据库名字
44         settings.NAME = user_item['screen_name']
45         # 设置粉丝数据页面 url, page 是页码
46         fans_url = "https://weibo.com/ajax/friendships/friends?
47         relate=fans&page=1&uid=" + settings.ID
48         self.flag = False
49         print("=====user yield into fans=====")
50         yield scrapy.Request(url=fans_url, callback=self.parse_fans,
51                               headers=self.fetch['headers'])
52     def parse_fans(self, response):
53         content = json.loads(response.text)
54         # 是 0 代表已经无法访问到
55         if 'users' not in content.keys():
56             print("=====fans data error=====")
57         if 'msg' in content.keys():
58             print(content['msg'])
59             # 用 return 结束 yield 继续执行
60             return
61         user_list = content["users"]
62         if not user_list:
63             print("===== 粉丝遍历完成 =====")

```

```
64     print(" 其内容为: ")
65     print(response.text)
66     print("=" * 28)
67     # 收集完粉丝信息后继续搜集博文信息
68     statuses_url = 'https://weibo.com/ajax/statuses/mymblog?uid=' +
69     settings.ID + '&page=1'
70     print(statuses_url)
71     yield scrapy.Request(url=statuses_url, callback=self.parse_statuses,
72     headers=self.fetch['headers'])
73     return
74 for user in user_list:
75     user_info = UserItem()
76     user_info = utils.gen_user_info(user_info, user)
77     user_info = utils.gen_user_detail(user_info, user)
78     # 详细查询每个用户信息会导致 414, IP 地址被服务器限制访问, 使用代理 IP
79     info_url = "https://weibo.com/ajax/profile/detail?uid=" +
80     str(user_info['uid'])
81     yield scrapy.Request(url=info_url, callback=self.parse_user_detail,
82     headers=self.fetch['headers'], meta={'user_item':
83     user_info, 'group': 1})
84     # 翻页遍历所有的粉丝信息
85     self.fan_offset += 1
86     fans_url = self.fan_url + str(self.fan_offset) + "&uid=" + settings.ID
87     print(" 页码: ", self.fan_offset)
88     yield scrapy.Request(url=fans_url, callback=self.parse_fans,
89     headers=self.fetch['headers'])
```

5.1.2 博文信息

有了上面的经验, 博文信息也可以轻松找到。其规律和粉丝一样, 都是每页 20 个, 每次 page 加 1, 可由 ok 字段判断是否结束。如图 15 所示。

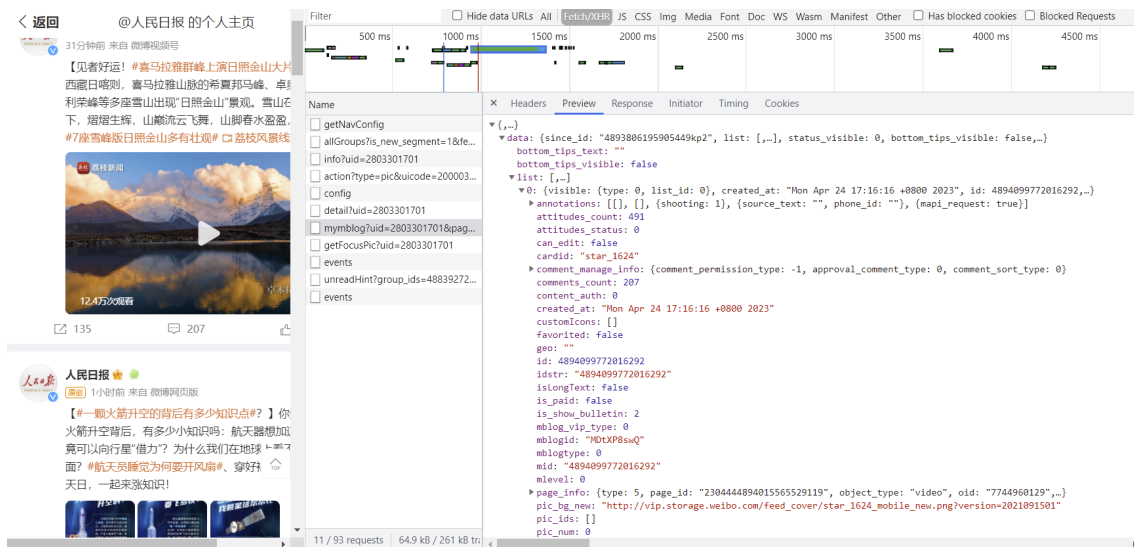


图 15: 博文信息

具体实现如下:

```

1 def parse_statuses(self, response):
2     content = json.loads(response.text)
3     statuses_list = content['data']['list']
4     if not statuses_list:
5         print("===== 博文遍历完成 =====")
6         return
7     # 遍历博文列表
8     for statuses in statuses_list:
9         statuses_info = StatusesItem()
10        statuses_info = utils.gen_statuses_info(statuses_info, statuses)
11        yield statuses_info
12        self.sid = statuses_info['sid']
13        # is_show_bulletin: 1 为按时间排序, 2 为按热度排序
14        comment_url = "https://weibo.com/ajax/statuses/buildComments?
15        id=" + self.sid + "&is_show_bulletin=2"
16        yield scrapy.Request(url=comment_url, callback=self.parse_comment,
17        headers=self.fetch['headers'])
18    # 下一页

```

```
19     self.statuses_offset += 1
20     statuses_url = self.statuses_url + str(self.statuses_offset)
21     print(" 页码: ", self.statuses_offset)
22     yield scrapy.Request(url=statuses_url, callback=self.parse_statuses,
23                          headers=self.fetch['headers'])
```

博文包含的信息在 StatusesItem 类中，其内容如下：

```
1 class StatusesItem(scrapy.Item):
2     """ 微博字段 """
3     sid = scrapy.Field() # 标识号
4     created_at = scrapy.Field() # 创建时间
5     text = scrapy.Field() # 文本
6     source = scrapy.Field() # 发送设备
7     reads_count = scrapy.Field() # 阅读数
8     reposts_count = scrapy.Field() # 转发数
9     comments_count = scrapy.Field() # 评论数
10    attitudes_count = scrapy.Field() # 点赞数
11    tag = scrapy.Field() # 话题
12    region_name = scrapy.Field() # 发布于
```

5.1.3 评论信息

获取评论的 url 为 https://weibo.com/ajax/statuses/buildComments?is_reload=1&id=4893374819862663&is_show_bulletin=2&is_mix=0, is_show_bulletin 字段的含义为：1 为按时间排序, 2 为按热度排序。

评论对应的 url 内容如图 16 所示。

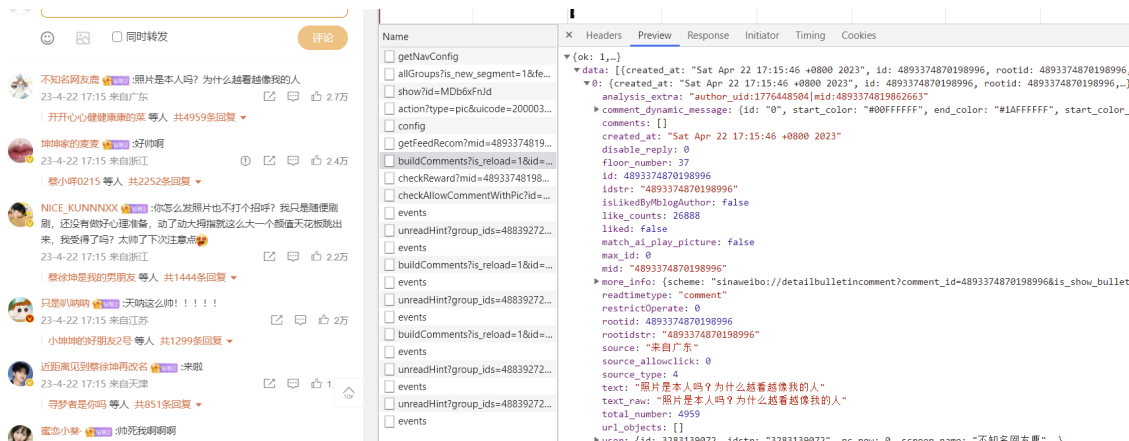


图 16: 评论信息

评论信息和上面两种的 page 加一的规律并不相同,其变化规律体现在 max_id 字段上,当前申请的 url 中的 max_id 字段指示了下一个请求 url 中的 max_id 的值。第一个 url 不携带 max_id 字段。具体实现如下:

```

1 def parse_comment(self, response):
2     """ 想要加载全部评论, 规律为 max_id 字段不断迁移 """
3     content = json.loads(response.text)
4     max_id = str(content['max_id'])
5     comment_list = content['data']
6     if not comment_list:
7         print("===== 评论遍历完成 =====")
8         followed_page_url = "https://weibo.com/ajax/friendships/friends?page="
9         + str(self.followed_offset) + "&uid=" + settings.ID
10        yield scrapy.Request(url=followed_page_url,
11                              callback=self.parse_followed, headers=self.fetch['headers'])
12        return
13    for comment in comment_list:
14        comment_info = CommentItem()
15        comment_info = utils.gen_comment_info(comment_info, comment, self.sid)
16        yield comment_info
17    next_page_url = "https://weibo.com/ajax/statuses/buildComments?flow=0
18    &is_reload=1&id=" + self.sid + "&is_show_bulletin=2&is_mix=0&max_id="

```

```
19     + max_id
20     print(" 最大 id: ", max_id)
21     yield scrapy.Request(url=next_page_url, callback=self.parse_comment,
22                           headers=self.fetch['headers'])
```

评论包含的信息在 CommentItem 类中，其内容如下：

```
1 class CommentItem(scrapy.Item):
2     """ 评论字段 """
3     sid = scrapy.Field() # 所属博文
4     cid = scrapy.Field() # 评论标识
5     comment_time = scrapy.Field() # 评论时间
6     text = scrapy.Field() # 评论文本
7     liked = scrapy.Field() # 作者是否点赞
8     comment_people_id = scrapy.Field() # 评论人 id
9     comment_people_name = scrapy.Field() # 评论人昵称
10    comment_likes = scrapy.Field() # 评论点赞数
11    total_number = scrapy.Field() # 评论回复总数
12    comment_comment = scrapy.Field() # 评论回复内容
```

5.1.4 数据存储

我使用 MySQL 数据库对爬取的数据进行存储，pipelines.py 中实现。实现思路为先用 isinstance 函数判断传过来的类属于三类中的哪一类。对于 UserItem 类，由于粉丝表和关注表基于该类进行存储，因为我在 UserItem 类中设置了 group 字段，0 -> 自己，1 -> 粉丝，2 -> 关注的人，以此来区分存入哪张表。

然后，判断要写入的表是否存在，若不存在则使用 CREATE TABLE 创建数据库，使用 PRIMARY KEY 将 uid, sid, cid 等字段设置为主键。随后，根据传入的对象包含的内容使用 insert ignore into 执行插入语句，可以避免插入重复元素。

下面以存储用户类型的数据为例进行展示：

```

1 class MyspiderPipeline:
2     def __init__(self):
3         self.connect = pymysql.connect(
4             host=settings.MYSQL_HOST,
5             db=settings.MYSQL_DBNAME,
6             user=settings.MYSQL_USER,
7             passwd=settings.MYSQL_PASSWD,
8             use_unicode=True)
9         self.cursor = self.connect.cursor()
10
11     def create_user_table(self, group, item):
12         """ 用于新建存储用户的数据库表单, group 为 ['_ 粉丝', '', '_ 关注'] """
13         # 选择 sina 数据库
14         self.cursor.execute("use sina")
15         # 获取所有表单
16         self.cursor.execute("show tables")
17         tables = self.cursor.fetchall()
18         table_list = [t[0] for t in tables]
19         name = settings.NAME.lower()
20         if name + group not in table_list:
21             print(" 名字 (粉丝表单): ", name)
22             # 需要设置主键
23             sql = 'CREATE TABLE ' + name + group +
24                 '(uid VARCHAR(255) PRIMARY KEY, 昵称 VARCHAR(255), '
25                 '真实姓名 VARCHAR(255), 所在地 VARCHAR(255), 性别 VARCHAR(255), '
26                 '生日 VARCHAR(255), QQ VARCHAR(255), 简介 VARCHAR(255), '
27                 '粉丝数 VARCHAR(255), 关注数 VARCHAR(255), '
28                 '微博数 VARCHAR(255), 微博认证用户 VARCHAR(255), 注册时间 '
29                 ' VARCHAR(255), 微博会员等级 VARCHAR(255), 职业信息 VARCHAR(255), '
30                 ' 教育信息 VARCHAR(255), 标签 VARCHAR(255), 信誉 VARCHAR(255), '
31                 'ip 属地 VARCHAR(255));'
32             self.cursor.execute(sql)

```

```

33     insert = 'insert ignore into ' + name + group + \
34     '(uid, 昵称, 真实姓名, 所在地, 性别, 生日, QQ, 简介, 粉丝数, 关注数, '
35     '微博数, 微博认证用户, 注册时间, 微博会员等级, 职业信息, 教育信息, '
36     '标签, 信誉, ip 属地) values (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s, '
37     '%s,%s,%s,%s,%s,%s,%s,%s,%s,%s); '
38     self.cursor.execute(insert,
39     [item['uid'], item['screen_name'], item['real_name'], item['location'],
40     item['gender'], item['birthday'], item['qq'], item['description'],
41     item['followers_count'], item['follow_count'], item['statuses_count'],
42     item['verified'], item['regist_time'], item['mbrank'],
43     item['profession'], item['education'], item['lable'], item['credit'],
44     item['ip_location']])
45     self.connect.commit() # 这行太重要啦!!!! 查了我半个小时
46     def process_item(self, item, spider):
47         dic = {0: '', 1: '_ 粉丝', 2: '_ 关注'}
48         # 传来的是 User 对象
49         if isinstance(item, UserItem):
50             group = item['group']
51             self.create_user_table(dic[group], item)

```

5.2 反反爬措施

只使用上述方法进行爬取并不能获取大量数据, 因为当我某段时间爬取数量达到限制是, 微博会将我的 ip 暂时封禁, 得到 414 响应码。

```

2023-04-25 09:36:17 [scrapy.core.engine] DEBUG: Crawled (200) <GET https://weibo.com/ajax/profile/info?uid=1776448504> (
2023-04-25 09:36:17 [scrapy.core.engine] DEBUG: Crawled (414) <GET https://weibo.com/ajax/profile/detail?uid=1776448504>
2023-04-25 09:36:17 [scrapy.spidermiddlewares.httperror] INFO: Ignoring response <414 https://weibo.com/ajax/profile/det

```

图 17: 无反爬措施

常见的微博反反爬方法有增加微博 cookie 池, 使用代理 IP, 设置访问时间间隔等。本项目旨在不降低爬取速率的情况下绕过微博的反爬措施, 因此只采用 cookie 池和代理 IP 的方法。

首先使用代理 IP 代理 IP 是指通过代理服务器获取的 IP 地址，用于隐藏客户端的真实 IP 地址并访问目标网站。代理 IP 的原理是，客户端将请求发送到代理服务器，代理服务器再将请求转发到目标网站，目标网站会认为请求来自代理服务器而不是客户端，从而隐藏了客户端的真实 IP 地址。

本项目使用四叶天高匿动态 IP，如图所示。

图 18: 四叶天代理 ip

四叶天代理 IP 每天可以提供 20 个免费的高匿 IP，为了降低代理 IP 的费用，我优先使用每天的免费 IP，在免费 IP 消耗完后再调用付费功能。代码调用 API 的方法为根据 api 链接获取。若申请失败，则根据返回的 code 进行错误处理。实现代码如下：

```
1 def get_ip(typ='free'):  
2     # 请求地址，简单提取具体需要根据实际情况获取记得添加白名单  
3     # http://www.siyetian.com/member/whitelist.html  
4     if typ == 'free':  
5         tiquApiUrl = ''  
6     elif typ == 'paid':  
7         tiquApiUrl = ''
```

```
8     else:
9         raise ZeroDivisionError('输入套餐类型不合法')
10    apiRes = requests.get(tiquApiUrl, timeout=5)
11    # 代理服务器
12    ipport = apiRes.text
13    # 返回错误代码
14    if "code" in ipport:
15        print(typ + " IP 申请失败")
16        print('申请 IP 错误信息: ', ipport)
17        code = re.findall(':(.*?),' , ipport)[0]
18        return "code:" + code
19    return ipport
20
21    def ip_list(num):
22        proxies_list = []
23        typ = "free"
24        while num > 0:
25            time.sleep(1)
26            ip = get_ip(typ)
27            print(ip)
28            if ip == "code:10019":
29                typ = "paid"
30                print("free ip 已达最大使用次数")
31            elif "code" not in ip:
32                proxies_list.append(ip)
33                print(typ + ' ip 使用成功')
34                num -= 1
35            elif ip == "code:10005":
36                print('该套餐已过期')
37                raise ZeroDivisionError('请前往'
38                    'https://www.siyetian.com/apis.html 手动获取最新 api 链接')
39    return proxies_list
```

接下来是对请求头的变化，我从浏览器类型，登录用户 cookie 两个角度进行更改，设置 browser 池和 cookie 池，每次随机进行选取。鉴于 cookie 每次从浏览器复制下来格式不是字典形式，scrapy 框架无法使用且有较多无用字段。经过测试只有'XSRF-TOKEN'、'SUB'、'SUBP' 和'WBPSESS' 为必须字段。我在 utils.py 中添加了预处理 cookie 函数，方便每次 cookie 的更新。

```
1 def operate_cookies(cookie):
2     dic = {} # 首先将以 ';' 分割字符串转化为字典
3     c_l = cookie.split("; ")
4     for item in c_l:
5         k_n = item.split('=')
6         dic[k_n[0]] = k_n[1]
7     # 其次提取有用字段进行返回
8     new_dic = {'XSRF-TOKEN': dic['XSRF-TOKEN'], 'SUB': dic['SUB'],
9               'SUBP': dic['SUBP'], 'WBPSESS': dic['WBPSESS']}
10    return new_dic
```

我还对这四个字段进行了调查，获取到以下信息：

1. XSRF-TOKEN

XSRF-TOKEN (跨站请求伪造令牌) 是一种用于防止跨站请求伪造攻击的机制。当用户访问一个网站时，服务器会生成一个随机的 XSRF-TOKEN，并将其存储到 cookie 中，然后将其发送给客户端。当用户在该网站上执行某些敏感操作时，客户端会将这个 XSRF-TOKEN 发送回服务器，服务器会验证该令牌是否有效，如果有效则允许该操作。

2. SUB

SUB 是新浪微博中的一个 cookie 字段，其值为用户的登录名（或者说是用户的昵称）。当用户登录成功后，服务器会将该字段存储到 cookie 中，并发送给客户端。客户端在以后的请求中会带上该 cookie 字段，以便服务器能够识别用户身份。

3. SUBP

SUBP 是新浪微博中的另一个 cookie 字段，其值包括用户的登录状态、用户 ID、加密信息等。该字段主要用于登录状态的维护和用户身份的验证。

4. WBPSESS

WBPSESS 是新浪微博中的一个 cookie 字段，其值为用户的会话 ID。当用户登录成功后，服务器会生成一个会话 ID，并将其存储到 cookie 中，然后将其发送给客户端。客户端在以后的请求中会带上该 cookie 字段，服务器可以通过该字段来识别用户的会话状态。

上面的代理 IP 每次也随机进行选取。但即便使用代理 IP 也有小概率被微博识别并进行封锁，因此我在选择代理 IP 的同时也会选择本机 IP，以 70% 选择代理 IP 池，30% 选择本机 IP 进行处理，在 middlewares.py 中实现。

```
1 class RequestMiddleware:
2     # 浏览器池
3     user_agents = [
4         'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; en-us)' \
5         'AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50',
6         'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us)' \
7         'AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50',
8         'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0;',
9         'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)',
10        'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv,2.0.1)' \
11        'Gecko/20100101 Firefox/4.0.1',
12        'Mozilla/5.0 (Windows NT 6.1; rv,2.0.1) Gecko/20100101 Firefox/4.0.1',
13        'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_0)' \
14        'AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11'
15    ]
16    # cookie 池
17    cookies = [收集的多个同学的 cookie(过长不展示)]
18    proxies_list = ip_list(4)
19    print('代理 IP 列表:', proxies_list)
20    @classmethod
```

```
21     def from_crawler(cls, crawler):
22         # 此函数用于创建爬虫
23         s = cls()
24         crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
25         return s
26     def process_request(self, request, spider):
27         request.headers['User-Agent'] = random.choice(self.user_agents)
28         request.cookies = utils.operate_cookies(random.choice(self.cookies))
29         # 是 proxy 一定是要写成 http:// 前缀, 否则会出现 to_bytes must
30         # receive a unicode, str or bytes object, got NoneType 的错误
31         choose = random.randint(0, 10)
32         if choose > 3:
33             request.meta['proxy'] = "http://" +
34             random.choice(self.proxies_list)
35         else:
36             request.meta['proxy'] = ''
37         return None
```

然而, 此时的系统并不稳定, 因为代理 IP 的存活时长为 1-5 分钟, 而爬取过程可能持续更长时间, 因此我需要动态更新代理 IP 池。可以根据程序的 exception 来判断是否选用了失效的代理 IP, 该 IP 失效后重新申请加入代理 IP 池。Scrapy 框架也提供了报错处理这一功能, 实现函数如下 (也在 middlewares.py 的类中)。

```
1 def process_exception(self, request, exception, spider):
2     # Called when a download handler or a process_request()
3     # (from other downloader middleware) raises an exception.
4     # Must either:
5     # - return None: continue processing this exception
6     # - return a Response object: stops process_exception() chain
7     # - return a Request object: stops process_exception() chain
8
9     # 处理失效代理 IP
```

```
10     print('发生错误: ', exception)
11     print('代理 IP ' + request.meta["proxy"] + '已失效')
12     self.proxies_list.remove(request.meta["proxy"])
13     self.proxies_list += ip_list(1)
14     print('现存代理 IP 池: ', self.proxies_list)
```

5.3 多线程加速及系统优化

在爬虫爬取过程中，我使用 `yield scrapy.Request` 函数请求下一个页面，然而实际上这个过程可以写在多个爬虫文件，在运行 scrapy 项目时使用多线程同时运行多个爬虫文件以达到同时爬取的目的。实现思路为把原项目分为四个线程，一个爬取关注者列表，一个爬取粉丝列表，一个爬取博文列表，最后一个爬取评论列表。

除此之外，由于第一次爬取需要大量时间且已经在数据库中存储，因此后续爬取时我不再需要重新遍历所有信息，只需要在遍历时遍历到已存储数据时停止即可。

5.4 前端实现

本项目的前端使用 Flask 框架，基于 html/css 和 JavaScript 实现。分为搜索界面和展示界面两个页面。

5.4.1 搜索页面

首先为用户提供一个搜索界面，用户输入需要查询的用户名来调用爬虫进行爬取。界面样式如图 19。



图 19: 搜索界面

当输入为空时，在页面上方会出现（以下拉形式）输入不能为空提示框，因此此页面的代码难点主要在搜索框的功能实现上。HTML 代码如下。

```
1 <body>
2   <div id="app">
3     <!-- 提示信息 -->
4     <transition>
5       <!--v-if判断变脸 isInfo 是否为真，为真则显示进入动画-->
6       <p v-if="isInfo" class="info" v-cloak>输入用户名不能为空</p>
7     </transition>
8     <!-- 搜索框 -->
9     <section class="searchBox">
10      <div class="clock">
11        微博认知安全监测系统
12      </div>
13      <div class="inputBox">
14        <input type="text" placeholder="Search" id="searchInput"
15          v-model="queryString" v-on:keyup.enter="search">
16        
17      </div>
18    </section>
19  </div>
```

```
20     <script type="text/javascript" src="https://www.ownthink.com
21     /bot/js/robot.js?appid=xiaosi"></script>
22 </body>
```

在代码中, `placeholder="Search"` 表示在输入框没有内容时显示的提示信息。`v-model="queryString"` 表示这个输入框的值会绑定到 Vue 实例的 `queryString` 属性上, 即当输入框的值改变时, `queryString` 的值也会跟着改变。`v-on:keyup.enter="search"` 表示当用户敲击回车键时, 会调用 Vue 实例的 `search` 方法。

JavaScript 的实现逻辑为设置 `isInfo` 标记位, 通过 `queryString` 是否为空来判断是否需要显示提示框和是否执行页面跳转。提示框的动态效果由 `index.css` 文件实现。下面的代码展示了 JS 的实现逻辑。

```
1 <script>
2 window.onload = function(){
3     // 设置input框自动获取焦点
4     let searchInput = document.getElementById("searchInput");
5     searchInput.focus();
6 };
7 </script>
8 <script src="https://cdn.bootcss.com/vue/2.6.11/vue.min.js"></script>
9 <script>
10 const vm = new Vue({
11     // 这个参数用来指定Vue实例所绑定的DOM元素, 即#app元素。
12     // 这意味着Vue实例将控制这个元素及其所有子元素。
13     el: "#app",
14     data:{
15         queryString: "", // 查询字符串
16         isInfo: false, // 是否显示提示信息
17     },
18     methods: {
19         // 按下enter键或点击搜索图标进行搜索
20         search: function(){
21             if(this.queryString === ""){
22                 this.isInfo = true;
```

```

23         let that = this;
24         // 1200毫秒之后关闭提示信息
25         setTimeout(function(){
26             that.isInfo = false;
27         }, 1200);
28     }else{
29         // 在当前窗口打开
30         window.location.href =
31             "http://127.0.0.1:8080/search.html?searchname=" +
32             this.queryString;
33     }
34 }
35 },
36 });
37 </script>

```

实现提示框出现消失的动画通过 Vue 来实现。其原理如图 20 所示。

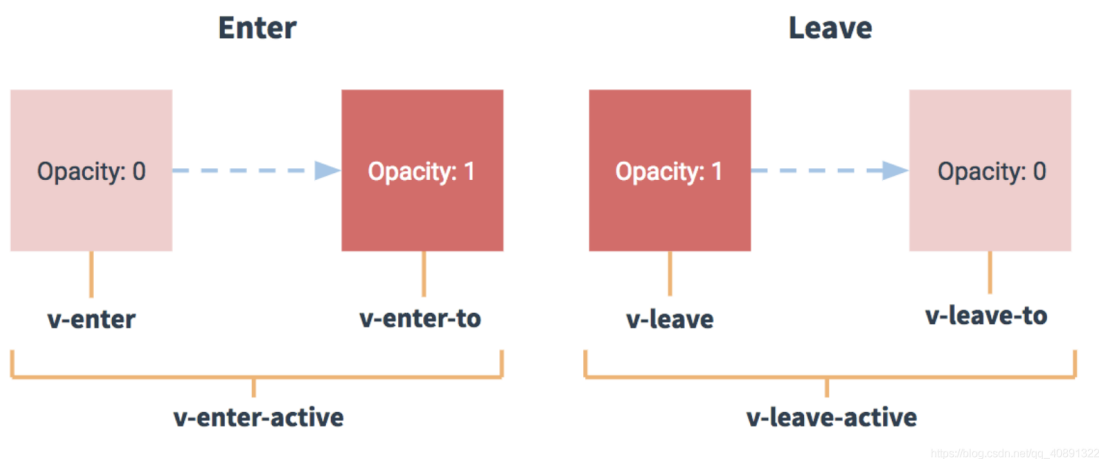


图 20: vue 进入离开效果原理

实现的 CSS 代码如下。

```
1  /* 和 transition 标签配合使用 */
2  /* 默认类名为 v */
3  .v-enter ,
4  .v-leave-to{
5      /* 透明度,0为透明 */
6      opacity: 0;
7      top: -20px;
8  }
9  /* 进入和离开状态 */
10 .v-enter-active ,
11 .v-leave-active{
12     /* 所有动画0.4s内进入 */
13     transition: all 0.4s ease-in-out;
14 }
```

5.4.2 展示界面

展示界面分为两部分，左侧为导航栏部分（类名 `content-left`），右侧为表格信息界面（类名 `content-right`），下面分开叙述讲解。

5.4.2.1 导航栏

导航栏的实现采用多级 `<div>` 标签形式，没有采纳 `` 格式，是因为 `<div>` 更为自由，对我这种初学者更为友好。在实现过程中，我使用了 `fontawesome` 图标库，下载本地文件引入 `all.min.css` 进行使用（`<i>` 标签）。使用 `<a>` 标签链接到右侧表格。由于我用的是嵌入式页面，`<a>` 标签无法传递参数，因此我在导航栏多放了一个不显示的 `` 标签用来传递参数。主要 `html` 代码如下。

```
1 <body>
2 <div class = "content_">
3     <!-- 左侧导航栏 -->
4     <div class = "content-left">
5         <div class = "left-title">
```

```
6         
7         <span>欢迎使用</span>
8     </div>
9     <!--水平线-->
10    <div class = "seg"></div>
11    <!--菜单栏导航-->
12    <div class = "nav_">
13    <!--每一个菜单栏项-->
14    <div class = "nav-menu">
15        <!--主标题-->
16        <div class = "nav-title">
17            <i class="far fa-address-book" aria-hidden="true"></i>
18            用户关系信息
19        </div>
20        <!--子标题-->
21        <div class = "nav-content">
22            <a href = "chart.html" onclick="setUrl('关注') "
23            rel="external">关注列表</a>
24            <a href = "chart.html" onclick="setUrl('粉丝') "
25            rel="external">粉丝列表</a>
26        </div>
27    </div>
28 </div>
29
30 <!--水平线-->
31 <div class = "seg"></div>
32 <!--用于给右侧界面提供打开表格信息-->
33 <div class = "chart-choose">
34     <span id="chart-info">123123</span>
35 </div>
36 </div>
37 <!--右侧内容区-->
38 <div class = "content-right"></div>
```

```
39 </div>
40 </body>
```

为了实现导航栏的折叠效果和右侧表格对应显示功能，JavaScript 代码的主要逻辑如下。首先隐藏所有菜单项的子标题。然后给所有菜单项中的主标题绑定点击事件，点击主标题时，判断其子标题的 `display` 属性是否为 `none`，是则将其隐藏，否则将其显示。同时，如果子标题处于隐藏状态，将其他的菜单项的子标题也隐藏。

给所有子标题中的链接绑定点击事件，点击时根据链接中的 `href` 属性加载相应的页面。同时，防止页面跳转，使用 `return false`。最后定义了一个名为 `setUrl` 的函数，用于设置图表的名称。该函数会将传入的名称设置给 `id` 为“`chart-info`”的元素（内容可以是‘粉丝’、‘关注’、‘微博’、‘评论’或”）。JS 代码如下：

```
1 <script>
2 $(function () {
3     // 隐藏所有子标题
4     $(".nav-menu").each(function () {
5         $(this).children(".nav-content").hide();
6     });
7     // 给菜单项中的所有主标题绑定事件
8     $(".nav-title").each(function () {
9         // 获取点击当前标签下所有子标签
10        let navConEle = $(this).parents(".nav-menu")
11            .children(".nav-content");
12        // 绑定点击事件，判断navConEle的display这个属性是否为none，
13        // 为none的话时隐藏状态
14        $(this).click(function () {
15            if (navConEle.css("display") !== "none") {
16                // 隐藏，传参数自带动画，单位为毫秒
17                navConEle.hide(300);
18            } else {
19                // 显示，传参数自带动画，单位为毫秒
20                $(".nav-menu").each(function () {
21                    $(this).children(".nav-content").hide(300);
```

```

22         });
23         navConEle.show(300);
24     }
25 });
26 });
27 $(".nav-content>a").each(function () {
28     $(this).click(function () {
29         let pageUrl = $(this).attr("href");
30         $(".content-right").load(pageUrl);
31         return false;
32     });
33 });
34 });
35 function setUrl(chart_name) {
36     let name = document.getElementById("chart-info");
37     name.innerHTML = chart_name;
38 }
39 </script>

```

最终的效果如图所示。

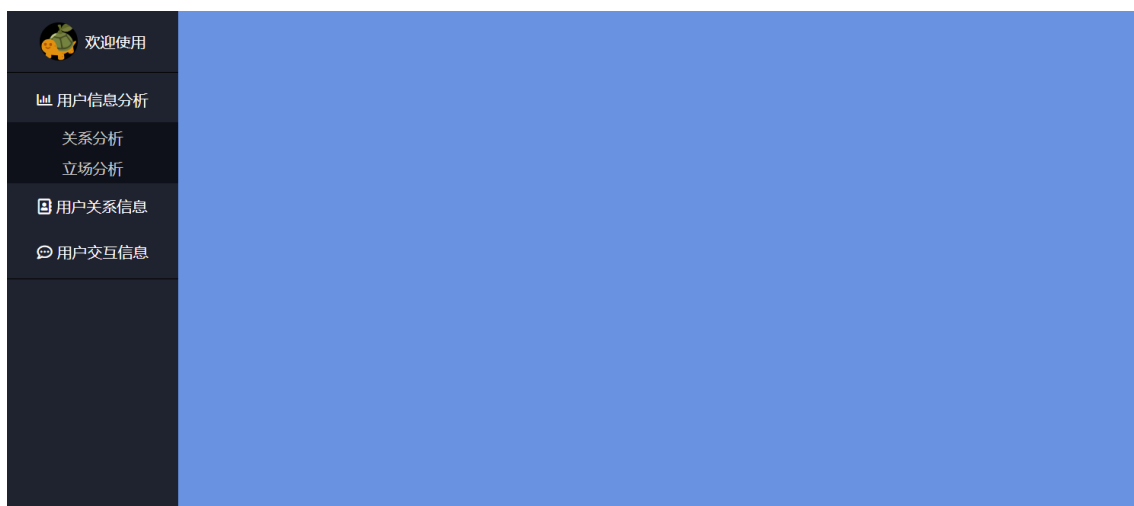


图 21: 导航栏动态效果

5.4.2.2 图表区

在本项目中有五个表，如果为每个表都写一个 html 文件的话太过冗余，鉴于每张表只是表头和内容不同，我才去 JavaScript 根据选择哪张表动态生成 html 代码对表格内容进行呈现。其中使用了 jQuery 库获取并添加页面元素。JS 代码的逻辑如下。

首先根据导航栏的'chart-info' 类告知我要加载哪个表获取需要加载的文件名称，然后使用.ajax 方法请求后端数据，分别写入表头和表格，具体代码如下：

```
1 <script>
2 $(function() {
3     // 获取指定 json 的路径
4     let searchname = location.search.slice(1);
5     let temp = searchname.split('=');
6     let form = document.getElementById('chart-info').innerHTML;
7     let ad = "";
8     if (form)
9         ad = "../static/json/" + temp[1] + "_" + form + ".json";
10    else
11        ad = "../static/json/" + temp[1] + ".json";
12    $.ajax({
13        url: ad,
14        dataType: "json",
15        success: function(data) {
16            let tableHead = $("#table-head");
17            let tableBody = $("#table-body");
18            // 获取 data 字典的键
19            const keys = Object.keys(data[0]);
20            // 写入表头
21            let header = $("<tr></tr>");
22            for (let key in keys) {
23                header.append($("<th></th>").text(keys[key]));
24            }
25            tableHead.append(header)
```

```

26         // 写入表的内容
27         $.each(data, function(index, content) {
28             let row = $("<tr></tr>");
29             for (let key in content) {
30                 let td = $("<td></td>")
31                 td.append($("<div class='td_width'></div>")
32                     .text(content[key]));
33                 row.append(td);
34             }
35             tableBody.append(row);
36         });
37     },
38     error: function(jqXHR, textStatus, errorThrown) {
39         console.log("读取数据失败：
40             "+textStatus+" "+errorThrown);
41     }
42 });
43 });
44 </script>

```

在设置图表 css 样式时，评论或博文内容过长会导致格式非常丑陋，我查阅资料找到一种给定表格宽度限制最大行数的 css 样式（webkit），可以很好的呈现出一大段文本，其内容如下。

```

1 .td_width {
2     overflow: hidden;
3     text-overflow: ellipsis;
4     display: -webkit-box;
5     -webkit-line-clamp: 10;
6     -webkit-box-orient: vertical;
7 }

```

最终的表格效果如图所示。

欢迎使用	粉丝数	关注数	微博数	微博认证用户	注册时间	微博会员等级	职业信息	教育信息
	0	6	0	False	2014-03-06 19:09:19	0		
用户信息分析	59	636	410	False	2014-03-06 22:14:00	0		
用户关系信息	42	81	39	False	2012-03-20 16:00:53	0		
关注列表	3	34	6	False	2014-03-07 11:42:07	0		
粉丝列表	21256	322	319	False	2010-09-11 23:10:22	4		
用户交互信息	370	725	2195	False	2010-03-20 18:52:48	0		东北林业大学
	1114	153	205	False	2011-07-05 12:21:14	0		金陵科技学院
	153	199	263	False	2011-08-10 17:57:25	6		
	33	86	22	False	2012-11-22 16:37:54	0		
	13	9	1	False	2014-03-08 01:55:21	0		
	0	5	0	False	2014-03-02 06:05:00	0		
	13	252	12	False	2012-03-16 22:57:32	1		
					2010-05-29			

图 22: 表格样式

6 实验总结

这次实验是我做的呕心沥血的一次，完全从零学起，无论是 Scrapy, Flask, html/css 还是 JavaScript，之前从未自己写过，接触到的也比较少。从着手学习到项目编写完成，历时两周时间，每天至少花费四五个小时在这个作业上，才最终把需要用到的东西学个大概，完成最终的项目编写。

在一开始选择技术路线时，本来想基于微博的 api 实现，但目前来看那个 api 对个人用户的限制过于强大，不足以满足实验要求，因此在查阅资料过后我了解到了 Scrapy 爬虫，在菜鸟教程上一步步地跟着做，才渐渐对它有了一个粗略的了解，能够完成 settings.py 的设置，爬虫文件的编写，和 items.py 的数据结构定义。在这个过程中，我遇到了非常多的困难，一开始根据网上的说法，weibo.com 的反爬措施过强，无法正常爬取，转而爬取微博的手机端 m.weibo.com，这个网站不需要我进行登录就可以爬取部分信息，但最终得到的用户信息属性较少，无法满足实验的要求，也无法满足我自己的要求，因此转战微博的主网站，由于对登录机制的不了解，我花费了大量时间研究如何不通过扫码登录微博，后来才终于在 fetch 中找到了我需要的 cookie，成功完成登录。

随后又遇到了一个困难，我需要提取信息的网页都需要用户的 ID，若规定搜索是给出用户的 ID 进行搜索的话，太不人性化了（虽然实验要求是给 ID），所以我需要用到微博 api 为数不多的几个可用功能根据用户名查询 ID，但这个过程的

code 需要手动获取（手动打开网页复制 code 并输入），非常麻烦，也非常不专业，因此我又摸索了很久自动登录的方法，用 request 等函数手动构造请求，在返回的 headers 中找到 code 的所在位置，这个过程也需要浏览器的 cookie。

终于，Scrapy 写好了，可以正常爬取了，但爬取小用户时没有问题，爬取人民日报等这种大人物不一会儿微博就会给我返回 414 错误代码，拒绝了我的访问请求，这就是其反爬措施造成的。因此我围绕这个进行了多种尝试，更换请求 headers 中的浏览器，更换请求 headers 中的 cookie（这个比较有效），使用代理 IP（解决根本问题）。这也是我第一次使用代理 IP，本来想找免费公开的高匿代理 IP，可尝试了多个都不起作用，才意识到这个是要收费使用的，随后才找了一家代理 IP 公司（四叶天）购买代理 IP 服务。这才让我的 Scrapy 非常高效完善。

相比于后端，我觉得前端更折磨人，我本想像大家一样使用现有框架，但我比较倒霉没有找到一个轻量级的，虽然很好看但动不动就几万行代码实在无法看懂再修改。而且想短时间学会前端并未易事，所以我采取面向需求的办法，我想写一个搜索界面，我就去 b 站上找怎么写搜索界面，想实现导航栏和表格效果也去 b 站上搜然后跟着做。做完这些事儿时，我已经对 HTML/CSS 有了一个大体的了解了，但把这些模块结合成一个项目也困难重重，且我的困难并不普遍，网上很难找到答案，只有通过别人的项目是如何书写的，再对照我的看看能否修改，在不断摸索的过程终于完成了我自己的所有需求。

这个实验让我学到了非常多东西，非常有收获，对我下一步竞赛的内容做出了有力的保障。