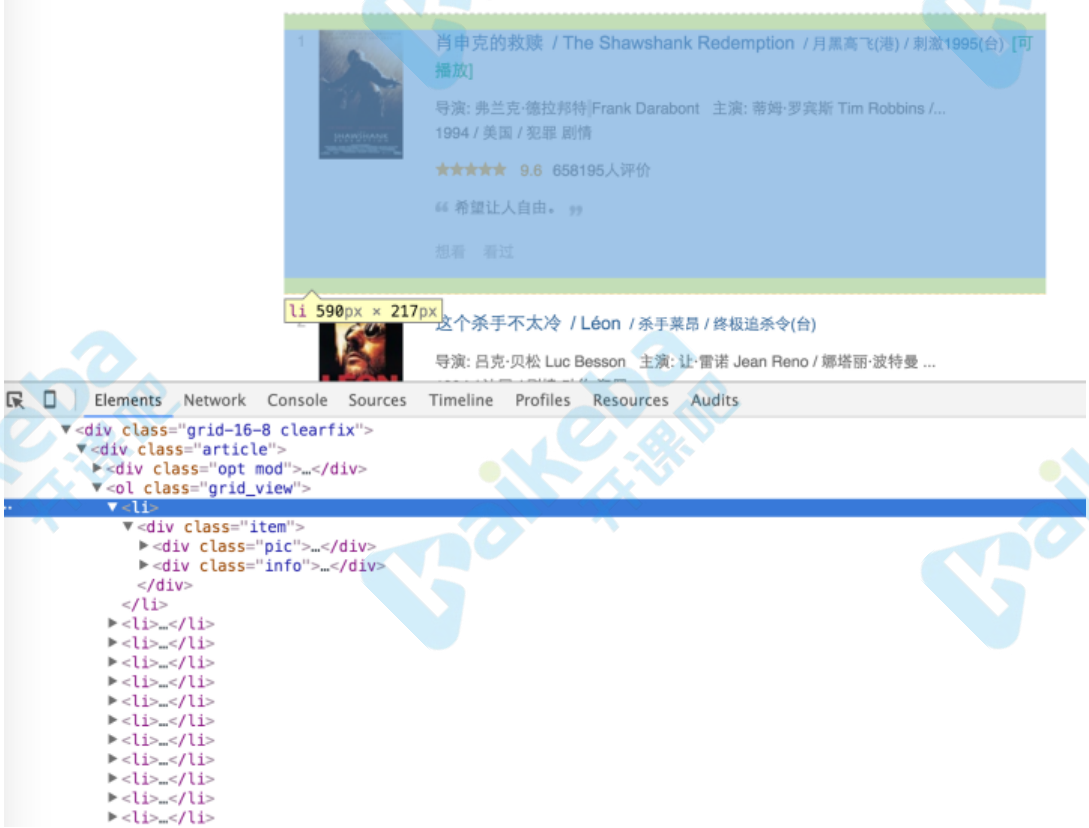
3.豆瓣电影TOP250

![xlzd](https://www.zhihu.com/people/xlzd)

今天，我们的目标是——豆瓣电影TOP250。

和完成其他代码一样，编写爬虫之前，我们需要先思考爬虫需要干什么、目标网站有什么特点，以及根据目标网站的数据量和数据特点选择合适的架构。编写爬虫之前，推荐使用Chrome的开发者工具来观察网页结构。在OS X上，通过”option+command+i”可以打开Chrome的开发者工具，在Windows和Linux，对应的快捷键是”F12”。效果如下：

OK，可以看出，这个页面其实有一个列表，其中放着25条电源信息。我们选中某一条电影，右键选择检查即可查看选中条目的HTML结构。如下图所示：



到这一步，我们已经得到的信息有如下：

1. 每页有25条电影，共有10页。
2. 电影列表在页面上的位置为一个class属性为grid_view的ol标签中。
3. 每条电影信息放在这个ol标签的一个li标签里。

到这一步，我们可以开始写代码了。先完成下载网页源码的代码吧，这里我们使用requests库：

```
#!/usr/bin/env python
# encoding=utf-8
#导包
import requests
#确定url
DOWNLOAD_URL = 'http://movie.douban.com/top250'
#定义函数，封装请求代码,content是返回二进制数据
def download_page(url):
    data = requests.get(url).content
    return data
def main():
    print download_page(DOWNLOAD_URL)
if __name__ == '__main__':
    main()
```

先来简单测试一下，没想到运行之后得到的结果是：

```
<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>dae</center>
</body>
</html>
```

产生403的原因，一般可能是因为需要登录的网站没有登录或者被服务器认为是爬虫而拒绝访问，这里很显然属于第二种情况。一般，浏览器在向服务器发送请求的时候，会有一个请求头——User-Agent，它用来标识浏览器的类型.当我们使用requests来发送请求的时候，默认的User-Agent是python-requests/2.8.1（后面的数字可能不同，表示版本号）。那么，我们试试看如果将User-Agent伪装成浏览器的，会不会解决这个问题呢?

```
#!/usr/bin/env python
# encoding=utf-8
#导包
import requests
#url
DOWNLOAD_URL = 'http://movie.douban.com/top250/'
#编写函数，封装请求代码
def download_page(url):
#编写请求头
    headers = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36'
    }
    #携带请求头发送请求，返回二进制数据
    data = requests.get(url, headers=headers).content
    return data
def main():
    print download_page(DOWNLOAD_URL)
if __name__ == '__main__':
    main()
```

上面的代码中，我们通过手动指定User-Agent为Chrome浏览器，再此访问就得到了真实的网页源码。服务器通过校验请求的U-A来识别爬虫，这算是最简单的一种反爬虫机制了，通过模拟浏览器的U-A，能够很轻松地绕过这个问题。

当我们拿到网页源码之后，就需要解析HTML源码了。这里，我们使用BeautifulSoup来搞定这件事。在使用之前，你需要通过运行pip install beautifulsoup4来安装BeautifulSoup。

使用BeautifulSoup解析网页的大致过程如下：

```
1. from bs4 import BeautifulSoup
2. #设置函数，封装解析html代码
3. def parse_html(html):
```

```
    #传入解析文本，返回bs对象
    soup = BeautifulSoup(html)
    #根据标签查找符合的数据
    movie_list_soup = soup.find('ol', attrs={'class': 'grid_view'})
    #遍历数据
    for movie_li in movie_list_soup.find_all('li'):
        #循环查找满足不同条件的数据
        detail = movie_li.find('div', attrs={'class': 'hd'})
        movie_name = detail.find('span', attrs={'class':
'title'}).getText()

        print movie_name
```

我将详细解释这段代码：import用来导入BeautifulSoup，这很容易理解。接着我们定义了函数parse_html，它接受html源码作为输入，并将这个网页中的电影名称打印到控制台。第5行我们创建了一个BeautifulSoup对象（这样的创建方式会产生一个warning，我们下一节再聊这个问题），然后紧接着在第7行使用刚刚创建的对象搜索这篇html文档中查找那个class为grid_view的ol标签（上面分析的第2步），接着通过find_all方法，我们得到了电影的集合，通过对它迭代取出每一个电影的名字，打印出来。至于for循环之间的内容，其实就是在解析每个li标签。你可以很简单的在刚才的浏览器窗口通过开发者工具查看li中的网页结构。

到这一步，我们已经得到了电影名称（由于只是演示BeautifulSoup的用法，这里不详细取出每条电影的所有信息），刚才提到一共有10页数据，怎么处理翻页的问题呢？一般在我们确定内容的前提下，可以直接在代码中写死如何跳转页面，但是为了让我们的爬虫更像爬虫，我们让它找到页码导航中的下一页的链接。

还是借助开发者工具，我们找到了下一页的链接放置在一个span标签中，这个span标签的class为next。具体链接则在这个span的a标签中，到了最后一页之后，这个span中的a标签消失了，就不需要再翻页了。于是，根据这段逻辑，我们将上面parse_html函数稍作修改：

```
def parse_html(html):
    soup = BeautifulSoup(html)
    movie_list_soup = soup.find('ol', attrs={'class': 'grid_view'})
```

```
    movie_name_list = []
    for movie_li in movie_list_soup.find_all('li'):
        detail = movie_li.find('div', attrs={'class': 'hd'})
        movie_name = detail.find('span', attrs={'class':
'title'}).getText()
        movie_name_list.append(movie_name)
    next_page = soup.find('span', attrs={'class': 'next'}).find('a')
    if next_page:
        return movie_name_list, DOWNLOAD_URL + next_page['href']
    return movie_name_list, None
```

我们需要在解析html之后取回我们需要的数据，于是将打印变成了返回一个包含电影名的list，以及下一页的链接，如果到了最后一页，则返回None。

到这里，大部分代码已经完成了，我们将其组装成一个完整的程序即可：

```
import codecs
def main():
    url = DOWNLOAD_URL
#以只读的方式打开文件，生成文件对象
    with codecs.open('movies', 'wb', encoding='utf-8') as fp:
        while url:
            html = download_page(url)
            movies, url = parse_html(html)
            #写入内容
            fp.write(u'{movies}\n'.format(movies='\n'.join(movies)))
```

上面的代码完成了对程序的拼装，并将结果输出到一个文件中，其中使用了codecs这个包是为了更方便处理中文编码。当程序运行结束之后，所有的电影名称就都写入到了movies这个文件中。

小结

这篇博客总结了最简单的处理反爬虫机制，以及简单的BeautifulSoup的使用，最后完成了将结果写入到文件中去。麻雀虽小，五脏俱全，这个程序虽然功能简单，但却算是一个完整的爬虫程序了。
接下来，我们将会面临更加复杂的反爬虫机制，面对更加复杂的网页结构，以及会使用数据库来持久化存储爬取结果。

完整的代码如下：

```
#!/usr/bin/env python
# encoding=utf-8
"""
爬取豆瓣电影TOP250 - 完整示例代码
"""
```

```
import codecs
import requests
from bs4 import BeautifulSoup
DOWNLOAD_URL = 'http://movie.douban.com/top250/'
def download_page(url):
    return requests.get(url, headers={
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36'
    }).content
def parse_html(html):
    soup = BeautifulSoup(html)
    movie_list_soup = soup.find('ol', attrs={'class': 'grid_view'})
    movie_name_list = []
    for movie_li in movie_list_soup.find_all('li'):
        detail = movie_li.find('div', attrs={'class': 'hd'})
        movie_name = detail.find('span', attrs={'class': 'title'}).getText()
        movie_name_list.append(movie_name)
    next_page = soup.find('span', attrs={'class': 'next'}).find('a')
    if next_page:
        return movie_name_list, DOWNLOAD_URL + next_page['href']
    return movie_name_list, None
def main():
    url = DOWNLOAD_URL
    with codecs.open('movies', 'wb', encoding='utf-8') as fp:
        while url:
            html = download_page(url)
            movies, url = parse_html(html)
            fp.write(u'{movies}\n'.format(movies='\n'.join(movies)))
if __name__ == '__main__':
    main()
```