# **Day02回顾**

## **爬取网站思路**

【1】先确定是否为动态加载网站  
【2】找URL规律  
【3】正则表达式  
【4】定义程序框架，补全并测试代码

## **数据持久化 - csv**

import csv  
 with open('xxx.csv','w',encoding='utf-8',newline='') as f:  
 writer = csv.writer(f)  
 writer.writerow([])  
 writer.writerows([(),(),()])

## **数据持久化 - MySQL**

import pymysql  
  
# \_\_init\_\_(self)：  
 self.db = pymysql.connect('IP',... ...)  
 self.cursor = self.db.cursor()  
   
# save\_html(self,r\_list):  
 self.cursor.execute('sql',[data1])  
 self.cursor.executemany('sql',[(data1),(data2),(data3)])  
 self.db.commit()  
   
# run(self):  
 self.cursor.close()  
 self.db.close()

## **数据持久化 - MongoDB**

import pymongo  
  
  
# \_\_init\_\_(self)：  
 self.conn = pymongo.MongoClient('IP',27017)  
 self.db = self.conn['cardb']  
 self.myset = self.db['car\_set']  
   
# save\_html(self,r\_list):  
 self.myset.insert\_one(dict)  
 self.myset.insert\_many([{},{},{}])  
  
# MongoDB - Commmand - 库->集合->文档  
mongo  
>show dbs  
>use db\_name  
>show collections  
>db.集合名.find().pretty()  
>db.集合名.count()  
>db.集合名.drop()  
>db.dropDatabase()

## **多级页面数据抓取**

【1】整体思路  
 1.1> 爬取一级页面,提取 所需数据+链接,继续跟进  
 1.2> 爬取二级页面,提取 所需数据+链接,继续跟进  
 1.3> ... ...   
  
【2】代码实现思路  
 2.1> 避免重复代码 - 请求、解析需定义函数

## **增量爬虫**

* **MySQL实现增量**
* 【1】数据库中创建指纹表，表中存放所有抓取过的'对URL地址进行md5加密'后的指纹  
  【2】代码实现流程模板  
  import pymysql  
  from hashlib import md5  
  import sys  
    
  class XxxIncrSpider:  
   def \_\_init\_\_(self):  
   self.db = pymysql.connect('localhost','root','123456','xxxdb',charset='utf8')  
   self.cursor = self.db.cursor()  
     
   def url\_md5(self,url):  
   """对URL进行md5加密函数"""  
   s = md5()  
   s.update(url.encode())  
   return s.hexdigest()  
     
   def run\_spider(self):  
   href\_list = ['url1','url2','url3','url4']  
   for href in href\_list:  
   href\_md5 = self.url\_md5(href)  
   if href\_md5 不在指纹表中:  
   开始进行数据抓取，完成后将指纹插入到指纹表中  
   else:  
   sys.exit()
* **Redis实现增量**
* 【1】原理  
   利用Redis集合特性，可将抓取过的指纹添加到redis集合中，根据返回值来判定是否需要抓取  
     
  【2】代码实现模板  
  import redis  
  from hashlib import md5  
  import sys  
    
  class XxxIncrSpider:  
   def \_\_init\_\_(self):  
   self.r = redis.Redis(host='localhost',port=6379,db=0)  
     
   def url\_md5(self,url):  
   """对URL进行md5加密函数"""  
   s = md5()  
   s.update(url.encode())  
   return s.hexdigest()  
     
   def run\_spider(self):  
   href\_list = ['url1','url2','url3','url4']  
   for href in href\_list:  
   href\_md5 = self.url\_md5(href)  
   if self.r.sadd('spider:urls',href\_md5) == 1:  
   返回值为1表示添加成功，即之前未抓取过，则开始抓取  
   else:  
   sys.exit()

## **Day03笔记**

## **百度图片抓取**

* **案例解析**
* 【1】百度图片官网指定图片抓取:  
   1.1> 百度图片官网: http://image.baidu.com/  
   1.2> 运行效果  
   请输入关键字: 赵丽颖  
   则自动创建文件夹: /home/tarena/images/赵丽颖/ 并把首页30张图片保存到此文件夹下  
     
  【2】注意   
   2.1> 一定要以响应内容为主来写正则表达式（右键 - 查看网页源代码）  
     
  【3】颠覆前两天课程认知的一个现实  
   3.1> 页面结构 - Elements，为页面最终渲染完成后的结构，和响应内容不一定完全一样  
   3.2> 原因1: 可能会有部分数据为动态加载的  
   原因2: 响应内容中存在JavaScript，对页面结构做了一定调整  
    
  【4】那我们写正则表达式时要以谁为准？  
   4.1> 必须以响应内容为准!!!!!! -> 右键,查看网页源代码为准  
   4.1> 必须以响应内容为准!!!!!! -> 右键,查看网页源代码为准  
   4.1> 必须以响应内容为准!!!!!! -> 右键,查看网页源代码为准  
     
   @@ 重要的事情说三遍,必须以响应内容为准 @@
* **实现步骤**
* 【1】右键,查看网页源码,搜索图片链接关键字 -> 存在  
  【2】分析URL地址规律  
   https://image.baidu.com/search/index?tn=baiduimage&word={}  
  【3】正则表达式 - 以响应内容为准  
   "thumbURL":"(.\*?)"  
  【4】代码实现  
   4.1> 知识点1 - Windows中路径如何表示  
   方式1: E:\\spider\\spider\_day03\\  
   方式2: E:/spider/spider\_day03/  
     
   4.2> 如何生成随机的User-Agent  
   sudo pip3 install fake\_useragent  
   from fake\_useragent import UserAgent  
   user\_agent = UserAgent().random
* **代码实现**
* import requests  
  import re  
  import time  
  import random  
  from fake\_useragent import UserAgent  
  import os  
  from urllib import parse  
    
  class BaiduImageSpider(object):  
   def \_\_init\_\_(self):  
   self.url = 'https://image.baidu.com/search/index?tn=baiduimage&word={}'  
   self.word = input('请输入关键字:')  
   self.directory = '/home/tarena/images/{}/'.format(self.word)  
   if not os.path.exists(self.directory):  
   os.makedirs(self.directory)  
    
   self.i = 1  
    
   def get\_images(self,one\_url):  
   # 使用随机的User-Agent  
   headers = { 'User-Agent':UserAgent().random }  
   one\_html = requests.get(url=one\_url,headers=headers).text  
   regex = '"thumbURL":"(.\*?)"'  
   pattern = re.compile(regex,re.S)  
   image\_src\_list = pattern.findall(one\_html)  
   for image\_src in image\_src\_list:  
   self.save\_image(image\_src)  
   # 控制爬取速度  
   time.sleep(random.uniform(0,1))  
    
   def save\_image(self,image\_src):  
   # 每次请求使用随机的User-Agent  
   headers = { 'User-Agent':UserAgent().random }  
   image\_html = requests.get(url=image\_src,headers=headers).content  
   filename = '{}{}\_{}.jpg'.format(self.directory,self.word,self.i)  
   with open(filename,'wb') as f:  
   f.write(image\_html)  
   print(filename,'下载成功')  
   self.i += 1  
    
   def run(self):  
   params = parse.quote(self.word)  
   one\_url = self.url.format(params)  
   self.get\_images(one\_url)  
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   spider = BaiduImageSpider()  
   spider.run()

## **Chrome浏览器安装插件**

* **安装方法**
* 【1】在线安装  
   1.1> 下载插件 - google访问助手  
   1.2> 安装插件 - google访问助手: Chrome浏览器-设置-更多工具-扩展程序-开发者模式-拖拽(解压后的插件)  
   1.3> 在线安装其他插件 - 打开google访问助手 - google应用商店 - 搜索插件 - 添加即可  
    
  【2】离线安装  
   2.1> 下载插件 - xxx.crx 重命名为 xxx.zip  
   2.2> Chrome浏览器-设置-更多工具-扩展程序-开发者模式  
   2.3> 拖拽 插件(或者解压后文件夹) 到浏览器中  
   2.4> 重启浏览器，使插件生效
* **爬虫常用插件**
* 【1】google-access-helper : 谷歌访问助手,可访问 谷歌应用商店  
  【2】Xpath Helper: 轻松获取HTML元素的xPath路径  
   打开/关闭: Ctrl + Shift + x  
  【3】JsonView: 格式化输出json格式数据  
  【4】Proxy SwitchyOmega: Chrome浏览器中的代理管理扩展程序

## ==**xpath解析**==

* **定义**
* XPath即为XML路径语言，它是一种用来确定XML文档中某部分位置的语言，同样适用于HTML文档的检索
* **示例HTML代码**
* <ul class="CarList">  
   <li class="bjd" id="car\_001" href="http://www.bjd.com/">  
   <p class="name">布加迪</p>  
   <p class="model">威航</p>  
   <p class="price">2500万</p>  
   <p class="color">红色</p>  
   </li>  
     
   <li class="byd" id="car\_002" href="http://www.byd.com/">  
   <p class="name">比亚迪</p>  
   <p class="model">秦</p>  
   <p class="price">15万</p>  
   <p class="color">白色</p>  
   </li>  
  </ul>
* **匹配演示**
* 【1】查找所有的li节点  
   //li  
  【2】获取所有汽车的名称: 所有li节点下的子节点p的值 (class属性值为name）  
   //li/p[@class="name"]  
   //p[@class="name"]  
   //ul[@class="CarList"]/li/p[@class="name"]  
  【3】获取ul节点下第2个li节点的汽车信息: 找比亚迪车的信息  
   //ul[@class="CarList"]/li[2]/p   
  【4】获取所有汽车的链接: ul节点下所有li子节点的href属性的值  
   //ul[@class="CarList"]/li/@href  
    
  【注意】   
   1> 只要涉及到条件,加 [] : //li[@class="xxx"] //li[2]  
   2> 只要获取属性值,加 @ : //li[@class="xxx"] //li/@href
* **选取节点**
* 【1】// : 从所有节点中查找（包括子节点和后代节点）  
  【2】@ : 获取属性值  
   2.1> 使用场景1（属性值作为条件）  
   //div[@class="movie-item-info"]  
   2.2> 使用场景2（直接获取属性值）  
   //div[@class="movie-item-info"]/a/img/@src  
     
  【3】练习 - 猫眼电影top100  
   3.1> 匹配电影名称  
   //div[@class="movie-item-info"]/p[1]/a/text()  
   3.2> 匹配电影主演  
   //div[@class="movie-item-info"]/p[2]/text()  
   3.3> 匹配上映时间  
   //div[@class="movie-item-info"]/p[3]/text()  
   3.4> 匹配电影链接  
   //div[@class="movie-item-info"]/p[1]/a/@href
* **匹配多路径（或）**
* xpath表达式1 | xpath表达式2 | xpath表达式3
* **常用函数**
* 【1】contains() : 匹配属性值中包含某些字符串节点  
   1.1> 查找id属性值中包含字符串 "car\_" 的 li 节点  
   //li[contains(@id,"car\_")]  
    
  【2】text() ：获取节点的文本内容  
   2.1> 查找所有汽车的价格  
   //ul[@class="CarList"]/li/p[@class="price"]/text()
* **终极总结**
* 【1】xpath表达式的末尾为: /text() 、/@href 得到的列表中为'字符串'  
     
  【2】其他剩余所有情况得到的列表中均为'节点对象'   
   [<element dd at xxxa>,<element dd at xxxb>,<element dd at xxxc>]  
   [<element div at xxxa>,<element div at xxxb>]  
   [<element p at xxxa>,<element p at xxxb>,<element p at xxxc>]
* **课堂练习**
* 【1】匹配汽车之家-二手车,所有汽车的链接 :   
   //li[@class="cards-li list-photo-li"]/a[1]/@href  
   //a[@class="carinfo"]/@href  
  【2】匹配汽车之家-汽车详情页中,汽车的  
   2.1)名称: //div[@class="car-box"]/h3/text()  
   2.2)里程: //ul/li[1]/h4/text()  
   2.3)时间: //ul/li[2]/h4/text()  
   2.4)挡位+排量: //ul/li[3]/h4/text()  
   2.5)所在地: //ul/li[4]/h4/text()  
   2.6)价格: //div[@class="brand-price-item"]/span[@class="price"]/text()

## **==lxml解析库==**

* **安装**
* 【1】Ubuntu: sudo pip3 install lxml  
  【2】Windows: python -m pip install lxml
* **使用流程**
* 1、导模块  
   from lxml import etree  
  2、创建解析对象  
   parse\_html = etree.HTML(html)  
  3、解析对象调用xpath  
   r\_list = parse\_html.xpath('xpath表达式')
* **xpath最常用**
* 【1】基准xpath: 匹配所有电影信息的节点对象列表  
   //dl[@class="board-wrapper"]/dd  
   [<element dd at xxx>,<element dd at xxx>,...]  
     
  【2】遍历对象列表，依次获取每个电影信息  
   item = {}  
   for dd in dd\_list:  
   item['name'] = dd.xpath('.//p[@class="name"]/a/text()').strip()  
   item['star'] = dd.xpath('.//p[@class="star"]/text()').strip()[3:]  
   item['time'] = dd.xpath('//p[@class="releasetime"]/text()').strip()[5:15]
* **猫眼电影-xpath**
* """  
  猫眼电影 - xpath  
  """  
  import requests  
  from lxml import etree  
  import time  
  import random  
    
  class MaoyanSpider(object):  
   def \_\_init\_\_(self):  
   self.url = 'https://maoyan.com/board/4?offset={}'  
   self.headers = {  
   '''Accept-Encoding''': '''gzip, deflate, br''',  
   '''Upgrade-Insecure-Requests''': '''1''',  
   '''User-Agent''': '''Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36''',  
   }  
   # 计数变量  
   self.i = 0  
    
   def get\_html(self,url):  
   html = requests.get(url=url,headers=self.headers).text  
   # 直接调用解析函数  
   self.parse\_html(html)  
    
   def parse\_html(self,html):  
   p = etree.HTML(html)  
   item = {}  
   # 1.基准xpath: dd节点对象列表 [dd1,dd2,dd3]  
   dd\_list = p.xpath('//dl[@class="board-wrapper"]/dd')  
   # 2.for循环遍历,依次提取每个电影信息  
   for dd in dd\_list:  
   item['name'] = dd.xpath('.//p[@class="name"]/a/text()')[0]  
   item['star'] = dd.xpath('.//p[@class="star"]/text()')[0].strip()  
   item['releasetime'] = dd.xpath('.//p[@class="releasetime"]/text()')[0].strip()[5:15]  
    
   print(item)  
   self.i += 1  
    
   def save\_html(self,film\_list):  
   item = {}  
   for film in film\_list:  
   item['name'] = film[0].strip()  
   item['star'] = film[1].strip()  
   item['time'] = film[2].strip()[5:15]  
   print(item)  
   self.i += 1  
    
   def run(self):  
   for offset in range(0,91,10):  
   url = self.url.format(offset)  
   self.get\_html(url)  
   # 休眠  
   time.sleep(random.uniform(0,1))  
   print('数量:',self.i)  
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   start = time.time()  
   spider = MaoyanSpider()  
   spider.run()  
   end = time.time()  
   print('执行时间:%.2f' % (end-start))

## **链家二手房案例（xpath)**

### 实现步骤

* **确定是否为静态**
* 打开二手房页面 -> 查看网页源码 -> 搜索关键字
* **xpath表达式**
* 【1】基准xpath表达式(匹配每个房源信息节点列表)  
   '此处滚动鼠标滑轮时,li节点的class属性值会发生变化,通过查看网页源码确定xpath表达式'  
   //ul[@class="sellListContent"]/li[@class="clear LOGVIEWDATA LOGCLICKDATA"]  
    
  【2】依次遍历后每个房源信息xpath表达式  
   2.1)名称: .//div[@class="positionInfo"]/a[1]/text()  
   2.2)地址: .//div[@class="positionInfo"]/a[2]/text()  
   2.3)户型+面积+方位+是否精装+楼层+年代+类型  
   info\_list: './/div[@class="houseInfo"]/text()' -> [0].strip().split('|')  
   a)户型: info\_list[0]  
   b)面积: info\_list[1]  
   c)方位: info\_list[2]  
   d)精装: info\_list[3]  
   e)楼层：info\_list[4]  
   f)年代: info\_list[5]  
   g)类型: info\_list[6]  
     
   2.4)总价+单价  
   a)总价: .//div[@class="totalPrice"]/span/text()  
   b)单价: .//div[@class="unitPrice"]/span/text()
* **代码实现**
* import requests  
  from lxml import etree  
  import time  
  import random  
  from fake\_useragent import UserAgent  
    
  class LianjiaSpider(object):  
   def \_\_init\_\_(self):  
   self.url = 'https://bj.lianjia.com/ershoufang/pg{}/'  
    
   def parse\_html(self,url):  
   headers = { 'User-Agent':UserAgent().random }  
   # 有问题页面,尝试3次,如果不行直接抓取下一页数据  
   for i in range(3):  
   try:  
   html = requests.get(url=url,headers=headers,timeout=3).content.decode('utf-8','ignore')  
   self.get\_data(html)  
   break  
   except Exception as e:  
   print('Retry')  
    
    
   def get\_data(self,html):  
   p = etree.HTML(html)  
   # 基准xpath: [<element li at xxx>,<element li>]  
   li\_list = p.xpath('//ul[@class="sellListContent"]/li[@class="clear LOGVIEWDATA LOGCLICKDATA"]')  
   # for遍历,依次提取每个房源信息,放到字典item中  
   item = {}  
   for li in li\_list:  
   # 名称+区域  
   name\_list = li.xpath('.//div[@class="positionInfo"]/a[1]/text()')  
   item['name'] = name\_list[0].strip() if name\_list else None  
   address\_list = li.xpath('.//div[@class="positionInfo"]/a[2]/text()')  
   item['address'] = address\_list[0].strip() if address\_list else None  
   # 户型+面积+方位+是否精装+楼层+年代+类型  
   # h\_list: ['']  
   h\_list = li.xpath('.//div[@class="houseInfo"]/text()')  
   if h\_list:  
   info\_list = h\_list[0].split('|')  
   if len(info\_list) == 7:  
   item['model'] = info\_list[0].strip()  
   item['area'] = info\_list[1].strip()  
   item['direct'] = info\_list[2].strip()  
   item['perfect'] = info\_list[3].strip()  
   item['floor'] = info\_list[4].strip()  
   item['year'] = info\_list[5].strip()[:-2]  
   item['type'] = info\_list[6].strip()  
   else:  
   item['model'] = item['area'] = item['direct'] = item['perfect'] = item['floor'] = item['year'] = item['type'] = None  
   else:  
   item['model'] = item['area'] = item['direct'] = item['perfect'] = item['floor'] = item['year'] = item['type'] = None  
    
   # 总价+单价  
   total\_list = li.xpath('.//div[@class="totalPrice"]/span/text()')  
   item['total'] = total\_list[0].strip() if total\_list else None  
   unit\_list = li.xpath('.//div[@class="unitPrice"]/span/text()')  
   item['unit'] = unit\_list[0].strip() if unit\_list else None  
    
   print(item)  
    
   def run(self):  
   for pg in range(1,101):  
   url = self.url.format(pg)  
   self.parse\_html(url)  
   time.sleep(random.randint(1,2))  
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   spider = LianjiaSpider()  
   spider.run()
* **后续自己完成**
* 【1】将数据存入MongoDB数据库  
  【2】将数据存入MySQL数据库

## **requests.get()参数**

### **查询参数-params**

* **参数类型**
* 字典,字典中键值对作为查询参数
* **使用方法**
* 1、res = requests.get(url=baseurl,params=params,headers=headers)  
  2、特点:   
   \* url为基准的url地址，不包含查询参数  
   \* 该方法会自动对params字典编码,然后和url拼接
* **示例**
* import requests  
    
  baseurl = 'http://tieba.baidu.com/f?'  
  params = {  
   'kw' : '赵丽颖吧',  
   'pn' : '50'  
  }  
  headers = {'User-Agent' : 'Mozilla/4.0'}  
  # 自动对params进行编码,然后自动和url进行拼接,去发请求  
  html = requests.get(url=baseurl,params=params,headers=headers).content.decode()

## **SSL证书认证参数-verify**

* **适用网站及场景**
* 【1】适用网站: https类型网站但是没有经过 证书认证机构 认证的网站  
  【2】适用场景: 抛出 SSLError 异常则考虑使用此参数
* **参数类型**
* 【1】verify=True(默认) : 检查证书认证  
  【2】verify=False（常用）: 忽略证书认证  
  【3】示例  
   res = requests.get(url=url,params=params,headers=headers,verify=False)

## **代理参数-proxies**

* **定义及分类**
* 【1】定义 : 代替你原来的IP地址去对接网络的IP地址  
    
  【2】作用 : 隐藏自身真实IP,避免被封  
    
  【3】种类  
   3.1) 高匿代理: Web端只能看到代理IP  
   3.2) 普通代理: Web端知道有人通过此代理IP访问，但不知用户真实IP  
   3.3) 透明代理: Web能看到用户真实IP，也能看到代理IP
* **普通代理**
* 【1】获取代理IP网站  
   西刺代理、快代理、全网代理、代理精灵、... ...  
    
  【2】参数类型  
   proxies = { '协议':'协议://IP:端口号' }  
   proxies = {  
   'http':'http://IP:端口号',  
   'https':'https://IP:端口号',  
   }
* **普通代理 - 示例**
* # 使用免费普通代理IP访问测试网站: http://httpbin.org/get  
  import requests  
    
  url = 'http://httpbin.org/get'  
  headers = {'User-Agent':'Mozilla/5.0'}  
  # 定义代理,在代理IP网站中查找免费代理IP  
  proxies = {  
   'http':'http://112.85.164.220:9999',  
   'https':'https://112.85.164.220:9999'  
  }  
  html = requests.get(url,proxies=proxies,headers=headers,timeout=5).text  
  print(html)