总论

2022年11月2日 10:37

ｕｒｌ：统一资源定位符：目标网页的路径

ｐｙｃｈａｒm没有播放器，所以通过爬虫下载的视频不能直接播放

url的组成：

协议	主机（服务器名+域名）	端口号	路径	参数	锚点
http/https	服务器名（一般是www）+ xxx.com	80/443		xx = xxxxx等	

http默认端口号是80，https是443

爬虫核心：１、爬取网页：爬取整个网页，包含网页所有内容

２、解析数据：将网页中你得到的数据进行解析

３、爬虫和反爬虫之间的博弈

爬虫的用途：数据分析 / 人工数据集

社交软件冷启动

舆情监控

竞争对手监控

爬虫的分类：１、通用爬虫

像百度、３６０搜索就是这类的，他们爬取所有网页，并整理后提供检索服务

缺点：抓取的数据大多是无用的

不能根据用户的需求来精准获取数据

２、聚焦爬虫

根据需求，实现爬虫程序，抓取所需要的数据

设计思路：

确定要爬取的ｕｒｌ（如何获取）

模拟浏览器通过ｈｔｔｐ协议访问ｕｒｌ（如何访问）

获取服务器返回的ｈｔｍｌ代码

解析ｈｔｍｌ字符串（如何解析）

反爬手段：

１、Ｕｓｅｒ-Ａｇｅｎｔ：

中文名为用户代理，简称UA，它是一个特殊字符串头，使得服务器能够识别用户使用的操作系统及版本，ｃｐｕ类型、浏览器版本、浏览器渲染引擎、语言、插件等

２、代理IP

透明代理：对方服务器可以知道你使用代理，并知道你真实IP

匿名代理：服务器知道你使用代理，且不知道真实IP

高匿名代理：服务器不知道使用代理，更不知道真实IP

代理服务器：

代理的常用功能：

1、突破自身IP访问限制，访问国外站点

2、访问有些单位或团体内部资源（比如大学里的资源）

3、提高访问速度：通常代理服务器都设置一个较大的硬盘缓冲区，当外界有信息通过时，同时也将其保存到缓存区中，当其他用户再访问相同的信息时，则直接由缓冲区中取出信息，传给用户，提高访问速度

4、隐藏真实IP，免受攻击

3、验证码（能解决）

打码平台、超级鹰等

4、动态加载网页

网站返回的是js数据，并不是网页真实的数据

selenium驱动真实的浏览器发送请求

5、数据加密

分析返回的js代码

爬虫中的异常：

1、URLError

2、HTTPError，是HTTPError的子类，因为url中的组成部分中有HTTP

导入包urllib.error.HTTPError，urllib.errorURLError

http错误：是针对浏览器无法连接到服务器而增加出来的错误提示，引导并告诉浏览者该页是哪里出了问题。

通过urllib发送请求的时候，有时候会发送失败，这个时候如果想让你的代码更加健壮，可以通过try-except进行捕获异常，异常有两类，即上述两个

```
importurllib.request
importurllib.error

#下面是个不存在的网页
url='http://www.nishizhu.com'
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)
AppleWebKit/537.36(KHTML,likeGecko)
Chrome/100.0.4896.75Safari/537.36'}
try:
request=urllib.request.Request(url=url,headers=headers)
response=urllib.request.urlopen(request)
content=response.read().decode('utf-8')
excepturllib.error.HTTPError:
```

```
print('有故障，但我不说在哪')
exceptурllib.error.URLError:
print('还是有故障，但我还是不说在哪')
```

其他和爬虫相关的知识

Web API是网站的一部分，用于与使用具体URL请求特定信息的程序交互，这种请求称为API调用。请求的数据将以予处理的格式（如JSON和CSV）返回，依赖外部数据源的大多数应用程序依赖于API调用。

urllib

2022年3月29日 10:24

urllib库（python）自带，但需导入

urllib.request.urlopen()模拟浏览器向服务器发送请求

response服务器返回的数据，其数据类型是HTTPResponse

字节->字符串

解码decode

字符串->字节

编码encode

read()字节形式读取二进制，按照一个字节一个字节去读。read（5）表示返回前5个字节

readline()读取一行

readlines()一行行读取，直至结束。

注意如果是这个方法读取，则后面不能跟.decode()语句，不然会报错

getcode()获取状态码

geturl()获取url

getheaders()获取headers

urllib.request。urlretrieve()

其中第一个参数是url，第二个参数是文件名（要加拓展名），可请求网页、请求图片、请求视频

在pycharm运行结果界面找到空位，ctrl+f可以进行结果里面的搜索

```
#使用urllib获取源码
#1、首先需要定义一个url，就是你要访问的地址
importurllib.request
url='http://www.baidu.com'
#2、模拟浏览器向服务器发送请求
response=urllib.request.urlopen(url)
#3、获取响应中的页面源码，用content（目录）这个变量来保存，read返回的是字节形式的二进制数据（b代表）
#将二进制数据->字符串（解码，decode(编码的格式)）
content=response.read().decode('utf-8')
#4、打印数据
print(content)
```

```
print(response.getcode())
```

打印这个response返回的状态码

可以通过状态码判断代码有没有问题，如果是200，是对的，如果是404等就是错了

```
importurllib.request
#url代表下载的路径，filename文件的名字，retrieve是返回、取回的意思
url_img='https://tse3-mm.cn.bing.net/th/id/OIP-C.aCIbCDfMNTOrckgrIJmwzgHaJ2?w=182&h=243&c=7&r=0&o=5&dpr=1.25&pid=1.7'
urllib.request.urlretrieve(url_img,'lisa.jpg')
```

```
importurllib.request
url='https://www.baidu.com'
#这个用户代理很像字典，我们可以用字典来存储它
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)AppleWebKit/537.36(KHTML,likeGecko)Chrome/99.0.4844.82Safari/537.36'}
#因为urlopen方法中不能存储字典，所以headers不能传递过去
#请求方法的定制
request=urllib.request.Request(url=url,headers=headers) #这里的传递参数，要传递关键字参数，不然会报错，因为第二个参数并不是headers，而是data，或者在这两个参数之间加个None也可以。
response=urllib.request.urlopen(request)
content=response.read().decode('utf-8')
print(content)
```

而urllib.parse.quote()方法就可以将汉字转化成Unicode编码（需提前导入urllib.parse库）

```
importurllib.parse
url='https://www.baidu.com/s?wd='
name=urllib.parse.quote('周杰伦')
url=url+name
```

这样的话，再按照原来的方法，就能成功运行，但这个方法在其他字符多的时候只能一个个转码

而如果要一次性转码，则需要使用urllib.parse.urlencode()方法

```
importurllib.parse
data={
'wd':'周杰伦',
'sex':'男',
'location':'中国'}
a=urllib.parse.urlencode(data)
print(a)
```

结果为wd=%E5%91%A8%E6%9D%B0%E4%BC%A6&sex=%E7%94%B7&location=%E4%B8%AD%E5%9B%BD

上面都是get请求方式，除此之外还有post请求方式
post请求方式是服务端和个人端双向通信的，而get方法只要求服务器传输数据来，前者相当于使用翻译时，你要传送你所需要翻译的文字给服务器，而后者相当于单纯下载东西
post请求的参数，必须要进行编码（data后面还要跟.encode('utf-8')），这里的data就是Request（）里面的第二个参数

post的请求的参数，是不会拼接在url后边的，而是需要放在请求对象定制的参数中

写爬虫时一定要注意：在headers里面，有一行复制过来的时候必须要注释掉：
' Accept-Encoding':'gzip, deflate, br'
不然的话会报错

headers不一定要把所有数据全部留下，必须留下的只有Cookies，如果还是报错，那么把UA（user-agent）留下。不同网站对需要的标头可能不一样，灵活选择

开始时爬虫最难的是寻找接口

下载豆瓣前10页，页数很多，get方法通过不同的页码观察请求url的不同来寻找规律：

```
importurllib.request
importurllib.parse

defcreate_request(page):
base_url='https://movie.douban.com/j/chart/top_list?type=6&interval_id=100%3A90&action=&'
data={'start':(page-1)*20,
'limit':20}
data=urllib.parse.urlencode(data)
url=base_url+data
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)AppleWebKit/537.36(KHTML,likeGecko)
Chrome/100.0.4896.75Safari/537.36'}
request=urllib.request.Request(url=url,headers=headers)
returnrequest

defget_content(request):
response=urllib.request.urlopen(request)
content=response.read().decode('utf-8')
returncontent

defdownload(page,content):
withopen('douban'+str(page)+'.json','w',encoding='utf-8')asfp:
fp.write(content)

if__name__=='__main__':
start_page=int(input('请输入起始页码：'))
end_page=int(input('请输入终止页码：'))
forpageinrange(start_page,end_page+1):
request=create_request(page)
content=get_content(request)
download(page,content)
```

X-Requested-With:
XMLHttpRequest
标头里面，UA最下面有这个一般都是ajax请求。post方法中请求url可能是一样的，但是参数方面不同页码肯定会有所不同。预览、载荷、标头都要看
Ajax请求，可以实现异步加载，在不重新加载整个网页的情况下更新一部分内容，像往下滑动网页不断有新内容出现、网页中的不同元素来源于不同的网页地址均用这个来实现。因此对于爬虫来说是一道很大的坎
一般而言，打印获取的网页源码，如果数据缺少，首先考虑请求头数据是否给足的问题，然后就要考虑是不是ajax请求的问题

post请求，必须要编码，还要encode，这是和get请求不同的地方
字符串不能是变量

无论是get还是post，都分三步走爬取网页：
1、请求对象定制，将标头、网页等制成一个request
2、获取网页源码：使用urlopen，并且read（）打开并接decode（）解码
3、下载，将下载来的源码命名，解码，并且写入文件当中

有些网站，必须要求你登陆后才能进行数据采集工作等操作。
这种时候，单凭url是进不去的，只能登陆
个人信息页面的确是utf-8的编码格式，但我们数据采集时仍然报错了，这是因为实际上我们根本没有进入个人信息界面，而是网站让你跳转到了登录界面，这个登录界面不是utf-8的编码格式
还是那个解决办法：访问不了，最有可能的情况是，请求头所包含的信息不够
起决定性作用的是cookie，这其中包含着登录信息，如果有登录之后的cookie，那么我们就可以携带着cookie进入到绝大多数页面
当然只保留cookie也不是万能的
referer：判断当前路径是不是由上一个路径进来的，一般用作图片防盗链，比如你不是通过登录页面而来的，就不让下载图片。有的网站验证referer，有的不验证

Handler处理器：
urllib.request.urlopen(url)不能定制请求头
urllib.request.Request(url, data, headers)可以定制请求头
而Handler可以定制更高级的请求头（随着业务逻辑的复杂，请求对象的定制已经满足不了我们的需求，动态cookie和代理不能使用请求对象的定制）

Handler处理器的基本使用:

```
importurllib.request

url=''
headers={}
request=urllib.request.Request(url,None,headers)
#(1)获取handler对象
handler=urllib.request.HTTPHandler()
#(2)获取opener对象
opener=urllib.request.build_opener(handler)
#(3)调用open方法，这一步很像urlopen
response=opener.open(request)

content=response.read().decode('utf-8')
print(content)
```

代码配置代理（用别人的IP来访问网站）：
创建Request对象
创建ProxyHandler对象
用handler对象创建opener对象
使用opener.open函数发送请求

使用handler的代理和handler的基本使用非常相似（注意http和https有所不同）：

```
importurllib.request

url=''
headers={}
request=urllib.request.Request(url,None,headers)
#proxy意思是代理权，代理人（.n），代理的（.adj）。proxies是复数形式，这里的是一个字典
#key是'http'和'https'，而值则是IP:PORT形式的字符串（主机：端口号）
proxies={'http':'121.230.211.142:3256'}
#(1)获取handler对象
handler=urllib.request.ProxyHandler(proxies=proxies)
#(2)获取opener对象
opener=urllib.request.build_opener(handler)
#(3)调用open方法，这一步很像urlopen
response=opener.open(request)

content=response.read().decode('utf-8')
print(content)
```

注意和上面没有代理的进行对比，两者其实非常像，只是加上了proxies这个变量，并在获取handler对象时换了个方法
网上有很多免费的代理，但大多都质量不高，质量高的都要钱

代理池：一个强大的爬虫批，肯定会有一个代理池，里面有许多高匿IP，在爬取网站的时候就不用担心被封IP了
我们可以自己写代理池，用列表来实现，而列表的元素就是一个个字典，字典即是proxies的表现形式。其实，我们可以用爬虫先爬取免费IP网站的那些IP地址和端口号，然后整理成列表，做成代理池，然后用这代理池继续爬取那些有限制的网站

```
importurllib.request
importrandom

proxies_pool=[]
proxies=random.choice(proxies_pool)
url=''
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)AppleWebKit/537.36(KHTML,likeGecko)
Chrome/100.0.4896.75Safari/537.36'}
request=urllib.request.Request(url=url,headers=headers)
handler=urllib.request.ProxyHandler(proxies=proxies)
opener=urllib.request.build_opener(handler)
response=opener.open(request)
content=response.read().decode('utf-8')
withopen('daili.html','w',encoding='utf-8')asfp:
fp.write(content)
```

解析网页元素

2022年4月5日　　22:06

使用解析来获取想要的数据，比如网页中的某一个特定的栏目，可以使用正则表达式完成
当然也有专门的插件（包）来完成，常见的有：xpath、jsonpath，beautifulsoup
xpath：要提前安装xpath，是在chrome的一个插件

用快捷键ctrl+shift+x打开和关闭xpath快捷窗口，这一操作在chrome的新标签页中无法生效
还要在pycharm中安装lxml库：pip install lxml

```
#xpath解析：（1）本地文件，（2）服务器响应的数据，即response.read().decode()[主要]
fromlxmlimportetree
#解析本地文件，使用etree.parse()方法
tree=etree.parse('xx.html')
#解析服务器响应文件，使用etree.HTML()方法
tree=etree.HTML(response.read().decode('utf-8'))
```

xpath基本语法：
1、路径查询：
//查找所有子孙节点，不考虑层级关系
/找直接子节点
2、谓词查询：
//div[@id] (这里的div视情况而定的，不是一个固定的语法，下面的id、 maincontent等同样)
//div[@id="maincontent"]
3、属性查询：
//@class
4、模糊查询
//div[contains(@id, "xx")]
//div[starts-with(@id, "xx")]
5、内容查询：
//div/h1/text()
6、逻辑运算：
//div[@id="head" and @class="s_down"]
//title | //price

其中模糊查询和逻辑运算用的不多，其他都非常重要，能熟练使用xpath和lxml，就可以很轻松地
获取网页中想要的数据，比正则表达式方便许多

熟练使用xpath基本语法，确定xpath路径（这其中就包含了网页位置、网页中想要的东西）
再通过tree.xpath('xpath路径')语句来完成工作。html_tree是变量名，其来源见上方代码。
下面是一些实例：

```
tree.xpath('//body/ul/li')
```

表示查找body下面的ul下面的li，/只能“找儿子而不能找孙子”，如果找不到，则会返回一个空列表，而//可以找所有血缘关系的

```
li_list=tree.xpath('//ul/li[@id]')
```

表示查找所有有id属性的li标签

```
li_list=tree.xpath('//ul/li[@id]/text()')
```
获取指定标签（id）的内容

```
li_list=tree.xpath('//ul/li[@id="l1"]/text()')
```
找到id为l1的标签，注意引号的问题，在单引号里面要加双引号，不然直接炸裂
标签id只是一个名字，可能很多不同的值都有着这个标签（id），但是值不一样

```
li=tree.xpath('//ul/li[@id="l1"]/@class')
```
查找到id为l1的li的class属性值

```
li_list=tree.xpath('//ul/li[contains(@id,"l")]')
```
查询id中包含l的li标签

```
li_list=tree.xpath('//ul/li[starts-with(@id,"c")]')
```
查询id的值以l为开头的li标签
可以把这些语法混合着用，比如上面的语句中，中括号后边可以直接接/text（），来获取里面的数据

在网页整个页面中，id都是唯一的，相当于数据的一个身份证
下方代码：从百度网页中获取百度一下四个字

```
importurllib.request

url='https://www.baidu.com'
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)
AppleWebKit/537.36(KHTML,likeGecko)Chrome/100.0.4896.75Safari/537.36'}
request=urllib.request.Request(url=url,headers=headers)
response=urllib.request.urlopen(request)
content=response.read().decode()
#解析网页源码，来获取我们想要的数据，利用lxml库
fromlxmlimportetree
#解析服务器响应文件，用HTML方法
tree=etree.HTML(content)
#找到所要文件的位置，利用网页中的xpath插件，验证是否正确，然后将插件
左边框的语句复制到下方的路径中
#注意引号的问题
result=tree.xpath('//input[@id="su"]/@value')
print(result)
```

xpath方法的返回值就是一个列表，而在上面result语句最后加个[0]即可得到字符串，通过序号访问列表元素

懒加载：一些涉及图片多的网站，一般都会使用名为懒加载的技术，就是在一开始不全部将网页中的全部图片一次性加载出来，此时未加载出来的图片是src2而不是src，而当你往下滑的时候，当加载了这张图片时，他的src2就会自动转变为src。这样的话按原来的代码就会报错
解决办法就是直接将src写成src2。

直接定位到图片的标签，然后右键-复制-复制xpath即可，不过后边一般要加/@xxx，将图片和名字，即src和alt区分开来，分别存储。如果没有等号的话，获取网页的文本，就加/text()，如果想获得特定标签的值的话，就用/@

用python做爬虫时，要想绕过封ip这道反爬虫机制，一般有两种方法

- 降低抓取速度。比如每两次请求之间间隔2秒，即设置time.sleep(2)（事先import time）。（你user-agent等伪装得当的情况下，如果2秒请求一次，这是人手动也可以操作出来的，网站就不敢确定你是一个爬虫程序，只有统一个ip 1秒请求很多次，网站才会直接判断你是爬虫）有时设置随机sleep时间，防止网站识别出你访问时间太过均匀
- 可以设置代理IP，使用不断变化的IP来请求，这样就不会被判定为同一个人，请求非常快也无法被判定为爬虫。大量抓取时，如果不想以牺牲爬虫速度为代价获取数据，一般都要设置代理IP。

查看网页编码格式：检查网页源代码-head-charest参数里面就是

```
<head>
        <meta charset="utf-8">
```

有setup.py的模块，进入到这个文件所在的文件夹中，然后双击运行，或者用idle打开，直接运行即可

```
import socket
socket.setdefaulttimeout(30)
```

代表经过30秒后如果还没有下载成功，就自动跳入下一次操作

第二种爬虫方法：jsonpath
适用于解析网页返回的json数据
使用pip安装
注意：jsonpath不像xpath那样可以解析服务器响应的文件，而是只能解析本地文件
基本使用：

```
importjson
importjsonpath
# 表示将json文件加载进来，注意必须有open
obj=json.load(open('json文件路径','r',encoding='utf-8'))
# 表示获取特定字符
ret=jsonpath.jsonpath(obj,'jsonpath语法')
```

找到所要解析的网页，先直接复制该网页然后访问，如果啥东西也没有，说明请求头上可能有问题，不仅仅会校验UA。直接访问网址和你从前一个网址正常进去是不一样的
注意：请求头参数中，以冒号开头的都是不能用的
另外还有一个accept-encoding 开头的也一定要注释掉，不然会报解码错误

获取网页源码后，不能直接使用，还要将开头的jsonp（以及结尾的）给去掉

requests

2022年4月12日 10:36

requests库：和urllib的作用几乎一样，但更好用

```
importrequests
url='https://www.baidu.com'
response=requests.get(url)
print(response.text)
```

返回的是字符串类型的网页源码，而字符串后边不能跟.decode解码

resopnse.encoding = 'utf-8'

设置解码格式

以下用r代表resoponse：

r.url 获取请求的url

r.content 返回的是二进制的源码，因为是二进制，在后边可以跟.decode()加上对应的编码格式来解码

就表现而言，是一堆字典键值对和十六进制东西，需要在后面加上网站编码格式，如.decode('utf-8')

r.status_code 响应的状态码

r.headers 响应的头信息

在urllib中有汉字的url不能直接作为url，必须先将汉字编码，然后再将两个字符串拼接起来

另外，在其urlopen方法中，因为不能存放数据等其他东西，所以必须有请求对象的定制

而在requests中

其get方法有多个参数，其中比较重要的是：url、params参数（传网页的参数）、

这个params也是一个字典，可以理解为params里面的内容拼接到url后面，才是一个完整的地址

使用get后，可以打印response.url，可以看到url已经拼接完成了

response.json()方法直接打印网页的json数据

kwargs字典，headers请求头

我们可以不用请求对象的定制，参数无需urlencode编码

```
importrequests

url='https://www.baidu.com/s?'
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)
AppleWebKit/537.36(KHTML,likeGecko)Chrome/100.0.4896.75Safari/537.36'}
data={'wd':'北京'}
response=requests.get(url,params=data,headers=headers)
content=response.text
print(content)
```

注意：如果网页后边有参数的话，哪网页后边要跟/s?，其中问号（连接符号）可以去掉

Ctrl +/键，可以快速多行注释

post请求方法：

requests.post()

参数有：url、data、kwargs、headers等，使用关键字参数避免出错，其中的data就是我们我们需要的请求参数，和get方法的parmas是一样的，名字不一样而已

post请求方法可以向服务器上传文件，只需把文件打开（必须使用rb，即二进制读取），然后再post方法中加入file的参数即可

在获得文本之后，因为是返回的json文件，如果想要看网页的中文，就需要import json

obj = json.loads(content)

print(obj)即可

代理：

proxy={'http':'代理ip：端口号'}

要使用代理，先将代理ip封装成这个样子，然后proxies = proxy直接传入到get方法中即可，非常方便

当要找需要登陆的网站的登录接口时，在网络里面的名字一栏，找带有login字样的文件，大概率是登录接口，是一个post请求，而这个url一般是以.aspx结尾的

我们会发现登录的时候需要的参数很多，比如说登录账号、密码等，这些都是后面所说的data的一部分

有些参数只在页面源码中，而不在登陆窗口中，所以需要页面源码，进行解析后得到他们的值，然后就可以获取了

在古诗文网的例子中，*__VIEWSTATE和__VIEWSTATEGENERATOR*是未知的参数

带有双下划线的基本是隐藏参数，涉及到隐藏域的问题

有些网站，在你每次进入的时候，请求头的参数都会发生改变，不像UA那样固定，不过虽然每次都变，但是我们在登陆的时候是可以获取到这些参数的，一般要去源码中找。只要把获取这些不断变化的请求头参数的语句放到循环语句中即可

至于验证码（仅针对图片验证码，比如输入字母啥的），将鼠标移动到验证码图片上，然后检查即可快速定位有关元素

在这里破解这类验证码的思路是找到每次随机的验证码所在位置，然后将其下载下来，本地利用超级鹰等平台破解，获取验证码后，往请求方法data里面添加就行，因为你输入的验证码的信息就包含在请求头里

Preserve log储存日志，在chrome检查网页，网络模块中，且一般没有勾选

如果chrome浏览器版本过低，就要把这东西给勾上

因为低版本chrome当你第二次及多次重复同一个操作时，比如多次登陆同一个网站时，登录接口就会被覆盖掉，不再出现。当然我们一般不勾上，因为勾上接口太多找不过来

```
importrequests
#通过cookie绕过登录，进入古诗文网，以bs4为解析网页工具
importrequests

url=''
headers={'User-Agent':'Mozilla/5.0(WindowsNT10.0;Win64;x64)
AppleWebKit/537.36(KHTML,likeGecko)Chrome/100.0.4896.75Safari/537.36'}
#以下两句是为了获取页面的源码
```

```
response=requests.get(url=url,headers=headers)
content=response.text

#解析页面源码，然后获取会发生变化的请求头，在这里是__VIEWSTATE和__VIEWSTATEGENERATOR
frombs4importBeautifulSoup
soup=BeautifulSoup(content,'lxml')

#获取_VIEWSTATE，注意select选择器在获取的元素的前面要加#号
#下面的语句表示对于网页中的元素，找到select里面包含的元素（是某个id），并获取其value值
#select方法返回的是一个列表，我们取第一个元素
viewstate=soup.select('#__VIEWSTATE')[0].attrs.get('value')
viewstategenerator=soup.select('#__VIEWSTATEGENERATOR')[0].attrs.get('value')

#获取验证码图片
code=soup.select('#imgcode')[0].attrs.get('src')
#注意这里获取到的只是src的值，不是一个网址，要想真正得到这张图片，还要加上前缀：https://so.gushiwen.cn，其他网页也类似
code_url='https://so.gushiwen.cn'+code
#这样我们就获得了验证码的地址，将code_url打印得到的地址，点进去就是验证码的图片，而且每次刷新都不一样
#服务器将验证码图片做出来，我们将验证码作为请求头参数传送过去，即可完成验证

#这里要千万注意：因为这里尝试下载这张验证码，每次请求链接都会响应不同验证码，所以到后边post传参的时候，验证码已经变了
#因此不能使用urlretrieve方法来下载图片
#importurllib.request
#urllib.request.urlretrieve(url=code_url,filename='code.jpg')

#requests里面有一个session()方法，通过session的返回值，就能使请求变成一个对象，以此获取验证码
session=requests.session()
#验证码的url的内容
response_code=session.get(code_url)
#注意此时要使用二进制数据，因为这里图片的下载要使用二进制
content_code=response_code.content
#wb模式就是将二进制数据写入到文件，这个可能速度有点慢，可以去文件夹里看文件
withopen('code.jpg','wb')asfp:
fp.write(content_code)
```

```
#获取了验证码的图片，下载到本地，然后观察验证码，然后在控制条输入验证码，然后就可以将其传递给data里面的参数
code_name=input('请输入验证码：')

#登录古诗文网，url_post是登录接口，post代表这是一个post请求
url_post=''
data_post={}
#这里不能使用requests了，而是session，这是为了和前面的获取验证码的操作保持为同一个请求，session=requests.session()
response_post=session.post(url=url_post,headers=headers,data=data_post)
content_post=response_post.text
withopen('gushiwen.html','w',encoding='utf-8')asfp:
fp.write(content_post)
```

破解验证码的平台：有很多，以超级鹰为例（虽然要钱，但还是比较便宜的，有机会可以试着玩玩）
去官网下载，解压，然后将里面的图片和.py文件移动到python文件夹里
打开.py文件，然后分别用用户名和密码代替有关参数，第三个参数按注释操作即可
这样的操作大部分打码平台都是一样的
超级鹰下载的文件最后一行print要加括号，里面的方法返回的是一个字典，验证码的key是pic_str，
所以直接在后边跟.get('pic_str')即可

搞定打码平台以后，就可以识别在线识别验证码了，不过要注意得先下载下来验证码，另外下载的验证码图片、打码文件以及python文件必须位于一个目录下

requests貌似没有像urlretrieve那样简单的方法去直接获取某个地址的文件
但urlretrieve方法除了代码少一点属实垃圾
对于requests的文件下载：
小文件，比如图片，直接写入

```
with open('名字','wb') as f:
    f.write(content)
```

图片一般是用二进制，所以用wb方式
而大文件，那么先下载下来的文件先放到内存中，内存压力比较大，所以为了防止内存不够，要将下载下来的文件 分块写入磁盘

selenium

2022年4月11日 18:24

Selenium：
是一个用于web应用程序测试的工具，直接运行在浏览器中，就像真正的用户在操作一样
支持通过各种driver驱动真实浏览器完成测试
支持无界面浏览器操作

为什么使用selenium?
模拟浏览器功能，自动执行的js代码，实现动态加载
正是因为我们是模拟的，所以才会有各种反爬，而这个工具就是用来解决部分反爬问题的
只是selenium速度有点慢

直接在想要查找的元素上右键然后检查，就可以直接在元素中找到，不用自己去找

有的时候，模拟浏览器向访问并下载网页，服务器不会把所有数据全部给你，比如京东首页不会给你秒杀有关的源码
因为在访问时会校验浏览器内核啥的，所以只能通过浏览器而不是一般的爬虫去获取
这个时候我们就需要使用selenium来驱动一个真实的浏览器解决这个问题

要使用这个工具驱动浏览器访问网页，首先要下载浏览器驱动
https://chromedriver.storage.googleapis.com/index.html
查看自己谷歌浏览器的版本，下载比自己版本同级或者更低的驱动，win32就行
然后将下载的压缩包解压，将解压完成的一个.exe文件放在python文件所在位置，不需要安装，也不需要运行
安装selenium
pip install selenium

使用步骤：
1、导入：from selenium import webdriver
2、创建谷歌浏览器的操作对象
path = 谷歌浏览器驱动文件路径（如果就在python文件同一个文件夹下，路径就是文件全名）
browser = webdriver.Chrome(path) ——注意是大写的C，有的时候也会命名为driver
3、访问网址
url = 网址
browser.get(url)
browser.back() 返回上一界面
如果浏览器没有安装在默认位置，就需要配置环境变量及代码指定浏览器位置

selenium元素定位：自动化要做的就是模拟鼠标和键盘来操作这些元素，点击输入等，操作之前
首先要找到他们，而webdriver提供了很多定位元素的方法
以下是例子：
根据id的值来进行寻找，id是标签，su是值，前几个都是这样，只有后边的@id=才是找标签
注意：在找元素时，有些网页可能是动态加载或者延迟加载，就是说网页元素还没加载出来就已
经开始定位了，这样是肯定定位不到元素的，可以在打开网页后先sleep一会儿再进行定位

```
button=browser.find_element_by_id('su')
name=browser.find_element_by_name('wd')
xpath1=browser.find_element_by_xpath('//input[@id="su"]')
names=browser.find_elements_by_tag_name('input')
my_input=browser.find_elements_by_css_selector('#kw')[0]
browser.find_element_by_link_text('新闻')
```

注意：找元素时，find_elements和find_element是不一样的，后者如果有多个结果只会返回第一个结果

注意：在查找元素的时候，认清元素和“元素的某一个值”的区别，所有查找的方法均是查找元素而不能查找元素的值，需要与下方的方法配合获取元素的值

注意：这样使用以上方法时，会提示方法已过时，只要先导入了warnings，并且忽略过时的检查，就没事。

最常用的就是根据id、xpath来使用

访问元素信息（必须在网页还开着的情况下使用）：

获取元素属性：.get_attribute('class')

获取元素文本：.text

获取id ： .id

获取标签名：.tag_name

查找id为su的元素，并打印其class的值

注意：按id找该元素，但该元素不止有id，还有class等属性

```
button=browser.find_element_by_id('su')
print(button.get_attribute('class'))
```

鼠标动作链：

from selenium.webdriver import ActionChains

ActionChains(browser).move_to_element(element).perform()

其中element表示网页中的一个元素，注意并不是某个元素某个属性的值，表示移动到某个元素所在的位置

ActionChains(browser).move_to_element(ac).click(ac).perform()

其中ac表示一个元素，表示点击某个元素（左键）

1. driver.quit() #退出相关驱动程序,并关闭所有窗口
2. driver.close() #关闭当前一个窗口

无界面浏览器：因为不需要进行css和gui渲染，其速度要比真实浏览器以及原生selenium快很多

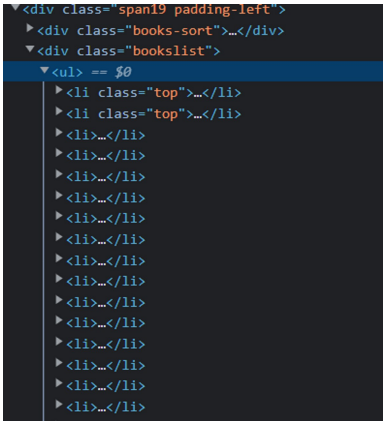
phantomjs和handless就是两个典型的无界面浏览器，他们均基于selenium

scrapy

2022年4月13日　1:01

scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架，可以应用在包括数据挖掘、信息处理或存储历史数据等一系列程序中，是企业级爬虫工程中用的最多的库。
框架可以让我们的效率更高，scrapy爬取速度快

结构性数据：所有网页所有的东西都是一个结构，例如：



像这个，网页的关键数据都是一行行列出来的，而在网页的元素中就表现出了这种结构性，结构性里边的元素都是高度类似的，只是某些值不一样

安装：pip install scrapy
在cmd中，d：就是去到d盘
cd 名字就是要去到的路径，一般用cd，然后把桌面python文件拖到cmd中，就能获取进入python文件的路径，在pycharm终端中同样能够直接拖
注意：scrapy项目文件名不能使用数字开头，也不能使用中文
注意：scrapy项目不是和平常的写python代码那样在pycharm中搞，而是要先通过cmd进入到想要创建项目的路径里面，然后用cmd创建项目
快速打开cmd：win+x然后点击powershell，
　　或者win+r，然后输入cmd
创建语法：scrapy startproject 项目名字
创建之后，在python文件夹里会出现一个新文件夹，专门用来搞爬虫的
因为有了pycharm，更推荐直接在pycharm的终端中创建scrapy框架，效果是一样的还省去了找路径的麻烦
项目组成：
自动创建的文件夹里，有多个.py文件和一个叫scrapy的文件夹
其中，__init__.py是初始化文件
items.py定义数据结构的文件，爬取的数据都包含哪些，是一个继承scrapy.item的类
middlewares.py中间件，代理
pipelines.py管道文件，里面只有一个类，用于处理下载数据的后续处理，默认是300优先级，值越小优先级越高（1-1000）
settings.py配置文件，比如是否遵守robots协议，UA定义等
而在scrapy文件夹里，只有一个初始化文件，而我们要创建一个新的.py文件，是实现爬虫的核心功能的文件，我们所有主要代码都在这里边
而这个新文件，需要我们在cmd或者pycharm的终端中进入到spiders文件夹中，然后创建
创建语法：scrapy genspider 爬虫文件的名字 要爬取的网页，如：
scrapy genspider baidu www.baidu.com
之后文件夹里就会出现一个叫baidu的新的py文件，如果没有，先收起文件夹再打开
注意看一下网址有没有搞错
一般情况下网址不需要写http的协议，直接三个w，他会自动给你加上

运行爬虫程序（注意要在文件所在的目录下运行终端）：
scrapy crawl 爬虫名字，比如scrapy crawl baidu

robots协议：一种约定，里面有写禁止爬取的网站区域，但该约定完全可以不遵守
打开robots协议：输入网站的 URL，然后在 URL 后面添加 "/robots.txt"，如果有就会打开，如果没有就
例如，以下是bilibili的一部分robots.txt：

```
User-agent: *
Disallow: /medialist/detail/
Disallow: /index.html

User-agent: Yisouspider
Allow: /

User-agent: Applebot
Allow: /
```

表示所有爬虫都不能访问medialist/detail以及index下的文件

日志信息和日志级别：
CRITICAL严重错误
ERROR一般错误
WARNING警告
INFO一般信息
DEBUG调试信息
默认的日志等级是debug，而只要出现了debug以上等级的日志，那么这些日志将会被打印，也就是说所有的日志信息都将被打印

setting文件设置：
在scrapy的这个配置文件中，加上语句
LOG_FILE:将屏幕显示的信息全部记录到文件中，屏幕不再显示，注意文件后缀一定是.log
比如下面的语句，就是将所有日志文件写入文件中而不是直接打印出来，文件名是自己取的，但是后缀必须是.log

```
LOG_FILE='LOGWENJIAN.log'
```

LOG_LEVEL:设置日志显示的等级，就是显示哪些，不显示哪些，如

```
LOG_LEVEL='WARNING'
```

就是指定只有warining及以上的日志信息才会被打印出来，不会影响我们看打印出来的数据

一般来说日志文件还是挺重要的，一般将日志文件写入文件中，要么就指定只有warning和以上的日志才打印

下面是Yisouspider、Applebot的爬虫可以访问所有内容

默认情况下是遵守的，所以在使用scrapy的时候有时会出错，我们进入到setting.py里面，把robotstxt_obey直接改为false，或者直接注释掉，所以每次创建完新新的scrapy项目后第一件事就是把这协议改掉
注意：不能将python文件名直接命名为任何模块的固定的名字，不然导入的时候很有可能就导入错误，显示在模块中找不到方法

```
importscrapy
classBaiduSpider(scrapy.Spider):
#爬虫的名字，用于运行爬虫的时候使用的值
name='baidu'
#允许访问的域名
allowed_domains=['www.baidu.com']
#起始的url地址，第一次访问的域名
#第二次访问的域名就是比如说第一次的爬完了，点进去下一页也是我们要爬的东西
start_urls=['http://www.baidu.com/']

#是执行了start_urls之后执行的方法，方法中的response就是返回的那个对象
#相当于response=urllib.request.urlopen()
#也相当于response=requests.get()
defparse(self,response):
pass
```

注意：parse方法会提示什么签名不匹配什么的，不用管
这里运行程序只能通过特定语句在cmd或者终端 中运行，不能直接运行
注意：有的时候，网页网址或者返回的数据里面包含有些特殊的字符，比如之前遇到的那个/，这样我们需要去处理，当然有的时候，只要能正常运行，也可以不管

在输入命令创建的python文件中，parse方法有个response参数这个参数和以前获取的得到的response参数很类似，方法也差不多，估计也是通过类似get或者urlopen这种方法得到的response
它的方法有：response.text得到的是响应网页源码的字符串
response.body得到的是二进制的源码（多了个b开头）
前两个用的不多，你都可以直接用xpath提取了还要源码干什么
response.xpath（'路径'）
没错，不再需要导入xpath，也不要使用lxml那些东西，直接就可以使用xpath方法，但是需要注意：这里的xpath返回的是一个selector对象，而不单纯是一个列表对象，因此需要需要使用下面的extract（）等方法
令 span = response.xpath()，返回的是有个列表，注意：这里的路径一定要搞正确，而且在xpath插件中显示正确的路径不一定就是我们需要的路径，一般还得在后边加点东西，比如@某个值或者加上/text()来获取文本
span.extract() 获取selector对象的data属性值，这个data就是我们需要的值，而这些属性值仍然存储在列表中，也就是说这个方法返回的仍然是一个列表，只是把数据提取出来了
这个selector（选择器）就是列表的意思
span.extract_first()提取的是selector列表的第一个数据，返回的不再是一个列表，如果要提取数据，可以用这个一步到位，其操作等同于span.extract()[0]
如果提示超出列表范围，可能是因为有的网页标签中是空值，使用try-except方法跳过这些空值，这个方法用的特别多，没有判断长度等问题

这个框架打印出来的东西有很多，找寻数据需要一定的时间

scrapy架构组成：
1、引擎：自动运行，无需关注，会自动组织所有请求对象，分发给下载器
2、下载器：从引擎处获取到请求对象后，请求数据
3、spiders：Spider类定义了如何爬取某个或某些网站，包括爬取的动作（例如：是否跟进链接）以及如何从网页中的内容中提取结构化数据（爬取item，可以理解为目标）。话句话说，spider就是定义爬取的动作及分析某个网页的地方
4、调度器：有自己的调度规则，无需关注
5、管道（Item pipeline）最终处理数据的管道，会预留接口供我们处理数据

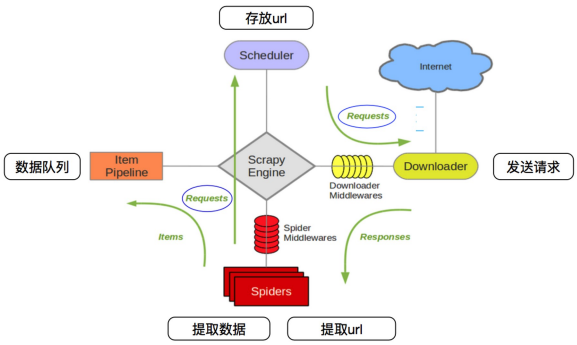
另外还有两个基本（暂时）用不到的东西
Downloader Middlewares（下载中间件）：可以自定义扩展下载功能的组件（代理、cokies等）。
Spider Middlewares（Spider中间件）：可以自定扩展和操作引擎和Spider中间通信的功能组件（比如进入Spider的Responses;和从Spider出去的Requests）

当item在spider中被收集之后，它会被传递到item pipeline，一些组件会按照一定的顺序执行对item的

处理。每个item pipeline组件（有时称为“item pipeline”）是实现了简单方法的python类，他们呢接收到item并通过它执行一些行为。同时也决定次item是否继续通过pipeline，或是被而不再进行处理

以下是item的一些典型应用：
1、清理html数据
2、验证爬取的数据（检查item包含某些字段）
3、查重（并丢弃）
4、将爬取的结果存储到数据库中
其典型语法为：xxx = scrapy.Field()
xxx即为赋值了数据的变量，比如
price=li.xpath('.//p[@class="price"]/span[1]/text()').extract_first()
这里的price变量存储了来自xpath路径下找到的有关图书的价格信息，通过extract_first()，price代表的是一个个价格的字符串，而item就可以处理这些字符串，xxx在这里即是price

爬虫工作原理：



Scrapy shell 是一个终端，供你在未启动spider的情况下调试你的代码，其本意是用来测试提取数据的代码，不过您可以将其作为正常的python终端，在上面测试任何python代码
该终端是用来测试xpath和css表达式，查看他们的工作方式，及从爬取的网页中提取数据，在编写你的spider的时候，该终端提供了交互性测试你的表达式代码的功能，免去了每次修改后运行spider的麻烦
一旦熟悉了spider终端后，会发现在其开发和调试spider时发挥的巨大作用
Scrapy shell也能导入库，比如from scrapy.linkextractors import LinkExtractor
流程、语法是一样的

注意：不要先进入scrapy shell再加入网址，而是：
直接输入语句：scrapy shell 网址（这个网址不需要加协议）
在任何路径下都可以这样用
如果想要高亮输入和自动补全等，需要安装ipython： pip install ipython
如果安装ipython，则scrapy终端就会使用ipython替代原有的终端，语法和原来一样的
并且里边已经给你封装好了一些参数，比如resquest和response都有
输入response.body即可打印二进制的网页源码，response.text输出字符串源码

这东西不需要你事先创建任何scrapy文件，只需要一个你想要的网址，相当于是一个已经封装好了的小爬虫，可以便捷获取网页数据，但熟悉了scrapy之后，貌似用的不多

数据结构：通常地说，就是说你要下载的数据都有什么
先尝试着下载所需要的东西，如果下载到了别的东西或者根本没有数据，那么其一是检查xpath路径，其二就是检查反爬，以当当网为例：你以为src值就是图片地址，然而其src值只有在你滑动到那个地方的时候才会变成地址，真正的地址一直存储在也该叫data-origin里边
还需注意：有的时候第一个是和后边不一样的，比如这里的第一张图片没有data-origin，src存储的就是实际地址

多注意自己单词有没有写错，这是会经常出现的问题

```
defparse(self,response):
li_list=response.xpath('//ul[@id="component_59"]/li')
#所有的selector对象都可以再次调用xpath方法
forliinli_list:
src=li.xpath('.//img/@data-original').extract_first()
#第一张图和其他图片的标签的属性是不一样的
ifsrc:
src=src
else:
src=li.xpath('.//img/@src').extract_first()
name=li.xpath('.//img/@alt').extract_first()
```

```
price=li.xpath('.//p[@class="price"]/span[1]/text()').extract_first()
print(src,name,price)
```

以上是在自己创建的.py文件中写的代码，其他地方都不用管便可以输出数据
注意多级调用xpath方法

数据的下载保存：
yield：带有yield的函数不再是一个普通函数，而是一个生成器generator，可用于迭代
是一个类似于return的关键字，迭代一次遇到yield后就返回yield后边的值。重点是：下一次迭代时，从上一次迭代遇到的yield后面的代码（下一行）开始执行
简要理解yield：就是return返回一个值，并且记住这个返回的位置，下次迭代就从这个位置后边（下一行）开始

如果想使用管道的话，必须在setting中开启管道
是
```
ITEM_PIPELINES={
'dangdangwang.pipelines.DangdangwangPipeline':300,
}
```
在这三行元素，将其注释解开
管道可以有很多个，300即优先级，优先级越小越高

在管道里，我们将数据写入文件可以是：
```
classDangdangwangPipeline:
#item就是yield后面的book对象
defprocess_item(self,item,spider):
#以下这种模式不推荐，因为每传递过来一个对象，那么就打开一次文件，对文件操作过于频繁
#1、write方法必须是写入字符串，而不是一个对象，解决方法就是str（）强制转换
#2、w模式，会每个对象都打开一次文件，覆盖以前的内容，所以不能使用w模式而应使用a模式（追加）
withopen('book.json','a',encoding='utf-8')asfp:
fp.write(item)
returnitem
```
但是以上的方法并不推荐，因为对文件的写入太过于频繁，所以我们可以这样：
```
classDangdangwangPipeline:
defopen_spider(self,spider):
#这里可以用w的模式是因为打开了，但一直没关
self.fp=open('book.json','w',encoding='utf-8')
defprocess_item(self,item,spider):
self.fp.write(str(item))
returnitem
defclose_spider(self,spider):
self.fp.close()
```

pipelines可以支持多条管道同时下载，一边下载图片一边下载网页的json数据，需要在pipeline文件里照着上边的样子再定义新的类，注意管道里是有返回值return的
1、首先定义管道类
2、再setting中手动开启管道
照着上面的语句抄就行了，注意 . 后面跟的名字要改一下，改成定义的管道类的名字，然后手动给一个优先级，将其加入到ITEM_PIPELINES这个字典里

下载下来的json数据，网址可能存在问题，比如没有http：这些东西啥的，这个时候在下载图片或者其他东西时就要手动在url面前拼接上

注意下载优先级的问题：比如在这个下载当当网数据的例子中，下载图片的url是通过item（也就是我们创建的book这样一个json数据文件中得到的），所以优先级下载json必须高于（即数字小于）下载图片

urlretrieve在下载图片、视频等文件时真的很方便，注意这个方法并不意味着我们在此之前必须使用urlopen等方法，而是我们可以单独使用它，只要用其他方法获得想要下载的文件的地址就行了

多页下载：绝大多数网站，每一页下载的东西在逻辑上是一样的，只是网址有点点不一样而已（网址之间也有规律），因此我们只需传入不同的网址，然后重复运行parse方法（我们下载第一页的方法）即可

截至目前scrapy各个模块的使用方法：

setting：将遵守的协议改为False，将默认被注释掉的管道给打开，如果多管道下载则需要将新管道添加进来，管道不打开仍然能在终端中打印数据，但是却不能将数据写入你的文件中
pipelines：构建下载方法的地方（如使用urlretrieve、get和write等方法下载图片或者其他文件），不同文件不同的下载方法，管道也不一样，注意构建完方法后将其添加到setting中
items：定义数据结构的地方，每要下载一种类型的数据，比如价格、图片1、图片2、名字等，就要写一行代码：标准语法：xxx = scrapy.Field()
而在自己创建的文件中，是构造反复反复执行获取网页、获取资源路径的方法的地方，比如xpath，不同类型的数据的定义都在此

当当网爬取多页的代码，以下分别是pipelines和自行创建文件的代码：
pipelines的下载方法都要有返回值item

```
#如果想使用管道的话，必须在setting中开启管道，使用如下类下载所需网页信息的json文件
classDangdangwangPipeline:
defopen_spider(self,spider):
#这里可以用w的模式是因为打开了，但一直没关，这一步就是为了打开文件
self.fp=open('book.json','w',encoding='utf-8')
defprocess_item(self,item,spider):
self.fp.write(str(item))
returnitem
defclose_spider(self,spider):
self.fp.close()

#下面这条管道用来下载图片，使用urlretrieve方法
importurllib.request
classDangDangDownloadPipeline:
defprocess_item(self,item,spider):
#可以直接从item中获取数据，item是上一个class中返回的东西，是一个字典，在这个例子就是book
#所以可以使用.get方法获得src和name的值
url='http:'+item.get('src')
filename='./books/'+item.get('name')+'.jpg'
filename=filename.replace('/','_')
urllib.request.urlretrieve(url=url,filename=filename)
returnitem
```

下面是自行定义文件的代码：

```
importscrapy
fromdangdangwang.itemsimportDangdangwangItem

classTushuSpider(scrapy.Spider):
name='tushu'
#如果是多页下载，那么必须调整allowed_domains（domian表示领土，范围）的范围，一般情况下只写主机，前后都要删掉一些，有www.的要保留www.
# allowed_domains一般都只写host，它代表的是网页的一个范围，不在这个范围的都不能发起请求，所以范围得搞大点
但是starturl必须是准确的网页地址，不能简化
allowed_domains=['category.dangdang.com']
start_urls=['http://category.dangdang.com/cp01.54.91.02.00.00.html']
page=1

defparse(self,response):
li_list=response.xpath('//ul[@id="component_59"]/li')
#所有的selector对象都可以再次调用xpath方法
forliinli_list:
src=li.xpath('.//img/@data-original').extract_first()
#第一张图和其他图片的标签的属性是不一样的
ifsrc:
src=src
else:
src=li.xpath('.//img/@src').extract_first()
name=li.xpath('.//img/@alt').extract_first()
price=li.xpath('.//p[@class="price"]/span[1]/text()').extract_first()
print(src,name,price)
```

```
book=DangdangwangItem(src=src,name=name,price=price)
#每获取一个book就将book交给pipelines
yieldbook

#每一页爬取的业务逻辑全都是一样的，只是网址有些不同，因此我们只需要多次执行parse方法即可
ifself.page<40:
self.page+=1
url='http://category.dangdang.com/pg'+str(self.page)+'-cp01.54.91.02.00.00.html'
#怎么去调用parse方法
#scrapy.Request就是scrapy的get请求
#url是请求地址，callback是要执行的函数，这里的parse方法不能加括号
callback只能写函数名字符串，callback='parse'，而如果callback就是要调用这个语句所在的函数，那么可以写callback=self.parse
yieldscrapy.Request(url=url,callback=self.parse)
```

注意这里是因为每一页的数据逻辑都是一样的，所以可以重复使用这个parse方法
但是如果访问逻辑不一样，比如先进去一个网页下载其电影名字然后再继续点进去，去下载这部电影的图片，那么就是逻辑不一样了，那样当然不能继续使用parse方法，但我们可以仿照parse方法自己构建一个方法，同样可以用在yield的这个语句中。详情看下面的下载电影天堂的方法

构建scrapy工程之后，首先可以在自己创建的文件中写入打印一些东西的代码，然后直接运行，以此测试该网站是不是有反爬手段，如果打印出来了就没反爬

有的时候xpath helper插件并不能准确显示路径中所得到的结果，经常会少些东西，需要注意
注意：为什么要用两次xpath路径而不直接一步到位呢？因为不这样的话，要同时从列表中解析src和name，要么需要判断长度然后再来搞，要么需要用两次for循环，如果要解析的东西过多，还会要更多的for循环，而两次xpath路径可以解决这个问题
注意第二次使用xpath路径，要加上个点.

截至目前，span和tbody标签都是xpath解析不了的，必须在路径中将其去除
如果拿不到数据，而能进去网页没有报错的情况下，首先考虑是xpath路径的问题

```
#在item中，这是等号左边的是定义的参数，而在自己构建的文件中则显示为关键字参数，下面这行就代表一个关键字参数的构建
src=scrapy.Field()
```

在item中这些语法，实际上是在为item定义关键字参数
所以这些变量一定要注意

CrawlSpider：
继承自scrapy.spider，可以定义规则，在解析html内容的时候，可以根据链接规则提取出指定的链接，然后再向这些链接发送请求，所以如果有需要跟进的链接，比如解析网页后到里边一个个下载图片，或者说提取了链接之后再次爬取，使用CrawlSpider是非常合适的

链接提取器：在这里就可以写规则提取指定链接
scrapy.linkextractors.LinkExtractor()
allow = 正则表达式，提取符合正则的链接
restrict_xpaths = () 提取符合xpath规则的链接
restrict_css = () 提取符合选择器规则的链接

用法：
正则：links1 = LinkExtractor（allow=r'某个正则表达式'），用的比较多
link = LinkExtractor(allow=r'/book/1175_\d+\.html')
其中\d表示数字，而+表示可以有很多个数字
xpath：Links2 = LinkExtractor(restract_xpaths = r'xpath路径')
注意无论是正则还是xpath都要是r开头，而且xpath要加s
注意：xpath的路径问题，后边不能加@来获取值，比如：
link1 = LinkExtractor(restrict_xpaths=r'//div[@class="pages"]/a')
而不能加@href
以上两个方法返回的是一个包含有我们需要的链接的值，但如果要获得链接，还要进行连接提取
提取链接：
link.extract_links(response)
返回的是许多个link对象，其中包含有我们需要的url等信息

这个crawlspider对于请求跟进非常有帮助，注意创建其文件的语法和普通爬虫项目有了改变：
创建项目，语法与前面一致：scrapy startproject 项目名
在项目最里边的spider文件夹中创建python文件：scrapy genspider -t crawl 爬虫文件名 网址
而文件运行方式也和原来是一样的：scrapy crawl 文件名
而由此创建出来的文件跟之前不一样了，不仅定义的类所继承的东西不一样了，下面的rules还出

现了类似正则的东西

正则表达式中点是通配符，除了“换行”以外都可以用点来匹配，因此如果要指定点的话，就需要进行转义，这就是为什么.html 开头的点可能不会生效，需要进行转义，即\.html

爬取数据的时候一定要注意网页的地址，比如第一页可能没有_x这样的东西在网页中，因此数据就没有爬取到

代理：
1、到setting.py中打开选项：
DOWNLOADER_MIDDLEWARES这个选项，将注释给他去掉
2、到middlewares.py中的process_request函数下写代码

scrapy的post请求：
post请求中，想要找到我们发送的参数，首先就需要找到post请求的所有接口，然后在这些接口中的请求头中，或者是载荷中寻找参数

注意post请求必须依赖参数才能成功发送请求，直接访问post的网址是会报错的
post请求如果没有参数，那么这个请求则没有意义
所有start_url没有用了，而依赖于start_url的parse方法也没有用了
因此我们需要自己写函数：

```
importscrapy
importjson

classPostSpider(scrapy.Spider):
name='post'
allowed_domains=['fanyi.baidu.com']
#start_urls=['https://fanyi.baidu.com/v2transapi?from=en&to=zh']
#
#defparse(self,response):
#pass
#系统为我们提供了post请求的方法，这个方法相当于创建一个start_url：
defstart_requests(self):
url='https://fanyi.baidu.com/v2transapi?from=en&to=zh'
data={"query":"final"}
#之前的scrapy.Request()方法是get请求，而post请求则是：
yieldscrapy.FormRequest(url=url,formdata=data,callback=self.parse2)
#如果不使用json，可能会报编码错误，因此导入json，下面这个方法是仿照parse方法写的
defparse2(self,response):
content=response.text
obj=json.loads(content)
print(obj)
```

js逆向技术

2023年4月19日 10:35

对于动态加载网页，有两种方法：

1、用selenium等真实浏览器驱动爬取网页元素

2、使用js逆向破解网站设置的各种反爬和加密，获得真实目标接口

第一种方法虽然能适用几乎所有场景，但效率低下且需要一直运行浏览器和加载没必要的网页，开销较大

第二种方法比较敏感，难度较高，需要不断发现--推断--尝试才能得到结果

JavaScript：一种函数优先的轻量级、解释型或即时编译型的脚本编程语言。广泛应用于web开发，长用来给网页添加各式各样的动态功能。在网页中，js脚本可嵌入在HTML中来实现自身功能，也可单独写成js文件。js可以在浏览器中直接执行

js逆向工程技术：

1、代码反混淆：许多js代码都是经过混淆的（加密），难以理解和修改。通过反混淆工具，还原js的代码，理解其原始意图

2、调试：通过调试工具，可以了解js代码的行为。在调试过程中，可以设置断点，单步执行代码，并监视变量的值。通过调试，可以理解代码内部的工作原理，找到其漏洞

3、注入：通过注入js代码，可以修改页面的行为，利用js执行各种任务

4、修改：通过修改js代码，以改变其行为（如绕过安全机制、隐藏恶意行为等）

方法：可在js文件里打上断点，逐步运行，一步步调试，重现js实现网页异步加载的过程，方便分析

断点：在源代码板块找到js文件（可能有多个，注意区分），点击某一行的行号即可打上断点

可在“网络”面板中的js文件中找到js，右键“在来源面板中打开”，即可打开源代码

常见问题和报错

2022年4月22日 22:08

解决json解析报错：Expecting value: line 1 column 1 (char 0)
一般的原因的返回的数据不是json格式的，可尝试普通网页的处理，使用response.content.decode()

selenium报错：StaleElementReferenceException
字面上的意思是，引用的元素已过时。原因是页面刷新了，此时当然找不到之前页面的元素，就算是后退回来的页面也是不一样的。有些牛马网页就喜欢搞这种，跳转回来了网页中的元素居然变了

使用selenium自动化的时候报错，错误提示：selenium.common.exceptions.WebDriverException: Message: invalid session id
通过对错误信息进行分析，无效的sessionid。后来通过对网上进行搜索查询，原因是在使用webdriver之前调用了driver.close()后将webdriver关闭了，则webdriver就失效了

ssl报错：利用requests方法中的get方法，将参数设置为verify=False

lxml.etree.XMLSyntaxError: StartTag: invalid element name, line 1, column 2
表示编码格式没搞好，在路径后面再加上即可parser=etree.HTMLParser(encoding='utf-8')

ssl握手超时，网站响应超时等：可能是网不好，多试几次

有的网站，虽然找到了资源的地址，但这个地址是经过加密的，以B站视频为例，其网址的前面还有一个blob：，这并不是一种新协议，而是html5中blob对象在赋给video标签后生成的一串标记，blob对象对象包含的数据，浏览器内部会解析

在使用xpath的时候，在路径结尾一般都要加/text()或者/@xxx
一般使用.get()和urlretrieve()方法配合使用以达到下载的目的，其中.get()方法能绕过检测

有些网页是静态网页，直接拿链接就可以获得所有你在这个网页中所看到的元素的数据
但有些动态网页（异步加载，ajax请求，实现页面局部刷新）里面的有些元素，是不是原本的链接里所包含的，而是二次加载进去的，这时，我们在页面元素中Ctrl+f搜索这一元素，会发现无法找到

ajax请求可能不会被归纳在Chrome的网络-XHR当中
因为有的数据包是通过前端生成的数据包，本身不是一个完整的json格式的数据，因此没有归纳在XHR

可以Ctrl+F在网页中查找定位元素
json数据是一种轻量化的数据格式，其数据结构的基础是字典，也就是说json数据由一个个字典组成
可以直接通过键来一步步找到特定的数据

response = resquests.get()
response.content表示在响应数据中将二进制数据提取出来，而.text则是提取文本文件
考虑到图片、视频都是二进制流存在的，所以可以以此写入文件，写入时直接用'wb'二进制写入
且不需要后面跟.decode方法
可以 with open('文件夹名\\'+文件名,mode = 'wb') as f: 也可以将文件写入指定文件夹

有些网站，若是其他一切正常，但获取的网页源码就是不对，可以将源码写入.html文件，然后使用浏览器打开，看看这到底是一个什么页面，然后再进行分析

有些网站使用了某些公司的反爬手段，如Cloudflare公司的强制等待5秒以判断是不是浏览器
具体情况具体分析，不同手段也有不同的破解手段

通过元素定位元素来源于哪个请求，可以在检查-网络里面直接搜索这个元素，会显示包含该元素的请求

有的网站直接使用.text方法打印出来的源码中存在乱码，这时就要先转化为二进制，再通过网站的编码方式进行解码

直接打印response可以得到状态码，以此分析问题出在哪

子元素的xpath后面跟/parent::*可以定位到该元素的父级，如

```
driver.find_element_by_xpath('//*[@id="sectitle0{}"]/parent::*'.format(str(i)))
```

有些网页（比如必应壁纸）会将整个网页的右键给屏蔽了，这样就不能直接打开检查了
但是可以通过按一下alt键，再按F12，就可以强制控制网页，然后呼出检查来，这样就可以继续找需要下载的数据的地址了，注意不是同时按alt和F12
某些网站不仅禁了右键，且一旦强制开启开发者模式就会自动关闭，我们可以在地址栏前加上：view-source： 即可从浏览器看到源代码
有的网站把右键ban了但是并没有把xpath做掉，这样就可以在源代码中定位元素
通过//*[@.....] 更快找到元素

除了tbody，还有tr、td等标签同样会使xpath路径出错，找不到所需要的资源，所以xpath路径中不能包含它们。加这种标签是常用的反爬手段

注意：在检查网页和网页源代码中得到的路径是不同的，xpath只能通过检查网页的结构来获取路径，也就是说源代码路径不可用

有的标签就只有一个名字，不要和后面的标签名看成一个标签了，比如：
target href='xxxx'，实际上xxxx的标签名就是href不包括target

'utf-8' codec can't decode byte：

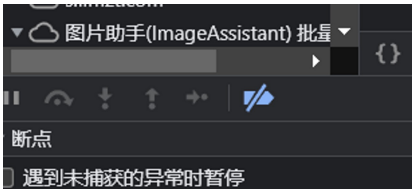
出现该问题一般是出现了无法进行转换的二进制数据所造成的
一般，只需要将decode的第二个参数，改为'ignore'即可（默认为'strict'）

```
content = response.content.decode('UTF-8','ignore')
```

a padding to disable MSIE and Chrome friendly error page：友好的错误界面

一般是爬虫出现该问题，问题来源于请求头所给的信息不够，因此禁止访问403。最常见的是加上refer标头
该问题是爬取today bing时发现的（2023.2.18），加上referer后可以继续正常爬取了

检查网页的时候，出现该栏，且代码中出现debuger段，导致网页无法刷新



先点击停用断点，再点击Paused in debugger后面的类似于播放的按钮，即可在调试时操作网页
出现这种反爬手段时，可能会出现消息轰炸，占用大量内存，注意内存占用。

若认为某网站采用Ajax技术，但网络的XHR中没有任何信息，则先把缓存停用，然后检查chrome开发者选项里“记录XMLhttprequest”的选项是否已经勾选
若XHR中实在没有信息，可能不是用的ajax技术更新网页，并且采用其他方式得到了请求数据的URL并进行请求

缓存用于缓存之前浏览的页面或者资源，如果禁用缓存，则网页非静态网页，则每加载一个资源都需要进行网络请求，便于抓包

refer标头：若网页无法正常访问时，优先添加该标头

网页服务器仅确认域名而不确认路径，例如：只要加入https://amlyu.com/能正常访问，那么https://amlyu.com/sddsf/dsf同样能正常访问，即使这个路径是不存在的

可使用eval将返回的字符串转变成字典

当字典中存在null等特殊符号时，Python里可能没有这个东西，就会报错。解决办法是提前定义null=None或''

要登录的网站进行爬取，一个非常有用的方法是在浏览器获取cookie后，作为请求标头的参数进行网页请求。

cookie过一段时间会发生变化，需要及时更新

项目代码

2022年10月12日 20:31

常用网站：https://m.shzx.org/a/162-4645-1.html（妹子，简单）

https://mmzztt.com/ （妹子，困难）

使用selenium浏览器爬取某文献网站（selenium库）

```
from selenium import webdriver
import warnings
import time

f = open('hx.txt','a',encoding='utf-8')
warnings.filterwarnings('ignore',category=DeprecationWarning)
path = 'chromedriver.exe'
driver = webdriver.Chrome(path)
# 爬取31-58页的数据
for x in range(40,59):
    url = 'https://www.sciencedirect.com/journal/journal-of-english-for-academic-purposes/vol/{}/suppl/C'.format(str(x))
    driver.get(url)
    xpath1 = driver.find_elements_by_xpath('//ol/li/dl/dt/h3/a')
    src_list = []
    # 获取每卷的文章链接并将链接写入列表
    for item in xpath1:
        src = item.get_attribute('href')
        src_list.append(src)
    # 打开每篇文章
    for src in src_list:
        driver.get(src)
        time.sleep(5.5)
        # 暴力搜索文章里是否有Author statement标题
        for i in range(80,150,5):
            xpath2 = driver.find_elements_by_id('sectitle0{:0>3}'.format(str(i)))
            for title in xpath2:
                if title.text == 'Author statement':
                    # 如果有Author statement，则获取其下面的文本
                    txt = driver.find_element_by_xpath('//*[@id="sectitle0{:0>3}"]/parent::*/p'.format(str(i))).text
                    time.sleep(1)
                    name = driver.find_element_by_xpath('//*[@id="screen-reader-main-title"]/span').text
                    f.write(name+'\n'+txt+'\n'+'\n')
                    print(name+'\n'+txt+'\n'+'\n')
        time.sleep(6)
f.close()
driver.quit()
```

使用requests库爬取某妹子网站图片（requests库）

```
import requests
import socket
from lxml import etree
import os

headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

```
Chrome/100.0.4896.75 Safari/537.36'}
socket.setdefaulttimeout(30)
urle = 'https://m.shzx.org/a/162-4645-'

while True:
    n = 1
    for y in range(40):
        url = urle+str(y)+'.html'
        response = requests.get(url=url,headers=headers)
        content = response.content.decode('utf-8')
        tree = etree.HTML(content)
        photoList = tree.xpath('/html/body/div[3]//img/@src')
        #文件夹名
        fileName = tree.xpath('/html/body/div[3]/h1/text()')[0]
        if os.path.exists('./xy/{}'.format(fileName)):
            pass
        else:
            os.mkdir('./xy/{}'.format(fileName))
        for i in range(len(photoList)):
            name = './xy/{}/{}.jpg'.format(fileName,n)
            response2 = requests.get(url=photoList[i],headers=headers,verify=False)
            content2 = response2.content
            try:
                with open(name,'wb') as f:
                    f.write(content2)
                print(fileName,'->',n,'->','下载完成')
                n += 1
            except:
                print('跳过这一张')
    urle = 'https://m.shzx.org/'+(tree.xpath('/html/body/div[5]/li[1]/a/@href')[0]).strip('0.html')
```

对某有着强力反爬机制的网站进行js逆向解密

```
from typing import List
import binascii
import json
import re
import requests
from lxml import etree
from Crypto.Cipher import AES
from Crypto.Hash import MD5

def decrypt(pid: str|int, cache_sign: str) -> List[str]:        # decrypt意为“解码、解读”
    pid = int(pid)
    IV = "".join([str(pid % i % 9) for i in range(2, 18)]).encode()
    key = MD5.new((f"{pid}6af0ce23e2f85cd971f58bdf61ed93a6").encode()).hexdigest()[8:24].encode()
    aes = AES.new(key, AES.MODE_CBC, IV)
    result = aes.decrypt(binascii.a2b_hex(cache_sign)).rstrip()

    result = re.findall(r'(\[.*\])', result.decode())[0]
    return json.loads(result)


def get_cache_sign(pid: str|int) -> str|None:
    url = "https://mmzztt.com/photo/{}".format(pid)
    res = requests.get(url, headers={
        "referer": "https://mmzztt.com/",
        "user-agent": "Mozilla/5.0"
    })
```

```
    if res.status_code == 200:
        html = etree.HTML(res.text)
        return html.xpath("//body/comment()")[0].__str__()[68:-3]

if __name__ == '__main__':
    pid = ''
    res = get_cache_sign(pid)

    resp = decrypt(pid, res)
    print(resp)
```