Ajax 数据爬取

一、什么是Ajax

Ajax 即 JavaScript 和 XML，它不是一种语言，而是**利用 JavaScript 在保证页面不被刷新、页面链接不改变的情况下与服务器交换数据并更新部分网页内容的技术。**对于传统的网页，如果想更新网页内容，就必须刷新整个页面，而使用 Ajax 之后，可以在页面不被全部更新的情况下更新。整个过程实际上就是页面在后台与服务器进行了数据交互，获取数据之后，再利用 JavaScript 改变页面，从而完成网页的更新。

1、基本原理：

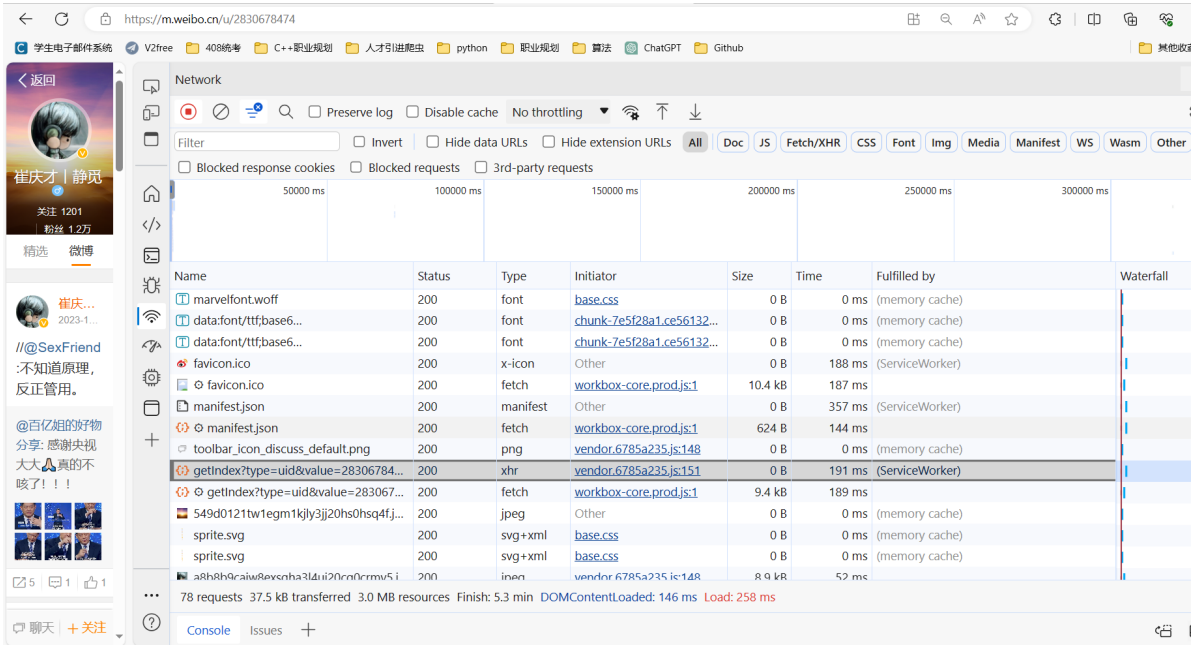
从发送 Ajax 请求到网页更新的这个过程可以简单分为3步——发送请求、解析内容、渲染网页。实际上，这三个步骤都是由 JavaScript 完成的。真实的网页数据就是一次次向服务器发送 Ajax 请求得到的，**要想抓取这些数据，需要知道 Ajax 请求到底是怎么发送的、发往哪里、发了哪些参数。**

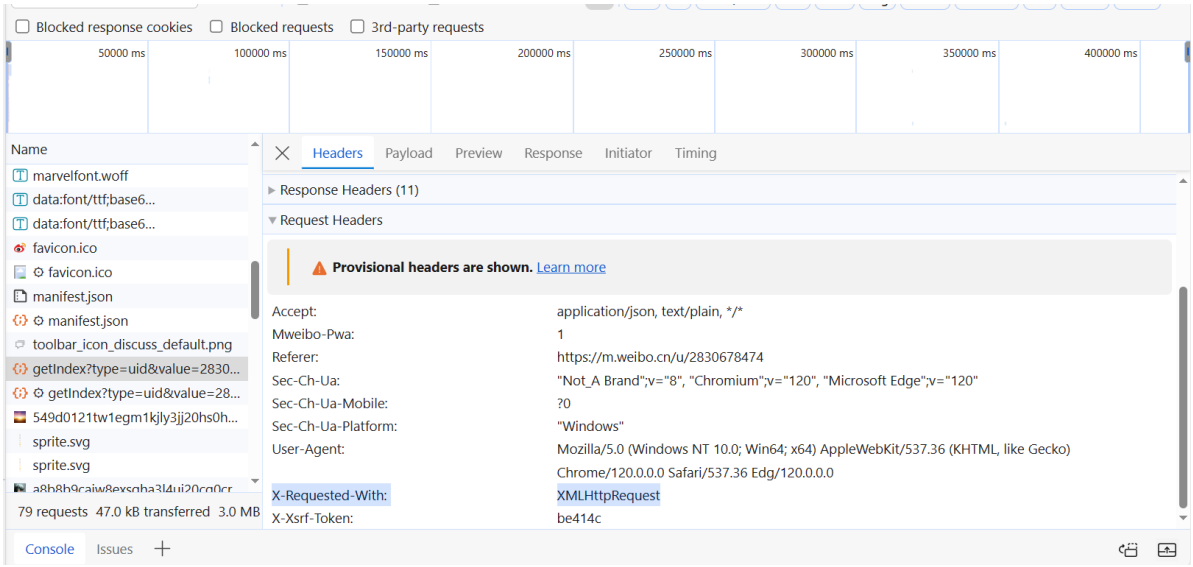
2、Ajax 分析：

以 微博 (weibo.cn) 为例进行分析：

（1）Ajax 有其特殊的请求类型：xhr

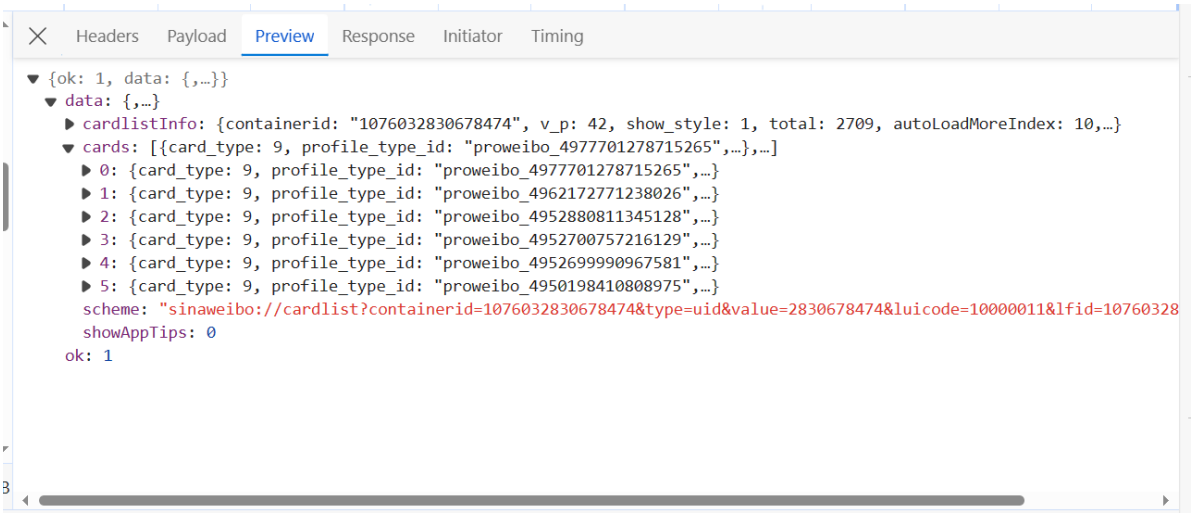
打开页面的开发者工具，点击 Network，查找 Type 为 xhr 的请求。单击此请求，可以观察其详细信息。其中， **Request Headers 中有一个信息为 X-Requested-With，其内容为 XMLHttpRequest。这就标记了此请求为 Ajax 请求。**





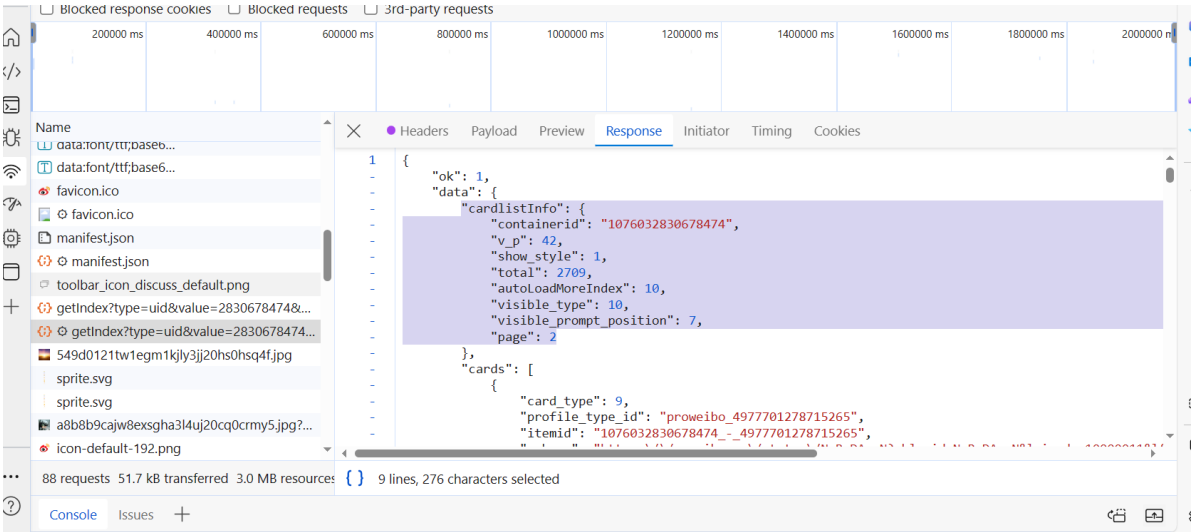
(2) Preview:

单击一下 Preview，可以查看响应内容（JSON 格式），这里浏览器为我们自动完成了解析。



(3) Response:

可以切换至 Response，从中观察真实的返回数据。



二、如何使用 Ajax 完成分析和爬取

1、目标任务：

- [ ] 分析页面数据的加载逻辑；
- [ ] 用 requests 实现 Ajax 数据的爬取；
- [ ] 将每部电影的数据保存至文件中。

2、初步尝试：

```
import requests

url = 'https://spa1.scrape.center/'
html = requests.get(url).text
print(html)
```

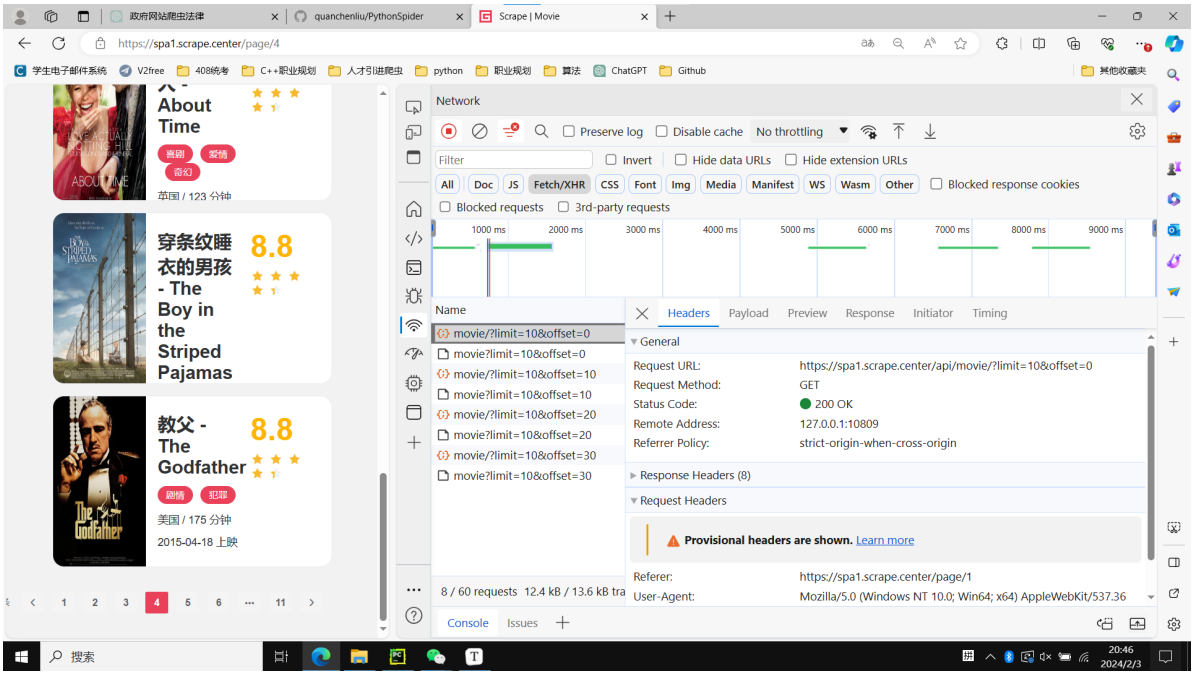
在 HTML 中，**我们只能看到一些 JavaScript 和 CSS 文件，并没有观察到任何电影数据信息。这说明我们整个页面都是由 JavaScript 渲染得到的，**浏览器执行了 HTML 中引用的 JavaScript 文件，JavaScript 通过调用一些数据加载和页面渲染方法，才最终呈现了该网页的视觉效果（'https://spa1.scrape.center/'）。网页中的数据是通过 Ajax 加载的，JavaScript 在后台调用 Ajax 数据接口，得到数据之后再对数据进行解析并渲染呈现出来，得到最终的页面。因此，**若想要爬取由 JavaScript 渲染的页面，直接爬取 Ajax 数据接口的数据即可。**

3、爬取列表页：

```
# 定义爬取方法，该方法专门处理 JSON 接口
def scrape_api(url):
    logging.info('scraping %s...', url)
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.json()          # 解析响应内容并将其转化为 json
字符串
        logging.error('get invalid status code %s while scraping %s',
response.status_code, url)
    except requests.RequestException:
        logging.error('error occurred while scraping %s', url,
exc_info=True)

# 定义爬取列表页的方法
def scrape_index(page):
    url = INDEX_URL.format(limit=LIMIT, offset=LIMIT * (page - 1))
    return scrape_api(url)
```

通过开发者工具，查看 xhr 选项卡对应的详情，观察请求URL、参数和响应内容是怎样的。



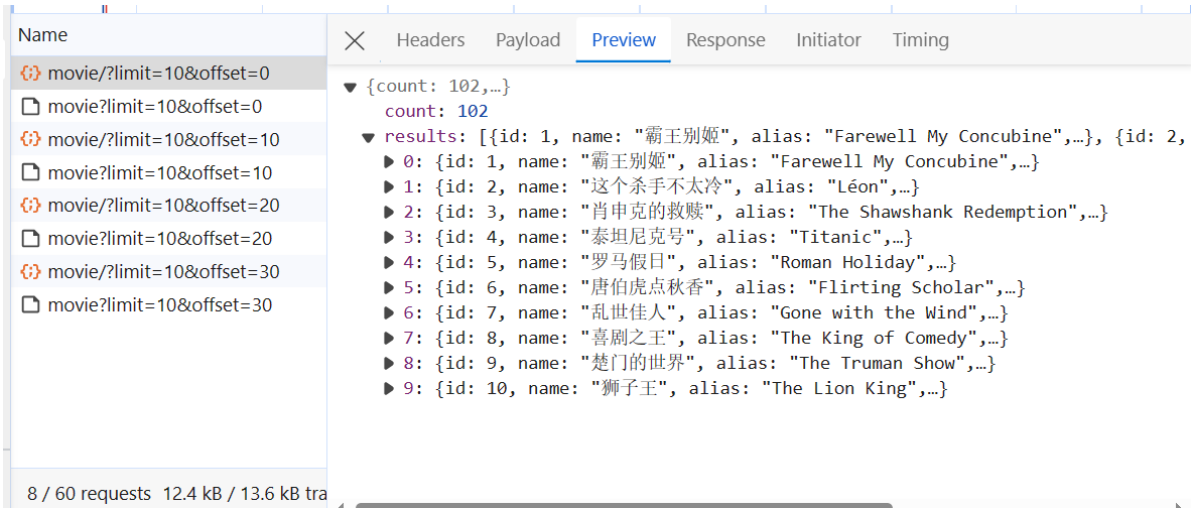
可以发现每个 Ajax 请求的 Request URL 具有统一的格式：

```
https://spa1.scrape.center/api/movie/?limit={limit}&offset={offset}
```

这里有两个参数，limit、offset：

- limit：用于限制每页的数据量；
- offset：代表页面的数据偏移量；

接着，观察响应内容（切换到 Preview 选项卡），如图：



结果就是一些 json 数据，其中有一个results 字段，是一个有若干字典组成的列表。观察列表中的内容，发现正好是我们需要的电影名称、电影类型、电影封面、电影时长、上映时间、评分等信息。因此，我们**只需要完成对 Ajax 接口的构造，就可以轻松获取所有列表页的数据。**

4、爬取详情页：

我们虽然从 Ajax 接口中获取了很多我们所需要的数据，但是仍然缺失一些我们需要的数据，如：剧情简介。因此我们需要对详情页进行进一步爬取。

```
# 定义爬取方法，该方法专门处理 JSON 接口
```

```
def scrape_api(url):
    logging.info('scraping %s...', url)
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.json()          # 解析响应内容并将其转化为 json 字符串
        logging.error('get invalid status code %s while scraping %s', response.status_code, url)
    except requests.RequestException:
        logging.error('error occurred while scraping %s', url, exc_info=True)

# 定义爬取详情页的方法
def scrape_detail(id):
    url = DETAIL_URL.format(id=id)
    return scrape_api(url)
```

通过观察开发者工具中的信息，可以发现，每个详情页对应的 Ajax 请求都有唯一的 URL，且具有统一格式：

```
https://spa1.scrape.center/api/movie/{id}
```

id 与某一部电影之间一一映射。同样，响应结果是结构化的 JSON 数据，其字段非常完整，直接爬取即可。

5、保存数据：

```
def save_data(data):
    name = data.get('name')
    data_path = f'{RESULTS_DIR}/{name}.json'
    json.dump(data, open(data_path, 'w', encoding='utf-8'),
              ensure_ascii=False, indent=2)
```

6、定义总的方法：

```
def main():
    for page in range(1, TOTAL_PAGE + 1):                    # 1~10
        index_data = scrape_index(page)                      # 爬取列表页的 Ajax 接口中的数据
        for item in index_data.get('results'):               # 获取 results 字段中的数据
            movie_id = item.get('id')                        # 获取电影的 id
            detail_data = scrape_detail(movie_id)            # 获取详情页的数据
            logging.info('detail data %s', detail_data)
            save_data(detail_data)                           # 保存数据
```

三、引入并发异步操作

在不引入并发的情况下，完成爬取任务所需的时间为：

```
Use time: 397.3165488243103
```

我们引入并发之后，所需时间为：

```
Use time: 45.07044768333435

进程已结束,退出代码0
```

是**原来耗时的 11.34%**

```
"""
 -*- coding : utf-8 -*-
 @Author    : quanchenliu
 @Time      : 2024/2/2
 @Function  : Ajax + json 存储
"""
import time
import aiohttp
import asyncio
import logging
import json
from os import makedirs
from os.path import exists

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %
(levelname)s: %(message)s')
INDEX_URL = 'https://spa1.scrape.center/api/movie/?limit={limit}&offset=
{offset}'
DETAIL_URL = 'https://spa1.scrape.center/api/movie/{id}'
LIMIT = 10
TOTAL_PAGE = 10
RESULTS_DIR = 'C:/Users/DELL/Desktop/python爬虫基础/3.AjaxTest'
exists(RESULTS_DIR) or makedirs(RESULTS_DIR)
CONCURRENCY = 5                                 # 声明爬虫最大并发数: 5
semaphore = asyncio.Semaphore(CONCURRENCY)  # 创建一个信号量对象 semaphore,
用于控制最大并发量

# 定义爬取方法，该方法专门处理 JSON 接口
async def scrape_api(url):
    logging.info('scraping %s...', url)
    try:
        async with semaphore:
            async with aiohttp.ClientSession() as session:
                async with session.get(url) as response:
                    if response.status == 200:
                        return await response.json()
        logging.error('get invalid status code %s while scraping %s',
response.status, url)
```

```
    except aiohttp.ClientError as e:
        logging.error('error occurred while scraping %s', url,
exc_info=True)

# 定义爬取列表页的方法
async def scrape_index(page):
    url = INDEX_URL.format(limit=LIMIT, offset=LIMIT * (page - 1))
    return await scrape_api(url)

# 定义爬取详情页的方法
async def scrape_detail(movie_id):
    url = DETAIL_URL.format(id=movie_id)
    return await scrape_api(url)

# 定义存储数据的方法
async def save_data(data):
    name = data.get('name')
    data_path = f'{RESULTS_DIR}/{name}.json'
    json_data = {
        "电影名称": name,
        "电影种类": data.get('categories'),
        "制作国家": data.get('regions'),
        "电影评分": data.get('score'),
        "电影时长": data.get('minute'),
        "电影简介": data.get('drama'),
        "上映时间": data.get('published_at')
    }
    json.dump(json_data, open(data_path, 'w', encoding='utf-8'),
              ensure_ascii=False, indent=2)

async def Scrape(page):
    index_data = await scrape_index(page)               # 爬取列表页的 Ajax
接口中的数据
    for item in index_data.get('results'):              # 获取 results 字段
中的数据
        movie_id = item.get('id')                       # 获取电影的 id
        detail_data = await scrape_detail(movie_id)     # 获取详情页的数据
        await save_data(detail_data)                    # 保存数据

async def main():
    start = time.time()

    tasks = [asyncio.ensure_future(Scrape(page)) for page in
range(TOTAL_PAGE+1)]
    await asyncio.gather(*tasks)

    end = time.time()
    print('Use time:', end - start)

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```