

# SPIDER-DAY07

## 1. scrapy处理cookie

### 1.1 方法一

```
1  【1】原理
2      利用settings.py中的 DEFAULT_REQUEST_HEADERS 参数
3  【2】实现
4      2.1》COOKIES_ENABLED = False
5      2.2》DEFAULT_REQUEST_HEADERS = {'Cookie':''}
```

### 1.2 方法二

```
1  【1】原理
2      利用爬虫文件中, scrapy.Request()中的cookies参数
3  【2】步骤
4      2.1》COOKIES_ENABLED = True
5      2.2》爬虫文件中,将抓取的cookie处理为字典,作为Request()参数
6          yield scrapy.Request(url=...,...,cookies={})
7
8  【处理Cookie为字典】
9  cook_str = 'antipas=1QT55g4e3k8p281404346979S; uuid=75783ae2-b0d2-4698-adf8-e4c0fac9a884;
10 user_city_id=12; clueSourceCode=%2A%2300'
11
12 def get_cookies(cook_str):
13     cookies = {}
14     for kv in cook_str.split('; '):
15         cookies[kv.split('=')[0]] = kv.split('=')[1]
16
17     return cookies
```

## 2. 腾讯招聘爬虫

- scrapy项目代码

```
1 见day07笔记: Tencent 文件夹
2  【1】一级页面
3      提取数据: 每个职位的PostId
4  【2】二级页面
5      提取数据: 1个职位的 名称、地点、类别、发布时间、职责、要求
```

## ■ 建库建表SQL

```
1  create database tencentdb charset utf8;
2  use tencentdb;
3  create table tencenttab(
4  job_name varchar(200),
5  job_type varchar(200),
6  job_duty varchar(2000),
7  job_require varchar(2000),
8  job_add varchar(100),
9  job_time varchar(100)
10 )charset=utf8;
```

## ■ MySQL数据持久化实现

```
1  【1】 pipelines.py新建MySQL管道类
2  from .settings import *
3  import pymysql
4
5  class TencentMysqlPipeline:
6      def open_spider(self, spider):
7          self.db = pymysql.connect('localhost', 'root', '123456', 'tencentdb',
8          charset='utf8')
9          self.cur = self.db.cursor()
10         self.ins = 'insert into tencenttab values(%s,%s,%s,%s,%s,%s)'
11
12         def process_item(self, item, spider):
13             li = [
14                 item['job_name'],
15                 item['job_type'],
16                 item['job_duty'],
17                 item['job_require'],
18                 item['job_add'],
19                 item['job_time'],
20             ]
21             self.cur.execute(self.ins, li)
22             self.db.commit()
23
24             return item
25
26         def close_spider(self, item, spider):
27             self.cur.close()
28             self.db.close()
29
30         【2】 settings.py添加
31         ITEM_PIPELINES = {
32             # 在原来基础上添加MySQL的管道
33             'Tencent.pipelines.TencentMysqlPipeline': 200,
34         }
```

## ■ MongoDB数据持久化实现

```
1  【1】 pipelines.py中新建MongoDB管道类
2  from .settings import *
3  import pymongo
4
5  class TencentMongoPipeline:
6      def open_spider(self, spider):
7          self.conn = pymongo.MongoClient(MONGO_HOST, MONGO_PORT)
8          self.db = self.conn[MONGO_DB]
9          self.myset = self.db[MONGO_SET]
10
11      def process_item(self, item, spider):
12          self.myset.insert_one(dict(item))
13
14  【2】 settings.py中添加
15  ITEM_PIPELINES = {
16      # 添加MongoDB管道
17      'Tencent.pipelines.TencentMongoPipeline': 400,
18  }
```

## ■ csv及json数据持久化实现

```
1  【1】 csv
2      scrapy crawl tencent -o tencent.csv
3
4  【2】 json
5      settings.py中添加变量: FEED_EXPORT_ENCODING = 'utf-8'
6      scrapy crawl tencent -o tencent.json
```

## 3. scrapy.Request()参数

```
1  【1】 url      : 指定URL地址
2  【2】 callback  : 指定解析函数
3  【3】 meta={}   : 不同解析函数间传递数据
4  【4】 dont_filter: 是否参与调度器的去重,默认为False,设置为True不去重
5  【5】 cookies={} : 请求头中的cookie
```

## 4. 分布式爬虫

### 4.1 分布式爬虫概述

- 1   **【1】原理**
- 2       多台主机共享1个爬取队列
- 3
- 4   **【2】实现**
- 5       2.1) 重写scrapy调度器(scrapy\_redis模块)
- 6       2.2) `sudo pip3 install scrapy_redis`
- 7
- 8   **【3】为什么使用redis**
- 9       3.1) Redis基于内存,速度快
- 10      3.2) Redis非关系型数据库,Redis中集合,存储每个request的指纹

## 4.2 scrapy\_redis详解

### ▪ GitHub地址

```
1 | https://github.com/rmax/scrapy-redis
```

### ▪ settings.py说明

```
1 | # 重新指定调度器: 启用Redis调度存储请求队列
2 | SCHEDULER = "scrapy_redis.scheduler.Scheduler"
3 |
4 | # 重新指定去重机制: 确保所有的爬虫通过Redis去重
5 | DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
6 |
7 | # 不清除Redis队列: 暂停/恢复/断点续爬(默认清除为False,设置为True不清除)
8 | SCHEDULER_PERSIST = True
9 |
10 | # redis管道
11 | ITEM_PIPELINES = {
12 |     'scrapy_redis.pipelines.RedisPipeline': 300
13 | }
14 |
15 | #指定连接到redis时使用的端口和地址
16 | REDIS_HOST = 'localhost'
17 | REDIS_PORT = 6379
```

## 4.3 腾讯招聘分布式爬虫

### ▪ 分布式爬虫完成步骤

- 1   **【1】** 首先完成非分布式scrapy爬虫 : 正常scrapy爬虫项目抓取
- 2   **【2】** 设置,部署成为分布式爬虫

### ▪ 分布式环境说明

```
1 【1】 分布式爬虫服务器数量：2（其中1台Windows,1台Ubuntu虚拟机）
2 【2】 服务器分工：
3     2.1) Windows : 负责数据抓取
4     2.2) Ubuntu : 负责URL地址统一管理,同时负责数据抓取
```

#### ■ 腾讯招聘分布式爬虫 - 数据同时存入1个Redis数据库

```
1 【1】 完成正常scrapy项目数据抓取（非分布式 - 拷贝之前的Tencent）
2
3 【2】 设置settings.py, 完成分布式设置
4     2.1-必须) 使用scrapy_redis的调度器
5         SCHEDULER = "scrapy_redis.scheduler.Scheduler"
6
7     2.2-必须) 使用scrapy_redis的去重机制
8         DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
9
10    2.3-必须) 定义redis主机地址和端口号
11        REDIS_HOST = '192.168.1.107'
12        REDIS_PORT = 6379
13
14    2.4-非必须) 是否清除请求指纹,True:不清除 False:清除（默认）
15        SCHEDULER_PERSIST = True
16
17    2.5-非必须) 在ITEM_PIPELINES中添加redis管道,数据将会存入redis数据库
18        'scrapy_redis.pipelines.RedisPipeline': 200
19
20 【3】 把代码原封不动的拷贝到分布式中的其他爬虫服务器,同时开始运行爬虫
21
22 【结果】：多台机器同时抓取,数据会统一存到Ubuntu的redis中, 而且所抓数据不重复
```

#### ■ 腾讯招聘分布式爬虫 - 数据存入MySQL数据库

```
1 """和数据存入redis步骤基本一样,只是变更一下管道和MySQL数据库服务器的IP地址"""
2 【1】 settings.py
3     1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
4     1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
5     1.3) SCHEDULER_PERSIST = True
6     1.4) REDIS_HOST = '192.168.1.105'
7     1.5) REDIS_PORT = 6379
8     1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}
9     1.7) MYSQL_HOST = '192.168.1.105'
10
11 【2】 将代码拷贝到分布式中所有爬虫服务器
12
13 【3】 多台爬虫服务器同时运行scrapy爬虫
14
15 # 赠送腾讯MySQL数据库建库建表语句
16 """
17 create database tencentdb charset utf8;
18 use tencentdb;
19 create table tencenttab(
20 job_name varchar(100),
21 job_type varchar(200),
22 job_duty varchar(5000),
23 job_require varchar(5000),
```

```
24 | job_address varchar(200),
25 | job_time varchar(200)
26 | )charset=utf8;
27 | ""
```

## 5. 图形验证码处理

### 5.1 机器视觉概述

- 1   **【1】作用**
- 2       处理图形验证码
- 3
- 4   **【2】三个重要概念** - OCR、tesseract-ocr、pytesseract
- 5       2.1) OCR
- 6           光学字符识别(Optical Character Recognition),通过扫描等光学输入方式将各种票据、报刊、书籍、
- 7           文稿及其它印刷品的文字转化为图像信息,再利用文字识别技术将图像信息转化为电子文本
- 8       2.2) tesseract-ocr
- 9           OCR的一个底层识别库(不是模块,不能导入),由Google维护的开源OCR识别库
- 10
- 11       2.3) pytesseract
- 12           Python模块,可调用底层识别库,是对tesseract-ocr做的一层Python API封装

### 5.2 安装

#### ■ 安装tesseract-ocr

- 1   **【1】安装tesseract-ocr**
- 2       1.1》Ubuntu安装 : `sudo apt-get install tesseract-ocr`
- 3       1.2》Windows安装
- 4           下载安装包,双击安装
- 5           添加到环境变量(Path)
- 6   **【2】测试 (终端 | cmd命令行)**
- 7       `tesseract xxx.jpg 文件名`

#### ■ 安装pytesseract

```

1  【1】 安装
2      sudo pip3 install pytesseract
3
4  【2】 使用示例
5      import pytesseract
6      # Python图片处理库
7      from PIL import Image
8
9      # 创建图片对象
10     img = Image.open('test1.jpg')
11     # 图片转字符串
12     result = pytesseract.image_to_string(img)
13     print(result)

```

#### ■ 面试问题: 如何处理爬虫中遇到的验证码

```

1  【1】 图形验证码
2      简单的图形验证码, 我使用tesseract-ocr去处理
3      对于一些复杂的验证码, 我们使用在线打码(图鉴、云打码)
4  【2】 滑块、缺口验证码
5      使用selenium处理
6      或者使用人工打码

```

## 6. 滑块缺口验证码

### 6.1 豆瓣网登录爬虫

#### 6.1.1 项目需求

```

1  【1】 URL地址: https://www.douban.com/
2  【2】 先输入几次错误的密码, 让登录出现滑块缺口验证, 以便于我们破解
3  【3】 模拟人的行为(总距离: 200)
4      3.1) 先快速滑动一部分距离(滑动160)
5      3.2) 剩余距离(40): 先匀加速( $40 \times 4 / 5 = 32$ ), 再匀减速( $40 \times 1 / 5 = 8$ )
6  【4】 详细看代码注释

```

#### 6.1.2 项目实现

```

1  """
2  说明: 先输入几次错误的密码, 出现滑块缺口验证码
3  """
4  from selenium import webdriver
5  # 导入鼠标事件类
6  from selenium.webdriver import ActionChains
7  import time
8
9  # 加速度函数
10 def get_tracks(distance):
11     """

```

```

12     拿到移动轨迹，模仿人的滑动行为，先匀加速后匀减速
13     匀变速运动基本公式：
14     ① $v=v_0+at$ 
15     ② $s=v_0t+\frac{1}{2}at^2$ 
16     """
17     # 初速度
18     v = 0
19     # 单位时间为0.3s来统计轨迹，轨迹即0.3s内的位移
20     t = 0.3
21     # 位置/轨迹列表，列表内的一个元素代表0.3s的位移
22     tracks = []
23     # 当前的位移
24     current = 0
25     # 到达mid值开始减速
26     mid = distance*4/5
27     while current < distance:
28         if current < mid:
29             # 加速度越小，单位时间内的位移越小，模拟的轨迹就越多越详细
30             a = 2
31         else:
32             a = -3
33
34         # 初速度
35         v0 = v
36         # 0.3秒内的位移
37         s = v0*t+0.5*a*(t**2)
38         # 当前的位置
39         current += s
40         # 添加到轨迹列表
41         tracks.append(round(s))
42         # 速度已经达到v，该速度作为下次的初速度
43         v = v0 + a*t
44     return tracks
45     # tracks: [第一个0.3秒的移动距离, 第二个0.3秒的移动距离, ...]
46
47     # 1、打开豆瓣官网 - 并将窗口最大化
48     driver = webdriver.Chrome()
49     driver.get(url='https://www.douban.com/')
50     driver.maximize_window()
51
52     # 2、切换到iframe子页面
53     iframe_node = driver.find_element_by_xpath('//div[@class="login"]/iframe')
54     driver.switch_to.frame(iframe_node)
55     # 3、密码登录 + 用户名 + 密码 + 登录豆瓣
56     driver.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
57     driver.find_element_by_xpath('//*[@id="username"]').send_keys('15110225726')
58     driver.find_element_by_xpath('//*[@id="password"]').send_keys('aaa')
59
60     while True:
61         try:
62             driver.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
63             time.sleep(3)
64
65             # 4、切换到新的iframe子页面 - 滑块验证
66             driver.switch_to.frame('tcaptcha_iframe')
67
68             # 5、按住开始滑动位置按钮 - 先移动180个像素

```



```

69     node = driver.find_element_by_xpath('//*[@id="tcaptcha_drag_button"]')
70     # click_and_hold(): 鼠标按住某个节点并保持
71     ActionChains(driver).click_and_hold(node).perform()
72     # 鼠标移动到距离某个节点水平 及 垂直的距离
73     ActionChains(driver).move_to_element_with_offset(to_element=node, xoffset=180,
yoffset=0).perform()
74     # 6、使用加速度函数移动剩下的距离-25个像素
75     # tracks: []
76     tracks = get_tracks(25)
77     for track in tracks:
78         # 鼠标移动到距离当前位置水平 及 垂直的距离
79         ActionChains(driver).move_by_offset(xoffset=track, yoffset=0).perform()
80
81     # 7、延迟释放鼠标: release()
82     time.sleep(0.5)
83     # release(): 释放鼠标
84     ActionChains(driver).release().perform()
85 except:
86     pass

```

## 7. Fiddler抓包工具

### 7.1 配置抓取浏览器数据包

#### ■ 配置Fiddler

```

1  【1】 Tools -> Options -> HTTPS
2      1.1) 添加证书信任: 勾选 Decrypt Https Traffic 后弹出窗口, 一路确认
3      1.2) 设置只抓浏览器的包: ...from browsers only
4
5  【2】 Tools -> Options -> Connections
6      2.1) 设置监听端口 (默认为8888)
7
8  【3】 配置完成后重启Fiddler ('重要')
9      3.1) 关闭Fiddler,再打开Fiddler

```

#### ■ 配置浏览器代理

```

1  【1】 安装Proxy SwitchyOmega谷歌浏览器插件
2
3  【2】 配置代理
4      2.1) 点击浏览器右上角插件SwitchyOmega -> 选项 -> 新建情景模式 -> myproxy(名字) -> 创建
5      2.2) 输入 HTTP:// 127.0.0.1 8888
6      2.3) 点击 : 应用选项
7
8  【3】 点击右上角SwitchyOmega可切换代理
9
10 【注意】: 一旦切换了自己创建的代理,则必须要打开Fiddler才可以上网

```

## 7.2 Fiddler使用说明

### ■ Fiddler常用菜单

- 1   【1】 Inspector : 查看数据包详细内容
- 2       2.1) 整体分为请求和响应两部分
- 3
- 4   【2】 Inspector常用菜单
- 5       2.1) Headers : 请求头信息
- 6       2.2) WebForms: POST请求Form表单数据 : <body>
- 7               GET请求查询参数: <QueryString>
- 8       2.3) Raw : 将整个请求显示为纯文本

## 8. 移动端app数据抓取

### 8.1 方式一(手机&Fiddler)

- 1   设置方法见文件夹 - 移动端抓包配置

### 8.2 方式二(F12模拟)

#### 8.2.1 有道翻译手机版爬虫

```
1  import requests
2  from lxml import etree
3
4  word = input('请输入要翻译的单词:')
5
6  post_url = 'http://m.youdao.com/translate'
7  post_data = {
8      'inputtext':word,
9      'type':'AUTO'
10 }
11
12 html = requests.post(url=post_url,data=post_data).text
13 parse_html = etree.HTML(html)
14 xpath_bds = '//ul[@id="translateResult"]/li/text()'
15 result = parse_html.xpath(xpath_bds)[0]
16
17 print(result)
```

