在上一课时我们了解了网站登录验证和模拟登录的基本原理。网站登录验证主要有两种实现，一种是基于 Session + Cookies 的登录验证，另一种是基于 JWT 的登录验证，那么本课时我们就通过两个实例来分别讲解这两种登录验证的分析和模拟登录流程。

准备工作

在本课时开始之前，请你确保已经做好了如下准备工作：

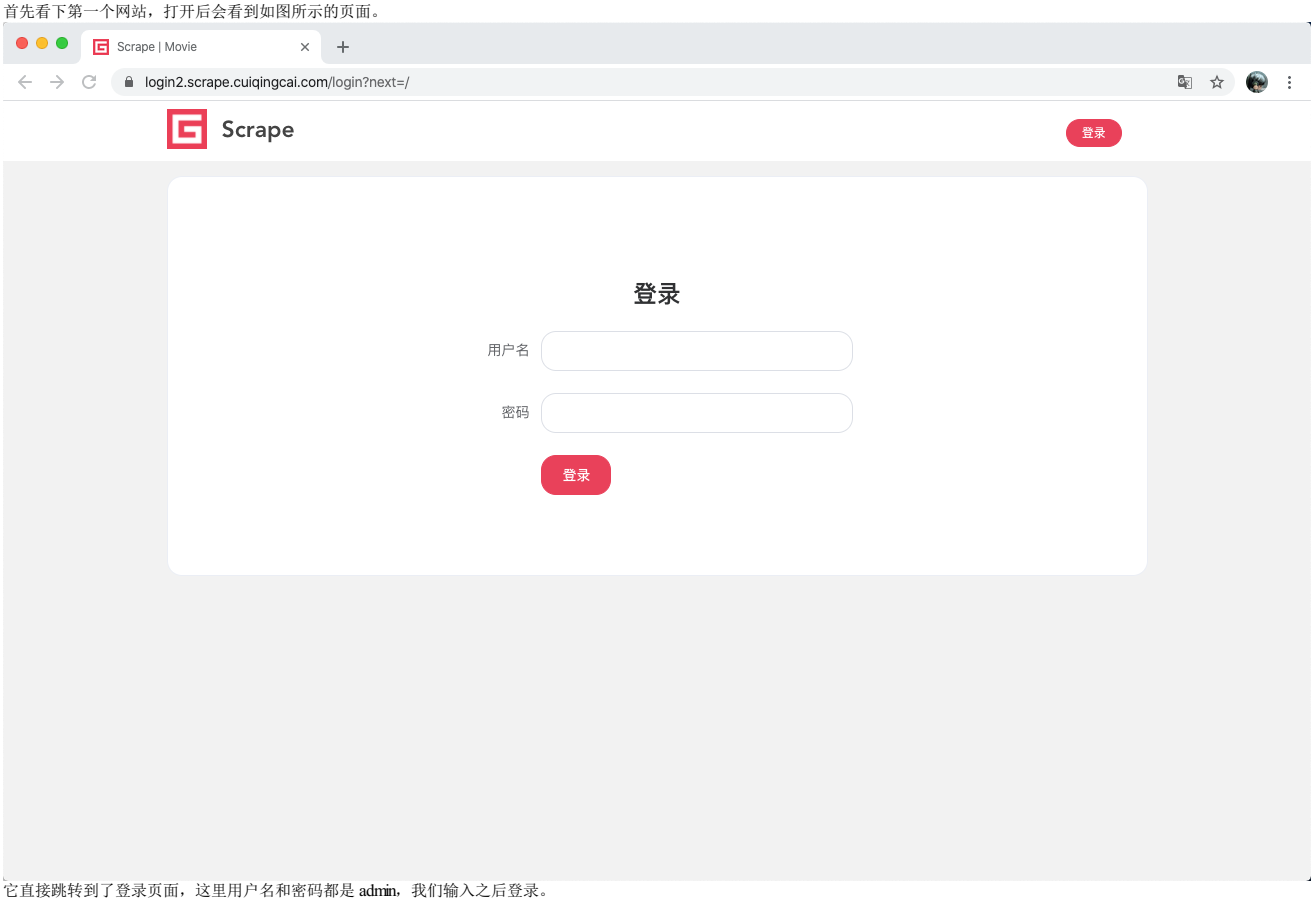
- 安装好了 Python（最好 3.6 及以上版本）并能成功运行 Python 程序；
- 安装好了 requests 请求库并学会了其基本用法；
- 安装好了 Selenium 库并学会了其基本用法。

下面我们就以两个案例为例来分别讲解模拟登录的实现。

案例介绍

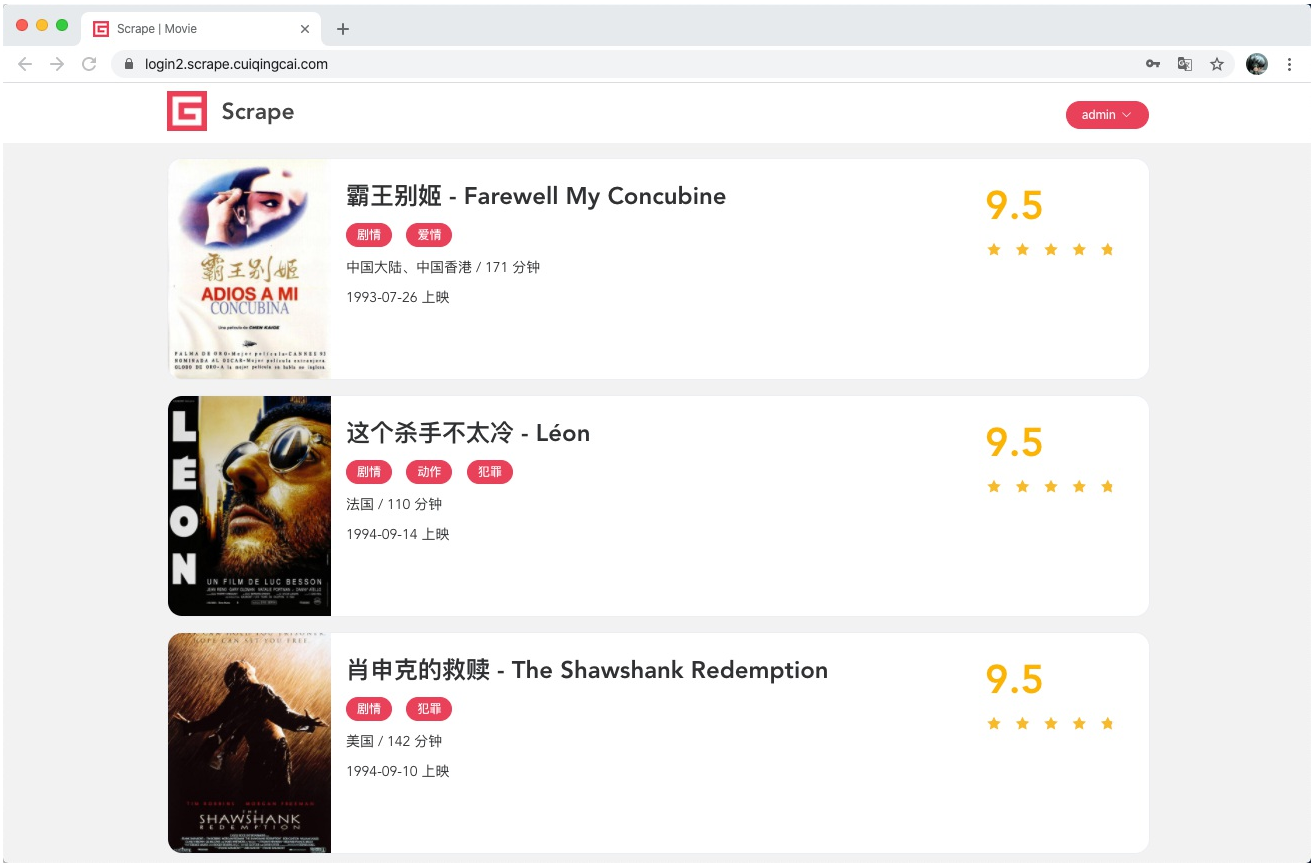
这里有两个需要登录才能抓取的网站，链接为 https://login2.scrape.cuiqingcai.com/ 和 https://login3.scrape.cuiqingcai.com/，前者是基于 Session + Cookies 认证的网站，后者是基于 JWT 认证的网站。

首先看下第一个网站，打开后会看到如图所示的页面。



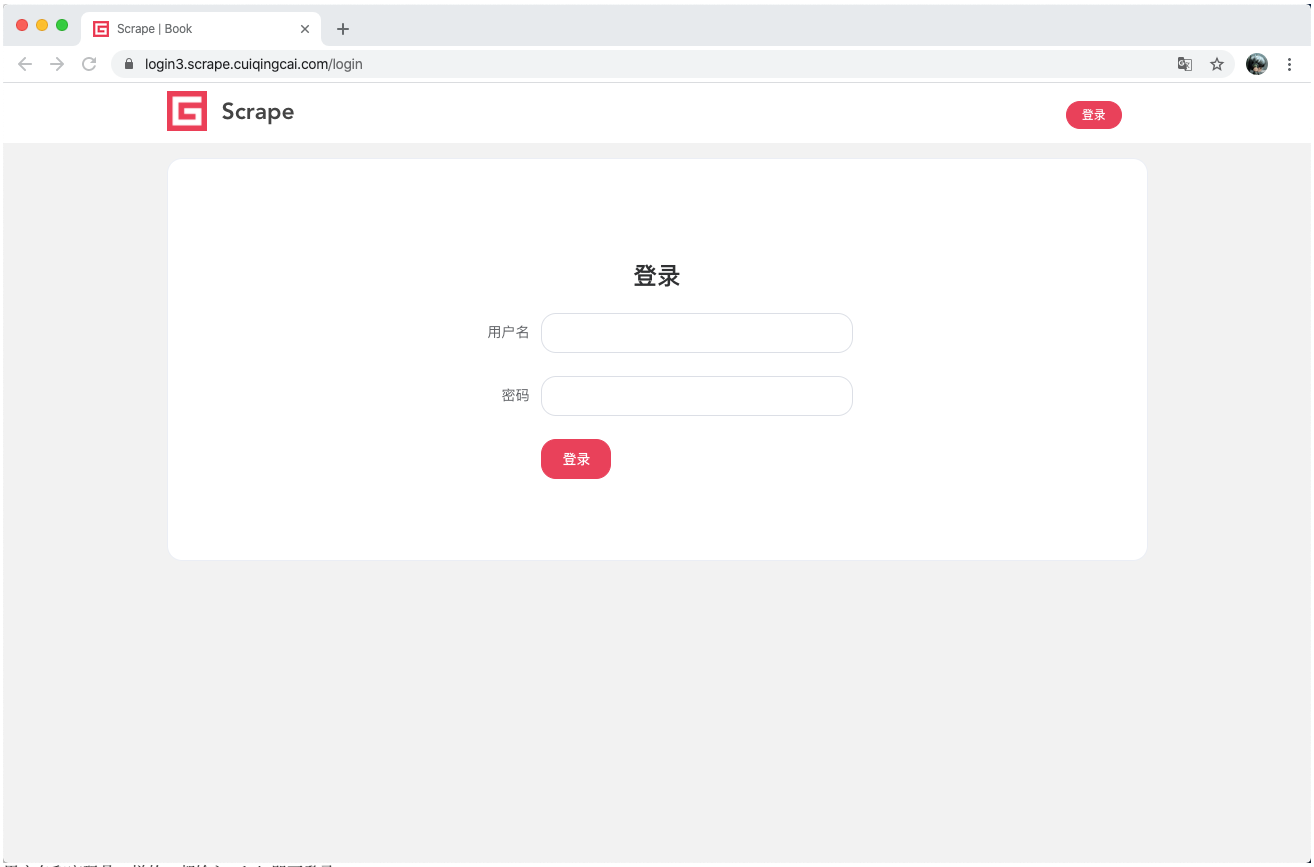
它直接跳转到了登录页面，这里用户名和密码都是 admin，我们输入之后登录。

登录成功之后，我们便看到了熟悉的电影网站的展示页面，如图所示。



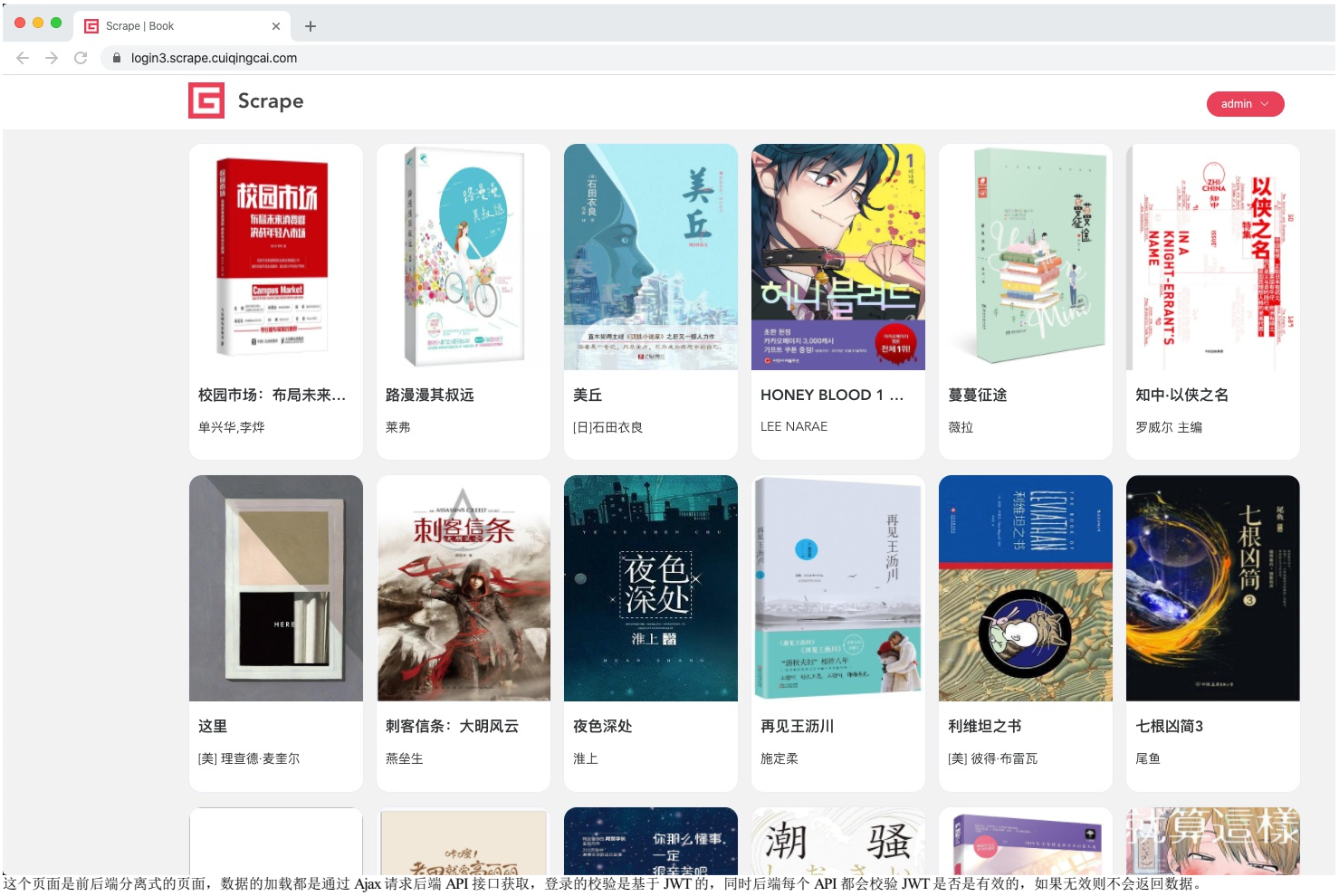
这个网站是基于传统的 MVC 模式开发的，因此也比较适合 Session + Cookies 的认证。

第二个网站打开后同样会跳到登录页面，如图所示。



用户名和密码是一样的，都输入 admin 即可登录。

登录之后会跳转到首页，展示了一些书籍信息，如图所示。

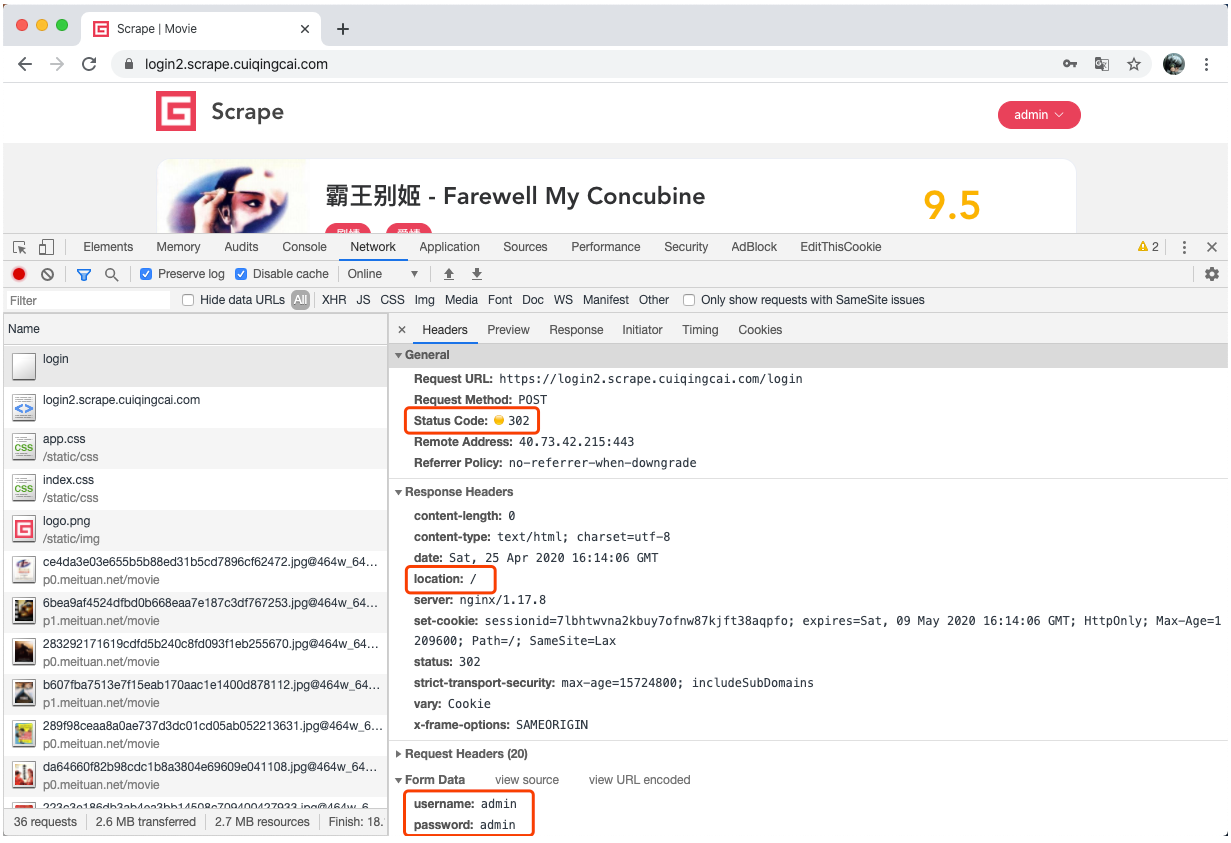


这个页面是前后端分离式的页面，数据的加载都是通过 Ajax 请求后端 API 接口获取，登录的校验是基于 JWT 的，同时后端每个 API 都会校验 JWT 是否是有效的，如果无效则不会返回数据。

接下来我们就分析这两个案例并实现模拟登录吧。

案例一

对于案例一，我们如果要模拟登录，就需要先分析下登录过程究竟发生了什么，首先我们打开 https://login2.scrape.cuiqingcai.com/，然后执行登录操作，查看其登录过程中发生的请求，如图所示。



这里我们可以看到其登录的瞬间是发起了一个 POST 请求，目标 URL 为 https://login2.scrape.cuiqingcai.com/login，通过表单提交的方式提交了登录数据，包括 username 和 password 两个字段，返回的状态码是 302，Response Headers 的 location 字段是根页面，同时 Response Headers 还包含了 set-cookie 信息，设置了 Session ID。

由此我们可以发现，要实现模拟登录，我们只需要模拟这个请求就好了，登录完成之后获取 Response 设置的 Cookies，将 Cookies 保存好，以后后续的请求带上 Cookies 就可以正常访问了。

好，那么我们接下来用代码实现一下吧。

requests 默认情况下每次请求都是独立互不干扰的，比如我们第一次先调用了 post 方法模拟登录，然后紧接着再调用 get 方法请求下主页面，其实这是两个完全独立的请求，第一次请求获取的 Cookies 并不能传给第二次请求，因此说，常规的顺序调用是不能起到模拟登录的效果的。

我们先来看一个无效的代码：

```
import requests
from urllib.parse import urljoin

BASE_URL = 'https://login2.scrape.cuiqingcai.com/'
LOGIN_URL = urljoin(BASE_URL, '/login')
INDEX_URL = urljoin(BASE_URL, '/page/1')
USERNAME = 'admin'
PASSWORD = 'admin'

response_login = requests.post(LOGIN_URL, data={
   'username': USERNAME,
   'password': PASSWORD
})

response_index = requests.get(INDEX_URL)
print('Response Status', response_index.status_code)
print('Response URL', response_index.url)
```

这里我们先定义了几个基本的 URL 和用户名、密码，接下来分别用 requests 请求了登录的 URL 进行模拟登录，然后紧接着请求了首页来获取页面内容，但是能正常获取数据吗？

由于 requests 可以自动处理重定向，我们最后把 Response 的 URL 打印出来，如果它的结果是 INDEX_URL，那么就证明模拟登录成功并成功爬取到了首页的内容。如果它跳回到了登录页面，那就说明模拟登录失败。

我们通过结果来验证一下，运行结果如下：

```
Response Status 200
Response URL https://login2.scrape.cuiqingcai.com/login?next=/page/1
```

这里可以看到，其最终的页面 URL 是登录页面的 URL，另外这里也可以通过 response 的 text 属性来验证页面源码，其源码内容就是登录页面的源码内容，由于内容较多，这里就不再输出比对了。

总之，这个现象说明我们并没有成功完成模拟登录，这是因为 requests 直接调用 post、get 等方法，每次请求都是一个独立的请求，都相当于是新开了一个浏览器打开这些链接，这两次请求对应的 Session 并不是同一个，因此这里我们模拟了第一个 Session 登录，而这并不能影响第二个 Session 的状态，因此模拟登录也就无效了。
那么怎样才能实现正确的模拟登录呢？

我们知道 Cookies 里面是保存了 Session ID 信息的，刚才也观察到了登录成功后 Response Headers 里面是有 set-cookie 字段，实际上这就是让浏览器生成了 Cookies。

Cookies 里面包含了 Session ID 的信息，所以只要后续的请求携带这些 Cookies，服务器便能通过 Cookies 里的 Session ID 信息找到对应的 Session，因此服务端对于这两次请求就会使用同一个 Session 了。而因为第一次我们已经完成了模拟登录，所以第一次模拟登录成功后，Session 里面就记录了用户的登录信息，第二次访问的时候，由于是同一个 Session，服务器就能知道用户当前是登录状态，就可以返回正确的结果而不再是跳转到登录页面了。

所以，这里的关键就在于两次请求的 Cookies 的传递。所以这里我们可以把第一次模拟登录后的 Cookies 保存下来，在第二次请求的时候加上这个 Cookies 就好了，所以代码可以改写如下：

```
import requests
from urllib.parse import urljoin

BASE_URL = 'https://login2.scrape.cuiqingcai.com/'
LOGIN_URL = urljoin(BASE_URL, '/login')
INDEX_URL = urljoin(BASE_URL, '/page/1')
USERNAME = 'admin'
PASSWORD = 'admin'

response_login = requests.post(LOGIN_URL, data={
   'username': USERNAME,
   'password': PASSWORD
}, allow_redirects=False)

cookies = response_login.cookies
print('Cookies', cookies)

response_index = requests.get(INDEX_URL, cookies=cookies)
print('Response Status', response_index.status_code)
print('Response URL', response_index.url)
```

由于 requests 可以自动处理重定向，所以模拟登录的过程我们要加上 allow_redirects 参数并设置为 False，使其不自动处理重定向，这里登录之后返回的 Response 我们赋值为 response_login，这样通过调用 response_login 的 cookies 就可以获取到网站的 Cookies 信息了，这里 requests 自动帮我们解析了 Response Headers 的 set-cookie 字段并设置了 Cookies，所以我们不需要手动解析 Response Headers 的内容了，直接使用 response_login 对象的 cookies 属性即可获取 Cookies。

好，接下来我们再次用 requests 的 get 方法来请求网站的 INDEX_URL，不过这里和之前不同，get 方法多加了一个参数 cookies，这就是第一次模拟登录完之后获取的 Cookies，这样第二次请求就能携带第一次模拟登录获取的 Cookies 信息了，此时网站会根据 Cookies 里面的 Session ID 信息查找到同一个 Session，校验其已经是登录状态，然后返回正确的结果。

这里我们还是输出了最终的 URL，如果其是 INDEX_URL，那就代表模拟登录成功并获取到了有效数据，否则就代表模拟登录失败。

我们看下运行结果：

```
Cookies <RequestsCookieJar[<Cookie sessionid=psnu8ij69f0ltecd5wasccyzc6ud41tc for login2.scrape.cuiqingcai.com/>]>
Response Status 200
Response URL https://login2.scrape.cuiqingcai.com/page/1
```

这下就没有问题了，这次我们发现其 URL 就是 INDEX_URL，模拟登录成功了！同时还可以进一步输出 response_index 的 text 属性看下是否获取成功。

接下来后续的爬取用同样的方式爬取即可。

但是我们发现其实这种实现方式比较烦琐，每次还需要处理 Cookies 并进行一次传递，有没有更简便的方法呢？

有的，我们可以直接借助于 requests 内置的 Session 对象来帮我们自动处理 Cookies，使用了 Session 对象之后，requests 会将每次请求后需要设置的 Cookies 自动保存好，并在下次请求时自动携带上去，就相当于帮我们维持了一个 Session 对象，这样就更方便了。

所以，刚才的代码可以简化如下：

```
import requests
from urllib.parse import urljoin

BASE_URL = 'https://login2.scrape.cuiqingcai.com/'
LOGIN_URL = urljoin(BASE_URL, '/login')
INDEX_URL = urljoin(BASE_URL, '/page/1')
USERNAME = 'admin'
PASSWORD = 'admin'

session = requests.Session()

response_login = session.post(LOGIN_URL, data={
   'username': USERNAME,
```

```
    'password': PASSWORD
})

cookies = session.cookies
print('Cookies', cookies)

response_index = session.get(INDEX_URL)
print('Response Status', response_index.status_code)
print('Response URL', response_index.url)
```

可以看到，这里我们无需再关心 Cookies 的处理和传递问题，我们声明了一个 Session 对象，然后每次调用请求的时候都直接使用 Session 对象的 post 或 get 方法就好了。

运行效果是完全一样的，结果如下：

```
Cookies <RequestsCookieJar[<Cookie sessionid=ssngkl4i7en9vm73bb36hxif05k10k13 for login2.scrape.cuiqingcai.com/>]>

Response Status 200

Response URL https://login2.scrape.cuiqingcai.com/page/1
```

因此，为了简化写法，这里建议直接使用 Session 对象来进行请求，这样我们就无需关心 Cookies 的操作了，实现起来会更加方便。

这个案例整体来说比较简单，但是如果碰上复杂一点的网站，如带有验证码，带有加密参数等等，直接用 requests 并不好处理模拟登录，如果登录不了，那岂不是整个页面都没法爬了吗？那么有没有其他的方式来解决这个问题呢？当然是有的，比如说，我们可以使用 Selenium 来通过模拟浏览器的方式实现模拟登录，然后获取模拟登录成功后的 Cookies，再把获取的 Cookies 交由 requests 等来爬取就好了。

这里我们还是以刚才的页面为例，我们可以把模拟登录这块交由 Selenium 来实现，后续的爬取交由 requests 来实现，代码实现如下：

```
from urllib.parse import urljoin
from selenium import webdriver
import requests
import time

BASE_URL = 'https://login2.scrape.cuiqingcai.com/'
LOGIN_URL = urljoin(BASE_URL, '/login')
INDEX_URL = urljoin(BASE_URL, '/page/1')
USERNAME = 'admin'
PASSWORD = 'admin'

browser = webdriver.Chrome()
browser.get(BASE_URL)
browser.find_element_by_css_selector('input[name="username"]').send_keys(USERNAME)
browser.find_element_by_css_selector('input[name="password"]').send_keys(PASSWORD)
browser.find_element_by_css_selector('input[type="submit"]').click()
time.sleep(10)

# get cookies from selenium
cookies = browser.get_cookies()
print('Cookies', cookies)
browser.close()

# set cookies to requests
session = requests.Session()
for cookie in cookies:
    session.cookies.set(cookie['name'], cookie['value'])

response_index = session.get(INDEX_URL)
print('Response Status', response_index.status_code)
print('Response URL', response_index.url)
```

这里我们使用 Selenium 先打开了 Chrome 浏览器，然后跳转到了登录页面，随后模拟输入了用户名和密码，接着点击了登录按钮，这时候我们可以发现浏览器里面就提示登录成功，然后成功跳转到了主页面。

这时候，我们通过调用 get_cookies 方法便能获取到当前浏览器所有的 Cookies，这就是模拟登录成功之后的 Cookies，用这些 Cookies 我们就能访问其他的数据了。

接下来，我们声明了 requests 的 Session 对象，然后遍历了刚才的 Cookies 并设置到 Session 对象的 cookies 上面去，接着再拿着这个 Session 对象去请求 INDEX_URL，也就能够获取到对应的信息而不会跳转到登录页面了。

运行结果如下：

```
Cookies [{'domain': 'login2.scrape.cuiqingcai.com', 'expiry': 1589043753.553155, 'httpOnly': True, 'name': 'sessionid', 'path': '/', 'sameSite': 'Lax', 'secure': False, 'value': 'rdag7

Response Status 200

Response URL https://login2.scrape.cuiqingcai.com/page/1
```

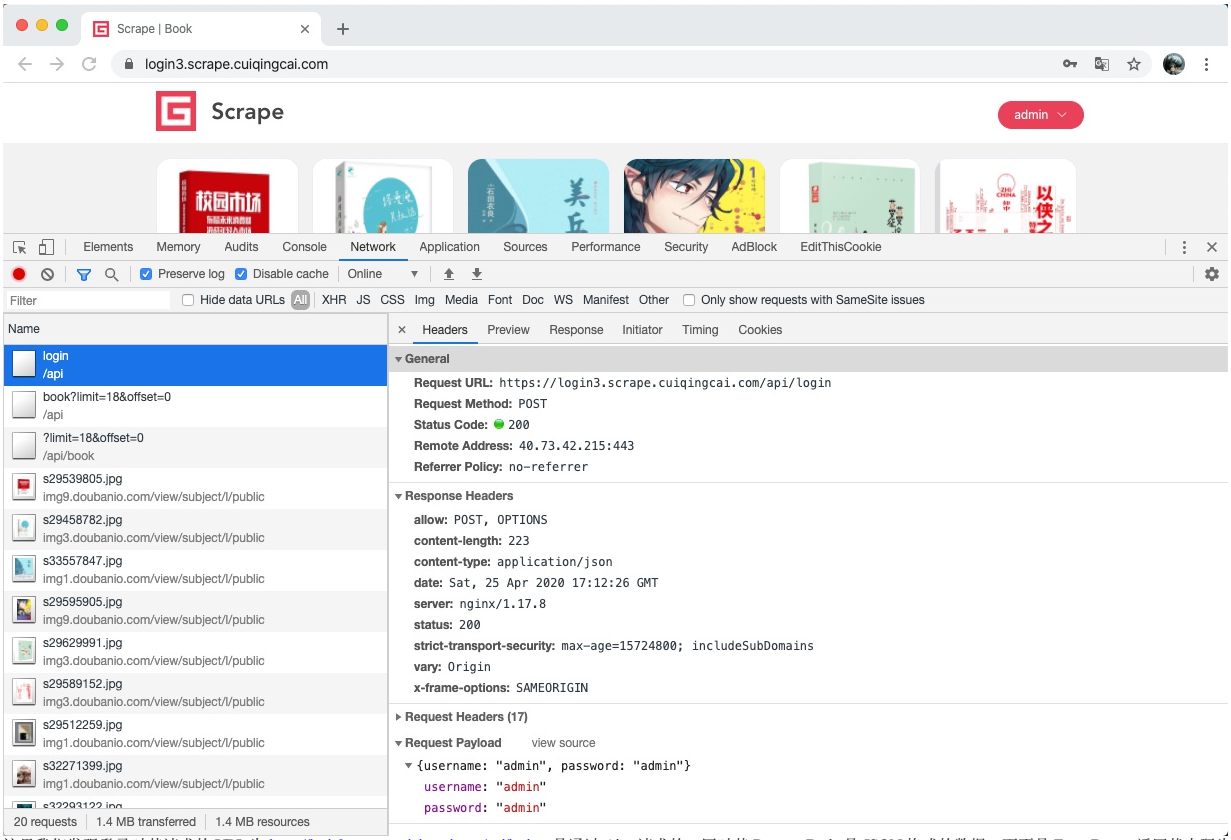
可以看到这里的模拟登录和后续的爬取也成功了。所以说，如果碰到难以模拟登录的过程，我们也可以使用 Selenium 或 Pyppeteer 等模拟浏览器操作的方式来实现，其目的就是取到登录后的 Cookies，有了 Cookies 之后，我们再用这些 Cookies 爬取其他页面就好了。

所以这里我们也可以发现，对于基于 Session + Cookies 验证的网站，模拟登录的核心要点就是获取 Cookies，这个 Cookies 可以被保存下来或传递给其他的程序继续使用。甚至说可以将 Cookies 持久化存储或传输给其他终端来使用。另外，为了提高 Cookies 利用率或降低封号几率，可以搭建一个 Cookies 池实现 Cookies 的随机取用。

案例二

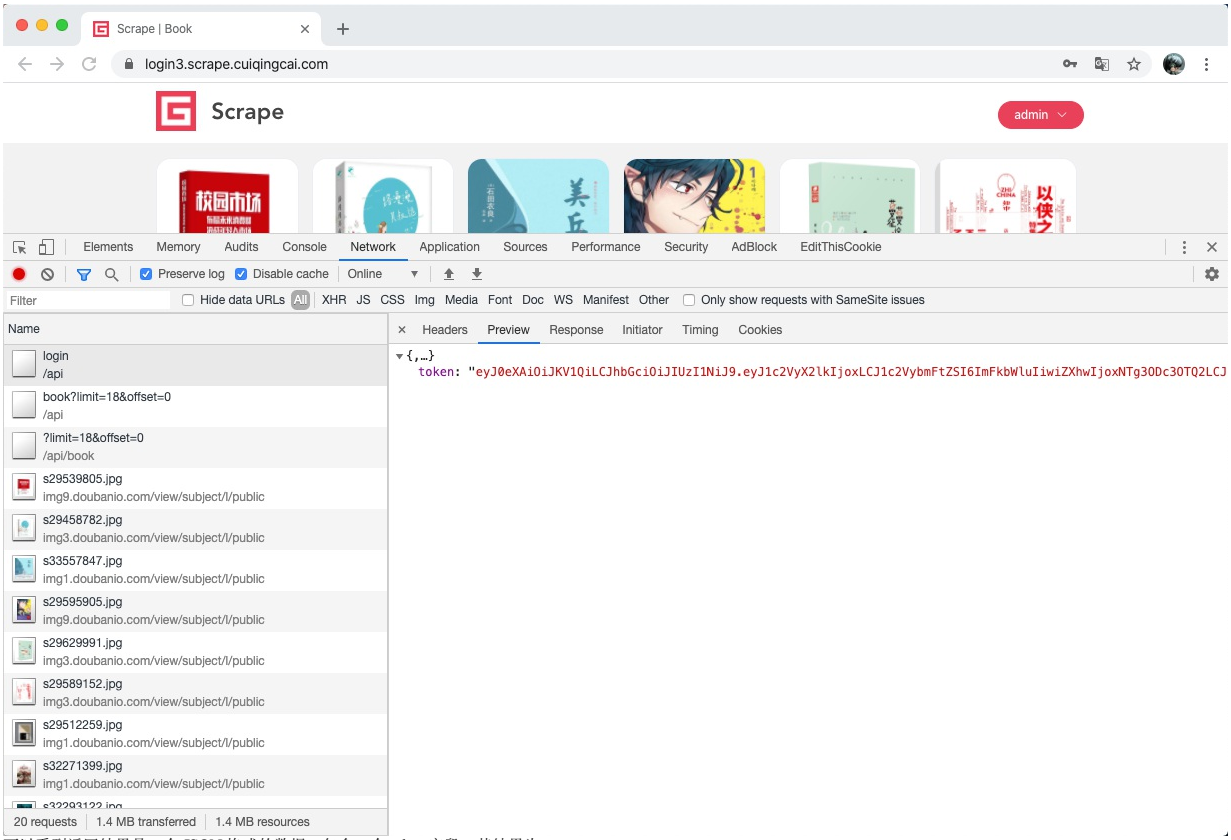
对于案例二这种基于 JWT 的网站，其通常都是采用前后端分离式的，前后端的数据传输依赖于 Ajax，登录验证依赖于 JWT 本身这个 token 的值，如果 JWT 这个 token 是有效的，那么服务器就能返回想要的数据。

下面我们先来在浏览器里面操作登录，观察下其网络请求过程，如图所示。



这里我们发现登录时其请求的 URL 为 https://login3.scrape.cuiqingcai.com/api/login，是通过 Ajax 请求的，同时其 Request Body 是 JSON 格式的数据，而不是 Form Data，返回状态码为 200。

然后再看下返回结果，如图所示。

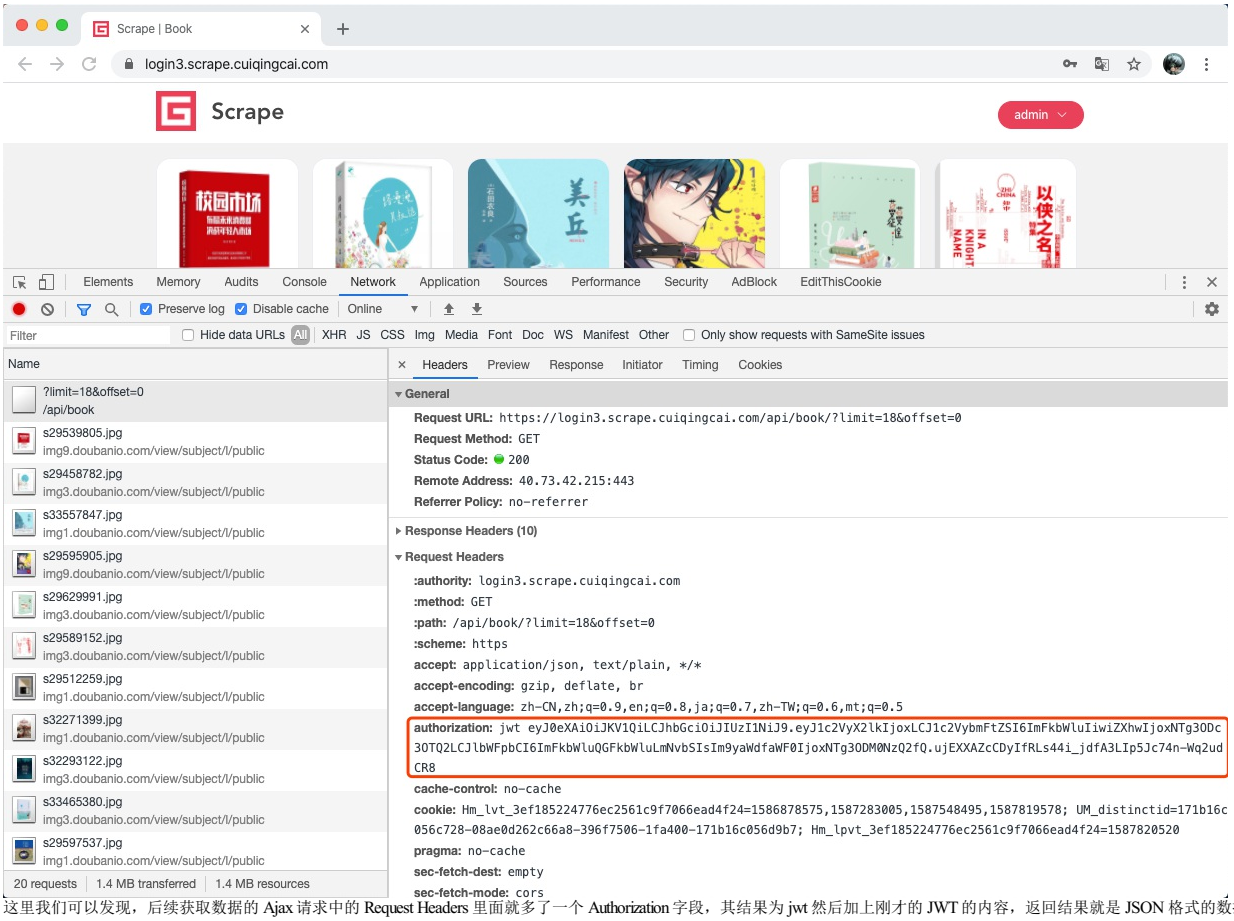


可以看到返回结果是一个 JSON 格式的数据，包含一个 token 字段，其结果为：

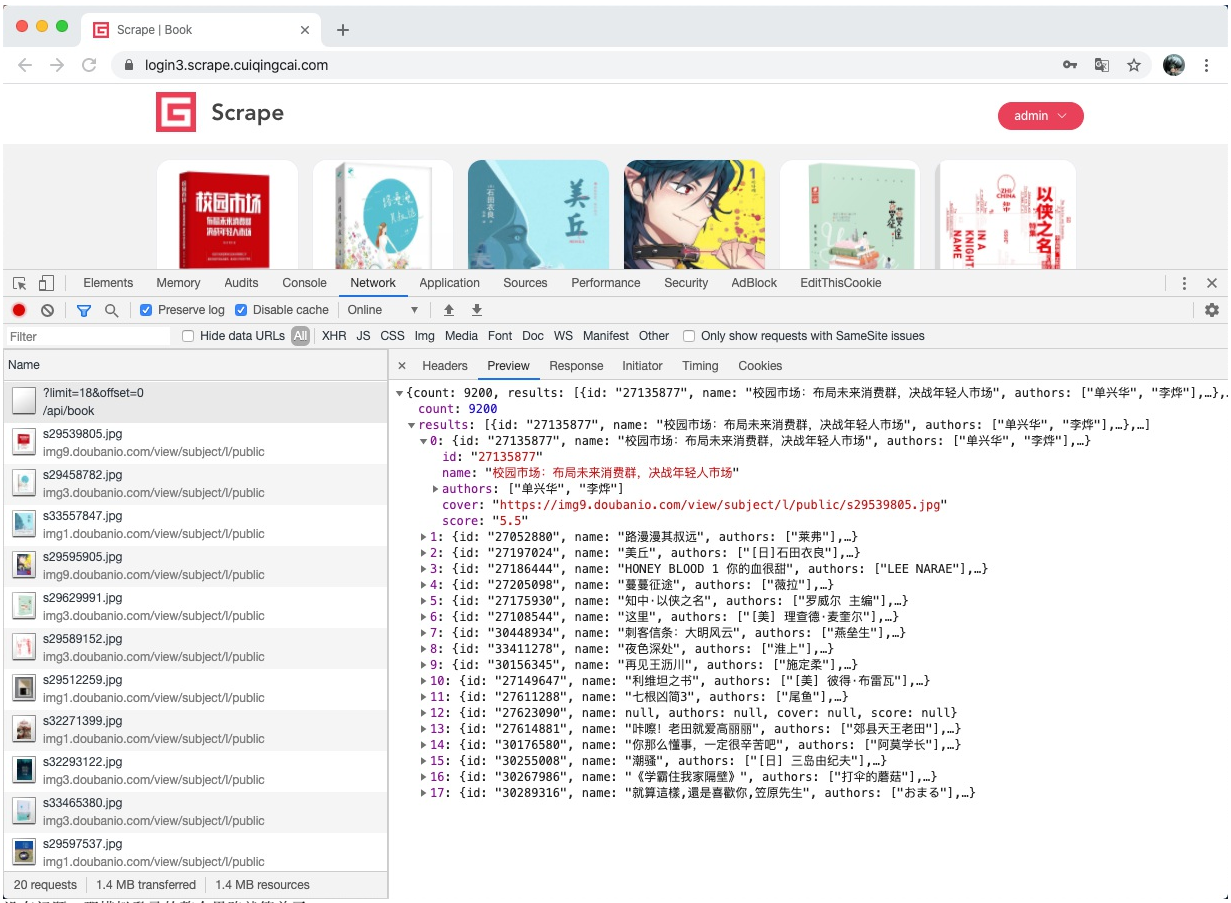
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNTg3ODc3OTQ2LCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsIm9yaWdfaWF0IjoxNTg3ODM0NzQ2fQ.ujEXXAZcCDyIfRLs
```

这就是我们上一课时所讲的 JWT 的内容，格式是三段式的，通过“.”来分隔。

那么有了这个 JWT 之后，后续的数据怎么获取呢？下面我们再来观察下后续的请求内容，如图所示。



这里我们可以发现，后续获取数据的 Ajax 请求中的 Request Headers 里面就多了一个 Authorization 字段，其结果为 jwt 然后加上刚才的 JWT 的内容，返回结果就是 JSON 格式的数据。



没有问题，那模拟登录的整个思路就简单了：
模拟请求登录结果，带上必要的登录信息，获取 JWT 的结果。

后续的请求在 Request Headers 里面加上 Authorization 字段，值就是 JWT 对应的内容。
好，接下来我们用代码实现如下：

```
import requests
```

```
from urllib.parse import urljoin

BASE_URL = 'https://login3.scrape.cuiqingcai.com/'
LOGIN_URL = urljoin(BASE_URL, '/api/login')
INDEX_URL = urljoin(BASE_URL, '/api/book')
USERNAME = 'admin'
PASSWORD = 'admin'

response_login = requests.post(LOGIN_URL, json={
    'username': USERNAME,
    'password': PASSWORD
})
data = response_login.json()
print('Response JSON', data)
jwt = data.get('token')
print('JWT', jwt)

headers = {
    'Authorization': f'jwt {jwt}'
}
response_index = requests.get(INDEX_URL, params={
    'limit': 18,
    'offset': 0
}, headers=headers)
print('Response Status', response_index.status_code)
print('Response URL', response_index.url)
print('Response Data', response_index.json())
```

这里我们同样是定义了登录接口和获取数据的接口，分别为 LOGIN_URL 和 INDEX_URL，接着通过 post 请求进行了模拟登录，这里提交的数据由于是 JSON 格式，所以这里使用 json 参数来传递。接着获取了返回结果中包含的 JWT 的结果。第二步就可以构造 Request Headers，然后设置 Authorization 字段并传入 JWT 即可，这样就能成功获取数据了。

运行结果如下：

```
Response JSON {'token': 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNTg3ODc4NzkxLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsIm9yaWdfaWF0IjoxNTg3OD

JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJ1c2VybmFtZSI6ImFkbWluIiwiZXhwIjoxNTg3ODc4NzkxLCJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsIm9yaWdfaWF0IjoxNTg3ODM1NTkxfQ.iUnu3Yhdi_a-

Response Status 200
Response URL https://login3.scrape.cuiqingcai.com/api/book/?limit=18&offset=0
Response Data {'count': 9200, 'results': [{'id': '27135877', 'name': '校园市场：布局未来消费群，决战年轻人市场', 'authors': ['单兴华', '李烨'], 'cover': 'https://img9.doubanio.com/view/subject/l
...
{'id': '30289316', 'name': '就算這樣,還是喜歡你,笠原先生', 'authors': ['おまる'], 'cover': 'https://img3.doubanio.com/view/subject/l/public/s29875002.jpg', 'score': '7.5'}]}
```

可以看到，这里成功输出了 JWT 的内容，同时最终也获取到了对应的数据，模拟登录成功！

类似的思路，如果我们遇到 JWT 认证的网站，也可以通过类似的方式来实现模拟登录。当然可能某些页面比较复杂，需要具体情况具体分析。

总结

以上我们就通过两个示例来演示了模拟登录爬取的过程，以后遇到这种情形的时候就可以用类似的思路解决了。

代码：https://github.com/Python3WebSpider/ScrapeLogin2、https://github.com/Python3WebSpider/ScrapeLogin3。