本科生课程论文规范与存档

1. 任课教师要对课程论文选题范围、结构、篇幅格式、上交时间、评分标准、成绩占期末考试比例等方面向学生提出具体要求。
2. 学生按照提供的论文模板格式撰写论文，打印签名后按时上交。
3. 任课老师按照评分标准，对论文的选题、分析论证、观点结论等方面给出简要评价，对论文进行打分。
4. 课程论文存档参照试卷存档要求，归档资料有①试卷存档封面（1份）；②学生成绩登记表（1份）；③试卷分析表（1份）；④课程论文要求（即相当于考题）（1份）；⑤论文评分标准（1份）；⑥学生论文。
5. 课程论文等材料为电子版如光盘等，请在试卷册封面备注栏进行情况说明。

附件 1、试卷存档封面

2、本科生课程论文封面和论文格式

本科生课程期末考试试卷档案

|  |  |
| --- | --- |
| 课程名称： | 信息系统安全 |
| 开课学院： | 软件学院 |
| 课程性质： | 必修课（ ）选修课（√）通选课（ ）附修课（ ） |
| 学生班级： | 2016 级 软件工程1,2 班 |
| 任课教师： | 陈春华 |
| 试卷份数： | 份 |
| 所属学期： | 2018-2019 学年 第 2 学期 |
| 考试日期： |  |
| 归档日期： |  |
| 档案卷数： |  |

阅卷登记：

|  |  |  |  |
| --- | --- | --- | --- |
| **考核方式** | **□论文**  **□其他 .** | **评阅日期** |  |
| **阅卷方式** | **□ 流水阅卷**  **□ 单人阅卷** | **评阅签名** |  |

|  |
| --- |
| 备注说明： |

（考核方式为论文、或者材料有其它情况的请备注。）

存档资料：①试卷存档封面（1份）；②学生成绩登记表（1份）；③试卷分析表（1份）；④课程论文要求（1份）；⑤论文评价标准（1份）；⑥学生论文（顺号）

**《信息系统安全》 课 程 论 文**

**(2018-2019学年第2学期)**

**基于Web的反反爬虫技术实现**

**学生姓名： 李林键 叶苏航 吴峻羽 刘一达**

**提交日期：2019年 6月 30日 学生签名：**

|  |  |  |  |
| --- | --- | --- | --- |
| **学 号** | **201630406969，201630666202，201630665892，201630665229**  **（按照姓名顺序）** | **座位编号** |  |
| **学 院** | **软件学院** | **专业班级** | **软件工程1,2班** |
| **课程名称** | **信息系统安全** | **任课教师** | **陈春华** |
| **教师评语：** | | | |
| **本论文成绩评定： 分** | | | |

目录

[基于Web端的反反爬虫技术实现 5](#_Toc12788383)

[1. 简介 6](#_Toc12788384)

[2. 反爬虫策略与反反爬虫实战 7](#_Toc12788385)

[2.1. robots.txt约束 7](#_Toc12788386)

[2.2. HTTP请求头部检查 8](#_Toc12788387)

[2.3. IP代理池 14](#_Toc12788388)

[2.4. JS文件数据加密的破解 18](#_Toc12788389)

[2.5. Ajax加密破解 22](#_Toc12788390)

[2.6. 蜜罐技术 26](#_Toc12788391)

[总结 29](#_Toc12788392)

[参考文献 30](#_Toc12788393)

基于Web端的反反爬虫技术实现

李林键 叶苏航 吴峻羽 刘一达

**摘要：**随着爬虫技术的兴起与广泛运用，各种反爬虫技术也被网站利用起来以防止数据被爬虫轻易爬取，进而保护网站关键数据。如今，当我们还想利用爬虫在网站上爬取数据时，就必须针对网站所使用的反爬虫技术，进行“反反爬虫”。本文主要针对robots.txt约束, HTTP请求头部检查，IP封锁，JS文件数据加密，Ajax破解加密以及蜜罐技术五个反爬虫技术进行了反反爬虫措施的讨论并使用Scrapy爬虫框架进行实战，成功实现“反反爬虫”。

**关键词：**反爬虫；反反爬虫；IP封锁；JS文件数据加密破解；ajax动态破解；蜜罐技术；Scrapy

# 1. 简介

网络爬虫，即Web Spider，是一个很形象的名字。把互联网比喻成一个蜘蛛网，那么Spider就是在网上爬来爬去的蜘蛛。网络蜘蛛是通过网页的链接地址来寻找网页的。

从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其它链接地址，然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页或者说，我们所想要的网页都抓取完为止。如果把整个互联网当成一个网站，那么网络蜘蛛就可以用这个原理把互联网上所有的网页都抓取下来。这样看来，网络爬虫就是一个爬行程序，一个抓取网页的程序。

然而，在如今的大数据时代，数据对于网站以及企业的重要不言而喻，若是网站轻而易举地把自己所有的数据拱手让人，定然会对网站的经营者与用户造成伤害，甚至于造成一些不可挽回的后果。这是所有的网页经营者所不想看到的。因此，伴随着网页爬虫的兴起，反爬虫技术也应运而生。反爬虫技术现在有许多种类，比如通过Headers检测进行反爬虫、基于用户行为反爬虫、动态页面反爬虫、使用会话token反爬虫、蜜罐技术等，还有越来越多的反爬虫技术正在被研究出来并运用到实践中。然而，面对网页数据这块大蛋糕，坐在电脑前的各位怎么可能不动心。而网页由于要把数据展示给网页的真实用户，因此必须把数据以及各项文件发送到网页的请求方。也就是说，网页既然要把数据展示给真实用户，就无法做到数据的绝对保密，这也让我们的“反反爬虫”有了可乘之机。反爬虫技术需要在真实用户与爬虫之间作一个权衡，而我们反反爬虫，就通过这个权衡之间的缝隙，去爬取数据。

本文主要针对robots.txt约束、 HTTP请求头部检查、IP封锁、JS文件数据加密、Ajax破解加密、蜜罐技术这6个反爬虫技术进行了反反爬虫措施的讨论并进行实战。针对robots.txt约束反爬虫技术，我们修改了爬虫以让他不遵守robots.txt协议；针对HTTP请求头部检查反爬虫技术，我们将爬虫爬取时的User-Agent改为与真实用户使用浏览器去浏览时一致；针对IP封锁，我们采用了IP代理池的办法降低单个IP对服务器的访问频率；针对JS文件数据加密，我们通过破解JS文件的数据格式进行反反爬虫；针对ajax动态加密反爬虫，我们通过两种方式，模拟用户操作和跳过操作部分直接调用加载功能，去获取我们需要的数据；针对蜜罐技术，我们没有进行实战，但我们通过我们华工的教务网站，体验了蜜罐技术的整体实现和运作的机制。

接下来，我将分别从我们的几项实战实例中，说明并分析我们小组反反爬虫项目的具体实现。

# 2. 反爬虫策略与反反爬虫实战

## 2.1. robots.txt约束

robots.txt约束可以归类为通过Headers反爬虫的一种。robots.txt 是一个爬虫规范协议，看名称就知道它是一个 txt 的文本，放在网站的根目录下。robots.txt 文件由一条或多条规则组成。每条规则可禁止（或允许）特定抓取工具抓取相应网站中的指定文件路径。它主要的作用就是来告诉爬虫，我这个网站，你哪些你能看，哪些你不能看的一个协议。

robots.txt 中主要有 4 个关键词。放在行首，用英文“:”分割内容部分或指令体。

* User-agent 网页抓取工具的名称
* Disallow 不应抓取的目录或网页
* Allow 应抓取的目录或网页
* Sitemap 网站的站点地图的位置

我们在爬取豆瓣的电影页面时，通过scrapy框架书写完爬虫代码后直接进行爬取，得到的反馈却是403，403代码的含义是被禁止，也就是被反爬虫了

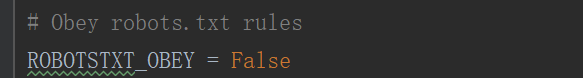


因此我们查看了淘宝网的robots.txt，看到了以下几行：



上面两条语句的意思是：对于所有的爬虫，禁止爬虫访问根目录以下的所有目录。我们现在的爬虫是设置为服从robots.txt的，因此后续爬取行为就失败了。

因此，我们通过修改settings.py文件中的ROBOTSTXT\_OBEY一项，使我们的爬虫不去遵守这个robots.txt协议，进而可以顺利进入下一步的爬取：



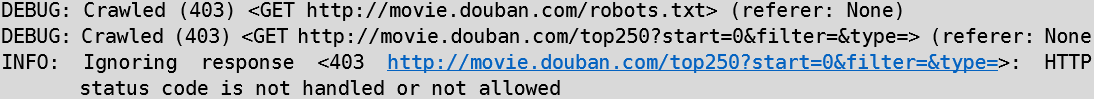
重新运行爬虫程序，得到以下输出



## 2.2. HTTP请求头部检查

通过HTTP头部识别爬虫同样也是大部分网站会采取的一种基础的反爬虫技术。爬虫默认的HTTP请求头部与我们人为的通过浏览器去真正浏览一个网站时的头部是不相同的，因此通过HTTP头部可以很轻松地识别出该访问请求是来自于真的的用户还是来自于爬虫。

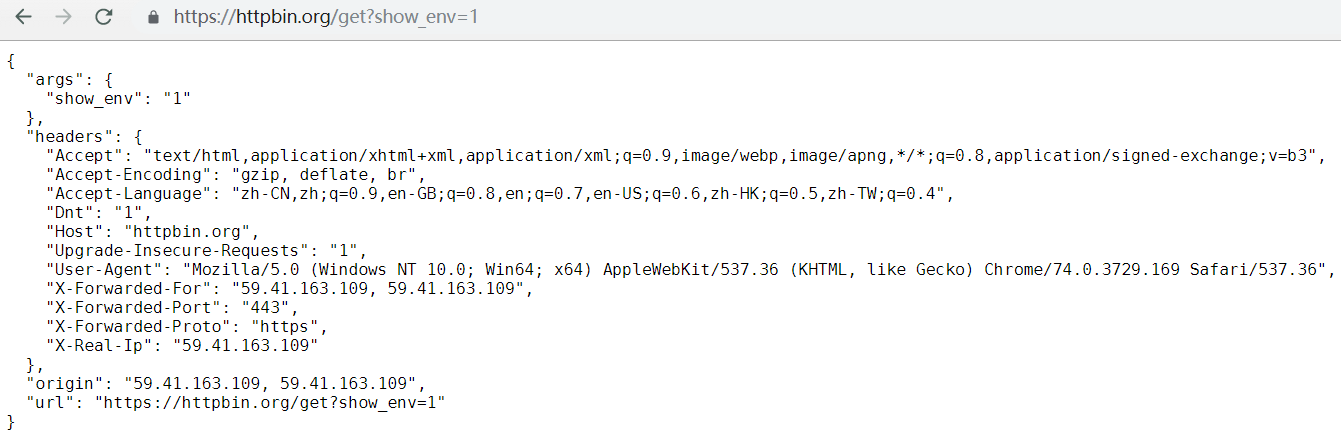
我们编写程序爬取豆瓣电影Top-250的页面内容，在运行爬虫程序之后，我们得到以下输出



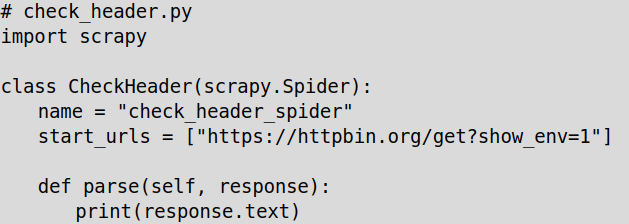
返回代码403就表示forbidden，被禁止了，也就是被“反爬虫”了。从上图第一行可以看到，我们甚至无法获取robots.txt文件，说明我们这次被反爬虫并不是简单的基于robots.txt文件的。

上网查询后得知，许多网站会检查HTTP请求的头部，爬虫默认的请求头部是很容易看出来该请求来自爬虫的。针对HTTP请求头部检查，我们采取的技术是通过将爬虫访问网站时发送的HTTP请求头部修改为与真实用户使用浏览器去浏览网站时的HTTP请求头部一致，进而使网站无法从HTTP请求头部检查识别出我们是一只爬虫，从而反掉这项反爬虫技术。

首先，我们在浏览器上打开链接 [https://[httpbin.org/get?show\_env=1](https://httpbin.org/get?show_env=1)] 可以查看我们真实用户使用浏览器去访问网页时的HTTP请求头部内容：



接下来我们新建check\_header.py，并用它来查看当使用爬虫的时候默认的请求头部内容。



然后运行名为check\_header\_spider的爬虫，可以得到以下输出

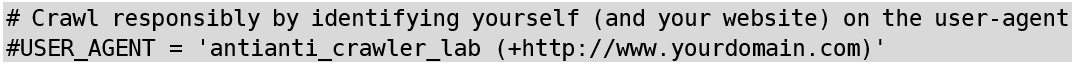


可以看到上面有一行“User-Agent”种对应的内容是“Scrapy/1.5.2”，表明这次访问是由爬虫程序发起的。而正常用浏览器访问的时候对应的内容为“Mozilla/5.0 (Windows NT 10.0; Win64; x64)……”。网站在检索到爬虫访问头部的User-Agent之后，就会禁止本次的访问，这也是常见的反爬虫策略。那么对应的策略就是伪装一个请求头部。既然我已经拿到了我正常使用浏览器访问时的请求头内容，那么我就以它作为伪装。

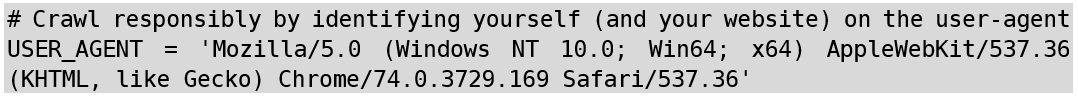
我参考 [<https://blog.csdn.net/mouday/article/details/80776030>] 这个网页。有4种方法可以伪装。

**方法1**，在settings.py里设置请求头部的“USER-AGENT”，此方法设置后该项目下的所有爬虫都使用修改后的默认头部。

在settings.py里找到以下2行内容

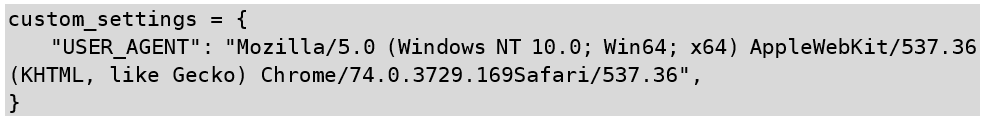


把第二行的注释取消，并且等号右边的内容替换成之前浏览器访问的请求头部的对应内容。



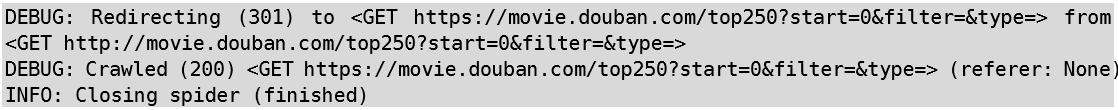
**方法2**，单独在爬虫里修改，此方法修改后仅对该爬虫起作用。

打开需要修改的爬虫的.py文件，在name后面加上custom\_settings



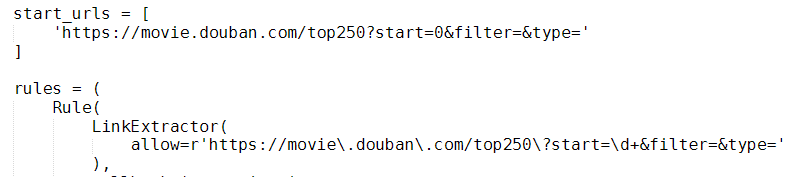
还有 {方法3: 链接设置} 和 {方法4: 中间件设置} 在此我就不去试验了。无论方法1还是方法2，我重新运行程序后得到的返回内容的User-Agent部分都与我用浏览器正常访问是相同的了。

这时我们重新爬取豆瓣电影Top 250的内容，执行entry\_point.py，返回内容包括



可以看到这时候返回代码已经不是403了，而是200，表示成功返回。

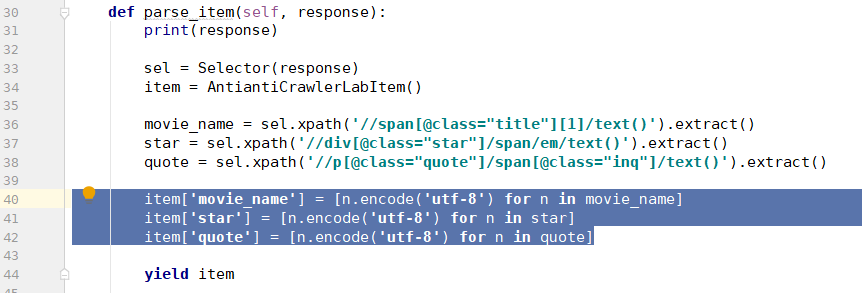
接下来继续探索。我们发现，前面的爬虫请求的返回代码是301 (Moved Permanently)，再看具体内容得知，豆瓣现在已经把所有HTTP访问重定向到了HTTPS，因此我们在爬虫对应的.py文件里修改URL。把start\_urls和rule里面的http改为https。



然后重新运行，得到以下输出



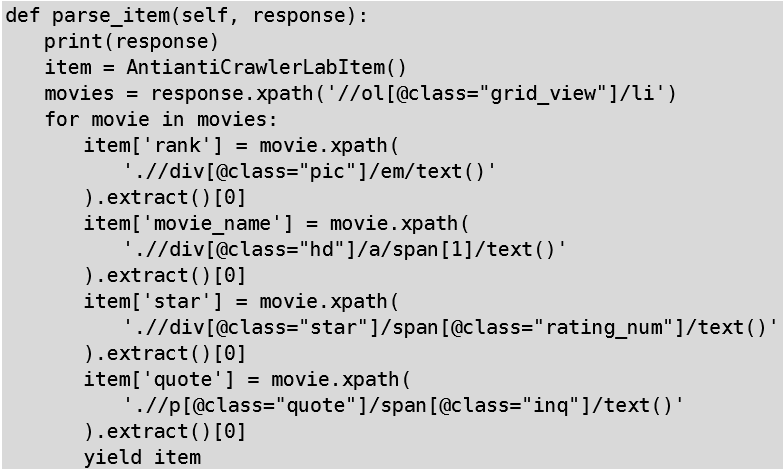
可以看到我们成功爬取到了内容。不过这里面显示的都是bytes，是因为我们的douban\_top250\_spider.py里面有这样三行代码



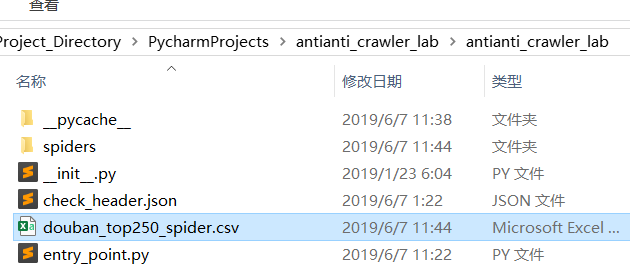
于是我们修改一下，让它能正确显示。并且，为了方便查看，我们修改一下需要爬取的内容以及输出位置。在items.py中，修改为如下内容



然后在执行的时候加上“-o douban\_top250\_spider.csv”表示我们需要把输出写入一个csv文件里。在douban\_top250\_spider.py中，把parse\_item函数修改为如下内容



然后运行爬虫，在项目目录下发现了douban\_top250\_spider.csv文件，打开后可以看到爬取的内容。（注：csv文件保存的时候使用的是UTF8编码，直接用Excel打开可能会显示乱码，用文本编辑器打开然后另存为ANSI编码后即可用Excel打开）

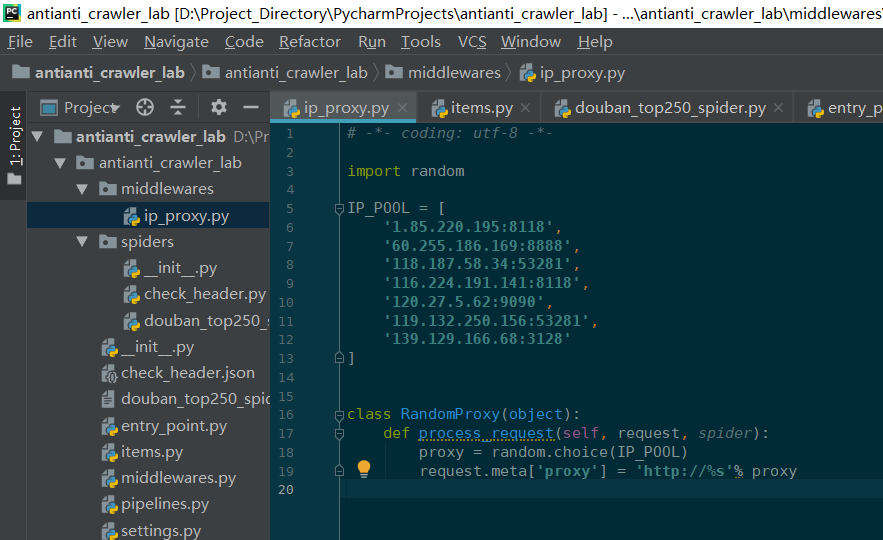




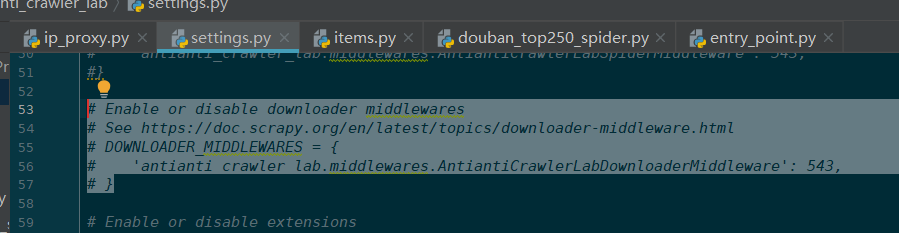
## 2.3. IP代理池

接下来尝试IP池与限制访问频率的策略。有很多网站的反爬虫策略是定期查看访问记录，如果发现某一个IP在短时间内高频率访问的话，就判定这个IP的访问是爬虫发起的，于是就在一定时间内封锁这个IP的访问。因此，对应的反反爬虫策略就有2个，第一是最简单的降低单个IP访问的频率，第二是使用一个IP代理池，每次访问都从IP代理池里取出一个IP，这样每次访问的源头IP就是不同的（严格来说，如果是随机取出，有可能取到同一个IP，但概率非常小）。

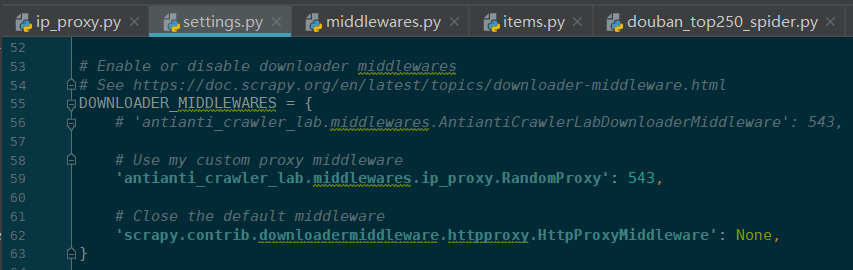
使用Scrapy做爬虫时，所有访问都会通过中间件，因此我们可以在中间件做设置，使得每次访问下载的时候的IP是从IP池里取出来的。首先我们在爬虫的工程目录里新建一个`middlewares`目录，然后新建一个`ip\_proxy.py`文件，然后我们人工输入一些备用IP到一个名为`IP\_POOL`的列表里。再然后我们定义一个`RandomProxy`类，可以每次从IP池里随机取出一个IP。最终的`ip\_proxy.py`文件内容如下



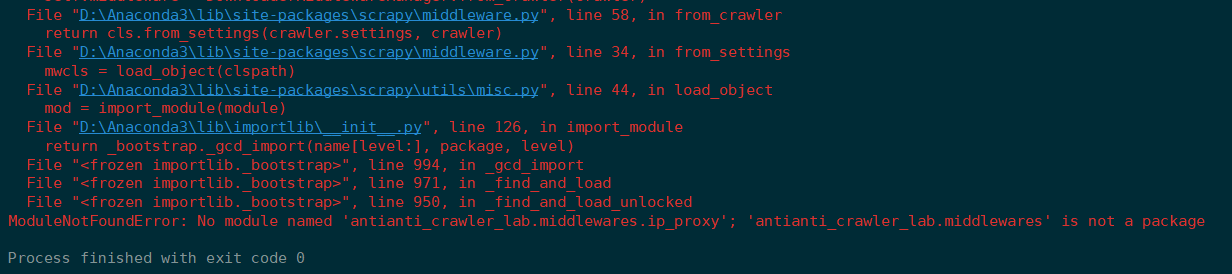
接下来我们就要在`settings.py`里设定中间件了。默认情况下，settings.py里面下载器中间件的设置是被注释掉的。



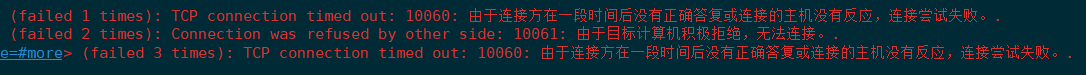
我们修改成如下

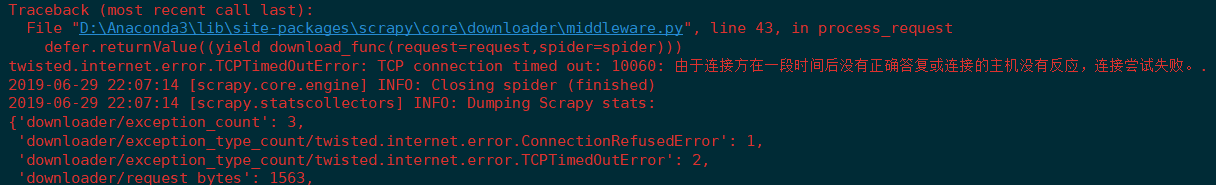


然后运行。发现如下错误



原因是我一开始新建的middlewares目录下没有`\_\_init\_\_.py`文件，于是我创建一个空白的`\_\_init\_\_.py`文件，重新运行就可以了。但是运行后又出现了以下内容



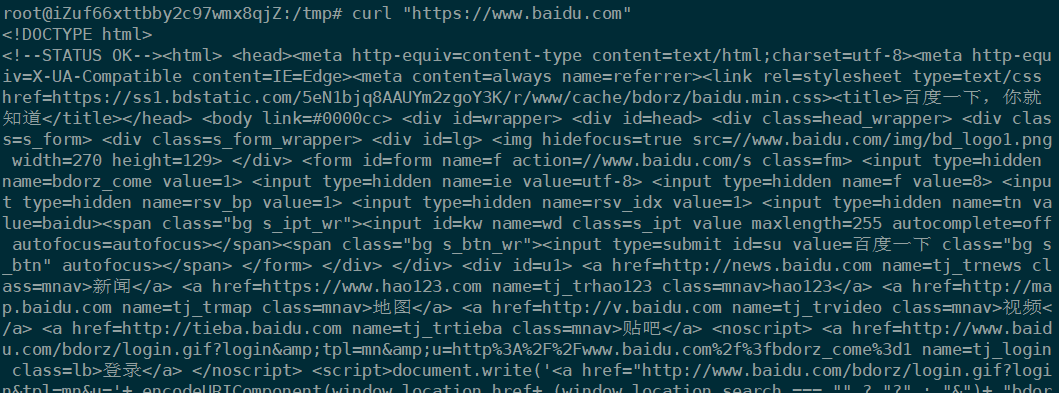


这是因为我最开始设置的列表`ip\_pool`里面的IP地址是随便在网上找到的，并没有验证过有效性，而我现在使用的时候这些IP地址全都已经失效了。

我们可以用一个最简单的方法检验IP是否失效。首先在Linux环境中执行

curl "https://www.baidu.com"

会得到以下内容

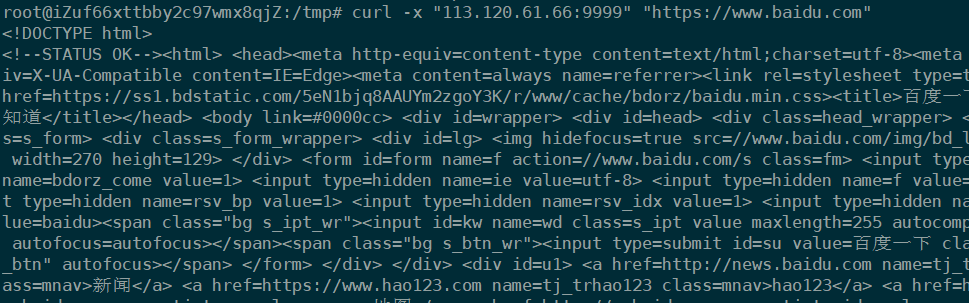


说明在不使用代理的时候，自己正在使用的IP是可用的。接下来我们执行

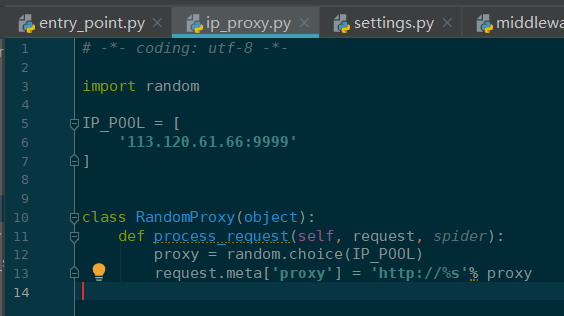
curl -x "1.85.220.195:8118" "https://www.baidu.com"

“-x”选项是表明使用代理，后面跟着的就是代理的IP地址，上面的指令中的代理IP是我们前面`ip\_pool`列表里的第一个IP地址。一直没有输出，说明这个代理IP已经失效了，其他的IP地址同理可以验证是失效的。

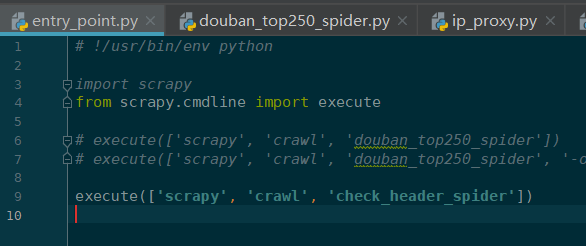
我在 [ <https://www.xicidaili.com/> ] 上面找到一个有效的IP代理地址。

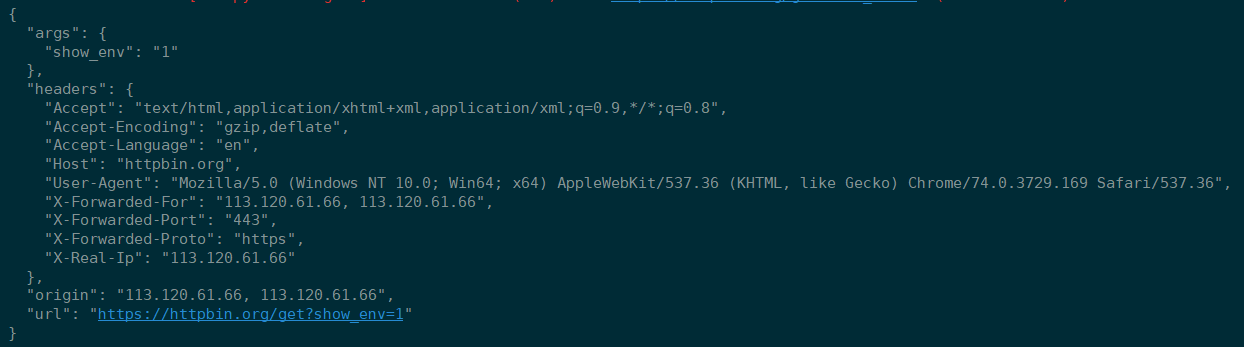


把它放到之前的`ip\_pool`列表里，然后删除之前失效的IP地址



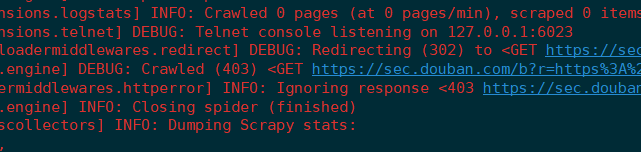
我们修改entry\_point.py，爬取之前能够查看请求头部的网站。然后运行。



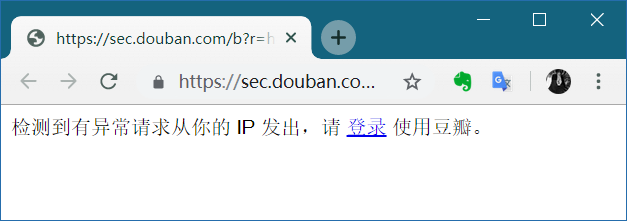


可以看到，现在我的IP已经变成了我所使用的代理IP。

重新运行爬虫爬取豆瓣网站，出现了一下错误。从输出中可以看到，请求首先被重定向到了另一个URL，然后新的URL返回了403。这里说明我其实已经又触发了豆瓣的反爬虫机制。其原因可能是我找到的有效代理IP是公开的免费的，在我使用之前已经被很多人使用过了，导致豆瓣封锁了这个IP。



我在浏览器中打开爬虫输出结果中返回302的URL（<https://sec.douban.com/b?r=https%3A%2F%2Fmovie.douban.com%2Ftop250%3Fstart%3D0%26filter%3D%26type%3D>），在浏览其中可以看到以下内容

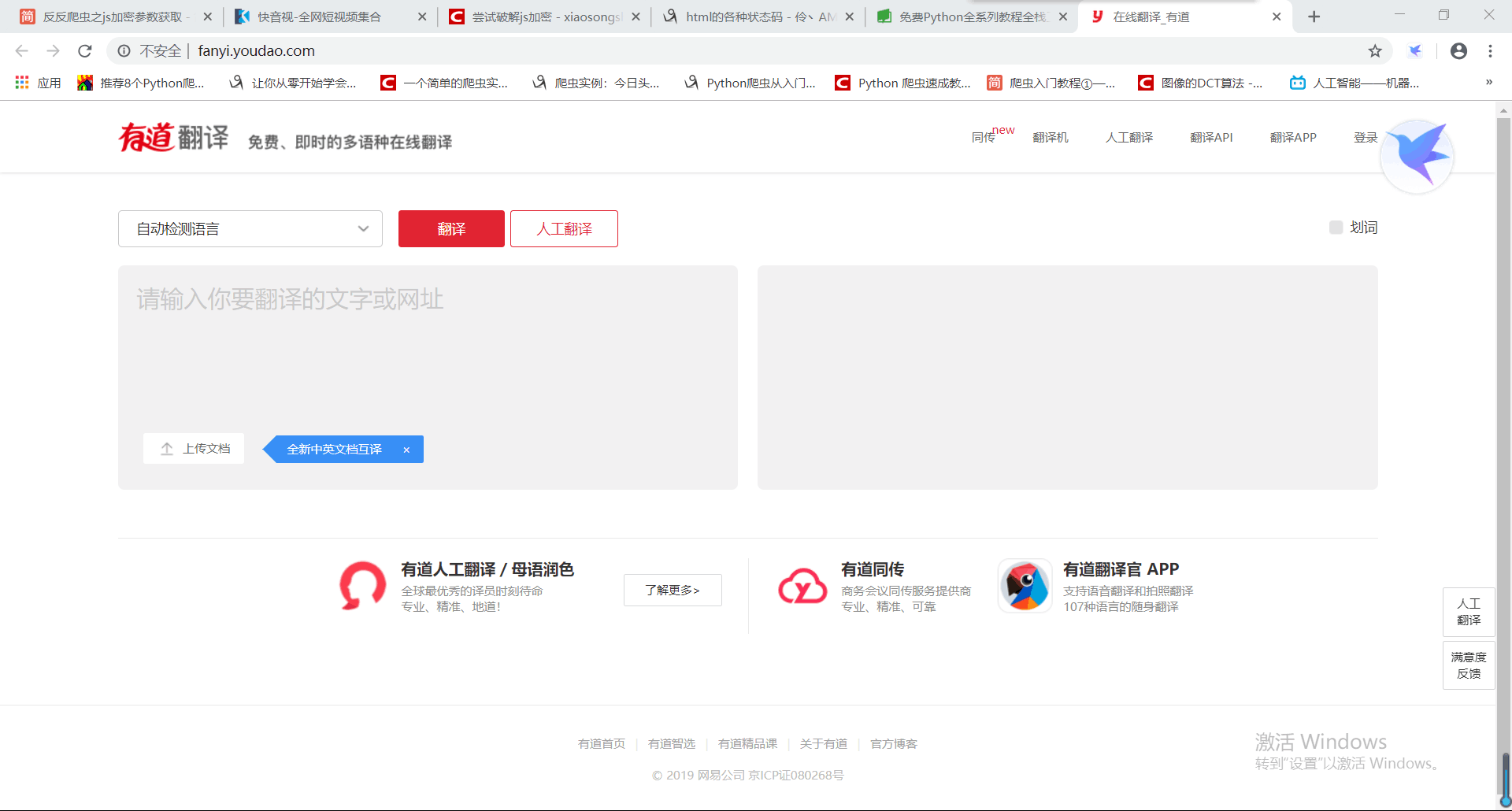


证明的确是这个IP被太多人过度使用了。由于优质的不被过度使用的IP需要在相关商家处付费获取，因此到这里我们并没有进行下去。但是我们已经熟悉了基于IP访问频率的反爬虫策略和对应的IP代理池反反爬虫策略。

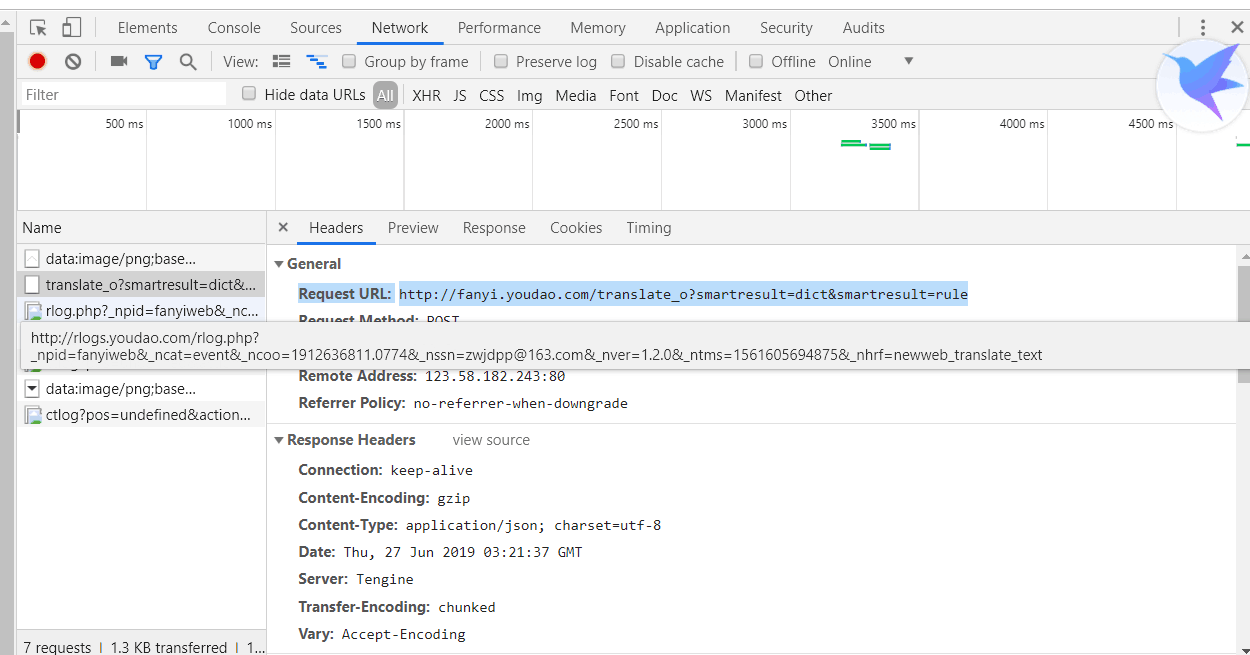
## 2.4. JS文件数据加密的破解

一些网站为了防止数据被爬，有时将重要的数据存放在JS文件中。同时，因为JS代码是要在浏览器中解释、执行的，无法做到绝对的保密，因此在反爬虫时，通常还会对JS代码或其中的数据运用加密或解密技术，以防止被爬虫获取。

针对JS文件数据加密技术，我们选择了有道翻译网页版进行反反爬虫操作的实现。以下是有道翻译网页版的网页界面：



在左边输入一串内容，网站将会在右边返回翻译的内容。很明显这是一个动态的页面。因此我们进入F12浏览器自带的抓包工具进行抓包。F12抓包，在完成一次翻译操作时，实际上的url地址与数据请求格式如下：

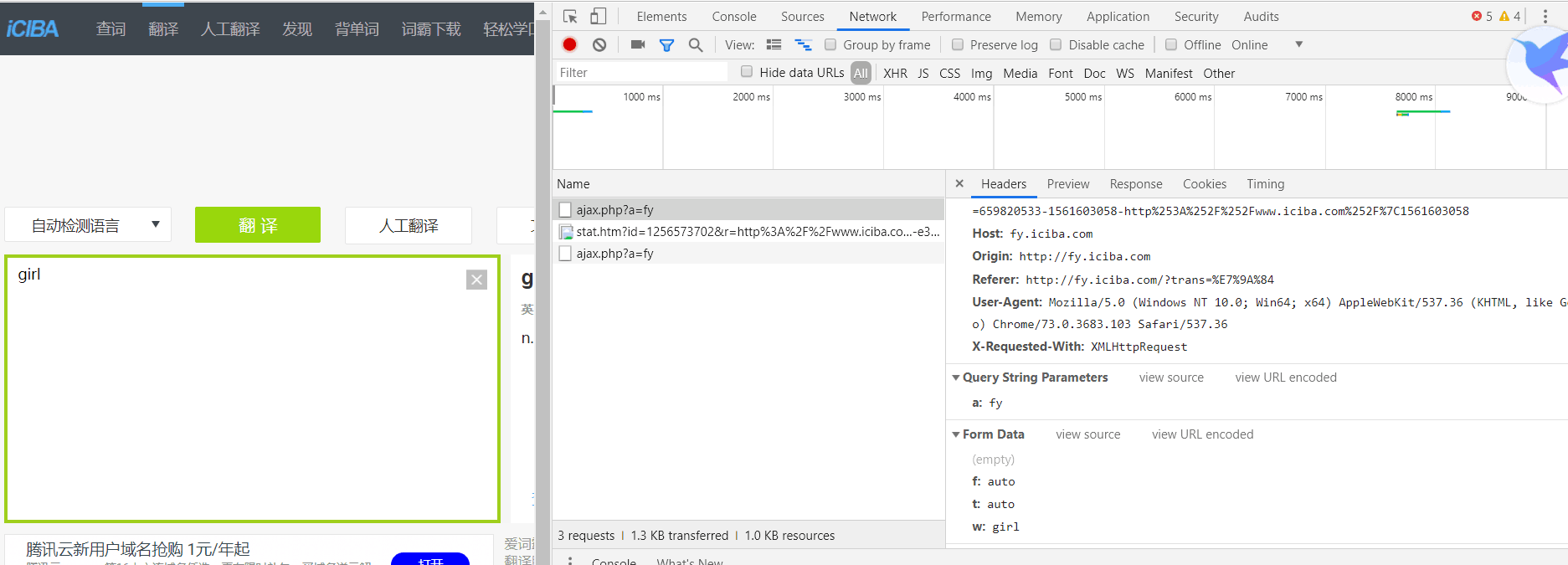


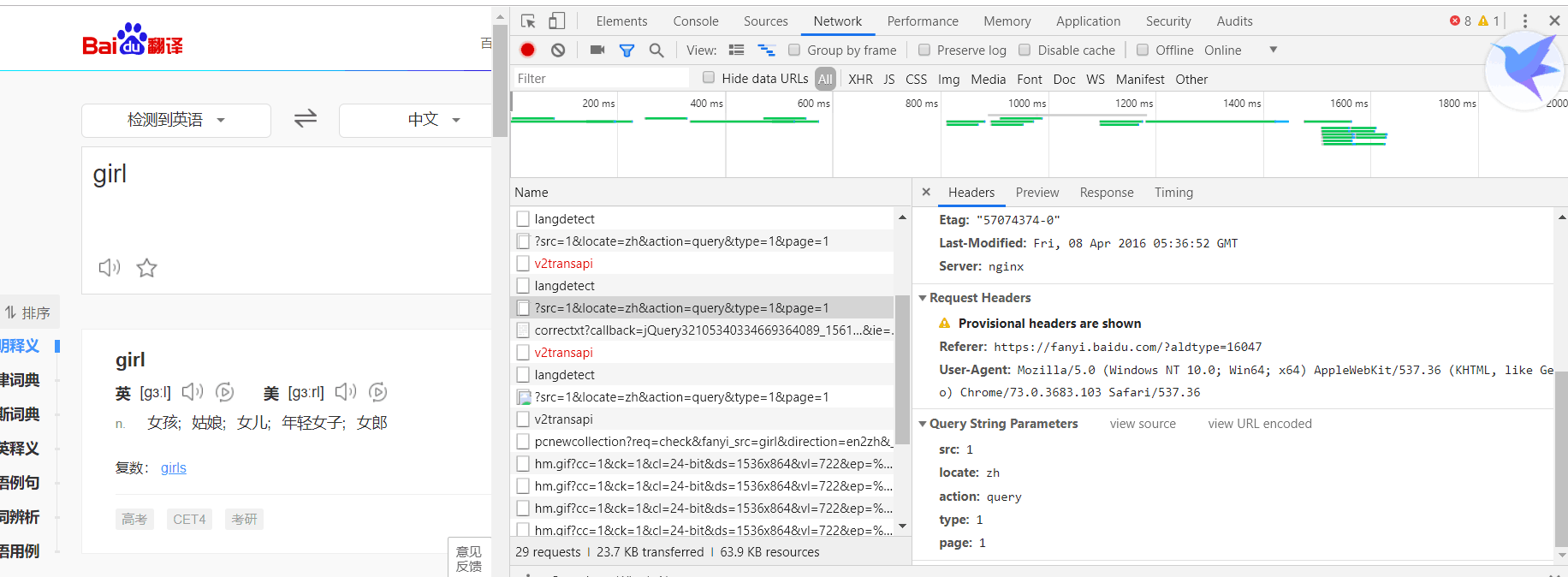
由此我们可以获取到实际的爬虫的url目标。我们根据获取到的实际url地址与数据请求格式修改爬虫，填充表头与数据表结构，避免基础反爬虫机制：



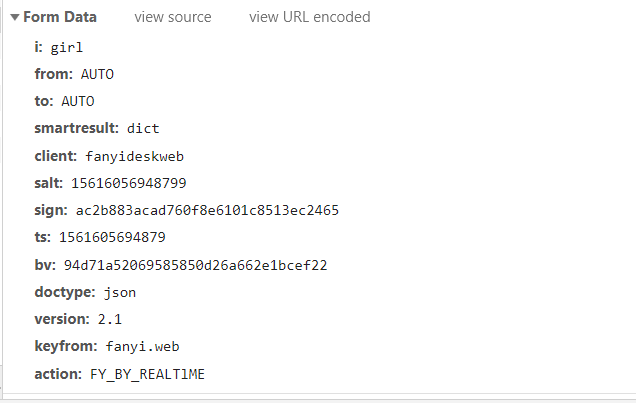
我们进行爬取，却得到了返回错误代码ErrorCode：50。

检查network数据，发现除去刚才操作的请求外，其余请求都是img申请，基本可以确认js加密发生在translate请求中。打开金山百度等翻译软件做一下数据表对比，同时查询有道数据表的各项含义。

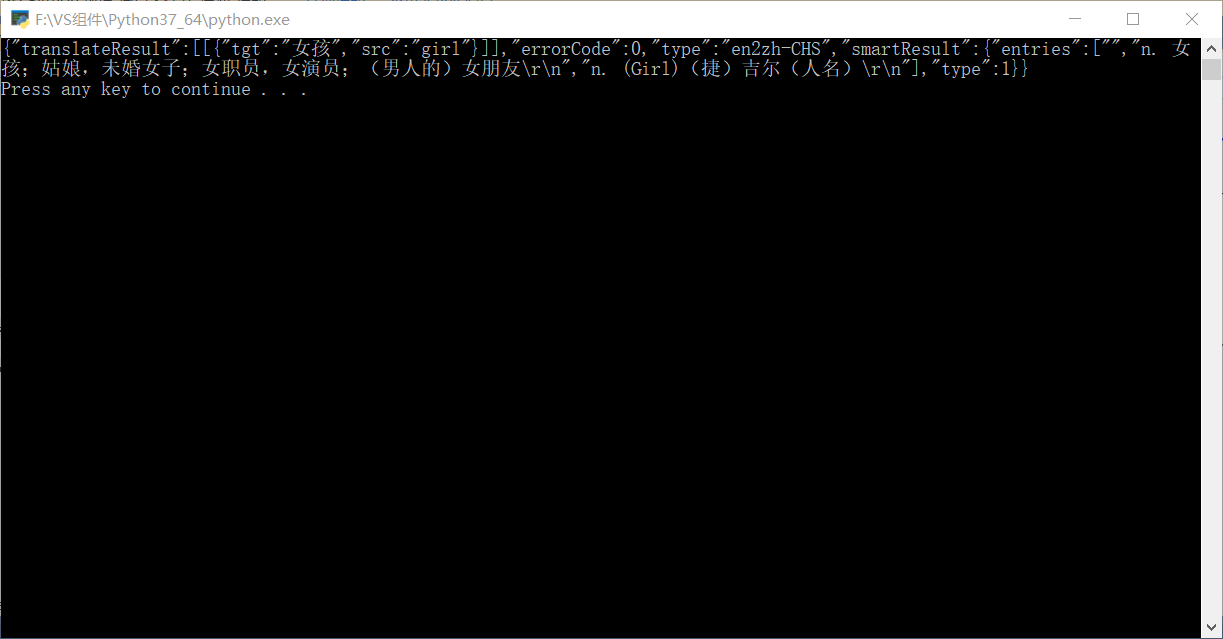




我们可以看到数据表：

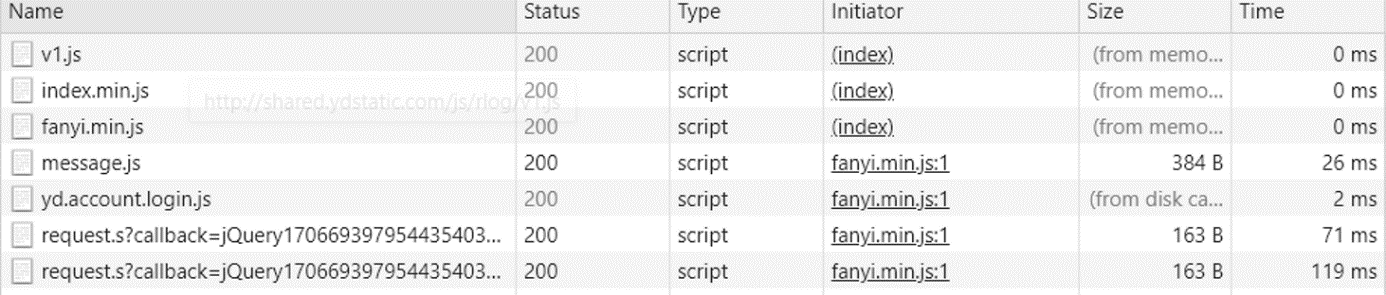


同时我们上网查询有道词典数据表的各项含义得知：i, from, to, version, doctype, action, kerfrom等均为常见结构。Salt，sign，ts，bv字段为主要搜索对象，尝试依次注释以上字段，尝试寻找问题字段，发现当ts值被注释时程序反而能正确运行，ts的值可能只是在数据表结构中，但并没有实际回传。尝试运行结果如下：

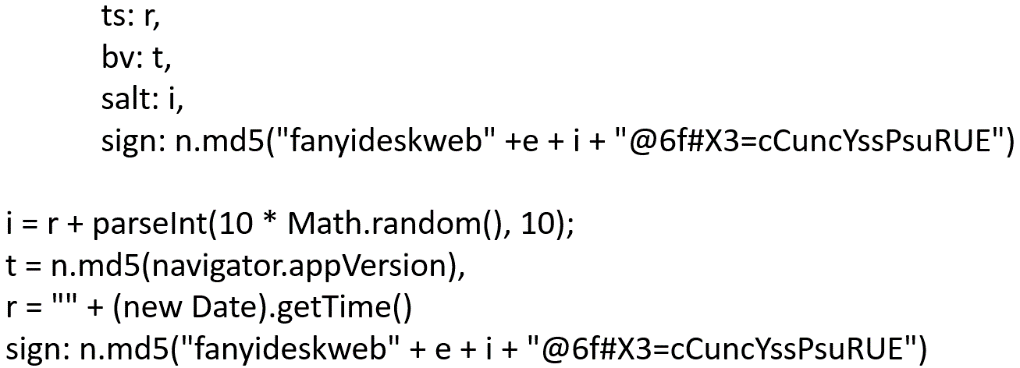


可以看到，我们本次的爬取是成功的。但是当我们将翻译的内容改为student后运行，却仍然得到了ErrorCode:50，也就是说反爬虫机制在此启动。

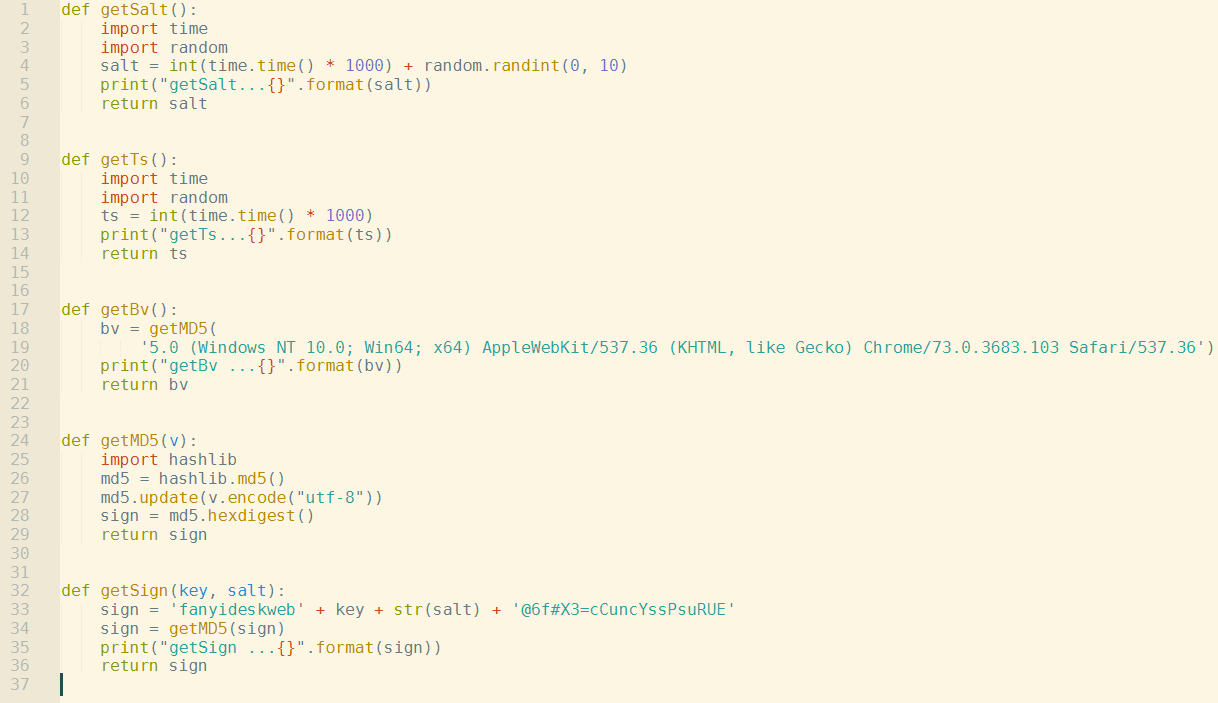
由于网页中只有js文件是可以进行数据加密的，于是我们刷新页面，再次查看js文件的加载情况：



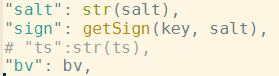
对这五个js文件进行格式化，并搜索刚才列为可疑范围的四个数据表字段，发现在fanyi.min.js中可以发现以上字段的加密代码：



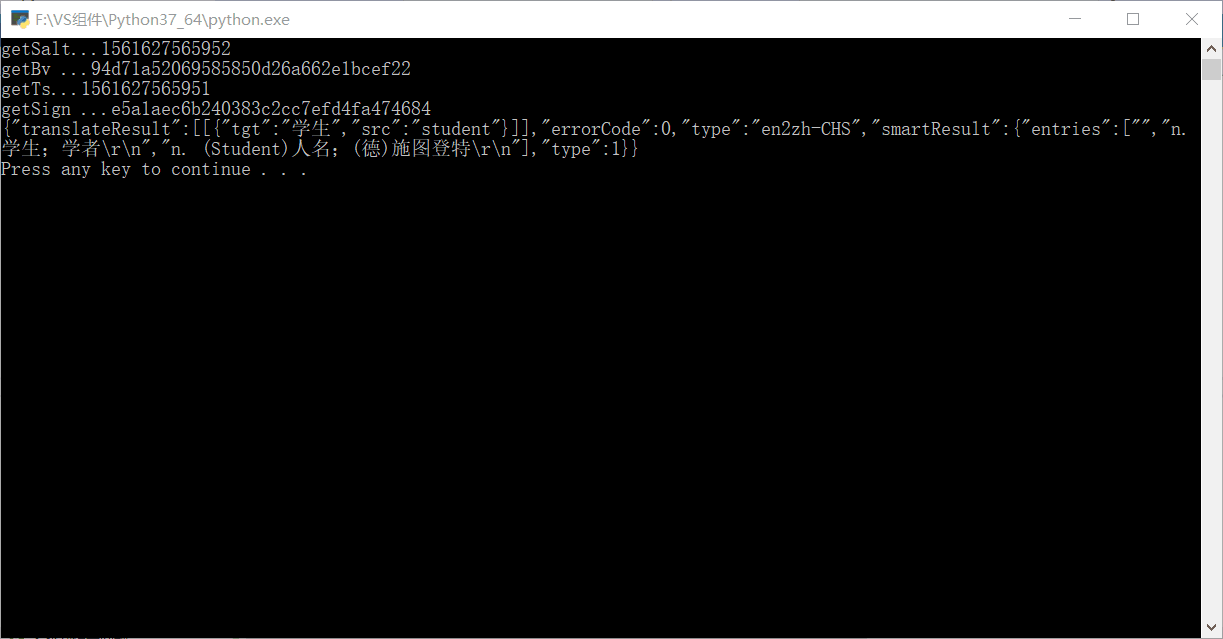
因此我们根据得到的JS代码，仿制Python代码，进行模仿加密：



将数据表结构替换为仿造js加密的python代码：



然后运行



到此，对有道翻译网页版进行的针对JS文件数据加密的反反爬虫技术实现成功。

## 2.5. Ajax加密破解

Ajax是asynchronous javascript and XML的简写，中文翻译是异步的javascript和XML，这一技术能够向服务器请求额外的数据而无须卸载页面，会带来更好的用户体验。虽然名字中包含XML，但ajax通信与数据格式无关。

作为前端，数据提交到后台之前，重要的数据要进行加密一下，虽然已经有 https 等技术，但是增加一道前端的加密还是相对更安全的。虽然，前端的加密很容易破解，但是有总比没有强。尤其是涉及到用户名和密码，最好加密后再进行发送 ajax 请求。

Ajax包括以下几步骤：1、创建AJAX对象；2、发出HTTP请求；3、接收服务器传回的数据；4、更新网页数据。概括起来，就是一句话，ajax通过原生的XMLHttpRequest对象发出HTTP请求，得到服务器返回的数据后，再进行处理。

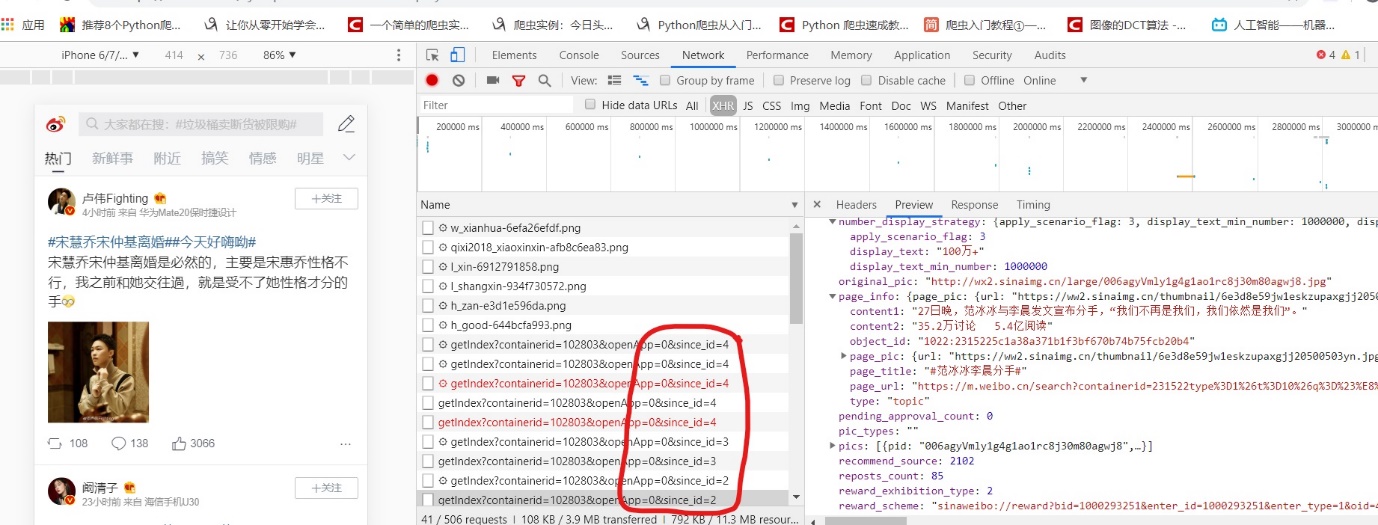
作为前端，数据提交到后台之前，重要的数据要进行加密一下，虽然已经有 https 等技术，但是增加一道前端的加密还是相对更安全的。虽然，前端的加密很容易破解，但是有总比没有强。尤其是涉及到用户名和密码，最好加密后再进行发送 ajax 请求。

所谓ajax反爬机制即为，当网页刚加载时，不会加载网页上的所有数据，必须伴随用户的操作数据实现实时加载，才能完成爬虫。

也就是说对于ajax的反反爬虫操作是两种模式，一种是完全模拟用户操作，一种是跳过操作部分，直接调用加载功能。

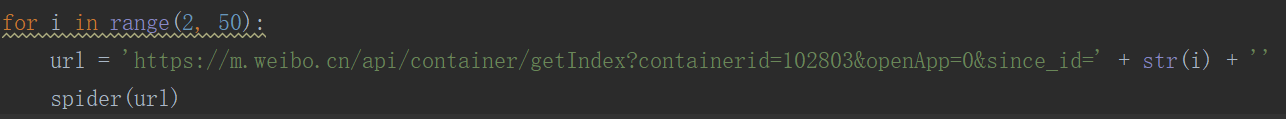
针对ajax加密反爬虫技术，我们分别对两种反反爬虫技术进行了两个实战实例，分别是使用跳过操作部分，直接调用加载功能的反反爬虫技术爬取微博热搜和使用selenium库以及chromedriiver插件来模仿用户操作爬取京东商品栏。

首先是微博热搜的爬取。我们在完成了robots.txt以及HTTP请求头部的修改后直接地去爬取，结果是不出所料地失败了，在当前的静态页面我们什么都抓不到。打开F12抓包工具network进行分析，搜索XHR请求，发现一个明显的标识符since\_id，同时我们滑动页面，会发现since\_id的值渐增，且在此请求中可以发现网页的内容。

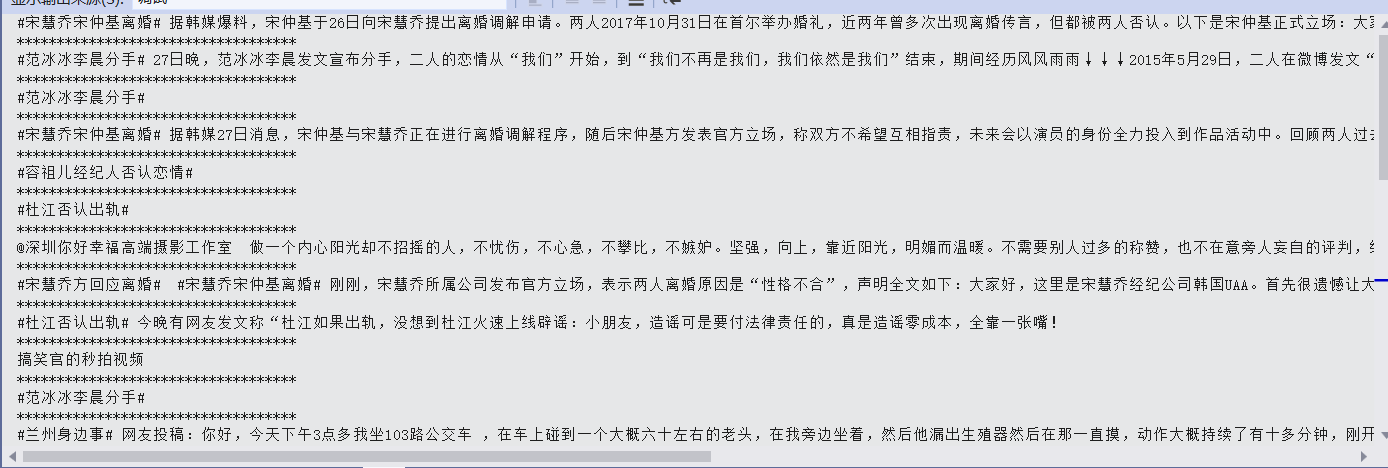


显而易见，只要我们能控制url地址最后的since\_id 就能攻破微博的反爬虫。

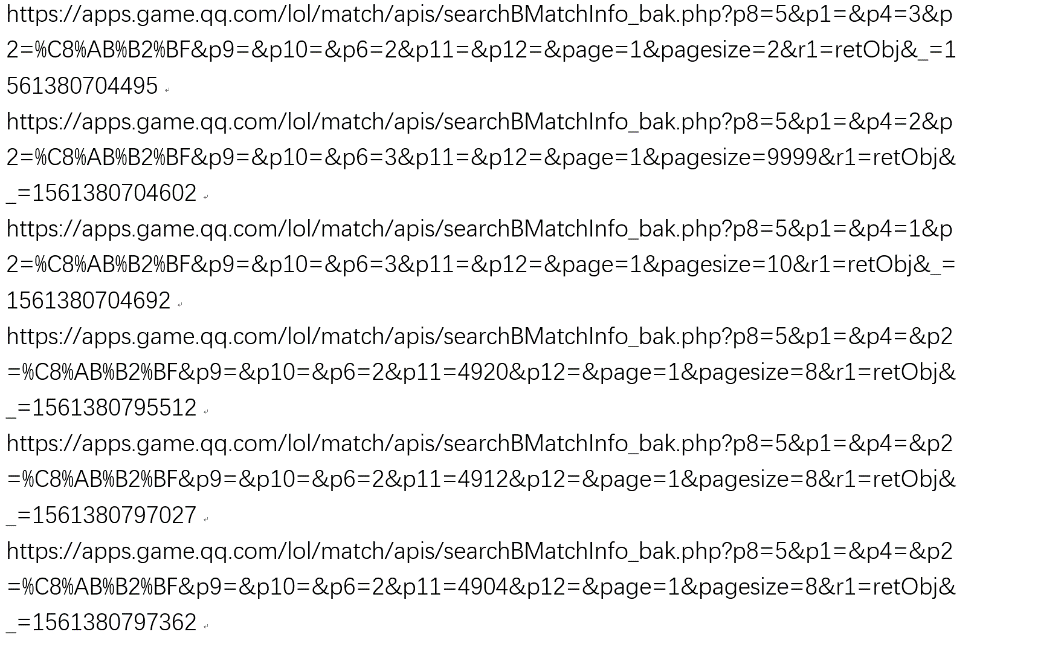
因此我们现在既然知道了微博动态ajax获取JS文件的格式，我们便能直接地通过我们已知的地址去获取到所需要的数据，如下图：



运行后就可以得到我们想要的数据啦！



这其实还算是一个相对较为简单的反反爬虫技巧，因为微博对于js文件的返回标识特别清晰，用了since\_id来标识，一目了然。我们之前在爬取腾讯的lpl官网的赛程时，返回的js地址几乎都是加密过的，如下图：

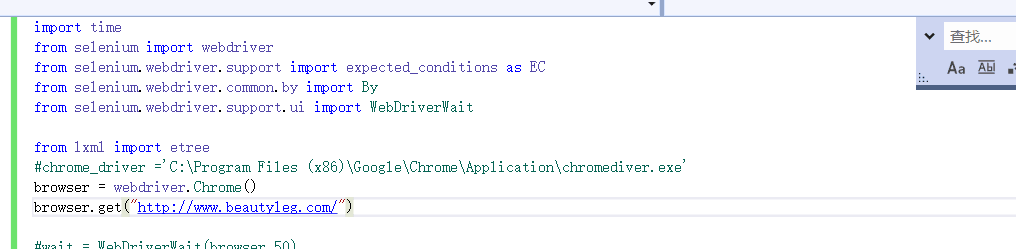


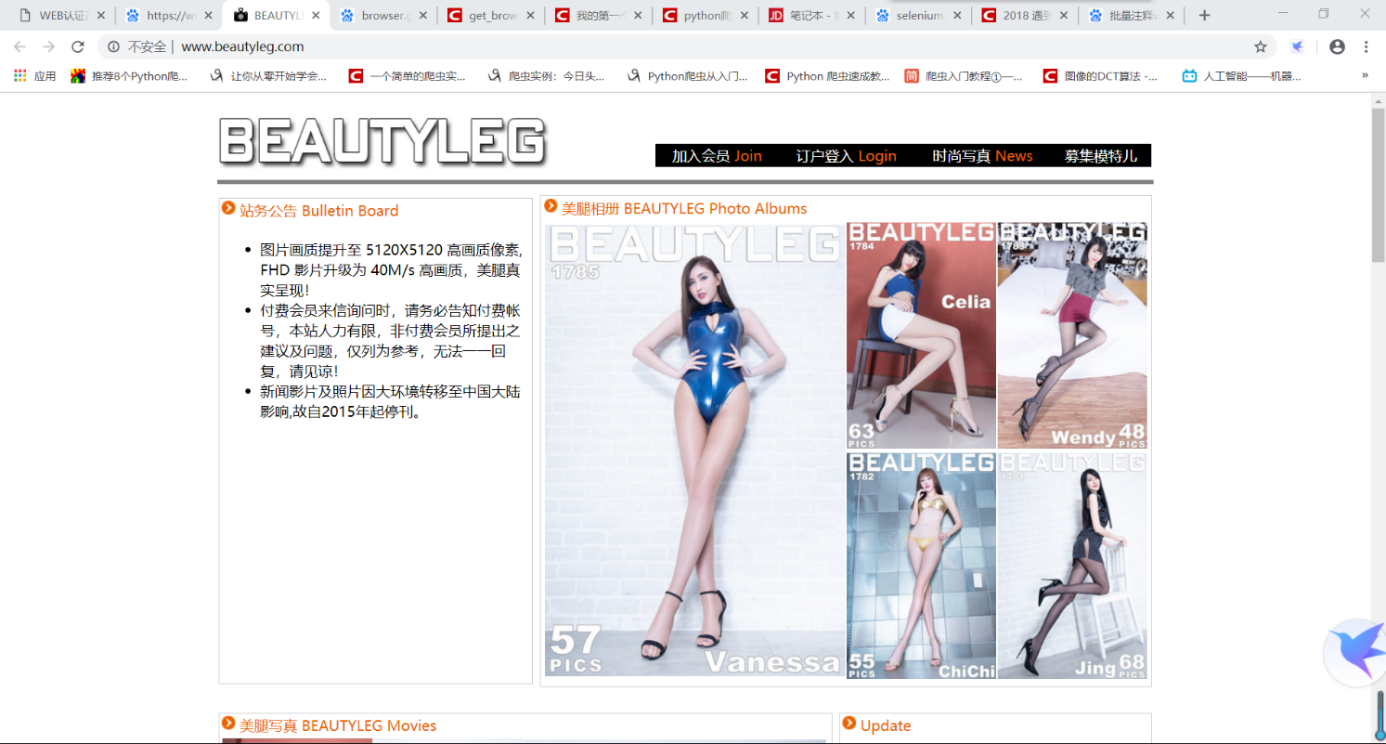
因此我们在破解时完全没有头绪，于是便放弃了使用跳过用户操作来爬取该网站的想法。

我们对于ajax动态加密做的第二个实例是使用selenium库结合模拟用户操作的方法获取js文件。

首先注意：selenium导入chromedriver路径语句会失效，必须得把chromedriver放在和爬虫脚本同路径下。

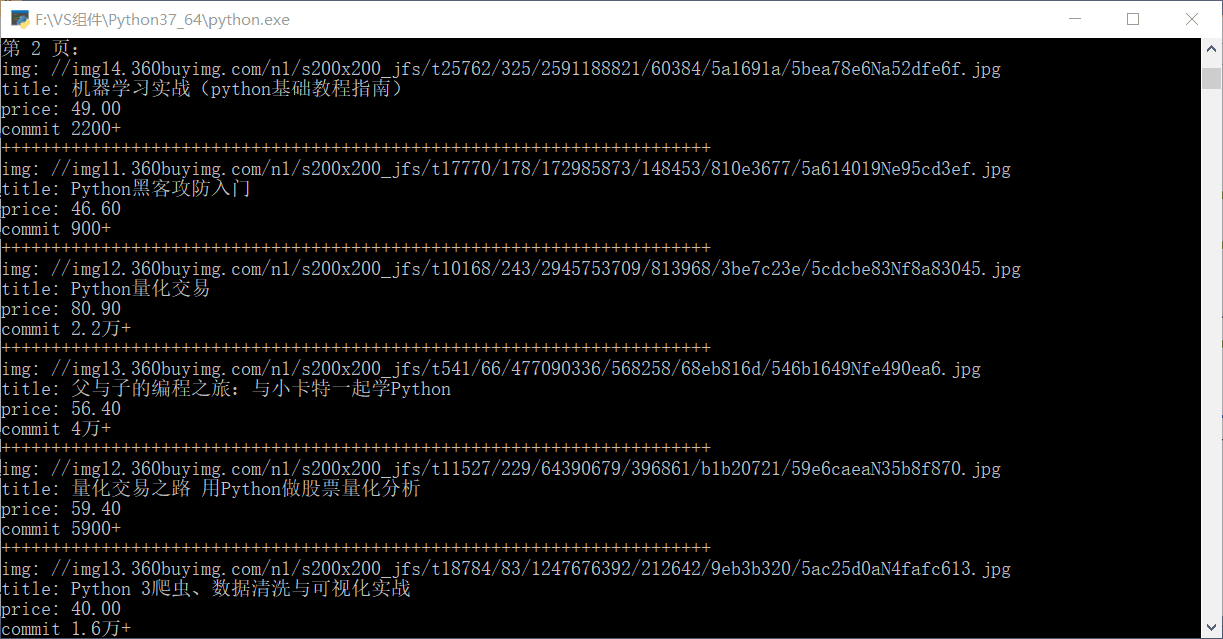
我们通过打开[www.beautyleg.com](http://www.beautyleg.com网站来测试一下selenium/) 网站来测试一下selenium库与chromedriver插件的协同工作效果，如下：

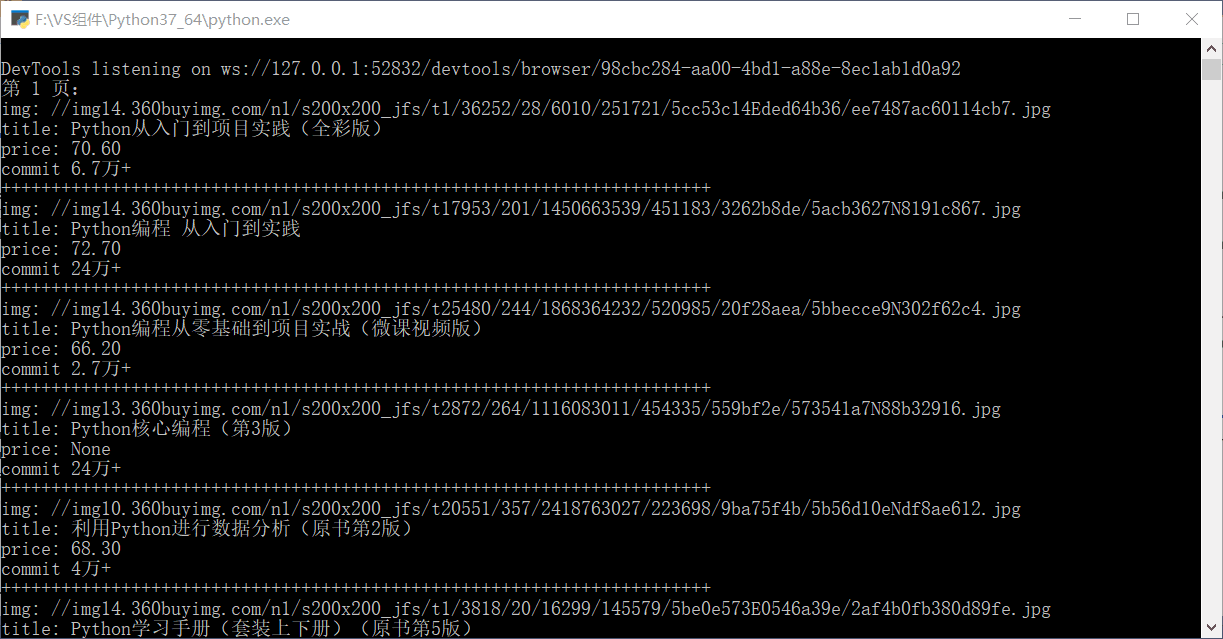




接下来我们对我们的目标：京东，发起冲击，目标任务是爬取京东商品搜索‘刘亦菲写真’的所有结果，包括商品名和价格。

爬取京东需要模拟三个操作，输入查询物品，滚动页面，翻页，模拟之后解析html得到如下结果。

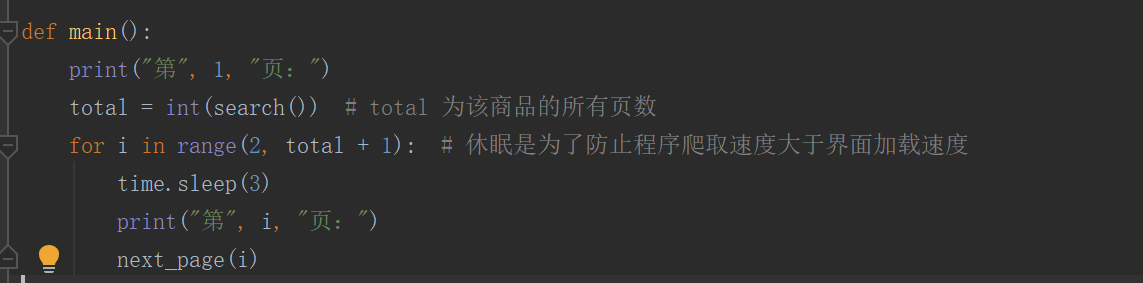




PS：在此，我们本来是想做一个通用爬虫，但是发现商品列表里，不同商品分区的html结构不一样，所以就只能复现教程操作了。

现在的反爬虫还有一项技术是：访问频率限制。作为一只网络爬虫，在如今的大数据时代，爬取数据的速度与效率是非常关键的。因此，爬虫在爬取数据时通常会达到一个真实人类浏览网页时所不可能达到的频率。同时，同一时间过量的访问量也有可能造成网站崩溃等异常。因此，网站安全负责人在反爬虫时，可采取对异常ip访问频率进行拉黑ip等操作等，以进行反爬虫操作。

针对此反爬虫手段进行反反爬虫的手段之一，是设置sleep()。但同时也应该注意，在sleep等待的时间与爬虫软件需求之间作一个权衡，选择一个合适的等待时间。本次几个实例在爬取数据时都没有达到网站限制的访问频率与访问量，因此没有触发此反爬虫机制。但我们在实例②的源代码中加入了time.sleep()以应对可能出现的针对访问频率限制的反爬虫机制，如下图：



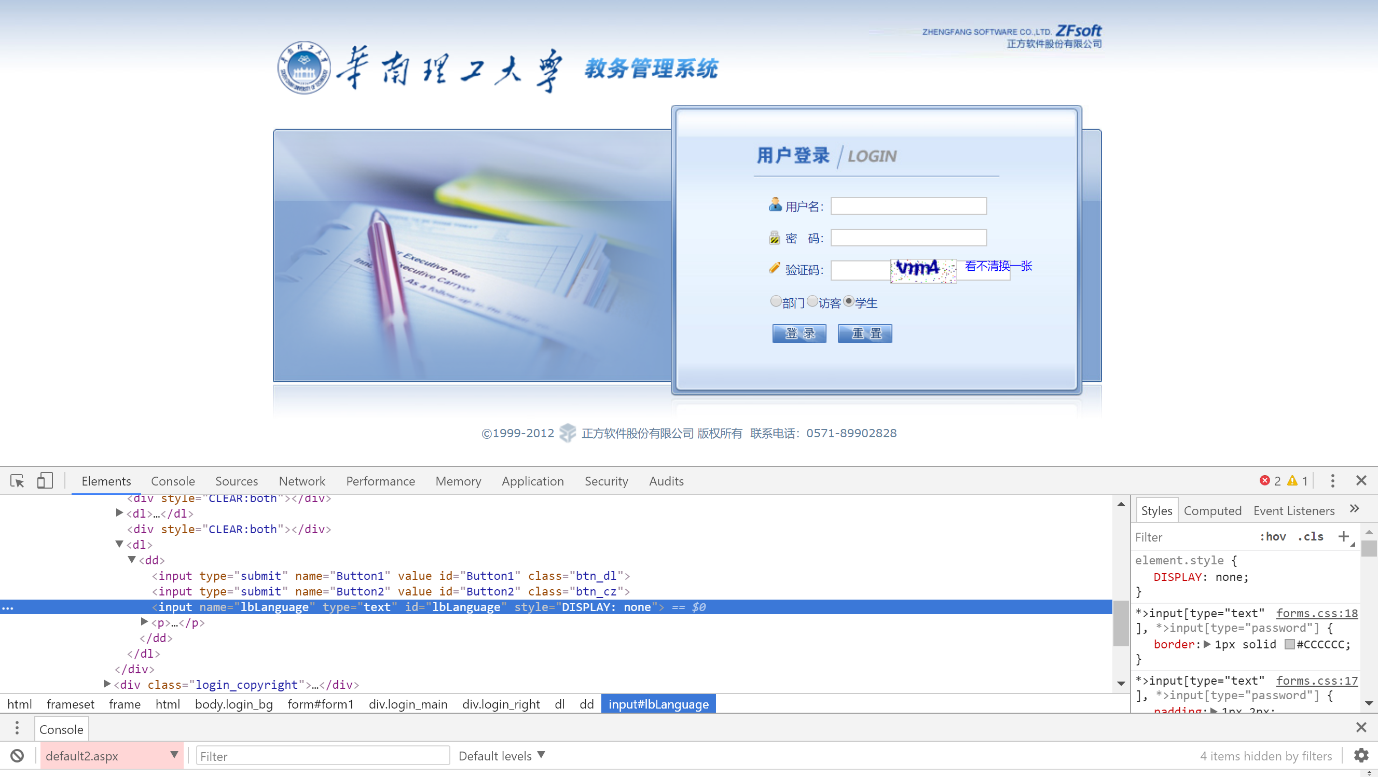
## 2.6. 蜜罐技术

蜜罐技术本质上是一种对攻击方进行欺骗的技术，通过布置一些作为诱饵的主机、网络服务或者信息，诱使攻击方对它们实施攻击，从而可以对攻击行为进行捕获和分析，了解攻击方所使用的工具与方法，推测攻击意图和动机，能够让防御方清晰地了解他们所面对的安全威胁，并通过技术和管理手段来增强实际系统的安全防护能力。

蜜罐好比是情报收集系统。蜜罐好像是故意让人攻击的目标，引诱黑客前来攻击。所以攻击者入侵后，你就可以知道他是如何得逞的，随时了解针对服务器发动的最新的攻击和漏洞。还可以通过窃听黑客之间的联系，收集黑客所用的种种工具，并且掌握他们的社交网络。

关于蜜罐技术，因为这已经属于比较高难度的层次了，因此我们小组没有进行对应的反反爬虫的实战实例。但是我们发现，在我们华工的新版与老版的教务系统中，都使用了蜜罐技术，因此虽然我们没有通过代码进行实践，但是我们可以去分析从而学习了解相关的内容。

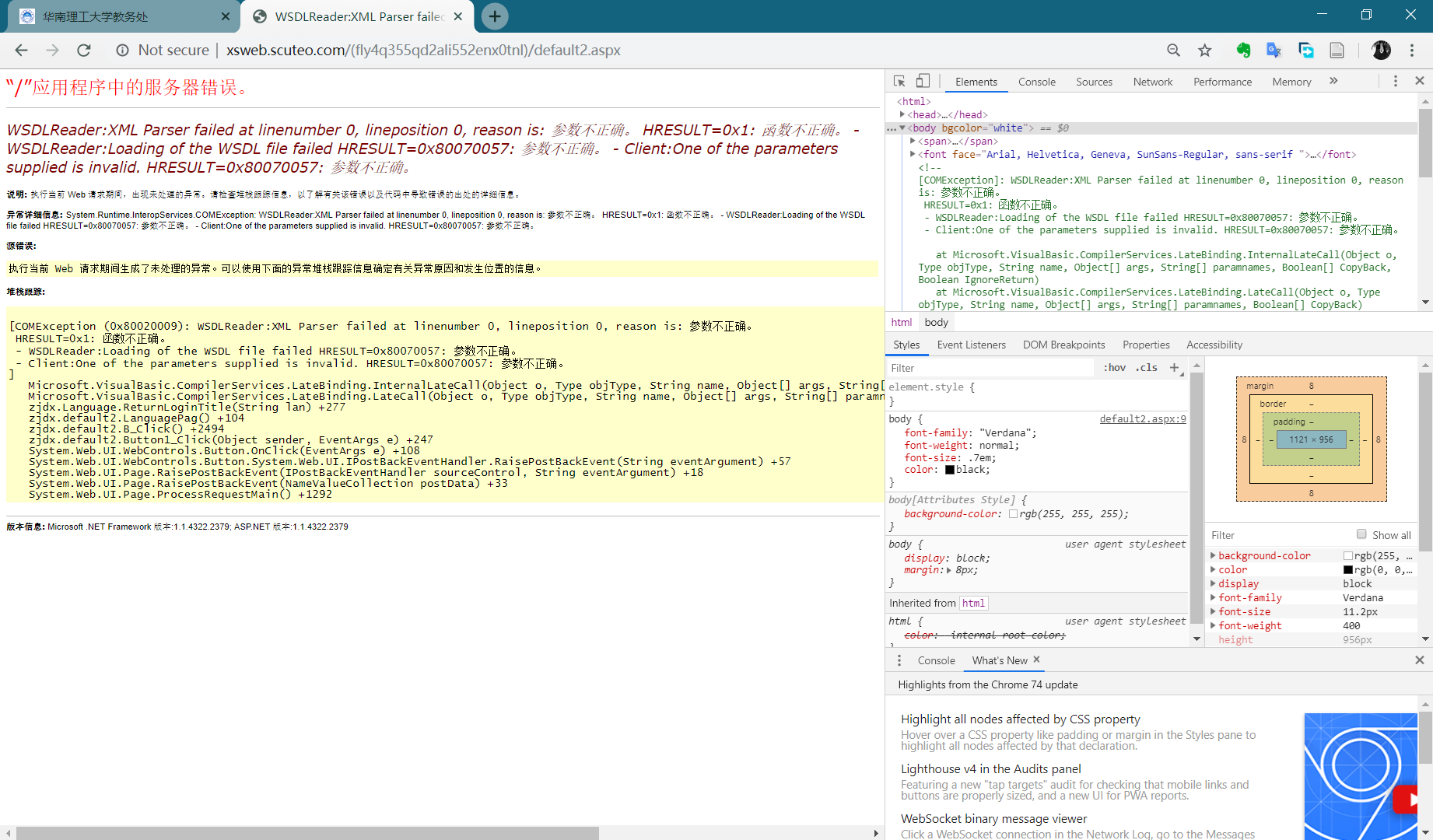
我们打开华工的教务管理系统，人类用户看到的就只有3个输入框和3个单选项目以及两个按钮。但其实页面中可交互的元素不止这些。

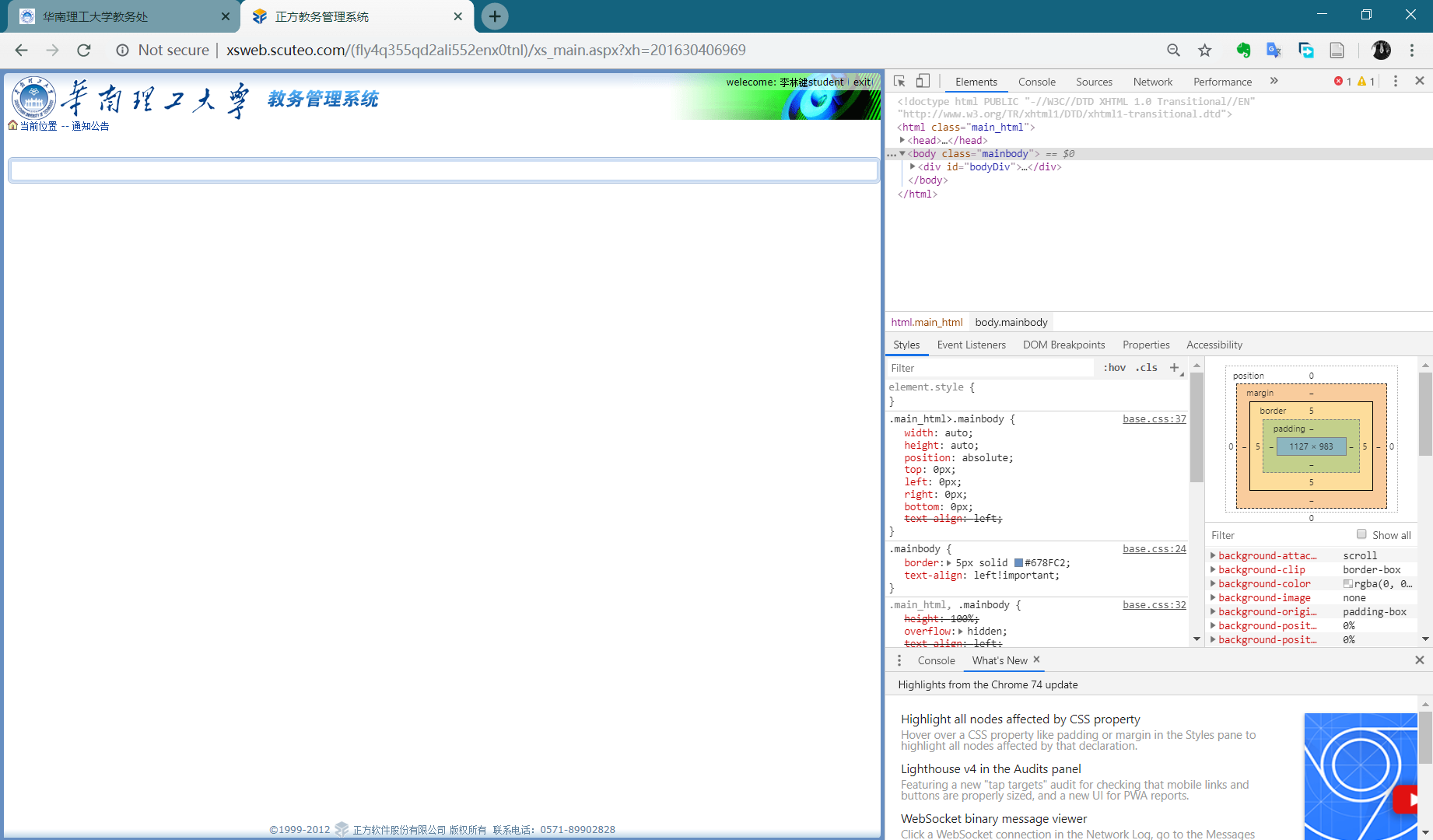




当我们在浏览器中检查页面元素时，我们发现有一些“<input>”的HTML标签的属性被设置为“DISPLAY: none”或者是“hidden”，浏览器在渲染的时候就不会让这些元素显示出来，因此正常人类用户是不知道它们的存在的，也就是看不见。然而爬虫的程序是可以看见这些内容的，程序在爬取数据的过程中就有可能在这些输入框里输入内容。

我们现在模仿爬虫的行为，看看会有什么结果。在我们将display=none取消，然后在原本不显示的输入框里输入一些内容后，即使我们在账号与密码栏输入了正确的数据，我们也确实“成功登陆”了。但是登陆的实际是一个如下图所示的“蜜罐”，我们在其中无法进行任何有效的操作或者获得任何有用的数据。





对于蜜罐技术我们由于时间原因没有进行反反爬虫技术的研究，因为对于教务系统的爬取还需要结合识别验证码等技术，对于都是小白的我们小组难度较大，目前在网上能查找到的对于蜜罐技术的反反爬虫也相对较少，我们小组在讨论之后认为可以将对应的反反爬虫策略的注意点集中在display=“none”(type=“hidden”)这些隐藏标签的识别上。但是蜜罐技术可不仅仅局限于此，因此之后遇到其他的蜜罐，也只能再具体问题具体分析了。

# 总结

反反爬虫是现在互联网大数据时代十分重要的一项技术。虽然本次课程设计的主题是“反反爬虫”，但是通过本次的课程设计，我们小组的成员对于爬虫、反爬虫、反反爬虫这三个方面的内容都有了一定深入的理解，并且掌握了一定的相关知识与技能，收获颇丰。

正如之前所说，由于现在HTTP通信机制，网站的设计者与安全维护人员需要在真实用户与爬虫之间划清一条明显的界限是十分困难的，而这也让我们这些“爬虫”，或者说，我们这些反反爬虫的人，有可趁之机。截至目前，还没有能完全杜绝爬虫的反爬虫机制出现。每当有新的反爬虫机制出现，就会有相应的反反爬虫技术出现。正所谓，“道高一尺，魔高一丈”，这场网络数据攻防战，任重而道远。

# 参考文献

[1] <https://zhuanlan.zhihu.com/p/31943138>

[2] <https://blog.csdn.net/mouday/article/details/80776030>

[3] <https://zhuanlan.zhihu.com/p/29733174>

[4] <https://study.163.com/course/courseLearn.htm?courseId=1004987028&from=study#/learn/video?lesso>nId=1052106893&courseId=1004987028

[5] https://www.jianshu.com/p/93d1e0df45a0

[6] <https://blog.csdn.net/qq_37393071/article/details/81409976#%E4%B8%80%E3%80%81%E4%BB%A>3%E7%A0%81%E7%9B%AE%E7%9A%84