

# Day09回顾

## settings.py常用变量

```
1  【1】 settings.py中常用变量
2      2.1) 设置日志级别
3          LOG_LEVEL = ''
4      2.2) 保存到日志文件(不在终端输出)
5          LOG_FILE = ''
6      2.3) 设置数据导出编码(主要针对于json文件)
7          FEED_EXPORT_ENCODING = 'utf-8'
8      2.4) 设置User-Agent
9          USER_AGENT = ''
10     2.5) 设置最大并发数(默认为16)
11         CONCURRENT_REQUESTS = 32
12     2.6) 下载延迟时间(每隔多长时间请求一个网页)
13         DOWNLOAD_DELAY = 1
14     2.7) 请求头
15         DEFAULT_REQUEST_HEADERS = {'User-Agent': 'Mozilla/'}
16     2.8) 添加项目管道
17         ITEM_PIPELINES = {'项目目录名.pipelines.类名': 优先级}
18     2.9) cookie(默认禁用,取消注释-True|False都为开启)
19         COOKIES_ENABLED = False
20     2.10) 非结构化数据存储路径
21         IMAGES_STORE = '/home/tarena/images/'
22         FILES_STORE = '/home/tarena/files/'
23     2.11) 添加下载器中间件
24         DOWNLOADER_MIDDLEWARES = {'项目名.middlewares.类名': 200}
```

## 非结构化数据抓取

```
1  【1】 spider
2      yield item['链接']
3
4  【2】 pipelines.py
5      from scrapy.pipelines.images import ImagesPipeline
6      import scrapy
7      class TestPipeline(ImagesPipeline):
8          def get_media_requests(self, item, info):
9              yield scrapy.Request(url=item['url'], meta={'item': item})
10
11          def file_path(self, request, response=None, info=None):
12              name = request.meta['item']
```

```

13         filename = name
14         return filename
15
16 【3】 settings.py
17 IMAGES_STORE = 'D:/Spider/images'

```

## Post请求

### ▪ 方法

```

1 scrapy.FormRequest(url=url, formdata=formdata, callback=self.xxx)

```

### ▪ 使用cookie

```

1 【1】 方法1
2     COOKIES_ENABLED = False
3     DEFAULT_REQUEST_HEADERS = {'Cookie': 'xxx'}
4
5 【2】 方法2
6     COOKIES_ENABLED = True
7     yield scrapy.Request(url=url, cookies={}, callback=self.xxx)
8     yield scrapy.FormRequest(url=url, formdata={}, cookies={}, callback=self.xxx)
9
10 【3】 方法3
11     COOKIES_ENABLED = True
12     class XxxCookieDownloaderMiddleware(object):
13         def process_request(self, request, spider):
14             request.cookies = {}

```

### ▪ 对象属性

```

1 【1】 请求对象request属性
2     1.1) request.url      : 请求URL地址
3     1.2) request.headers : 请求头 - 字典
4     1.3) request.meta    : item数据传递、定义代理
5     1.4) request.cookies : Cookie
6
7 【2】 响应对象response属性
8     2.1) response.url     : 返回实际数据的URL地址
9     2.2) response.text    : 响应内容 - 字符串
10    2.3) response.body     : 响应内容 - 字节串
11    2.4) response.encoding : 响应字符编码
12    2.5) response.status   : HTTP响应码

```

# Day10笔记

# 中间件

## 设置中间件(随机User-Agent)

### ■ 大量UA设置 - 使用middlewares.py中间件

```
1  【1】middlewares.py新建中间件类
2      class RandomUseragentMiddleware(object):
3          def process_request(self, request, spider):
4              agent = UserAgent().random
5              request.headers['User-Agent'] = agent
6
7  【2】settings.py添加此下载器中间件
8      DOWNLOADER_MIDDLEWARES = {'': 优先级}
```

## 设置中间件(随机代理)

### ■ 代理IP中间件

```
1  class RandomProxyDownloaderMiddleware(object):
2      def process_request(self, request, spider):
3          request.meta['proxy'] = xxx
4
5      # 捕获异常的方法,一旦代理不能用,会被此方法捕获,并重新包装请求再次发送
6      def process_exception(self, request, exception, spider):
7          return request
```

## 设置中间件(Cookie)

### ■ Cookie中间件

```
1  class BaiduCookieDownloaderMiddleware(object):
2      def process_request(self, request, spider):
3          cookies = self.get_cookies()
4          print('middleware3', cookies)
5          # 利用请求对象request的cookies属性
6          request.cookies = cookies
7
8
9      def get_cookies(self):
10         costr = ''
11         cookies = {}
12         for kv in costr.split('; '):
13             cookies[kv.split('=')[0]] = kv.split('=')[1]
14
15         return cookies
```

## ■ 练习

```
1 | 将有道翻译案例的cookie使用中间件的方式来实现
```

# 分布式爬虫

## ■ 分布式爬虫介绍

```
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
```

## ■ 为什么使用redis

```
1 | 【1】 Redis基于内存,速度快
2 | 【2】 Redis非关系型数据库,Redis中集合,存储每个request的指纹
```

# 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 | # 优先级队列 (默认)
11 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.PriorityQueue'
12 | #可选用的其它队列
13 | # 先进先出
14 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
15 | # 后进先出
16 | SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.LifoQueue'
17 |
18 | # redis管道
19 | ITEM_PIPELINES = {
20 |     'scrapy_redis.pipelines.RedisPipeline': 300
```

```
21 }
22
23 #指定连接到redis时使用的端口和地址
24 REDIS_HOST = 'localhost'
25 REDIS_PORT = 6379
```

## 腾讯招聘分布式改写

### ■ 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(1000),
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 """

```

## 新的篇章

# 腾讯招聘分布式改写之方法二

### ■ 使用redis\_key改写（同时存入MySQL数据库）

```

1 【1】 settings.py和方法一中写法一致
2     1.1) SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
3     1.2) DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
4     1.3) SCHEDULER_PERSIST = True
5     1.4) REDIS_HOST = '192.168.1.105'
6     1.5) REDIS_PORT = 6379
7     1.6) ITEM_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}
8     1.7) MYSQL_HOST = '192.168.1.105'
9
10 【2】 爬虫文件:tencent.py（必须基于start_urls）
11     from scrapy_redis.spiders import RedisSpider
12     class TencentSpider(RedisSpider):
13         # 1. 去掉start_urls
14         # 2. 定义redis_key
15         redis_key = 'tencent:spider'
16         def parse(self, response):
17             pass
18
19 【3】把代码复制到所有爬虫服务器，并启动项目
20

```

```
21 【4】到redis命令行, 执行LPUSH命令压入第一个要爬取的URL地址
22 >LPUSH tencent:spider 第1页的URL地址
23
24 【注意】: 项目爬取结束后无法退出, 如何退出?
25 setting.py
26 CLOSESPIDER_TIMEOUT = 3600
27 # 到指定时间(3600秒)时, 会自动结束并退出
```

## ■ Linux中周期性计划任务

```
1 【1】进入Linux周期性计划任务: crontab -e
2 【2】使用说明
3 * * * * * python3 /home/tarena/xxx.py
4 分 时 日 月 周
5 分: 0-59
6 时: 0-23
7 日: 1-31
8 月: 1-12
9 周: 0-6
10
11 - : 代表时间段
12 / : 代表时间间隔的频率
13 , : 指定多个时间点
14
15 【3】使用示例
16 3.1) 每天早晨9点整去执行爬虫文件
17 0 9 * * * python3 /home/tarena/spider.py
18
19 3.2) 每天9:00-18:00之间, 每隔2个小时去执行1次爬虫文件
20 0 9-18/2 * * * python3 /home/tarena/spider.py
21
22 3.3) 每月的1号和15号的9:00去执行爬虫
23 0 9 1,15 * * python3 /home/tarena/spider.py
```

# 机器视觉与tesseract

## ■ 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
7 籍、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
9 2.2) tesseract-ocr
10 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
11
12 2.3) pytesseract
13 Python模块, 可调用底层识别库, 是对tesseract-ocr做的一层Python API封装
```

## ■ 安装tesseract-ocr

```
1  【1】Ubuntu安装
2      sudo apt-get install tesseract-ocr
3
4  【2】Windows安装
5      2.1) 下载安装包
6      2.2) 添加到环境变量(Path)
7
8  【3】测试 (终端 | cmd命令行)
9      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)
```

# 在线打码平台

## ■ 为什么使用在线打码

```
1  | tesseract-ocr识别率很低,文字变形、干扰,导致无法识别验证码
```

## ■ 云打码平台使用步骤

```
1  | 【1】下载并查看接口文档
2  | 【2】调整接口文档,调整代码并接入程序测试
3  | 【3】真正接入程序,在线识别后获取结果并使用
```

# 破解云打码网站验证码

## ■ 1 - 下载并调整接口文档,封装成函数,打码获取结果

```
1  | import http.client, mimetypes, urllib, json, time, requests
2
3  | #####
```



```

4
5 class YDMHttp:
6
7     apiurl = 'http://api.yundama.com/api.php'
8     username = ''
9     password = ''
10    appid = ''
11    appkey = ''
12
13    def __init__(self, username, password, appid, appkey):
14        self.username = username
15        self.password = password
16        self.appid = str(appid)
17        self.appkey = appkey
18
19    def request(self, fields, files=[]):
20        response = self.post_url(self.apiurl, fields, files)
21        response = json.loads(response)
22        return response
23
24    def balance(self):
25        data = {'method': 'balance', 'username': self.username, 'password': self.password,
26'appid': self.appid, 'appkey': self.appkey}
27        response = self.request(data)
28        if (response):
29            if (response['ret'] and response['ret'] < 0):
30                return response['ret']
31            else:
32                return response['balance']
33        else:
34            return -9001
35
36    def login(self):
37        data = {'method': 'login', 'username': self.username, 'password': self.password,
38'appid': self.appid, 'appkey': self.appkey}
39        response = self.request(data)
40        if (response):
41            if (response['ret'] and response['ret'] < 0):
42                return response['ret']
43            else:
44                return response['uid']
45        else:
46            return -9001
47
48    def upload(self, filename, codetype, timeout):
49        data = {'method': 'upload', 'username': self.username, 'password': self.password,
50'appid': self.appid, 'appkey': self.appkey, 'codetype': str(codetype), 'timeout':
51str(timeout)}
52        file = {'file': filename}
53        response = self.request(data, file)
54        if (response):
55            if (response['ret'] and response['ret'] < 0):
56                return response['ret']
57            else:
58                return response['cid']
59        else:
60            return -9001

```

```

57
58     def result(self, cid):
59         data = {'method': 'result', 'username': self.username, 'password': self.password,
60 'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid)}
61         response = self.request(data)
62         return response and response['text'] or ''
63
64     def decode(self, filename, codetype, timeout):
65         cid = self.upload(filename, codetype, timeout)
66         if (cid > 0):
67             for i in range(0, timeout):
68                 result = self.result(cid)
69                 if (result != ''):
70                     return cid, result
71                 else:
72                     time.sleep(1)
73             return -3003, ''
74         else:
75             return cid, ''
76
77     def report(self, cid):
78         data = {'method': 'report', 'username': self.username, 'password': self.password,
79 'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid), 'flag': '0'}
80         response = self.request(data)
81         if (response):
82             return response['ret']
83         else:
84             return -9001
85
86     def post_url(self, url, fields, files=[]):
87         for key in files:
88             files[key] = open(files[key], 'rb');
89         res = requests.post(url, files=files, data=fields)
90         return res.text
91
92     #####
93     def get_result(filename):
94         # 用户名
95         username = 'yibeizi001'
96         # 密码
97         password = 'zhishouzhethian001'
98         # 软件ID, 开发者分成必要参数。登录开发者后台【我的软件】获得!
99         appid = 1
100         # 软件密钥, 开发者分成必要参数。登录开发者后台【我的软件】获得!
101         appkey = '22cc5376925e9387a23cf797cb9ba745'
102         # 验证码类型, # 例: 1004表示4位字母数字, 不同类型收费不同。请准确填写, 否则影响识别率。在此查
103         询所有类型 http://www.yundama.com/price.html
104         codetype = 5000
105         # 超时时间, 秒
106         timeout = 60
107         # 初始化
108         yundama = YDMHttp(username, password, appid, appkey)
109         # 登陆云打码
110         uid = yundama.login()
111         # 查询余额
112         balance = yundama.balance()
113         # 开始识别, 图片路径, 验证码类型ID, 超时时间(秒), 识别结果

```

```

111     cid, result = yundama.decode(filename, codetype, timeout)
112
113     return result
114     #####

```

## ■ 2 - 访问云打码网站，获取验证码并在线识别

```

1  '''识别云打码官网的验证码'''
2  from selenium import webdriver
3  from ydmapi import *
4  from PIL import Image
5
6  class YdmSpider(object):
7      def __init__(self):
8          self.browser = webdriver.Chrome()
9          self.browser.maximize_window()
10         self.browser.get('http://www.yundama.com/')
11
12         # 获取验证码图片截取出来
13         def get_image(self):
14             # 1.获取页面截图
15             self.browser.save_screenshot('index.png')
16             # 2.获取验证码节点坐标,把图片截取出来
17             # location: 获取节点左上角的坐标(x y)
18             location = self.browser.find_element_by_xpath('//*[@id="verifyImg"]').location
19             # size: 获取节点的大小(宽度和高度)
20             size = self.browser.find_element_by_xpath('//*[@id="verifyImg"]').size
21             # 四个坐标
22             left_x = location['x']
23             left_y = location['y']
24             right_x = left_x + size['width']
25             right_y = left_y + size['height']
26             # 从index.png中截图图片,注意crop()方法参数为元组
27             img = Image.open('index.png').crop((left_x, left_y, right_x, right_y))
28             img.save('verify.png')
29
30         # 获取识别结果
31         def get_result(self):
32             result = get_result('verify.png')
33             print('识别结果:', result)
34
35         # 入口函数
36         def run(self):
37             self.get_image()
38             self.get_result()
39             self.browser.close()
40
41         if __name__ == '__main__':
42             spider = YdmSpider()
43             spider.run()

```

## ■ 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) 设置之抓浏览器的包： ...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
```

## ■ Fiddler常用菜单

```
1  【1】 Inspector：查看数据包详细内容
2      1.1) 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
```

# 移动端app数据抓取

## ■ 方法1 - 手机 + Fiddler

```
1  设置方法见文件夹 - 移动端抓包配置
```

## ■ 方法2 - F12浏览器工具

# 有道翻译手机版破解案例

```
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)

```

## 补充 - 滑块缺口验证码案例

### 豆瓣网登录

#### ■ 案例说明

- 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
  - 4 3.1) 先快速滑动
  - 5 3.2) 到离重点位置不远的地方开始减速
- 6 【4】 详细看代码注释

#### ■ 代码实现

```

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     ①v=v0+at
15     ②s=v0t+½at²
16     """
17     # 初速度
18     v = 0
19     # 单位时间为0.3s来统计轨迹, 轨迹即0.3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
48 # 1、打开豆瓣官网 - 并将窗口最大化
49 browser = webdriver.Chrome()
50 browser.maximize_window()
51 browser.get('https://www.douban.com/')
52
53 # 2、切换到iframe子页面
54 login_frame = browser.find_element_by_xpath('//*[id="anony-reg-new"]/div/div[1]/iframe')
55 browser.switch_to.frame(login_frame)
56
57 # 3、密码登录 + 用户名 + 密码 + 登录豆瓣
58 browser.find_element_by_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()
59 browser.find_element_by_xpath('//*[id="username"]').send_keys('15110225726')
60 browser.find_element_by_xpath('//*[id="password"]').send_keys('zhanshen001')
61 browser.find_element_by_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()
62 time.sleep(4)
63
64 # 4、切换到新的iframe子页面 - 滑块验证
65 auth_frame = browser.find_element_by_xpath('//*[id="TCaptcha"]/iframe')
66 browser.switch_to.frame(auth_frame)
67
68 # 5、按住开始滑动位置按钮 - 先移动180个像素
69 element = browser.find_element_by_xpath('//*[id="tcaptcha_drag_button"']')
70 # click_and_hold(): 按住某个节点并保持
71 ActionChains(browser).click_and_hold(on_element=element).perform()
72 # move_to_element_with_offset(): 移动到距离某个元素(左上角坐标)多少距离的位置
73 ActionChains(browser).move_to_element_with_offset(to_element=element,xoffset=180,yoffset=0)
74 .perform()
75
76 # 6、使用加速度函数移动剩下的距离
77 tracks = get_tracks(28)

```

```

77 for track in tracks:
78     # move_by_offset() : 鼠标从当前位置移动到某个坐标
79     ActionChains(browser).move_by_offset(xoffset=track,yoffset=0).perform()
80
81 # 7、延迟释放鼠标: release()
82 time.sleep(0.5)
83 ActionChains(browser).release().perform()

```

## 爬虫总结

```

1 # 1、什么是爬虫
2 爬虫是请求网站并提取数据的自动化程序
3
4 # 2、robots协议是什么
5 爬虫协议或机器人协议,网站通过robots协议告诉搜索引擎哪些页面可以抓取, 哪些页面不能抓取
6
7 # 3、爬虫的基本流程
8 1、请求得到响应
9 2、解析
10 3、保存数据
11
12 # 4、请求
13 1、urllib
14 2、requests
15 3、scrapy
16
17 # 5、解析
18 1、re正则表达式
19 2、lxml+xpath解析
20 3、json解析模块
21
22 # 6、selenium+browser
23
24 # 7、常见反爬策略
25 1、Headers : 最基本的反爬手段, 一般被关注的变量是UserAgent和Referer, 可以考虑使用浏览器中
26 2、UA : 建立User-Agent池,每次访问页面随机切换
27 3、拉黑高频访问IP
28 数据量大用代理IP池伪装成多个访问者,也可控制爬取速度
29 4、Cookies
30 建立有效的cookie池, 每次访问随机切换
31 5、验证码
32 验证码数量较少可人工填写
33 图形验证码可使用tesseract识别
34 其他情况只能在线打码、人工打码和训练机器学习模型
35 6、动态生成
36 一般由js动态生成的数据都是向特定的地址发get请求得到的, 返回的一般是json
37 7、签名及js加密
38 一般为本地JS加密,查找本地JS文件,分析,或者使用execjs模块执行JS
39 8、js调整页面结构
40 9、js在响应中指向新的地址
41
42 # 8、scrapy框架的运行机制
43

```

44 # 9、分布式爬虫的原理  
45 多台主机共享一个爬取队列