爬虫

基础：基本语法与数据结构；文件/数据库相关知识；函数/面向对象编程；异常处理；并发处理。

相关概念：http协议；get/post; cookie/useragent/proxy….

模块：urllib/requests/scrapy

re /bs4/xpath/css

快速构建爬虫

http详解：

使用get/post 方法，web浏览器/抓包工具 进行http协议的分析，页面信息定位；

urllib模块详解：

get方法行进抓取；post方法进行登录

正则表达式与BS4模块提取页面信息：

反扒与处理方式：

代理使用：

高并发代理爬虫实现：

Requests模块与文件上传：

http:超文本传输协议，客户端和服务器之间数据传输协议（基于TCP）

服务器默认端口：80

浏览器首先进行DNS域名的一个解析，得到一个IP,连接服务器形成一个通路，就可以数据传输，请求得到一个HTML

http请求：

请求行：

请求头部：

请求数据：

http协议

http协议方法

request头信息

response（应答）状态码与头信息

请求方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| GET \* | 请求资源 |
| POST \* | 提交表单或者文件，可能修改服务器资源（登录） |
| PUT | 更新资源 |
| DELETE | 删除指定资源 |
| OPTIONS | 获取服务器支持的HTTP请求方法；检查服务器性能 |

http请求header

|  |  |
| --- | --- |
| Header字段 | 说明 |
| Host | 请求的服务器的域名 |
| User-Agent | 用户请求信息：浏览器，操作系统等信息（标识请求者的信息） |
| Accept | 请求段接受数据类型 |
| Accept-Encoding | 指定接受数据的压缩编码类型 |
| Cookie | Cookie信息（保存客户端的状态） |
| Date | 日期 |
| Keep-Alive | 长连接 |

Http应答：状态行，消息报头，正文

状态码

|  |  |
| --- | --- |
| 状态码 | 说明 |
| 1XX | 请求已接受，继续处理 |
| 2XX | 请求成功 |
| 3XX | 重定向，继续处理 |
| 4XX | 客户端请求错误（403Forbidden/404 Not Found/400 Bad Request） |
| 5XX | 服务端错误 |

Http应答Headers

|  |  |
| --- | --- |
| Headers字段 | 说明 |
| Content-length | 数据长度 |
| Content-Type | 传输内容格式 |
| Content-Range | 断点续传范围，例如：200-400/400 |
| Set-Cookie | 客户端状态 |

urllib模块介绍

四个常用模块

|  |  |
| --- | --- |
| 模块 | 介绍 |
| urllib.request | 打开读取URL |
| urllib.error | 异常信息 |
| urllib.parse | 解析URL |
| urllib.robotparser | 解析robots.txt |

Urllib请求第一个页面

打开连接：from urllib import request

urlopen(url,data=None,timeout=xx,cafile=None,capath=None,cadefault=False,context=None)

参数详解

[url: 连接地址](url:连接地址)；

data: None,使用get方法，Not None,使用post方法；

timeout：超时设置；

示例：req=request.urlopen(‘http://www.baidu.com’)

应答信息

状态码：req.code, req.status

状态：req.reason

应答消息头：req.headers:获取headers对象

req.getheaders():获取所有headers信息

req.getheader(key):根据key获取对应头信息

（获取页面的编码格式：req.headers.get\_content\_charset() ）

读取页面信息

|  |  |
| --- | --- |
| 方法 |  |
| req.readable() | 是否可读 |
| req.read() | 读取所有内容 |
| req.readline() | 读取一行 |
| req.readlines() | 读取多行 |
| req.readinto(b) | 读取bytearray中 |
| req.close() | 关闭连接 |

例：con=req.read()

解码 con.decode(‘utf-8’)

读取完了之后，再次读取就会是一个空

5

动态构建URL

使用百度进行关键字搜索

1. 百度搜索api
2. 拼接url
3. 保存搜索结果

From urllib import request

url = ‘http://www.baidu.com/s?’+’wd=python’

s?表示搜索的意思，上述例子搜索的关键字为python.

然后保存到电脑上使用如下语句

f = open(**r'D:\test\python.html'**,**'w'**,encoding=**'utf-8'**)  
f.write(con.decode(**'utf-8'**))  
f.close()

当要搜索的关键字为汉字时，需要引入parse这个模块，使用urlencode函数，需要的是一个字典，默认的是utf-8形式

**from** urllib **import** request

**from** urllib **import** parse  
kv = {**'wd'**:**'金刚川'**}  
surl = parse.urlencode(kv) //  
url = **'http://www.baidu.com/s?'**+surl  
req = request.urlopen(url)  
con = req.read()  
con.decode(**'utf-8'**)  
f = open(**r'D:\python爬虫\金刚川.html'**,**'w'**,encoding=**'utf-8'**)  
f.write(con.decode(**'utf-8'**))  
f.close()

6

爬取豆瓣电影TOP250

爬取地址：<https://movie.douban.com/top250>

1,获取页面；

2,解析页面；电影名，海报；

3,获取下一页面；解析页面：正则表达式，字符串解析。

爬取02.py

7

8页面元素解析

bs4模块

安装：pip install beautifulsoup4

使用：import bs4

From bs4 import BeautifulSoup

支持：html,xml解析

创建BeautifulSoup对象：

obj = BeautifulSoup(html,’html5lib’)#指定解析库

浏览数据基本操作

|  |  |
| --- | --- |
| 操作 |  |
| obj.title | 获取title |
| obj.title.name | 获取title名字 |
| obj.title.string | 获取title内容 |
| obj.p | 第一个<p>tag |
| obj.p.attrs | tag属性 |
| obj.a | 第一个<a>tag |
| obj.get\_text() | 获取所有内容 |

from bs4 import BeautifulSoup

from urllib import request

Import re

url = **'http://www.baidu.com/s?'**  
req = request.urlopen(url)  
con = req.read().decode(**'utf-8'**)

obj= BeautifulSoup(con,’html5lib’)

9

查找操作

find\_all(name=None,attrs={},recursive=True,text=None,\*\*kwargs)

recursive=True 查找所有节点，为False的时候只查找当前对象下的直接子节点

返回匹配所有节点，（find返回匹配第一个节点）

|  |  |
| --- | --- |
| 操作 | 说明 |
| find\_all(‘div’) | 查找tag为div的所有节点(默认递归方式True) |
| find\_all([‘a’,’b’]) | 查找tag为a和b的所有节点 |
| find\_all(re.compile(“^b”)) | 参数为正则表达式  查找以b开头的所有节点 |
| find\_all(has\_id) | 参数为函数，查找含有id属性的所有节点  def has\_id(tag):  return tag.has\_attr(‘id’) |
| find\_all(id=’XX’) | 查找属性id为XX的所有节点 |
| find\_all(‘div’,calss\_=’XX’) | 查找属性calss为XX的div所有节点 |
| find\_all(calss\_=re.compile(“itl”)) | 返回所有calss属性包含itl的子节点 |
| find\_all(text=’123’) | 返回所有text为123的子节点 |
| find\_all(‘a’,text=’评价’) | 查找a中文字为‘评价’的节点 |
| find\_all(‘a’,text=re.complite(评价’)) | 查找a中文字包含‘评价’的节点 |

10

查找

方法：obj.select():通过css选择元素

|  |  |
| --- | --- |
| 语法 | 说明 |
| tag.select(‘a’) | 返回查找tag列表 |
| tag.select(‘div a’) | 按层级查找（查找div 下的 a节点） |
| tag.select(‘.css1’) | 通过calss值查找 |
| tag.select(“#id1”) | 通过id值查找 |
| tag.select(‘div[class]’) | 查找(div)存在某个(class)属性值的元素  例如<span class=”title”>…</span>  这里span就具有class属性 |
| tag.select(‘a[href=’XXXX’]’)  tag.select(a[href^=’http://img3’])  tag.select(a[href$=’http:title’]) | 通过属性值查找  匹配开头  匹配结尾 |

obj.select(‘img[src^=https://img3]’)

提取信息

tag相关属性与方法

|  |  |
| --- | --- |
| 属性与方法 | 说明 |
| tag.children | 子节点 |
| tag.attrs | 节点属性 |
| tag.text | 获取节点所有文本 |
| tag.string | 节点文本 |
| tag.strings | 节点所有文本 |
| tag.stripped\_strings | 节点所有不为空文本 |

11：爬取影视信息分析

信息：电影名，电影详细信息，豆瓣评分

存储：CSV文件，Mysql或者mongodb数据库；

1. 代码实现

解析内容：页面跳转

使用面向对象方式实现

代码健壮性

12

13.

14.

存储

CSV文件

15

16.构建请求头

直接urlopen打开连接，我们做了什么事情？请求信息？

如果服务器端拒绝非浏览器访问，如何处理？

"User-Agent": "Python-urllib/3.6",伪装成一个浏览器

浏览器请求页面做了什么事情？

查看请求头

http://httpbin.org/get

构建请求header

request.Request(url,data=None,headers={},origin\_req\_host=None,

unverifiable=False,method=None)

url:请求地址

Data:请求附加数据

Headers:请求头信息

Method:设置http请求方式

Data为None,请求：get, Data不为None,请求：post

添加请求header

头信息：

myhead={Host:www.baidu.com,User-Agent:”Mozilla/5.0(Windows……Firefox/54.0”}

构建头：

**hd=request.Request(‘http://www.baidu.com’,headers=myhead)**

**req=request.urlopen(hd)**

**req.read()**

更换user-agent

User-agent产生

安装fake\_usergent

pip install fake\_useragent

pip install -U fake-useragent

**from fake\_useragent import UserAgent**

**genua=UserAgent()**

genua.ie

genua.chrome

genua.firefox

genua.safari

随机产生一个

Genua.random

17

Post详解

Post方法：提交更新数据

**from urllib import request, parse**

Post应用场景：登录，提交表单，上传文件等：

使用：**request.urlopen(url, data=postdayta)**

Postdata为字节流(编译的时候，一定要进行编码，否则会报错)

第一个post请求：

测试网站：<http://httpbin.org/post>

第一个post:

Post内容：用户名密码

步骤：

1. post内容：
2. 数据编码
3. post消息
4. 查看应答消息；

18

登录github

1. 浏览器分析github登录行为
2. 浏览器登录github
3. 分析登录路径与提交post数据
4. 动态获取登录相关数据
5. 构建post数据
6. 验证是否登录成功

H/OeTMt0kPuDA78MN/Zgpo8ojn9T+IE9A5snUADU/Dwa1wN/1iewGskRHzsplVxUIjh+iPBP+sTBcH4Cas8H9w==

KXout0+aO78fHKHwsC7793lsgpvl17QgQ4jCXcyC0J1OBtu3vHkpRpAk6mkGjBGNRC7qmkZvhESjsj8nSfUzHQ==

mWl1zEEVo0nySkSPEWA++OdfrVRNq956B+KBo7T3kz1BGo6pYD9zPcuJhGNOG3fDdC6kQTRvPrcZxIKekAvEXQ==

19.登录失败分析

登录未成功

1. urllib交互消息，需要查看log
2. 根据log分析问题

问题解决方式：引入cookie处理

20，引入cookie成功登录

添加cookie处理

Opener默认对cookie不做处理，需要单独处理

引入cookie相关处理方式：

1. 创建cookier：

from http import cookiejar

ckobj = cookiejar.CookieJar()

1. 添加到opener中：

ckhandler = request.HTTPCookieProcessor(ckobj)

opener = build\_opener(ckhandler)

重新登陆

21：使用cookie直接登录

cookie存储与使用

MozillaCookieJar ,LWPCookieJar

保存：

ckobj = LWPCookieJar(‘cookie.txt’)

ckobj = save(ignore\_discard=True,ignore\_expires=True)

**ignore\_discard: cookie无效也存储**

**ignore\_expires:覆盖文件中原来cookie**

导入并使用cookie访问页面

ckobj = LWPCookieJar(‘cookie.txt’)

ckobj.load(‘m.txt’,ignore\_discard=True,ignore\_expires=True)

**反爬虫及反爬虫处理**

Robots协议

Robots：通过robots协议告诉搜索引擎那些页面可以抓取，那些页面不能抓取；

位置：根目录下，网址/robots.txt;

例如：

<https://www.baidu.com/robots.txt>

<https://www.jd.com/robots.txt>

在实际工作中，对一个网站进行爬取，要遵循Robots协议，这里有两种方法，

1. 就是在上述网站这样的形式，把文件下载下来，然后分析
2. 采用自带的库函数来解决

from urllib import robotparser

url = ‘https://www.douban.com/robots.txt’

robot = robotparser.RobotFileParser()

robot.set\_url(url)

robot.read()

robot.can\_fetch(‘\*’,’/’)

注：\*表示user-agent,表示任意一种，/表示根目录，，这里返回的是bool值吧

robots.txt语法

|  |  |
| --- | --- |
| 语法 | 说明 |
| User-agent | 搜索引擎种类 |
| Disallow | 禁止爬取 |
| Allow | 允许爬取 |
| Sitemap | 网站资源地图，告知搜索引擎可以爬取资源 |
| Crawl-delay | 爬取延时 |
| Disallow:// | 禁止爬取所有资源 |
| Disallow:/abc | 禁止爬取abc及其下所有资源 |
| Disallow:/abc/ | 禁止爬取abc下的所有资源，可以爬取abc |
| Disallow:/\*.png | 禁止爬取所有png图片 |
| Disallow:/\*?\* | 禁止爬取动态url |

反爬虫处理方式

1. useragent
2. 延时
3. 代理
4. 代理使用
5. 代理IP池构建
6. 并发访问（多进程多线程）

23.常见反扒机制与处理方式

拒绝非浏览器访问：检查请求header中的useragent,referer,cookie等信息；

解决办法：

1. 搜集user-agent并保存，爬取过程中更换user-agent;
2. Referer使用浏览器分析提取referer（跳转信息）
3. 登录过程中保留cookie（登录过程中将 cookie保留下来）;

url = ‘http://www.xicidaili.com/’

24.urllib代理使用

表现行为：同一时间IP短时间内多次访问网站

解决方式：代理

1. 爬取代理ip 与port
2. 验证代理有效性
3. 构建代理ip池
4. 使用代理爬取信息

看自己的IP

url = ‘**http://httpbin.org/ip**’

使用代理’;

**from urllib import request**

**ip\_proxy = {‘http’:’192.168.1.2:8080’} #代理ip**

**proxy = request.ProxyHandler(ip\_proxy) #代理handler**

**opener = request.build\_opener(proxy)**

**req =opener.open(‘http://httpbin.org/ip’)**

**print(req.read())** #查看IP地址

25.构建代理IP池分析

<https://www.fanqieip.com/>

<http://www.66ip.cn/>

验证代理IP的有效性，生成IP代理池

代码实现

提取<http://www.66ip.cn/>代理

From urllib import request

From bs4 import BeautifulSoup

Import re

from fake\_useragent import FakeUserAgent

ua = FakeUserAgent()

headinfo = {‘UserAgent’:ua.random}

url =’http://www’

reqhd = request.Request(url,headers=headinfo)

req = request.urlopen(reqhd)

con = req.read().decode(‘XXX’)

obj = BeautifulSoup(con,’html5lib’)

for x in obj.stripped\_strings:

print(x)

listip = [item for item in obj.stripped\_strings if re.match(r’\d’，item)]

//匹配以数字开头

26

Api提取

请求主页

地域

Page

代理提取

代理验证

28

代理有效性验证与高并发验证实现

29；爬虫项目设计

34 requests模块详解

官网：

<http://docs.python-requests.org/>

安装：

Pip install requests

GitHub地址：

<http://github.com/requests/requests>

get

url = <https://github.com/>

请求页面：

R = requests.get(url)

带参数请求：

Baseurl = <https://github.com/search?>

Params = {“utf8”:”√”,”q”:”python”,”type”:””}

Req = requests.get(url,params=params )