

DAY06

Day05回顾

控制台抓包

打开方式及常用选项

- 1 1、打开浏览器，F12打开控制台，找到Network选项卡
- 2 2、控制台常用选项
 - 3 1、Network：抓取网络数据包
 - 4 1、ALL：抓取所有的网络数据包
 - 5 2、XHR：抓取异步加载的网络数据包
 - 6 3、JS：抓取所有的JS文件
 - 7 2、Sources：格式化输出并打断点调试JavaScript代码，助于分析爬虫中一些参数
 - 8 3、Console：交互模式，可对JavaScript中的代码进行测试
- 9 3、抓取具体网络数据包后
 - 10 1、单击左侧网络数据包地址，进入数据包详情，查看右侧
 - 11 2、右侧：
 - 12 1、Headers：整个请求信息
General、Response Headers、Request Headers、Query String、Form Data
 - 13 2、Preview：对响应内容进行预览
 - 14 3、Response：响应内容

有道翻译过程梳理

- 1 1. 打开首页
- 2 2. 准备抓包：F12开启控制台
 - 3 3. 寻找地址
页面中输入翻译单词，控制台中抓取到网络数据包，查找并分析返回翻译数据的地址
 - 4 4. 发现规律
找到返回具体数据的地址，在页面中多输入几个单词，找到对应URL地址，分析对比 Network - All(或者XHR) - Form Data，发现对应的规律
 - 5 5. 寻找JS文件
右上角 ... -> Search -> 搜索关键字 -> 单击 -> 跳转到Sources，左下角格式化符号{}
 - 6 6. 查看JS代码
搜索关键字，找到相关加密方法
 - 7 7. 断点调试
 - 8 8. 完善程序

增量爬取思路

- 1 1、将爬取过的地址存放到数据库中
- 2 2、程序爬取时先到数据库中查询比对, 如果已经爬过则不会继续爬取

动态加载网站数据抓取

- 1 1、F12打开控制台, 页面动作抓取网络数据包
- 2 2、抓取json文件URL地址
- 3 # 控制台中 XHR : 异步加载的数据包
- 4 # XHR -> Query String Parameters(查询参数)

数据抓取最终梳理

- 1 # 响应内容中存在
- 2 1、确认抓取数据在响应内容中是否存在
- 3 2、分析页面结构, 观察URL地址规律
- 4 1、大体查看响应内容结构, 查看是否有更改 -- (百度视频案例)
- 5 2、查看页面跳转时URL地址变化, 查看是否新跳转 -- (民政部案例)
- 6 3、开始码代码进行数据抓取
- 7
- 8 # 响应内容中不存在
- 9 1、确认抓取数据在响应内容中是否存在
- 10 2、F12抓包, 开始刷新页面或执行某些行为, 主要查看XHR异步加载数据包
- 11 1、GET请求: Request Headers、Query String Paramters
- 12 2、POST请求: Request Headers、FormData
- 13 3、观察查询参数或者Form表单数据规律, 如果需要进一步抓包分析处理
- 14 1、比如有道翻译的 salt+sign, 抓取并分析JS做进一步处理
- 15 2、此处注意请求头中的cookie和referer以及User-Agent
- 16 4、使用res.json()获取数据, 利用列表或者字典的方法获取所需数据

Day06笔记

豆瓣电影数据抓取案例

■ 目标

- 1 1、地址: 豆瓣电影 - 排行榜 - 剧情
- 2 2、目标: 电影名称、电影评分

■ F12抓包 (XHR)

```
1 1、Request URL(基准URL地址) : https://movie.douban.com/j/chart/top_list?
2 2、Query String(查询参数)
3
4 # 抓取的查询参数如下:
5 type: 13 # 电影类型
6 interval_id: 100:90
7 action: ''
8 start: 0 # 每次加载电影的起始索引值 0 20 40 60
9 limit: 20 # 每次加载的电影数量
```

■ 代码实现 - 全站抓取 - 完美接口 - 指定类型所有电影信息

```
1 import requests
2 import time
3 import random
4 import re
5 from useragents import ua_list
6
7 class DoubanSpider(object):
8     def __init__(self):
9         self.url = 'https://movie.douban.com/j/chart/top_list?'
10        self.i = 0
11
12        # 获取随机headers
13        def get_headers(self):
14            headers = {'User-Agent': random.choice(ua_list)}
15
16            return headers
17
18        # 获取页面
19        def get_page(self, params):
20            headers = self.get_headers()
21            res = requests.get(url=self.url, params=params, headers=headers)
22            res.encoding = 'utf-8'
23            # 返回 python 数据类型
24            html = res.json()
25            self.parse_page(html)
26
27        # 解析并保存数据
28        def parse_page(self, html):
29            item = {}
30            # html为大列表 [{电影1信息}, {}, {}]
31            for one in html:
32                # 名称 + 评分
33                item['name'] = one['title'].strip()
34                item['score'] = float(one['score'].strip())
35                # 打印测试
36                print(item)
37                self.i += 1
38
39        # 获取电影总数
40        def total_number(self, type_number):
41            # F12抓包抓到的地址
```

```

42     url = 'https://movie.douban.com/j/chart/top_list_count?type=
43     {}&interval_id=100%3A90'.format(type_number)
44     headers = self.get_headers()
45     html = requests.get(url=url,headers=headers).json()
46     total = int(html['total'])
47
48     return total
49
50 # 获取所有电影的名字和对应type值
51 def get_all_type_films(self):
52     # 获取 类型和类型码
53     url = 'https://movie.douban.com/chart'
54     headers = self.get_headers()
55     html = requests.get(url=url,headers=headers).text
56     re_bds = r'<a href=.*?type_name=(.*?)&type=(.*?)&.*?</a>'
57     pattern = re.compile(re_bds,re.S)
58     r_list = pattern.findall(html)
59     # 存放所有类型和对应类型码大字典
60     type_dict = {}
61     menu = ''
62     for r in r_list:
63         type_dict[r[0].strip()] = r[1].strip()
64         # 获取input的菜单，显示所有电影类型
65         menu += r[0].strip() + '|'
66
67     return type_dict,menu
68
69 # 主函数
70 def main(self):
71     # 获取type的值
72     type_dict,menu = self.get_all_type_films()
73     menu = menu + '\n请做出你的选择:'
74     name = input(menu)
75     type_number = type_dict[name]
76     # 获取电影总数
77     total = self.total_number(type_number)
78     for start in range(0,(total+1),20):
79         params = {
80             'type' : type_number,
81             'interval_id' : '100:90',
82             'action' : '',
83             'start' : str(start),
84             'limit' : '20'
85         }
86         # 调用函数,传递params参数
87         self.get_page(params)
88         # 随机休眠1-3秒
89         time.sleep(random.randint(1,3))
90     print('电影数量:',self.i)
91
92 if __name__ == '__main__':
93     spider = DoubanSpider()
94     spider.main()

```

腾讯招聘数据抓取

■ 确定URL地址及目标

- 1、URL：百度搜索腾讯招聘 - 查看工作岗位
- 2、目标：职位名称、工作职责、岗位要求

■ 要求与分析

- 1、通过查看网页源码,得知所需数据均为 Ajax 动态加载
- 2、通过F12抓取网络数据包,进行分析
- 3、一级页面抓取数据：职位名称
- 4、二级页面抓取数据：工作职责、岗位要求

■ 一级页面json地址(index在变,timestamp未检查)

- 1 `https://careers.tencent.com/tencentcareer/api/post/Query?timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attrId=&keyword=&pageIndex={}&pageSize=10&language=zh-cn&area=cn`

■ 二级页面地址(postId在变,在一级页面中可拿到)

- 1 `https://careers.tencent.com/tencentcareer/api/post/ByPostId?timestamp=1563912374645&postId={}&language=zh-cn`

■ 代码实现

```
1 import requests
2 import json
3 import time
4 import random
5 from useragents import ua_list
6
7 class TencentSpider(object):
8     def __init__(self):
9         self.one_url = 'https://careers.tencent.com/tencentcareer/api/post/Query?
timestamp=1563912271089&countryId=&cityId=&bgIds=&productId=&categoryId=&parentCategoryId=&attr
rId=&keyword=&pageIndex={}&pageSize=10&language=zh-cn&area=cn'
10         self.two_url = 'https://careers.tencent.com/tencentcareer/api/post/ByPostId?
timestamp=1563912374645&postId={}&language=zh-cn'
11         # 打开文件
12         self.f = open('tencent.json', 'a')
13         # 存放抓取的item字典数据
14         self.item_list = []
15
16         # 获取响应内容函数
17         def get_page(self, url):
18             headers = {'User-Agent': random.choice(ua_list)}
19             html = requests.get(url=url, headers=headers).text
20             # json格式字符串 -> Python
21             html = json.loads(html)
22
```

```

23     return html
24
25 # 主线函数: 获取所有数据
26 def parse_page(self, one_url):
27     html = self.get_page(one_url)
28     item = {}
29     for job in html['Data']['Posts']:
30         # postId
31         post_id = job['PostId']
32         # 拼接二级地址, 获取职责和要求
33         two_url = self.two_url.format(post_id)
34         item['name'], item['duty'], item['require'] = self.parse_two_page(two_url)
35
36         print(item)
37         # 添加到大列表中
38         self.item_list.append(item)
39
40 # 解析二级页面函数
41 def parse_two_page(self, two_url):
42     html = self.get_page(two_url)
43     # 职位名称
44     name = html['Data']['RecruitPostName']
45     # 用replace处理一下特殊字符
46     duty = html['Data']['Responsibility']
47     duty = duty.replace('\r\n', '').replace('\n', '')
48     # 处理要求
49     require = html['Data']['Requirement']
50     require = require.replace('\r\n', '').replace('\n', '')
51
52     return name, duty, require
53
54 # 获取总页数
55 def get_numbers(self):
56     url = self.one_url.format(1)
57     html = self.get_page(url)
58     numbers = int(html['Data']['Count']) // 10 + 1
59
60     return numbers
61
62 def main(self):
63     number = self.get_numbers()
64     for page in range(1, 3):
65         one_url = self.one_url.format(page)
66         self.parse_page(one_url)
67
68     # 保存到本地json文件: json.dump
69     json.dump(self.item_list, self.f, ensure_ascii=False)
70     self.f.close()
71
72 if __name__ == '__main__':
73     spider = TencentSpider()
74     spider.main()

```

多线程爬虫

应用场景

- 1、多进程：CPU密集程序
- 2、多线程：爬虫(网络I/O)、本地磁盘I/O

知识点回顾

■ 队列

```
1 # 导入模块
2 from queue import Queue
3 # 使用
4 q = Queue()
5 q.put(url)
6 q.get() # 当队列为空时, 阻塞
7 q.empty() # 判断队列是否为空, True/False
```

■ 线程模块

```
1 # 导入模块
2 from threading import Thread
3
4 # 使用流程
5 t = Thread(target=函数名) # 创建线程对象
6 t.start() # 创建并启动线程
7 t.join() # 阻塞等待回收线程
8
9 # 如何创建多线程
10 t_list = []
11
12 for i in range(5):
13     t = Thread(target=函数名)
14     t_list.append(t)
15     t.start()
16
17 for t in t_list:
18     t.join()
```

小米应用商店抓取(多线程)

目标

- 1、网址：百度搜 - 小米应用商店，进入官网
 - 2、目标：所有应用分类
- 应用名称
- 应用链接

实现步骤

■ 1、确认是否为动态加载

- 1 1、页面局部刷新
- 2 2、右键查看网页源代码，搜索关键字未搜到
- 3 # 此网站为动态加载网站，需要抓取网络数据包分析

■ 2、F12抓取网络数据包

- 1 1、抓取返回json数据的URL地址 (Headers中的Request URL)
- 2 `http://app.mi.com/categotyAllListApi?page={}&categoryId=2&pageSize=30`
- 3
- 4 2、查看并分析查询参数 (headers中的Query String Parameters)
- 5 `page: 1`
- 6 `categoryId: 2`
- 7 `pageSize: 30`
- 8 # 只有page在变, 0 1 2 3 ... , 这样我们就可以通过控制page的值拼接多个返回json数据的URL地址

■ 将抓取数据保存到csv文件

- 1 # 注意多线程写入的线程锁问题
- 2 `from threading import Lock`
- 3 `lock = Lock()`
- 4 # 加锁
- 5 `lock.acquire()`
- 6 python语句
- 7 # 释放锁
- 8 `lock.release()`

■ 整体思路

- 1 1、在 `__init__(self)` 中创建文件对象，多线程操作此对象进行文件写入
- 2 `self.f = open('xiaomi.csv', 'a', newline='')`
- 3 `self.writer = csv.writer(self.f)`
- 4 `self.lock = Lock()`
- 5 2、每个线程抓取1页数据后将数据进行文件写入，写入文件时需要加锁
- 6 `def parse_html(self):`
- 7 `app_list = []`
- 8 `for xxx in xxx:`
- 9 `app_list.append([name, link, typ])`
- 10 `self.lock.acquire()`
- 11 `self.wirter.writerow(app_list)`
- 12 `self.lock.release()`
- 13 3、所有数据抓取完成关闭文件
- 14 `def main(self):`
- 15 `self.f.close()`

■ 代码实现

- 1 `import requests`
- 2 `from threading import Thread`
- 3 `from queue import Queue`


```

4 import time
5 from useragents import ua_list
6 from lxml import etree
7 import csv
8 from threading import Lock
9 import random
10
11 class XiaomiSpider(object):
12     def __init__(self):
13         self.url = 'http://app.mi.com/categoryAllListApi?page={}&categoryId={}&pageSize=30'
14         # 存放所有URL地址的队列
15         self.q = Queue()
16         self.i = 0
17         # 存放所有类型id的空列表
18         self.id_list = []
19         # 打开文件
20         self.f = open('xiaomi.csv', 'a')
21         self.writer = csv.writer(self.f)
22         # 创建锁
23         self.lock = Lock()
24
25
26     def get_cateid(self):
27         # 请求
28         url = 'http://app.mi.com/'
29         headers = { 'User-Agent': random.choice(ua_list)}
30         html = requests.get(url=url, headers=headers).text
31         # 解析
32         parse_html = etree.HTML(html)
33         xpath_bds = '//ul[@class="category-list"]/li'
34         li_list = parse_html.xpath(xpath_bds)
35         for li in li_list:
36             typ_name = li.xpath('./a/text()')[0]
37             typ_id = li.xpath('./a/@href')[0].split('/')[1]
38             # 计算每个类型的页数
39             pages = self.get_pages(typ_id)
40             self.id_list.append( (typ_id, pages) )
41
42         # 入队列
43         self.url_in()
44
45     # 获取counts的值并计算页数
46     def get_pages(self, typ_id):
47         # 每页返回的json数据中,都有count这个key
48         url = self.url.format(0, typ_id)
49         html = requests.get(
50             url=url,
51             headers={'User-Agent': random.choice(ua_list)}
52         ).json()
53         count = html['count']
54         pages = int(count) // 30 + 1
55
56         return pages
57
58     # url入队列
59     def url_in(self):
60         for id in self.id_list:

```

```

61     # id为元组,('2',pages)
62     for page in range(1,id[1]+1):
63         url = self.url.format(page,id[0])
64         # 把URL地址入队列
65         self.q.put(url)
66
67 # 线程事件函数: get() - 请求 - 解析 - 处理数据
68 def get_data(self):
69     while True:
70         if not self.q.empty():
71             url = self.q.get()
72             headers = {'User-Agent':random.choice(ua_list)}
73             html = requests.get(url=url,headers=headers).json()
74             self.parse_html(html)
75         else:
76             break
77
78 # 解析函数
79 def parse_html(self,html):
80     # 存放1页的数据 - 写入到csv文件
81     app_list = []
82
83     for app in html['data']:
84         # 应用名称 + 链接 + 分类
85         name = app['displayName']
86         link = 'http://app.mi.com/details?id=' + app['packageName']
87         typ_name = app['level1CategoryName']
88         # 把每一条数据放到app_list中,目的是为了 writerows()
89         app_list.append([name,typ_name,link])
90
91         print(name,typ_name)
92         self.i += 1
93
94     # 开始写入1页数据 - app_list
95     self.lock.acquire()
96     self.writer.writerows(app_list)
97     self.lock.release()
98
99 # 主函数
100 def main(self):
101     # URL入队列
102     self.get_cateid()
103     t_list = []
104     # 创建多个线程
105     for i in range(1):
106         t = Thread(target=self.get_data)
107         t_list.append(t)
108         t.start()
109
110     # 回收线程
111     for t in t_list:
112         t.join()
113
114     # 关闭文件
115     self.f.close()
116     print('数量:',self.i)
117

```

```

118 if __name__ == '__main__':
119     start = time.time()
120     spider = XiaomiSpider()
121     spider.main()
122     end = time.time()
123     print('执行时间:%.2f' % (end-start))

```

cookie模拟登录

适用网站及场景

- 1 抓取需要登录才能访问的页面

cookie和session机制

- 1 # http协议为无连接协议
- 2 cookie: 存放在客户端浏览器
- 3 session: 存放在Web服务器

人人网登录案例

▪ 方法一 - 登录网站手动抓取Cookie

- 1 1、先登录成功1次,获取到携带登录信息的Cookie
- 2 登录成功 - 个人主页 - F12抓包 - 刷新个人主页 - 找到主页的包(profile)
- 3 2、携带着cookie发请求
- 4 ** Cookie
- 5 ** User-Agent

```

1 # 1、将self.url改为 个人主页的URL地址
2 # 2、将Cookie的值改为 登录成功的Cookie值
3 import requests
4 from lxml import etree
5
6 class RenrenLogin(object):
7     def __init__(self):
8         self.url = 'xxxxxxx'
9         self.headers = {
10             'Cookie': 'xxxxxx',
11             'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
12             Gecko) Chrome/76.0.3809.100 Safari/537.36'
13         }
14
15     def get_html(self):
16         html = requests.get(url=self.url, headers=self.headers).text
17         self.parse_html(html)

```

```

18     def parse_html(self,html):
19         parse_html = etree.HTML(html)
20         r_list = parse_html.xpath('//*[@id="operate_area"]/div[1]/ul/li[1]/span/text()')
21         print(r_list)
22
23     if __name__ == '__main__':
24         spider = RenrenLogin()
25         spider.get_html()

```

▪ 方法二 - requests模块处理Cookie

原理思路及实现

```

1  # 1. 思路
2  requests模块提供了session类,来实现客户端和服务端的会话保持
3
4  # 2. 原理
5  1、实例化session对象
6     session = requests.session()
7  2、让session对象发送get或者post请求
8     res = session.post(url=url,data=data,headers=headers)
9     res = session.get(url=url,headers=headers)
10
11 # 3. 思路梳理
12 浏览器原理：访问需要登录的页面会带着之前登录过的cookie
13 程序原理：同样带着之前登录的cookie去访问 - 由session对象完成
14 1、实例化session对象
15 2、登录网站：session对象发送请求,登录对应网站,把cookie保存在session对象中
16 3、访问页面：session对象请求需要登录才能访问的页面,session能够自动携带之前的这个cookie,进行请求

```

具体步骤

```

1  1、寻找Form表单提交地址 - 寻找登录时POST的地址
2     查看网页源码,查看form表单,找action对应的地址: http://www.renren.com/PLogin.do
3
4  2、发送用户名和密码信息到POST的地址
5     * 用户名和密码信息以什么方式发送? -- 字典
6     键 : <input>标签中name的值(email,password)
7     值 : 真实的用户名和密码
8     post_data = {'email':'','password':''}
9
10    session = requests.session()
11    session.post(url=url,data=data)

```

程序实现

```

1  # 把Formdata中的 email 和 password 的改为自己真实的用户名和密码
2  import requests
3  from lxml import etree
4
5  class RenrenSpider(object):
6     def __init__(self):
7         self.post_url = 'http://www.renren.com/PLogin.do'
8         self.get_url = 'http://www.renren.com/967469305/profile'

```

```

9      # 实例化session对象
10     self.session = requests.session()
11
12     def get_html(self):
13         # email和密码为<input>节点中name的属性值
14         form_data = {
15             'email' : 'xxxx',
16             'password' : 'xxx'
17         }
18         # 先session.post()
19         self.session.post(url=self.post_url,data=form_data)
20         # 再session.get()
21         html = self.session.get(url=self.get_url).text
22         self.parse_html(html)
23
24     def parse_html(self,html):
25         parse_html = etree.HTML(html)
26         r_list = parse_html.xpath('//li[@class="school"]/span/text()')
27         print(r_list)
28
29     if __name__ == '__main__':
30         spider = RenrenSpider()
31         spider.get_html()

```

▪ 方法三

原理

- 1 1、把抓取到的cookie处理为字典
- 2 2、使用requests.get()中的参数:cookies

处理cookie为字典

```

1  # 处理cookies为字典
2  cookies_dict = {}
3  cookies = 'xxxx'
4  for kv in cookies.split('; '):
5      cookies_dict[kv.split('=')[0]] = kv.split('=')[1]

```

代码实现

```

1  import requests
2  from lxml import etree
3
4  class RenrenLogin(object):
5      def __init__(self):
6          self.url = 'http://www.renren.com/967469305/profile'
7          self.headers = {
8              'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/76.0.3809.100 Safari/537.36'
9          }
10
11     # 获取字典形式cookie的函数
12     def get_cookie_dict(self):

```

```

13     cookie_dict = {}
14     cookies = 'anonymid=jy87mc5fx4xvjj; _r01_=1; jebe_key=a04238bc-adc0-4418-a770-
519d74219f15%7C2e9beece3ead42fe6a26739d515f14df%7C1563911475551%7C1%7C1563911475689;
ln_uact=13603263409; depovince=GW; jebecookies=3720f008-2502-4422-acfe-8b78b4c3611d||||;
JSESSIONID=abcUijruA6U375Qz-tHZw; ick_login=60cb66a4-e407-4fd5-a2ee-1eae3220a102;
_de=4DBCFCC17D9E50C8C92BCDC45CC5C3B7; p=415429a0f0b3067e9061fd8387c269c45; first_login_flag=1;
ln_hurl=http://hdn.xnimg.cn/photos/hdn421/20190815/1435/main_91u0_81d40000ca0d1986.jpg;
t=da27ae8094836b90d439e88e21ed73ac5; societyguester=da27ae8094836b90d439e88e21ed73ac5;
id=967469305; xnsid=1eafd54d; ver=7.0; loginfrom=null; jebe_key=a04238bc-adc0-4418-a770-
519d74219f15%7C2012cb2155debcd0710a4bf5a73220e8%7C1567148226573%7C1%7C1567148227902;
wp_fold=0'
15     for kv in cookies.split('; '):
16         # kv: 'td_cookie=184xxx'
17         key = kv.split('=')[0]
18         value = kv.split('=')[1]
19         cookie_dict[key] = value
20
21     return cookie_dict
22
23 def get_html(self):
24     # 获取cookies
25     cookies = self.get_cookie_dict()
26     html = requests.get(
27         url=self.url,
28         headers=self.headers,
29         cookies=cookies,
30     ).text
31     self.parse_html(html)
32
33 def parse_html(self,html):
34     parse_html = etree.HTML(html)
35     r_list = parse_html.xpath('//*[@id="operate_area"]/div[1]/ul/li[1]/span/text()')
36     print(r_list)
37
38 if __name__ == '__main__':
39     spider = RenrenLogin()
40     spider.get_html()

```

json解析模块

json.loads(json)

■ 作用

```
1 把json格式的字符串转为Python数据类型
```

■ 示例

```
1  html_json = json.loads(res.text)
```

json.dumps(python)

■ 作用

```
1 把 python 类型 转为 json 类型
```

■ 示例

```
1 import json
2
3 # json.dumps()之前
4 item = {'name':'QQ','app_id':1}
5 print('before dumps',type(item)) # dict
6 # json.dumps之后
7 item = json.dumps(item)
8 print('after dumps',type(item)) # str
```

json.load(f)

作用

```
1 将json文件读取,并转为python类型
```

示例

```
1 import json
2
3 with open('D:\\spider_test\\xiaomi.json','r') as f:
4     data = json.load(f)
5
6 print(data)
```

json.dump(python,f,ensure_ascii=False)

■ 作用

```
1 把python数据类型 转为 json格式的字符串
2 # 一般让你把抓取的数据保存为json文件时使用
```

■ 参数说明

```
1 第1个参数: python类型的数据(字典, 列表等)
2 第2个参数: 文件对象
3 第3个参数: ensure_ascii=False # 序列化时编码
```

■ 示例1

```
1 import json
2
3 item = {'name':'QQ','app_id':1}
4 with open('小米.json','a') as f:
5     json.dump(item,f,ensure_ascii=False)
```

■ 示例2

```
1 import json
2
3 item_list = []
4 for i in range(3):
5     item = {'name':'QQ','id':i}
6     item_list.append(item)
7
8 with open('xiaomi.json','a') as f:
9     json.dump(item_list,f,ensure_ascii=False)
```

练习: 将腾讯招聘数据存入到json文件

json模块总结

```
1 # 爬虫最常用
2 1、数据抓取 - json.loads(html)
3     将响应内容由: json 转为 python
4 2、数据保存 - json.dump(item_list,f,ensure_ascii=False)
5     将抓取的数据保存到本地 json文件
6
7 # 抓取数据一般处理方式
8 1、txt文件
9 2、csv文件
10 3、json文件
11 4、MySQL数据库
12 5、MongoDB数据库
13 6、Redis数据库
```

今日作业

```
1 1、多线程改写 - 腾讯招聘案例
2 2、多线程改写 - 链家二手房案例
3 3、尝试破解百度翻译
```