# **Day09回顾**

## **settings.py常用变量**

【1】settings.py中常用变量  
 2.1) 设置日志级别  
 LOG\_LEVEL = ''  
 2.2) 保存到日志文件(不在终端输出)  
 LOG\_FILE = ''  
 2.3) 设置数据导出编码(主要针对于json文件)  
 FEED\_EXPORT\_ENCODING = 'utf-8'  
 2.4) 设置User-Agent  
 USER\_AGENT = ''  
 2.5) 设置最大并发数(默认为16)  
 CONCURRENT\_REQUESTS = 32  
 2.6) 下载延迟时间(每隔多长时间请求一个网页)  
 DOWNLOAD\_DELAY = 1  
 2.7) 请求头  
 DEFAULT\_REQUEST\_HEADERS = {'User-Agent':'Mozilla/'}  
 2.8) 添加项目管道  
 ITEM\_PIPELINES = {'项目目录名.pipelines.类名':优先级}  
 2.9) cookie(默认禁用,取消注释-True|False都为开启)  
 COOKIES\_ENABLED = False  
 2.10) 非结构化数据存储路径  
 IMAGES\_STORE = '/home/tarena/images/'  
 FILES\_STORE = '/home/tarena/files/'  
 2.11) 添加下载器中间件  
 DOWNLOADER\_MIDDLEWARES = {'项目名.middlewares.类名':200}

## **非结构化数据抓取**

【1】spider  
 yield item['链接']  
   
【2】pipelines.py  
 from scrapy.pipelines.images import ImagesPipeline  
 import scrapy  
 class TestPipeline(ImagesPipeline):  
 def get\_media\_requests(self,item,info):  
 yield scrapy.Request(url=item['url'],meta={'name':item['name']})  
   
 def file\_path(self,request,response=None,info=None):  
 name = request.meta['name']  
 filename = name  
 return filename  
   
【3】settings.py  
 IMAGES\_STORE = 'D:/Spider/images'

## **Post请求**

* **方法**
* scrapy.FormRequest(url=url,formdata=formdata,callback=self.xxx)
* **使用cookie**
* 【1】方法1  
   COOKIES\_ENABLED = False  
   DEFAULT\_REQUEST\_HEADERS = {'Cookie':'xxxx'}  
     
  【2】方法2  
   COOKIES\_ENABLED = True  
   yield scrapy.Request(url=url,cookies={},callback=self.xxxx)  
   yield scrapy.FormRequest(url=url,formdata={},cookies={},callback=self.xxxx)  
     
  【3】方法3  
   COOKIES\_ENBALED = True  
   class XxxCookieDownloaderMiddleware(object):  
   def process\_request(self,request,spider):  
   request.cookies = {}
* **对象属性**
* 【1】请求对象request属性  
   1.1) request.url : 请求URL地址  
   1.2) request.headers : 请求头 - 字典  
   1.3) request.meta : item数据传递、定义代理  
   1.4) request.cookies : Cookie  
    
  【2】响应对象response属性  
   2.1) response.url : 返回实际数据的URL地址  
   2.2) response.text : 响应内容 - 字符串  
   2.3) response.body : 响应内容 - 字节串  
   2.4) response.encoding ：响应字符编码  
   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】middlewares.py新建中间件类  
   class RandomUseragentMiddleware(object):  
   def process\_request(self,reuqest,spider):  
   agent = UserAgent().random  
   request.headers['User-Agent'] = agent  
     
  【2】settings.py添加此下载器中间件  
   DOWNLOADER\_MIDDLEWARES = {'' : 优先级}

### **设置中间件(随机代理)**

* **代理IP中间件**
* class RandomProxyDownloaderMiddleware(object):  
   def process\_request(self,request,spider):  
   request.meta['proxy'] = xxx  
     
   # 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  
   def process\_exception(self,request,exception,spider):  
   return request

### **设置中间件(Cookie)**

* **Cookie中间件**
* class BaiduCookieDownloaderMiddleware(object):  
   def process\_request(self,request,spider):  
   cookies = self.get\_cookies()  
   print('middleware3', cookies)  
   # 利用请求对象request的cookies属性  
   request.cookies = cookies  
    
    
   def get\_cookies(self):  
   costr = ''  
   cookies = {}  
   for kv in costr.split('; '):  
   cookies[kv.split('=')[0]] =kv.split('=')[1]  
    
   return cookies
* **练习**
* 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】原理  
   多台主机共享1个爬取队列  
     
  【2】实现  
   2.1) 重写scrapy调度器(scrapy\_redis模块)  
   2.2) sudo pip3 install scrapy\_redis
* **为什么使用redis**
* 【1】Redis基于内存,速度快  
  【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**
* https://github.com/rmax/scrapy-redis
* **settings.py说明**
* # 重新指定调度器: 启用Redis调度存储请求队列  
  SCHEDULER = "scrapy\_redis.scheduler.Scheduler"  
    
  # 重新指定去重机制: 确保所有的爬虫通过Redis去重  
  DUPEFILTER\_CLASS = "scrapy\_redis.dupefilter.RFPDupeFilter"  
    
  # 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)  
  SCHEDULER\_PERSIST = True  
    
  # 优先级队列 （默认）  
  SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.PriorityQueue'  
  #可选用的其它队列  
  # 先进先出  
  SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.FifoQueue'  
  # 后进先出  
  SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.LifoQueue'  
    
  # redis管道  
  ITEM\_PIPELINES = {  
   'scrapy\_redis.pipelines.RedisPipeline': 300  
  }  
    
  #指定连接到redis时使用的端口和地址  
  REDIS\_HOST = 'localhost'  
  REDIS\_PORT = 6379

## **腾讯招聘分布式改写**

* **分布式爬虫完成步骤**
* 【1】首先完成非分布式scrapy爬虫 : 正常scrapy爬虫项目抓取  
  【2】设置,部署成为分布式爬虫
* **分布式环境说明**
* 【1】分布式爬虫服务器数量: 2（其中1台Windows,1台Ubuntu虚拟机）  
  【2】服务器分工:  
   2.1) Windows : 负责数据抓取  
   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】完成正常scrapy项目数据抓取（非分布式 - 拷贝之前的Tencent）  
    
  【2】设置settings.py，完成分布式设置  
   2.1-必须) 使用scrapy\_redis的调度器  
   SCHEDULER = "scrapy\_redis.scheduler.Scheduler"  
     
   2.2-必须) 使用scrapy\_redis的去重机制  
   DUPEFILTER\_CLASS = "scrapy\_redis.dupefilter.RFPDupeFilter"  
     
   2.3-必须) 定义redis主机地址和端口号  
   REDIS\_HOST = '192.168.1.107'  
   REDIS\_PORT = 6379  
     
   2.4-非必须) 是否清除请求指纹,True:不清除 False:清除（默认）  
   SCHEDULER\_PERSIST = True  
     
   2.5-非必须) 在ITEM\_PIPELINES中添加redis管道,数据将会存入redis数据库  
   'scrapy\_redis.pipelines.RedisPipeline': 200  
     
  【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  
    
  【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**
* """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"""  
  【1】settings.py  
   1.1) SCHEDULER = 'scrapy\_redis.scheduler.Scheduler'  
   1.2) DUPEFILTER\_CLASS = 'scrapy\_redis.dupefilter.RFPDupeFilter'  
   1.3) SCHEDULER\_PERSIST = True  
   1.4) REDIS\_HOST = '192.168.1.105'  
   1.5) REDIS\_POST = 6379  
   1.6) ITEM\_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}  
   1.7) MYSQL\_HOST = '192.168.1.105'  
     
  【2】将代码拷贝到分布式中所有爬虫服务器  
    
  【3】多台爬虫服务器同时运行scrapy爬虫  
    
  # 赠送腾讯MySQL数据库建库建表语句  
  """  
  create database tencentdb charset utf8;  
  use tencentdb;  
  create table tencenttab(  
  job\_name varchar(1000),  
  job\_type varchar(200),  
  job\_duty varchar(5000),  
  job\_require varchar(5000),  
  job\_address varchar(200),  
  job\_time varchar(200)  
  )charset=utf8;  
  """

## **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】settings.py和方法一中写法一致  
   1.1) SCHEDULER = 'scrapy\_redis.scheduler.Scheduler'  
   1.2) DUPEFILTER\_CLASS = 'scrapy\_redis.dupefilter.RFPDupeFilter'  
   1.3) SCHEDULER\_PERSIST = True  
   1.4) REDIS\_HOST = '192.168.1.105'  
   1.5) REDIS\_PORT = 6379  
   1.6) ITEM\_PIPELINES = {'Tencent.pipelines.TencentMysqlPipeline' : 300}  
   1.7) MYSQL\_HOST = '192.168.1.105'  
     
  【2】爬虫文件:tencent.py (必须基于start\_urls)  
   from scrapy\_redis.spiders import RedisSpider  
   class TencentSpider(RedisSpider):  
   # 1. 去掉start\_urls  
   # 2. 定义redis\_key  
   redis\_key = 'tencent:spider'  
   def parse(self,response):  
   pass  
     
  【3】把代码复制到所有爬虫服务器，并启动项目  
    
  【4】到redis命令行，执行LPUSH命令压入第一个要爬取的URL地址  
   >LPUSH tencent:spider 第1页的URL地址  
    
  【注意】: 项目爬取结束后无法退出，如何退出？  
  setting.py  
  CLOSESPIDER\_TIMEOUT = 3600  
  # 到指定时间(3600秒)时,会自动结束并退出

## **机器视觉与tesseract**

* **概述**
* 【1】作用  
   处理图形验证码  
    
  【2】三个重要概念 - OCR、tesseract-ocr、pytesseract  
   2.1) OCR  
   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、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  
    
   2.2) tesseract-ocr  
   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  
    
   2.3) pytesseract  
   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】Ubuntu安装  
   sudo apt-get install tesseract-ocr  
    
  【2】Windows安装  
   2.1) 下载安装包  
   2.2) 添加到环境变量(Path)  
    
  【3】测试（终端 | cmd命令行）  
   tesseract xxx.jpg 文件名
* **安装pytesseract**
* 【1】安装  
   sudo pip3 install pytesseract  
     
  【2】使用示例  
   import pytesseract  
   # Python图片处理库  
   from PIL import Image  
    
   # 创建图片对象  
   img = Image.open('test1.jpg')  
   # 图片转字符串  
   result = pytesseract.image\_to\_string(img)  
   print(result)

## **在线打码平台**

* **为什么使用在线打码**
* 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】下载并查看接口文档  
  【2】调整接口文档，调整代码并接入程序测试  
  【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**
* import http.client, mimetypes, urllib, json, time, requests  
    
  ######################################################################  
    
  class YDMHttp:  
    
   apiurl = 'http://api.yundama.com/api.php'  
   username = ''  
   password = ''  
   appid = ''  
   appkey = ''  
    
   def \_\_init\_\_(self, username, password, appid, appkey):  
   self.username = username   
   self.password = password  
   self.appid = str(appid)  
   self.appkey = appkey  
    
   def request(self, fields, files=[]):  
   response = self.post\_url(self.apiurl, fields, files)  
   response = json.loads(response)  
   return response  
     
   def balance(self):  
   data = {'method': 'balance', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey}  
   response = self.request(data)  
   if (response):  
   if (response['ret'] and response['ret'] < 0):  
   return response['ret']  
   else:  
   return response['balance']  
   else:  
   return -9001  
     
   def login(self):  
   data = {'method': 'login', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey}  
   response = self.request(data)  
   if (response):  
   if (response['ret'] and response['ret'] < 0):  
   return response['ret']  
   else:  
   return response['uid']  
   else:  
   return -9001  
    
   def upload(self, filename, codetype, timeout):  
   data = {'method': 'upload', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey, 'codetype': str(codetype), 'timeout': str(timeout)}  
   file = {'file': filename}  
   response = self.request(data, file)  
   if (response):  
   if (response['ret'] and response['ret'] < 0):  
   return response['ret']  
   else:  
   return response['cid']  
   else:  
   return -9001  
    
   def result(self, cid):  
   data = {'method': 'result', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid)}  
   response = self.request(data)  
   return response and response['text'] or ''  
    
   def decode(self, filename, codetype, timeout):  
   cid = self.upload(filename, codetype, timeout)  
   if (cid > 0):  
   for i in range(0, timeout):  
   result = self.result(cid)  
   if (result != ''):  
   return cid, result  
   else:  
   time.sleep(1)  
   return -3003, ''  
   else:  
   return cid, ''  
    
   def report(self, cid):  
   data = {'method': 'report', 'username': self.username, 'password': self.password, 'appid': self.appid, 'appkey': self.appkey, 'cid': str(cid), 'flag': '0'}  
   response = self.request(data)  
   if (response):  
   return response['ret']  
   else:  
   return -9001  
    
   def post\_url(self, url, fields, files=[]):  
   for key in files:  
   files[key] = open(files[key], 'rb');  
   res = requests.post(url, files=files, data=fields)  
   return res.text  
    
  ######################################################################  
  def get\_result(filename):  
   # 用户名  
   username = 'yibeizi001'  
   # 密码  
   password = 'zhishouzhetian001'  
   # 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！  
   appid = 1  
   # 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！  
   appkey = '22cc5376925e9387a23cf797cb9ba745'  
   # 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。在此查询所有类型 http://www.yundama.com/price.html  
   codetype = 5000  
   # 超时时间，秒  
   timeout = 60  
   # 初始化  
   yundama = YDMHttp(username, password, appid, appkey)  
   # 登陆云打码  
   uid = yundama.login()  
   # 查询余额  
   balance = yundama.balance()  
   # 开始识别，图片路径，验证码类型ID，超时时间（秒），识别结果  
   cid, result = yundama.decode(filename, codetype, timeout)  
    
   return result  
  ######################################################################
* **2 - 访问云打码网站，获取验证码并在线识别**
* '''识别云打码官网的验证码'''  
  from selenium import webdriver  
  from ydmapi import \*  
  from PIL import Image  
    
  class YdmSpider(object):  
   def \_\_init\_\_(self):  
   self.browser = webdriver.Chrome()  
   self.browser.maximize\_window()  
   self.browser.get('http://www.yundama.com/')  
    
   # 获取验证码图片截取出来  
   def get\_image(self):  
   # 1.获取页面截图  
   self.browser.save\_screenshot('index.png')  
   # 2.获取验证码节点坐标,把图片截取出来  
   # location: 获取节点左上角的坐标(x y)  
   location = self.browser.find\_element\_by\_xpath('//\*[@id="verifyImg"]').location  
   # size: 获取节点的大小(宽度和高度)  
   size = self.browser.find\_element\_by\_xpath('//\*[@id="verifyImg"]').size  
   # 四个坐标  
   left\_x = location['x']  
   left\_y = location['y']  
   right\_x = left\_x + size['width']  
   right\_y = left\_y + size['height']  
   # 从index.png中截图图片,注意crop()方法参数为元组  
   img = Image.open('index.png').crop((left\_x,left\_y,right\_x,right\_y))  
   img.save('verify.png')  
    
   # 获取识别结果  
   def get\_result(self):  
   result = get\_result('verify.png')  
   print('识别结果:',result)  
    
   # 入口函数  
   def run(self):  
   self.get\_image()  
   self.get\_result()  
   self.browser.close()  
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   spider = YdmSpider()  
   spider.run()

## **Fiddler抓包工具**

* **配置Fiddler**
* 【1】Tools -> Options -> HTTPS  
   1.1) 添加证书信任: 勾选 Decrypt Https Traffic 后弹出窗口，一路确认  
   1.2) 设置之抓浏览器的包: ...from browsers only  
    
  【2】Tools -> Options -> Connections  
   2.1) 设置监听端口（默认为8888）  
    
  【3】配置完成后重启Fiddler（重要）  
   3.1) 关闭Fiddler,再打开Fiddler
* **配置浏览器代理**
* 【1】安装Proxy SwitchyOmega谷歌浏览器插件  
    
  【2】配置代理  
   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  
   2.2) 输入 HTTP:// 127.0.0.1 8888  
   2.3) 点击 ：应用选项  
     
  【3】点击右上角SwitchyOmega可切换代理  
    
  【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】Inspector ：查看数据包详细内容  
   1.1) 整体分为请求和响应两部分  
     
  【2】Inspector常用菜单  
   2.1) Headers ：请求头信息  
   2.2) WebForms: POST请求Form表单数据 ：<body>  
   GET请求查询参数: <QueryString>  
   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**
* 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**

import requests  
from lxml import etree  
  
word = input('请输入要翻译的单词:')  
  
post\_url = 'http://m.youdao.com/translate'  
post\_data = {  
 'inputtext':word,  
 'type':'AUTO'  
}  
  
html = requests.post(url=post\_url,data=post\_data).text  
parse\_html = etree.HTML(html)  
xpath\_bds = '//ul[@id="translateResult"]/li/text()'  
result = parse\_html.xpath(xpath\_bds)[0]  
  
print(result)

## **补充 - 滑块缺口验证码案例**

### **豆瓣网登录**

* **案例说明**
* 【1】URL地址: https://www.douban.com/  
  【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】模拟人的行为  
   3.1) 先快速滑动  
   3.2) 到离重点位置不远的地方开始减速  
  【4】详细看代码注释
* **代码实现**
* """  
  说明：先输入几次错误的密码，出现滑块缺口验证码  
  """  
  from selenium import webdriver  
  # 导入鼠标事件类  
  from selenium.webdriver import ActionChains  
  import time  
    
  # 加速度函数  
  def get\_tracks(distance):  
   """  
   拿到移动轨迹，模仿人的滑动行为，先匀加速后匀减速  
   匀变速运动基本公式：  
   ①v=v0+at  
   ②s=v0t+½at²  
   """  
   # 初速度  
   v = 0  
   # 单位时间为0.3s来统计轨迹，轨迹即0.3内的位移  
   t = 0.3  
   # 位置/轨迹列表,列表内的一个元素代表0.3s的位移  
   tracks = []  
   # 当前的位移  
   current = 0  
   # 到达mid值开始减速  
   mid = distance\*4/5  
   while current < distance:  
   if current < mid:  
   # 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  
   a = 2  
   else:  
   a = -3  
    
   # 初速度  
   v0 = v  
   # 0.3秒内的位移  
   s = v0\*t+0.5\*a\*(t\*\*2)  
   # 当前的位置  
   current += s  
   # 添加到轨迹列表  
   tracks.append(round(s))  
   # 速度已经达到v，该速度作为下次的初速度  
   v = v0 + a\*t  
   return tracks  
   # tracks: [第一个0.3秒的移动距离,第二个0.3秒的移动距离,...]  
    
    
  # 1、打开豆瓣官网 - 并将窗口最大化  
  browser = webdriver.Chrome()  
  browser.maximize\_window()  
  browser.get('https://www.douban.com/')  
    
  # 2、切换到iframe子页面  
  login\_frame = browser.find\_element\_by\_xpath('//\*[@id="anony-reg-new"]/div/div[1]/iframe')  
  browser.switch\_to.frame(login\_frame)  
    
  # 3、密码登录 + 用户名 + 密码 + 登录豆瓣  
  browser.find\_element\_by\_xpath('/html/body/div[1]/div[1]/ul[1]/li[2]').click()  
  browser.find\_element\_by\_xpath('//\*[@id="username"]').send\_keys('15110225726')  
  browser.find\_element\_by\_xpath('//\*[@id="password"]').send\_keys('zhanshen001')  
  browser.find\_element\_by\_xpath('/html/body/div[1]/div[2]/div[1]/div[5]/a').click()  
  time.sleep(4)  
    
  # 4、切换到新的iframe子页面 - 滑块验证  
  auth\_frame = browser.find\_element\_by\_xpath('//\*[@id="TCaptcha"]/iframe')  
  browser.switch\_to.frame(auth\_frame)  
    
  # 5、按住开始滑动位置按钮 - 先移动180个像素  
  element = browser.find\_element\_by\_xpath('//\*[@id="tcaptcha\_drag\_button"]')  
  # click\_and\_hold(): 按住某个节点并保持  
  ActionChains(browser).click\_and\_hold(on\_element=element).perform()  
  # 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  
  ActionChains(browser).move\_to\_element\_with\_offset(to\_element=element,xoffset=180,yoffset=0).perform()  
    
  # 6、使用加速度函数移动剩下的距离  
  tracks = get\_tracks(28)  
  for track in tracks:  
   # move\_by\_offset() : 鼠标从当前位置移动到某个坐标  
   ActionChains(browser).move\_by\_offset(xoffset=track,yoffset=0).perform()  
    
  # 7、延迟释放鼠标: release()  
  time.sleep(0.5)  
  ActionChains(browser).release().perform()

## **爬虫总结**

# 1、什么是爬虫  
 爬虫是请求网站并提取数据的自动化程序  
  
# 2、robots协议是什么  
 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  
  
# 3、爬虫的基本流程  
 1、请求得到响应  
 2、解析  
 3、保存数据  
  
# 4、请求  
 1、urllib  
 2、requests  
 3、scrapy  
  
# 5、解析  
 1、re正则表达式  
 2、lxml+xpath解析  
 3、json解析模块  
  
# 6、selenium+browser  
  
# 7、常见反爬策略  
 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  
 2、UA ： 建立User-Agent池,每次访问页面随机切换  
 3、拉黑高频访问IP  
 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  
 4、Cookies  
 建立有效的cookie池，每次访问随机切换  
 5、验证码  
 验证码数量较少可人工填写  
 图形验证码可使用tesseract识别  
 其他情况只能在线打码、人工打码和训练机器学习模型  
 6、动态生成  
 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  
 7、签名及js加密  
 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  
 8、js调整页面结构  
 9、js在响应中指向新的地址  
  
# 8、scrapy框架的运行机制  
  
# 9、分布式爬虫的原理  
 多台主机共享一个爬取队列