注：报告中有些代码块的显示可能会有格式错误，因为我是直接复制过来的，单行字符数过多时MS Word会自动调整格式，导致看起来很混乱。在MS Word打开此报告，直接复制代码块到文本编辑器或IDE中就能更好地显示。

操作系统：Windows 10

IDE：PyCharm 2018.3.4

爬虫框架：Scrapy 1.5.2

环境：Python 3.6.5

本次实验我打算首先开始尝试爬取豆瓣电影Top 250的内容。

执行如下指令新建Scrapy项目

scrapy startproject antianti\_crawler\_lab

然后创建(若原本不存在)并编辑以下文件

items.py

import scrapy

class AntiantiCrawlerLabItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

movie\_name = scrapy.Field()

star = scrapy.Field()

quote = scrapy.Field()

entry\_point.py

# !/usr/bin/env python

import scrapy

from scrapy.cmdline import execute

execute(['scrapy', 'crawl', 'douban\_top250\_spider'])

douban\_top250\_spider.py

from scrapy.spiders import Spider

from scrapy.http import Request

from scrapy.selector import Selector

from scrapy.spiders import Rule, CrawlSpider

from scrapy.linkextractors import LinkExtractor

from antianti\_crawler\_lab.items import AntiantiCrawlerLabItem

class DoubanTop250Spider(CrawlSpider):

name = "douban\_top250\_spider"

download\_delay = 1

allowed\_domains = []

start\_urls = [

'http://movie.douban.com/top250?start=0&filter=&type='

]

rules = (

Rule(

LinkExtractor(

allow=r'http://movie\.douban\.com/top250\?start=\d+&filter=&type='

),

callback='parse\_item',

follow=False), # Set follow=True when need to crawl multiple pages.

)

def parse\_item(self, response):

print(response)

sel = Selector(response)

item = AntiantiCrawlerLabItem()

movie\_name = sel.xpath('//span[@class="title"][1]/text()').extract()

star = sel.xpath('//div[@class="star"]/span/em/text()').extract()

quote = sel.xpath('//p[@class="quote"]/span[@class="inq"]/text()').extract()

item['movie\_name'] = [n.encode('utf-8') for n in movie\_name]

item['star'] = [n.encode('utf-8') for n in star]

item['quote'] = [n.encode('utf-8') for n in quote]

yield item

然后在PyCharm里运行entry\_point.py。此时会发现程序的输出包括了以下内容

DEBUG: Crawled (403) <GET http://movie.douban.com/robots.txt> (referer: None)

DEBUG: Crawled (403) <GET http://movie.douban.com/top250?start=0&filter=&type=> (referer: None)

INFO: Ignoring response <403 <http://movie.douban.com/top250?start=0&filter=&type=>>: HTTP status code is not handled or not allowed

返回代码403就表示forbidden，被禁止了，也就是被“反爬虫”了。到这里我就觉得奇怪，为什么我连爬取robots.txt文件都被403 forbidden了？于是我尝试爬取一下淘宝的首页，简单地把douban\_top250\_spider.py中的start\_urls里面的URL改为“https://www.taobao.com/”，重新运行entry\_point.py，会发现程序的输出有

DEBUG: Crawled (200) <GET https://www.taobao.com/robots.txt> (referer: None)

DEBUG: Forbidden by robots.txt: <GET https://www.taobao.com>

可以看到，我在抓取robots.txt的时候返回代码是200，即成功，后续抓取行为失败是因为淘宝的robots.txt里有

User-Agent: \*

Disallow: /

禁止了爬虫访问根目录下的所有内容。而我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。

现在我们继续在淘宝的首页进行尝试。打开settings.py文件，看到里面有以下内容

# Obey robots.txt rules

ROBOTSTXT\_OBEY = True

说明我们的项目设置是遵守robots.txt的，于是我们把True改为False，运行entry\_point.py，输出结果中有

DEBUG: Crawled (200) <GET https://www.taobao.com> (referer: None)

INFO: Closing spider (finished)

没有了前面出现过的

DEBUG: Forbidden by robots.txt: <GET https://www.taobao.com>

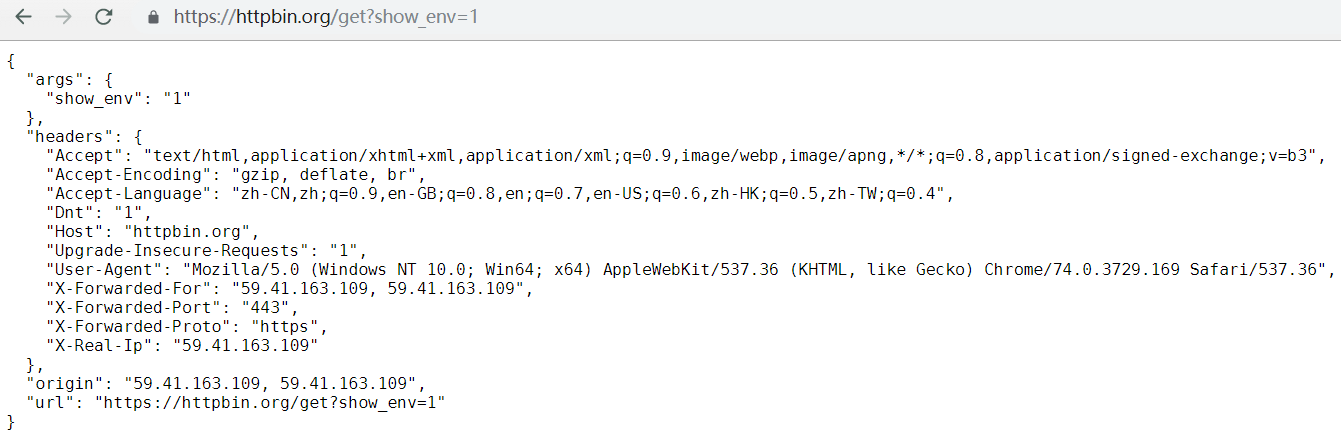
说明此时我们的程序已经不遵守robots.txt的约束，成功爬取淘宝网的首页了。不过因为我前面只改了start\_urls为淘宝网，没有改其他内容，因此也没有爬下什么有意义的数据，只是验证了程序能够不被robots.txt所约束。

因此，现在我们已经探索出了第一条反爬虫和反反爬虫的规则，即

|  |  |
| --- | --- |
| 反爬虫 | 反反爬虫 |
| robots.txt约束 | 设置程序不遵守robots.txt |

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。

我们首先在自己的浏览器上打开 [<https://httpbin.org/get?show_env=1>] 这个链接，就可以看到我们使用浏览器上网时的请求头部的内容，如下



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。

# check\_header.py

import scrapy

class CheckHeader(scrapy.Spider):

name = "check\_header\_spider"

start\_urls = ["https://httpbin.org/get?show\_env=1"]

def parse(self, response):

print(response.text)

然后把entry\_point.py其他execute指令注释掉，然后加入下面的指令。

execute(['scrapy', 'crawl', 'check\_header\_spider'])

运行entry\_point.py，得到如下结果。

{

"args": {

"show\_env": "1"

},

"headers": {

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8",

"Accept-Encoding": "gzip,deflate",

"Accept-Language": "en",

"Host": "httpbin.org",

"User-Agent": "Scrapy/1.5.2 (+https://scrapy.org)",

"X-Forwarded-For": "59.41.163.109, 59.41.163.109",

"X-Forwarded-Port": "443",

"X-Forwarded-Proto": "https",

"X-Real-Ip": "59.41.163.109"

},

"origin": "59.41.163.109, 59.41.163.109",

"url": "https://httpbin.org/get?show\_env=1"

}

可以看到上面有一行

"User-Agent": "Scrapy/1.5.2 (+https://scrapy.org)",

而正常用浏览器访问的时候对应的内容为

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",

因此这也是常见的反爬虫策略之一，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

# Crawl responsibly by identifying yourself (and your website) on the user-agent

#USER\_AGENT = 'antianti\_crawler\_lab (+http://www.yourdomain.com)'

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。

# Crawl responsibly by identifying yourself (and your website) on the user-agent

USER\_AGENT = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36'

**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

custom\_settings = {

"USER\_AGENT": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169Safari/537.36",

}

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

DEBUG: Redirecting (301) to <GET https://movie.douban.com/top250?start=0&filter=&type=> from <GET http://movie.douban.com/top250?start=0&filter=&type=>

DEBUG: Crawled (200) <GET https://movie.douban.com/top250?start=0&filter=&type=> (referer: None)

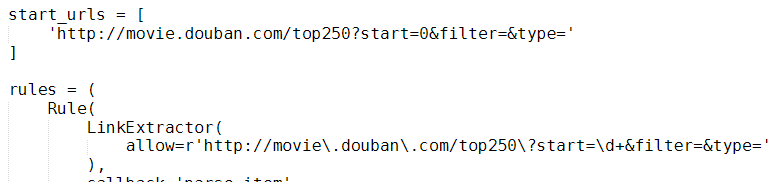
INFO: Closing spider (finished)

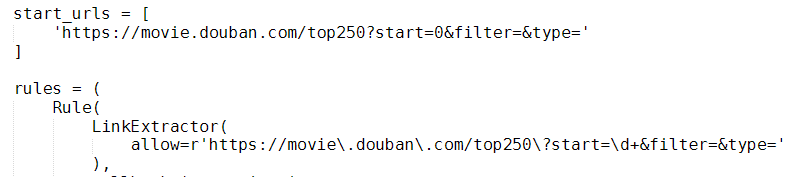
可以看到这时候返回代码已经不是403了。

到此，我们又得到了新的反爬虫和反反爬虫的规则，即

|  |  |
| --- | --- |
| 反爬虫 | 反反爬虫 |
| robots.txt约束 | 设置程序不遵守robots.txt |
| 检查HTTP请求头部 | 伪装HTTP请求头部 |

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douban\_top250\_spider.py文件里修改URL。把start\_urls和rule里面的http改为https。



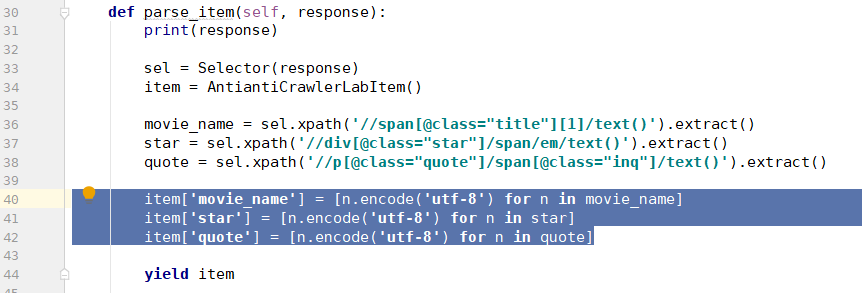


然后重新运行entry\_point.py





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

import scrapy

class AntiantiCrawlerLabItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

rank = scrapy.Field()

movie\_name = scrapy.Field()

star = scrapy.Field()

quote = scrapy.Field()

在entry\_point.py中，修改为如下内容，表示我们需要输出到一个csv文件里。

import scrapy

from scrapy.cmdline import execute

execute(['scrapy', 'crawl', 'douban\_top250\_spider', '-o', 'douban\_top250\_spider.csv'])

在douban\_top250\_spider.py中，把parse\_item函数修改为如下内容

def parse\_item(self, response):

print(response)

item = AntiantiCrawlerLabItem()

movies = response.xpath('//ol[@class="grid\_view"]/li')

for movie in movies:

item['rank'] = movie.xpath(

'.//div[@class="pic"]/em/text()'

).extract()[0]

item['movie\_name'] = movie.xpath(

'.//div[@class="hd"]/a/span[1]/text()'

).extract()[0]

item['star'] = movie.xpath(

'.//div[@class="star"]/span[@class="rating\_num"]/text()'

).extract()[0]

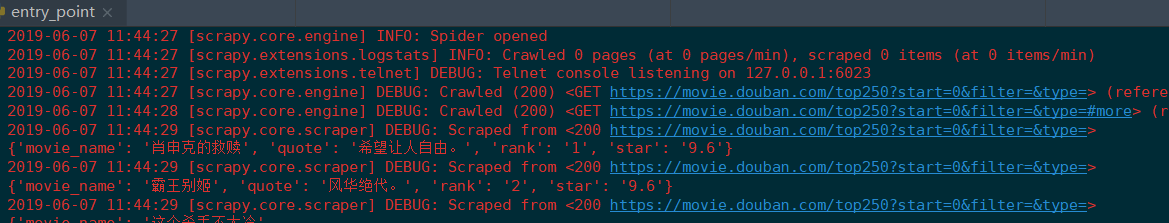
item['quote'] = movie.xpath(

'.//p[@class="quote"]/span[@class="inq"]/text()'

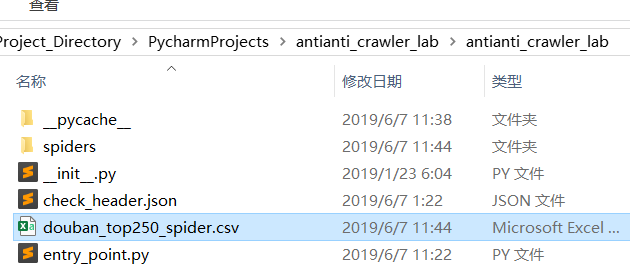
).extract()[0]

yield item

然后运行entry\_point.py，得到以下输出



并且在项目目录下也发现了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）





我们看到，这里爬下来的数据只有25部电影，我们打开豆瓣电影Top250首页可以看到，第一页就只有这25部电影。说明我们的爬虫只爬取了第一页，没有爬取后面的页面。这是因为我在douban\_top250\_spider.py里面的一行代码设置

rules = (

Rule(

LinkExtractor(

allow=r'https://movie\.douban\.com/top250\?start=\d+&filter=&type='

),

callback='parse\_item',

follow=False), # Set follow=True when need to crawl multiple pages.

)

这里我设置了“follow=False”，表示我不会跟随爬取更多的页面。当我要爬取后面的页面时，我设置为“follow=True”。

rules = (

Rule(

LinkExtractor(

allow=r'https://movie\.douban\.com/top250\?start=\d+&filter=&type='

),

callback='parse\_item',

follow=True), # Set follow=True when need to crawl multiple pages.

)

然后重新运行entry\_point.py，发现仍然只爬了第一页的内容下来。从控制台输出内容中可以看到，在爬完第一页的数据之后，控制台输出了以下内容

DEBUG: Filtered duplicate request:

<GET <https://movie.douban.com/top250?start=0&filter=&type=#more>>

- no more duplicates will be shown (see DUPEFILTER\_DEBUG to show all duplicates)

INFO: Closing spider (finished)

INFO: Stored csv feed (25 items) in: douban\_top250\_spider.csv

我目前还不能很熟练的正确使用crawling rules，因此我打算用另一种方法实现爬取多页的逻辑。打开douban\_top250\_spider.py，然后在parse\_item函数内部最后面加上以下内容

next\_url = response.xpath('//span[@class="next"]/a/@href').extract()

if next\_url:

next\_url = 'https://movie.douban.com/top250' + next\_url[0]

yield Request(next\_url)

然后重新运行entry\_point.py，就能看到程序成功把250部电影的信息都爬取下来了。



还有许多常见的反爬虫策略，比如

* 监控IP访问频率，频率过高则判定为机器人
* 动态加载页面使得简单爬虫无法直接从HTML文件拿到数据
* 使用token
* 蜜罐——在页面中加入一些合法人类用户正常使用浏览器访问时看不到的内容，但爬虫可以看到，一旦爬虫访问这些特殊的内容，则触发反爬虫机智
* 验证码等图灵测试手段
* ……

**现在这个课程设计的report内容还是太少了，感觉答辩的时候很难通过，老师说每个人都要有2个问题，这远不够。**

**这一页写的其他常见反爬虫策略，有一些是可以简单的自己实现的。反正我们实验3、4是要搭web服务，我觉得可能可以自己实现简单的反爬虫策略，然后自己去尝试违反策略和绕开策略？即，自己做简单的反爬虫，然后自己再做简单的反反爬虫。不用多高大上的，就只是实现非常简单的版本。这样就可以写得出很多东西，答辩也不怎么担心了。**