第 3 章

CHAPTER 3

静态网页爬取

在网站设计中，纯粹 HTML（标准通用标记语言下的一个应用）格式的网页通常被称为“静态网页”。静态网页是相对于动态网页而言的，是指没有后台数据库、不含程序和不可交互的网页。静态网页的更新相对比较麻烦，适用于一般更新较少的展示型网站。容易让人产生误解的是静态页面都是 HTML 这类页面，实际上静态也不是完全静态，它也可以出现各种动态的效果，如 GIF 格式的动画、Flash、滚动字幕等。

在网络爬虫中，静态网页的数据比较容易获取，因为所有数据都呈现在网页的 HTML 代码中。相对而言，使用 AJAX 动态加载网络的数据不一定会出现在 HTML 代码中，这就给爬虫增加了困难。

在静态网页中，有一个强大的 Requests 库能够让我们方便地发送 HTTP 请求，这个库功能完善，而且操作非常简单。

3.1 Requests 的安装

在 Windows 系统下，Requests 库可以通过 pip 安装。打开 cmd 或 terminal，输入：

```
pip install requests
```

即可完成安装，可以输入 import requests 命令来试试是否安装成功，如图 3-1 所示即显示安装成功。

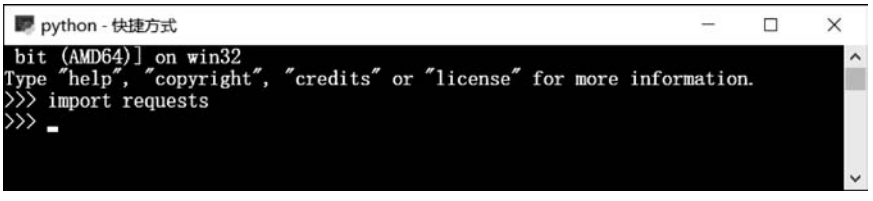

图 3-1　成功安装 Requests

在 Requests 中，最常用的功能就是获取某个网页内容。现在使用 Requests 获取个人博客主页的内容。

```
>>> import requests
>>> r = requests.get('http://www.zhidaow.com')  # 发送请求
```

```
>>> r.status_code                          # 返回码
200
>>> r.headers['content-type']              # 返回头部信息
'text/html; charset=utf-8'
>>> r.encoding                             # 编码信息
'utf-8'
>>> r.text    #内容部分(PS,由于编码问题,建议这里使用 r.content)
'\n<!DOCTYPE html>\n<html>\n<head>\n<script type="text/javascript" src="/fb
…
```

其中，

(1) r.text 是服务器响应的内容，会自动根据响应头部的字符编码进行解码。

(2) r.encoding 是服务器的内容所使用的文本编码。

(3) r.status_code 用于检测响应的状态码，如果返回 200，则表示请求成功；如果返回的是 4xx，则表示客户端错误；如果返回 5xx，则表示服务器错误响应。可以用 r.status_code 来检测请求是否正确响应。

(4) r.content 是字节方式的响应体，会自动解码 gzip 和 deflate 编码的响应数据。

3.2 获取响应内容

在 Python 爬虫网络中，可以使用 r.encoding 获取网页编码。

```
>>> import requests
>>> r = requests.get('http://www.zhidaow.com')
>>> r.encoding
'utf-8'
```

在 Python 中，当发送请求时，Requests 会根据 HTTP 头部来猜测网页编码，当使用 r.text 时，Requests 就会使用这个编码。当然你还可以修改 Requests 的编码形式。例如：

```
>>> r = requests.get('http://www.zhidaow.com')
>>> r.encoding
'utf-8'
>>> r.encoding = 'ISO-8859-1'
>>>
```

像上面的例子，对 encoding 修改后就直接会用修改后的编码去获取网页内容。

3.3 JSON 数据库

JSON 全称为 JavaScript Object Notation，也就是 JavaScript 对象标记，它通过对象和数组的组合来表示数据，构造简洁但是结构化程度非常高，是一种轻量级的数据交换格式。下面进行简单的介绍，第 7 章将对其进行详细介绍。

3.3.1 JSON 的使用

像 urllib1 和 urllib2，如果用到 JSON，就要引入新模块，如 JSON 和 simplejson，但在 Requests 中已经有了内置的函数——r.json()。以查询 IP 的 API 为例：

```
>>> r = requests.get('http://ip.taobao.com/service/getIpInfo.php?ip=122.88.60.28')
>>> r.json()['data']['country']
'中国'
```

3.3.2　爬取抽屉网信息

此外，还可以利用 Requests 和 JSON 爬取网络信息。

【例 3-1】 爬取抽屉网信息(JSON 数据)。

```
import requests
from fake_useragent import UserAgent
agent = UserAgent()
import json #
# # pip search 工具包名字
# # pip install fake_useragent
url = "https://dig.chouti.com/getTopTenLinksOrComments.json?_=1529764992551"
# # 通过浏览器获取的操作一般都是 get 请求
headers = { "Host":"dig.chouti.com",
    "User-Agent":"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0",
    "Accept": "application/json,text/javascript, */*; q=0.01",
    "Accept-Language": "zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3",
    "Accept-Encoding":"gzip,deflate, br",
    "X-Requested-With":"XMLHttpRequest",
    "Referer":"https://dig.chouti.com/", "Cookie":"gpsd=0b08b9f5b945fd53eac7868a2e8945a8;
JSESSIONID=aaazCSOWV2s7FcALFeHqw;gpid=55eaeb947f15445b82467624c476521f;_pk_id.1.a2d5=
dbeb24b52f36519f.1529741245.1.1529741290.1529741245.;_pk_ses.1.a2d5=*",
    "Connection":"keep-alive"}
data = {"_":"1529742010062"}
res = requests.post(url, headers=headers, data=data)
rs_js = json.loads(res.content)
print(rs_js['result']['data'])
```

运行程序，输出如下：

```
    raise FakeUserAgentError('Maximum amount of retries reached')
fake_useragent.errors.FakeUserAgentError: Maximum amount of retries reached
[{'action': 1, 'actiontime': 1558879802717000, 'actiontimeStr': '8 分钟前', 'closeIp': False,
'commentsCount': 0, 'comm…
…
```

3.4　传递 URL 参数

为了请求特定的数据，需要在 URL 的查询字符串中加入某些数据。如果你是自己构建 URL，那么数据一般会跟在一个问号后面，并且以键-值的形式放在 URL 中，如 http://httpbin.org/get?key1=value1。

在 Requests 中，可以直接把这些参数保存在字典中，用 params 构建至 URL 中。例如，将 key1=value1 和 key2=value2 传递到 http://httpbin.org/get，可以这样编写：

```
import requests
key_dict={'key1':'value1','key2':'value2'}
r=requests.get('http://httpbin.org/get',params=key_dict)
```

```
print('URL 已经正确编码: ',r.url)
print('字符串方式的响应体: \n',r.text)
```

通过上述代码的输出结果可以发现 URL 已经正确编码：

```
URL 已经正确编码: http://httpbin.org/get?key1 = value1&key2 = value2
```

字符串方式的响应体：

```
{
  "args": {
    "key1": "value1",
    "key2": "value2"
  },
  "headers": {
    "Accept": " * / * ",
    "Accept - Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User - Agent": "python - requests/2.19.1"
  },
  "origin": "14.27.49.210, 14.27.49.210",
  "url": "https://httpbin.org/get?key1 = value1&key2 = value2"
}
```

3.5 获取响应内容

在 Requests 中，可以通过 r. text 来获取网页的内容。例如：

```
>>> import requests
>>> r = requests.get('https://www.baidu.com')
>>> r.text
'<!DOCTYPE html >\r\n <! -- STATUS OK -- >< html >< head >< meta http - equiv = content - type
content = text/html;charset = utf - 8 >< meta htt …
…
```

在 Requests 中，还会自动将内容转码，大多数 unicode 字体都会无缝转码。此外，还可以通过 r. content 来获取页面内容。

```
>>> r = requests.get('https://www.baidu.com')
>>> r.content
b'<!DOCTYPE html >\r\n <! -- STATUS OK -- >< html >< head >< meta http - equiv = content - type
content = text/html;charset = utf - 8 >< meta ht …
…
```

r. content 是以字节的方式去显示，所以在 IDLE 中以 b 开头。但在 cygwin 中用起来并没有，下载网页正好，所以就替代了 urllib2 的 urllib2. urlopen(url). read()功能。

3.6 获取网页编码

在 Requests 中，可以用 r. status_code 来检查网页的状态码。例如：

```
>>> r = requests.get('http://www.mengtiankong.com')
>>> r.status_code
```

```
200
>>> r = requests.get('http://www.mengtiankong.com/123123/')
>>> r.status_code
404
>>> r = requests.get('http://www.baidu.com/link?url=QeTRFOS7TuUQRppa0wlTJJr6FfI-YI1DJprJukx4Qy0XnsDO_s9bao08u1wvjxgqN')
>>> r.url
'http://www.zhidaow.com/'
>>> r.status_code
200
```

前两个例子很正常，能正常打开的返回 200，不能正常打开的返回 404。但第三个就有点奇怪了，那个是百度搜索结果中的 302 跳转地址，但状态码显示是 200，接下来用其他方法进行实验：

```
>>> r.history
[<Response [302]>]
>>>
```

这里能看出是使用了 302 跳转。也许有人认为这样可以通过判断来获取跳转的状态码了，其实还有个更简单的方法：

```
>>> r = requests.get('http://www.baidu.com/link?url=QeTRFOS7TuUQRppa0wlTJJr6FfIYI1D-JprJukx4Qy0XnsDO_s9bao08u1wvjxgqN', allow_redirects=False)
>>> r.status_code
302
```

只要加上一个参数 allow_redirects，禁止了跳转，就会直接出现跳转的状态码了。

3.7　定制请求头

请求头 Headers 提供了关于请求、响应或其他发送实体的信息。对于爬虫而言，请求头十分重要，尽管在上一个例子中并没有制定请求头。如果没有指定请求头或请求的请求头与实际网页不一致，就可能无法返回正确的结果。

Requests 并不会基于定制的请求头 Headers 的具体情况改变自己的行为，只是在最后的请求中，所有的请求头信息都会被传递进去。

在 Requests 中可以通过 r.headers 获取响应头内容。例如：

```
>>> r.headers
{'Cache-Control': 'private, no-cache, no-store, proxy-revalidate, no-transform', 'Connection': 'Keep-Alive', 'Content-Encoding': 'gzip', 'Content-Type': 'text/html', 'Date': 'Mon, 27 May 2019 04:47:21 GMT', 'Last-Modified': 'Mon, 23 Jan 2017 13:27:57 GMT', 'Pragma': 'no-cache', 'Server': 'bfe/1.0.8.18', 'Set-Cookie': 'BDORZ=27315; max-age=86400; domain=.baidu.com; path=/', 'Transfer-Encoding': 'chunked'}
```

由结果可以看到是以字典的形式返回了全部内容，也可以访问部分内容。例如：

```
>>> r.headers['Content-Type']
'text/html'
>>> r.headers.get('content-type')
'text/html'
```

而请求头内容可以用 r.request.headers 来获取。例如：

```
>>> r.request.headers
{'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*',
'Connection': 'keep-alive'}
```

3.8 发送 POST 请求

除了 GET 请求外，有时还需要发送一些编码为表单形式的数据，如在登录的时候请求就为 POST，因为如果用 GET 请求，密码就会显示在 URL 中，这是非常不安全的。如果要实现 POST 请求，那么只需要简单地传递一个字典给 Requests 中的 data 参数，这个数据字典就会在发出请求的时候自动编码为表单形式。例如：

```
import requests
key_dict = {'key1':'value1','key2':'value2'}
r = requests.post('http://httpbin.org/post',data = key_dict)
print(r.text)
```

运行程序，输出如下：

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key1": "value1",
    "key2": "value2"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Content-Length": "23",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.19.1"
  },
  "json": null,
  "origin": "14.27.49.210, 14.27.49.210",
  "url": "https://httpbin.org/post"
}
```

可以看到，form 变量的值为 key_dict 输入的值，这样一个 POST 请求就发送成功了。

3.9 设置超时

有时爬虫会遇到服务器长时间不返回，这时爬虫程序就会一直等待，造成爬虫程序没能顺利地执行。因此，可以用 Requests 在 timeout 参数设定的秒数结束之后停止等待响应。也就是说，如果服务器在 timeout 秒内没有应答，就返回异常。

把这个秒数设置为 0.001 秒，看看会抛出什么异常，这是为了让大家体验 timeout 异常的效果而设置的值，一般会把这个值设置为 20 秒。

```
>>> requests.get('http://github.com', timeout = 0.001)
Traceback (most recent call last):
  File "C:\Users\ASUS\Anaconda3\lib\site - packages\urllib3\connection.py", line 141, in _
new_conn
    (self.host, self.port), self.timeout, ** extra_kw)
  File "C:\Users\ASUS\Anaconda3\lib\site - packages\urllib3\util\connection.py", line 83, in
create_connection
    raise err
  File "C:\Users\ASUS\Anaconda3\lib\site - packages\urllib3\util\connection.py", line 73, in
create_connection
    sock.connect(sa)
socket.timeout: timed out
During handling of the above exception, another exception occurred:
…
```

3.10　代理访问

采集时为避免被封 IP,经常会使用代理。Requests 也有相应的 proxies 属性。例如：

```
import requests
proxies = {
  "http": "http://10.10.1.10:3128",
  "https": "http://10.10.1.10:1080",
}
requests.get("http://www.zhidaow.com", proxies = proxies)
```

如果代理需要账户和密码,则需如下这样：

```
proxies = {
    "http": "http://user:pass@10.10.1.10:3128/",
}
```

3.11　自定义请求头部

在 Requests 中,伪装请求头部是采集时经常用的,可以用如下方法来隐藏：

```
import requests
r = requests.get('http://www.zhidaow.com')
print(r.request.headers['User - Agent'])
headers = {'User - Agent': 'alexkh'}
r = requests.get('http://www.zhidaow.com', headers = headers)
print(r.request.headers['User - Agent'])
```

运行程序,输出如下：

```
python - requests/2.19.1
alexkh
```

3.12　Requests 爬虫实践

至此,已经介绍了利用爬虫网络对静态网页进行爬取,下面直接通过两个实例来演示爬虫的实践。

3.12.1 状态码 521 网页的爬取

1. 问题发现

在做代理池的时候，发现了一种以前没有见过的反爬虫机制。在用常规的 requests.get(url)方法对目标网页进行爬取时，其返回的状态码(status_code)为 521，这是一种以前没有见过的状态码。再输出它的爬取内容(text)，发现是一些 JavaScript 代码，如图 3-2 所示。下面来探索一下。

```
521
<html><body><script language="javascript"> window.onload=setTimeout("lr(74)", 200);
    function lr(RK) {var qo, mo="", no="", oo = [0xfa,0x8a,0xc3,0xb5,0x91,0x85,0x71,0x67
    ,0x5e,0x39,0xef,0xe1,0xd7,0xcd,0xd3,0xcd,0xc5,0xb1,0x55,0x03,0x09,0x4f,0x37,0x24,0x
    16,0x09,0xf6,0xe6,0xbb,0x9e,0x8e,0x83,0x75,0x61,0x05,0x85,0x72,0x5e,0x49,0x3c,0xdc,
    0x79,0x20,0x12,0xaf,0x50,0xeb,0xd7,0x73,0x6e,0x10,0xaf,0x51,0x3d,0x30,0xd1,0xf1,0xde
    ,0xd0,0x6c,0x09,0xa6,0x47,0x37,0xec,0x89,0x78,0x64,0x00,0xb4,0x0d,0xa9,0x45,0xe8,0x
    8f,0x43,0x36,0xac,0x4d,0x40,0xdc,0x90,0x2d,0xcc,0x5a,0xf8,0x97,0x84,0x20,0x10,0xb2,
    0x84,0x76,0x14,0x06,0xa7,0x5b,0xfc,0x70,0x0c,0xa8,0x4a,0xf1,0x91,0x31,0x1d,0xc0,0x5d
    ,0xf9,0xa3,0x54,0x40,0x15,0xfd,0xde,0xd6,0xb7,0xa3,0x85,0x76,0x1a,0x15,0xf2,0xde,0x
    91,0x42,0x84,0x23,0xbf,0x73,0x80,0x6c,0x5e,0x13,0xc8,0x68,0x0f,0xbf,0x60,0x07,0x5a,
    0x03,0xa4,0x41,0xeb,0x8c,0x27,0xed,0x9e,0xac,0xc0,0xbb,0x65,0x16,0x3a,0x25,0x1b,0x0f
    ,0x00,0xf8,0xed,0x3c,0xdf,0x96,0x33,0xd1,0xc9,0xaa,0xb3,0x6a,0x5c,0x51,0xfc,0xab,0x
    8c,0x34,0x24,0x00,0xf7,0x9b,0x51,0x00,0xf0,0x9a,0x49,0x27,0x1f,0x16,0x01,0xda,0xd0,
    0xae,0x63,0x50,0x46,0x38,0xd9,0xb5,0xa9,0x95,0x8c,0x67,0x1e,0xab,0xa0,0x96,0x88,0x78
    ,0x53,0x4b,0xe7,0xdd,0xd4,0x78,0x63,0x59,0x4b,0x23,0x00,0xf4,0xe0,0xd5,0xb2,0x67,0x
    a7,0xa3,0xa6,0xc1,0x3b];qo = "qo=227; do{oo[qo]=(-oo[qo])&0xff; oo[qo]=(((oo[
    qo]>>1)|((oo[qo]<<7)&0xff))-153)&0xff;} while(--qo>=2);"; eval(qo);qo = 226; do { oo
    [qo] = (oo[qo] - oo[qo - 1]) & 0xff; } while (-- qo >= 3 );qo = 1; for (;;) { if (qo
     > 226) break; oo[qo] = ((((((oo[qo] + 101) & 0xff) + 40) & 0xff) << 1) & 0xff) | ((
    (((oo[qo] + 101) & 0xff) + 40) & 0xff) >> 7); qo++;}po = ""; for (qo = 1; qo < oo.
    length - 1; qo++) if (qo % 7) po += String.fromCharCode(oo[qo] ^ RK);eval("
    qo=eval;qo(po);");} </script> </body></html>
```

图 3-2 状态码和爬取内容

2. 分析原理

打开 Fiddler，爬取访问网站的包，如图 3-3 所示，发现浏览器对于同一网页连续访问了两次，第一次的访问状态码为 521，第二次为 200(正常访问)。看来网页加了反爬虫机制，需要两次访问才可返回正常网页。

#	结果	协议	Host	URL	正文	缓存	Content-Type	进程
⚠ 7	521	HTTP	www....	/	855	no-cache...	text/html	chrome...
⇔ 8	200	HTTP	www....	/	3,746		text/html	chrome...

图 3-3 Fiddler 抓包信息

下面来对比两次请求的区别。521 的请求如图 3-4 所示；200 的请求如图 3-5 所示。

```
请求标头                                                    [原始] [头定义]
GET / HTTP/1.1
Cookies
  Cookie
    Hm_lvt_1761fabf3c988e7f04bec51acd4073f4=1543141191,1543571983,1543831372,1543904668
安全
  Upgrade-Insecure-Requests: 1
传输
  Connection: keep-alive
  Host:
客户端
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
  Accept-Encoding: gzip, deflate
  Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36
```

图 3-4 521 请求

图 3-5　200 请求

通过对比两次请求头，可发现第二次访问带了新的 Cookie 值。再考虑上面程序对爬取结果的输出为 JavaScript 代码，可以考虑其操作过程为：第一次访问时服务器返回一段可动态生成 Cookie 值的 JavaScript 代码；浏览器运行 JavaScript 代码生成 Cookie 值，并带 Cookie 重新进行访问；服务器被正常访问，返回页面信息，浏览器渲染加载。

3. 执行流程

弄清楚浏览器的执行过程后，就可以模拟其行为通过 Python 进行网页爬取。操作步骤如下：

- 用 request.get(url)获取 JavaScript 代码。
- 通过正则表达式对代码进行解析，获得 JavaScript 函数名，JavaScript 函数参数和 JavaScript 函数主体，并将执行函数 eval()语句修改为 return 语句返回 Cookie 值。
- 调用 execjs 库的 executeJS()功能执行 JavaScript 代码获得 Cookie 值。
- 将 Cookie 值转化为字典格式，用 request.get(url, Cookie = Cookie)方法获取得到正确的网页信息。

4. 实现代码

根据以上流程步骤，实现代码主要表现在：

(1) 实现程序所需要用到的库。

```
import re              #实现正则表达式
import execjs          #执行 JavaScript 代码
import requests        #爬取网页
```

(2) 第一次爬取获得包含 JavaScript 函数的页面信息后，通过正则表达式对代码进行解析，获得 JavaScript 函数名、JavaScript 函数参数和 JavaScript 函数主体，并将执行函数 eval()语句修改为 return 语句返回 Cookie 值。

```
# js_html 为获得的包含 JavaScript 函数的页面信息
# 提取 JavaScript 函数名
js_func_name = ''.join(re.findall(r'setTimeout\(\"(\D+)\(\d+\)\"', js_html))
# 提取 JavaScript 函数参数
js_func_param = ''.join(re.findall(r'setTimeout\(\"\D+\((\d+)\)\"', js_html))
# 提取 JavaScript 函数主体
js_func = ''.join(re.findall(r'(function .*?)</script>', js_html))
```

(3) 将执行函数 eval()语句修改为 return 语句返回 Cookie 值。

```
# 修改 JavaScript 函数,返回 Cookie 值
js_func = js_func.replace('eval("qo=eval;qo(po);")', 'return po')
```

(4) 调用 execjs 库的 executeJS()功能执行 JavaScript 代码获得 Cookie 值。

```
# 执行 JavaScript 代码的函数,参数为 JavaScript 函数主体,JavaScript 函数名和 JavaScript 函数参数
def executeJS(js_func, js_func_name, js_func_param):
jscontext = execjs.compile(js_func) #调用 execjs.compile()加载 JavaScript 函数主体内容
func = jscontext.call(js_func_name,js_func_param) #使用 call()通过函数名和参数执行该函数
return func
cookie_str = executeJS(js_func, js_func_name, js_func_param)
```

(5) 将 Cookie 值转化为字典格式。

```
#将 Cookie 值解析为字典格式,方便后面调用
def parseCookie(string):
 string = string.replace("document.cookie='", "")
 clearance = string.split(';')[0]
 return {clearance.split('=')[0]: clearance.split('=')[1]}
cookie = parseCookie(cookie_str)
```

至此,在获得 Cookie 后,采用带 Cookie 的方式重新进行爬取,即可获得我们需要的网页信息。

3.12.2 TOP250 电影数据

本实践项目的目的是获取豆瓣电影 TOP250 的所有电影名称,网页地址为:https://movie.douban.com/top250。在此爬虫中,将请求头定制为实际浏览器的请求头。

1. 网站分析

打开豆瓣电影 TOP250 的网站,右击网页的任意位置,在弹出的快捷菜单中单击“审查元素”命令即可打开该网页的请求头,如图 3-6 所示。

图 3-6 豆瓣电影 TOP250 的网站

提取网站中重要的请求头代码为：

```
import requests                    #爬取网页
headers = {'user - agent':'Mozilla/5.0(Windows NT 6.1;Win64;x64) AppleWebKit/537.36(KHTML,
like Gecko) Chrome/52.0.2743.82 Safari/537.36','Host':'movie.douban.com'}
```

第一页只有 25 部电影，如果要获取所有的 250 页电影，就需要获取总共 10 页的内容。通过单击第二页可以发现网页地址变成了：

```
https://movie.douban.com/top250?start = 25
```

第三页的地址为 https://movie.douban.com/top250? start＝50，这很容易理解，每多一页，就给网页地址的 start 参数加上 25。

2. 项目实践

通过以上分析，可以使用 requests 获取电影网页的代码，并利用 for 循环翻页。代码为：

```
import requests                    #爬取网页
def get_movies():
    headers = {'user - agent':'Mozilla/5.0(Windows NT 6.1;Win64;x64) AppleWebKit/537.36
(KHTML,like Gecko) Chrome/52.0.2743.82 Safari/537.36','Host':'movie.douban.com'}
    for i in range(0,10):
        link = 'http://movie.douban.com/top250?start = ' + str(i * 25)
        r = requests.get(link,headers = headers,timeout = 10)
        print(str(i + 1),"页响应状态码: ",r.status_code)
        print(r.text)
get_movies()
```

运行程序，输出如下：

```
1 页响应状态码: 200
</p>
<div class = "star">
<span class = "rating45 - t"></span>
<span class = "rating_num" property = "v:average"> 8.8 </span>
<span property = "v:best" content = "10.0"></span>
<span> 244205 人评价</span>
</div>
<p class = "quote">
<span class = "inq">穷尽一生，我们要学会的，不过是彼此拥抱.</span>
</p>
</div>
</div>
</div>
</li>
<li>
<div class = "item">
<div class = "pic">
<em class = ""> 117 </em>
<a href = "https://movie.douban.com/subject/1291990/">
<img width = "100" alt = "爱在日落黄昏时" src = "https://img3.doubanio.com/view/photo/s_
ratio_poster/public/p1910924055.jpg" class = "">
…
```

这时，得到的结果只是网页的 HTML 代码，还需要从中提取需要的电影名称。下面代

码实现网页的内容解析：

```
import requests
from bs4 import BeautifulSoup
# 通过 find 定位标签
# BeautifulSoup 文档：https://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html
def bs_parse_movies(html):
    movie_list = []
    soup = BeautifulSoup(html, "html")
    # 查找所有 class 属性为 hd 的 div 标签
    div_list = soup.find_all('div', class_ = 'hd')
    # 获取每个 div 中的 a 中的 span(第一个),并获取其文本
    for each in div_list:
        movie = each.a.span.text.strip()
        movie_list.append(movie)
    return movie_list
# css 选择器定位标签
# 更多 ccs 选择器语法：http://www.w3school.com.cn/cssref/css_selectors.asp
# 注意：BeautifulSoup 并不是每个语法都支持
def bs_css_parse_movies(html):
    movie_list = []
    soup = BeautifulSoup(html, "lxml")
    # 查找所有 class 属性为 hd 的 div 标签下的 a 标签的第一个 span 标签
    div_list = soup.select('div.hd > a > span:nth-of-type(1)')
    # 获取每个 span 的文本
    for each in div_list:
        movie = each.text.strip()
        movie_list.append(movie)
    return movie_list
# Xpath 定位标签
# 更多 Xpath 语法：https://blog.csdn.net/gongbing798930123/article/details/78955597
def xpath_parse_movies(html):
    et_html = etree.HTML(html)
    # 查找所有 class 属性为 hd 的 div 标签下的 a 标签的第一个 span 标签
    urls = et_html.xpath("//div[@class = 'hd']/a/span[1]")
    movie_list = []
    # 获取每个 span 的文本
    for each in urls:
        movie = each.text.strip()
        movie_list.append(movie)
    return movie_list
def get_movies():
    headers = {
        'user-agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/52.0.2743.82 Safari/537.36',
        'Host': 'movie.douban.com'
    }
    link = 'https://movie.douban.com/top250'
    r = requests.get(link, headers = headers, timeout = 10)
    print("响应状态码:", r.status_code)
    if 200 != r.status_code:
        return None
    # 3 种定位元素的方式：
    # 普通 BeautifulSoup find
```

```
        return bs_parse_movies(r.text)
        return bs_css_parse_movies(r.text)
        return xpath_parse_movies(r.text)
    movies = get_movies()
    print(movies)
```

运行程序，输出如下：

```
响应状态码: 200
to this:
 BeautifulSoup(YOUR_MARKUP, "html5lib")
  markup_type=markup_type))
['肖申克的救赎', '霸王别姬', '这个杀手不太冷', '阿甘正传', '美丽人生', '泰坦尼克号', '千与千寻', '辛德勒的名单', '盗梦空间', '忠犬八公的故事', '机器人总动员', '三傻大闹宝莱坞', '海上钢琴师', '放牛班的春天', '楚门的世界', '大话西游之大圣娶亲', '星际穿越', '龙猫', '教父', '熔炉', '无间道', '疯狂动物城', '当幸福来敲门', '怦然心动', '触不可及']
```

3.13　习题

1. 请求头 Headers 提供了关于_____、_____或其他_____的信息。

2. 什么是静态网页？

3. 利用 pip 安装第三方库 wheel。

4. 利用 Requests 中常用功能获取个人计算机浏览器首页内容。

5. 利用 POST 发送 JSON 数据 str＝"loginAccount":"xx","password":"xxx","userType":"individua"}。