网络爬虫（又被称为网页蜘蛛，网络机器人，在FOAF社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。

根据使用场景，网络爬虫可分为 通用爬虫 和 聚焦爬虫 两种.

通用爬虫

通用网络爬虫是捜索引擎抓取系统（Baidu、Google、Yahoo等）的重要组成部分。主要目的是将互联网上的网页下载到本地，形成一个互联网内容的镜像备份。

通用搜索引擎（Search Engine）工作原理

通用网络爬虫，从互联网中搜集网页，采集信息，这些网页信息用于为搜索引擎建立索引从而提供支持，它决定着整个引擎系统的内容是否丰富，信息是否即时，因此其性能的优劣直接影响着搜索引擎的效果。

通用搜索引擎大多提供基于关键字的检索，难以支持根据语义信息提出的查询，无法准确理解用户的具体需求。

Robots协议（也叫爬虫协议、机器人协议等），全称是“网络爬虫排除标准”（Robots Exclusion Protocol），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



聚焦爬虫

聚焦爬虫，是"面向特定主题需求"的一种网络爬虫程序，它与通用搜索引擎爬虫的区别在于： 聚焦爬虫在实施网页抓取时会对内容进行处理筛选，尽量保证只抓取与需求相关的网页信息。

HTTP和HTTPS

HTTP协议（HyperText Transfer Protocol，超文本传输协议）：是一种发布和接收 HTML页面的方法。

HTTPS（Hypertext Transfer Protocol over Secure Socket Layer）简单讲是HTTP的安全版，在HTTP下加入SSL层。

SSL（Secure Sockets Layer 安全套接层）主要用于Web的安全传输协议，在传输层对网络连接进行加密，保障在Internet上数据传输的安全。

HTTP的端口号为80，HTTPS的端口号为443

URL（Uniform / Universal Resource Locator的缩写）：统一资源定位符，是用于完整地描述Internet上网页和其他资源的地址的一种标识方法。

基本格式：scheme://host[:port#]/path/…/[?query-string][#anchor]

scheme：协议(例如：http, https, ftp)

host：服务器的IP地址或者域名

port#：服务器的端口（如果是走协议默认端口，缺省端口80）

path：访问资源的路径

query-string：参数，发送给http服务器的数据

anchor：锚（跳转到网页的指定锚点位置）

urllib库

urlopen( )

网页抓取数据

from urllib import request  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 向指定的url发送请求，并返回服务器响应的类文件对象  
 response = request.urlopen("https://my.alipay.com/portal/i.htm")  
 html = response.read()  
 print(html.decode('utf-8'))

urlopen()返回的对象，可以使用read()进行读取

同样也可以使用geturl()方法、info()方法、getcode()方法。

geturl()返回的是一个url的字符串；

info()返回的是一些meta标记的元信息，包括一些服务器的信息；

getcode()返回的是HTTP的状态码，如果返回200表示请求成功。

向服务器发送数据

使用data参数，向服务器发送数据。GET用于信息获取，POST向服务器提交数据

如果没有设置urlopen()函数的data参数，HTTP请求采用GET方式，也就是从服务器获取信息，如果设置data参数，HTTP请求采用POST方式，也就是我们向服务器传递数据。

data参数有自己的格式，使用urllib.parse.urlencode()函数将字符串自动转换。

url可以是一个字符串或者Request对象，Request存在的意义是便于在请求的时候传入一些信息，如果请求中需要加入headers（请求头）等信息，我们就可以利用更强大的Request类来构建一个请求

# 使用urlencode方法转换标准格式  
data = parse.urlencode(Form\_Data).encode('utf-8')

urllib.error异常

urllib.error可以接收有urllib.request产生的异常。urllib.error有两个方法，URLError和HTTPError。

URLError是OSError的一个子类，HTTPError是URLError的一个子类，服务器上HTTP的响应会返回一个状态码，根据这个HTTP状态码，我们可以知道我们的访问是否成功。

if \_\_name\_\_ == '\_\_main\_\_':  
 # URLError  
 url='http://www.iloveyou.com/' #一个不存在的连接  
 # HTTPError  
 url = 'http://www.douyu.com/Jack.html' #请求的资源没有在服务器上找 到,www.douyu.com这个服务器是存在的，但是我们要查找的Jack.html资源是没有的  
 req=request.Request(url)  
 try:  
 response=request.urlopen(req)  
 # html=response.read().decode('utf-8')  
 # print(html)  
 # except error.URLError as e:  
 # print(e.reason)  
 except error.HTTPError as e:  
 print(e.code)

URLError和HTTPError混合使用

如果想用HTTPError和URLError一起捕获异常，那么需要将HTTPError放在URLError的前面，因为HTTPError是URLError的一个子类。如果URLError放在前面，出现HTTP异常会先响应URLError，这样HTTPError就捕获不到错误信息了。

也可以使用hasattr函数判断URLError含有的属性，如果含有reason属性表明是URLError，如果含有code属性表明是HTTPError

User Agent和代理IP隐藏身份

User Agent的中文名为用户代理，简称UA。User Agent的中文名为用户代理，简称UA。

常见的User Agent

1.Android

Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Safari/535.19

2.Firefox

Mozilla/5.0 (Windows NT 6.2; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0

3.Google Chrome

Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.94 Safari/537.36

4.iOS

Mozilla/5.0 (iPad; CPU OS 5\_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3

设置User Agent

1. request.Request()
2. 在创建Request对象的时候，填入headers参数(包含User Agent信息)，这个Headers参数要求为字典；
3. 在创建Request对象的时候不添加headers参数，在创建完成之后，使用add\_header()的方法，添加headers。

IP代理

代理ip选取：http://www.xicidaili.com/

使用代理步骤

1. 调用urlib.request.ProxyHandler()，proxies参数为一个字典。
2. 创建Opener：request.build\_opener（ ）
3. 安装Opener，install\_opener方法之后，会将程序默认的urlopen方法替换掉，如果不想替换掉，只是想临时使用一下，可以使用opener.open(url)，这样就不会对程序默认的urlopen有影响。

url='http://www.whatismyip.com.tw/'  
proxy={'http':'183.159.80.231:18118'} #设置代理ip  
proxy\_support=request.ProxyHandler(proxy) #创建ProxyHandler  
opener=request.build\_opener(proxy\_support) #创建opener  
opener.addheaders = [('User-Agent','Mozilla/5.0 (Windows NT 6.2; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0')]  
request.install\_opener(opener) #安装opener  
response=request.urlopen(url) #使用自己安装好的Opener  
html=response.read().decode('utf-8') #读取相应信息并解码  
print(html)

Scrapy

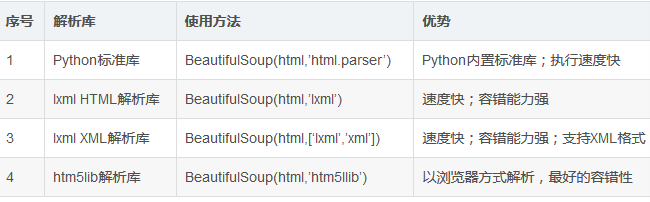
使用Cookie和使用代理IP一样，也需要创建一个自己的opener。在HTTP包中，提供了cookiejar模块，用于提供对Cookie的支持。

可以利用本模块的CookieJar类的对象来捕获cookie并在后续连接请求时重新发送，比如可以实现模拟登录功能。该模块主要的对象有CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

CookieJar–派生–>FileCookieJar–派生–>MozillaCookieJar和LWPCookieJar

创建一个带有cookie的opener，在访问登录的URL时，将登录后的cookie保存下来，然后利用这个cookie来访问其他网址。查看登录之后才能看到的信息。

创建Beautiful Soup对象



Beautiful Soup四大对象种类

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种:

Tag：soup.标签名

Tag有两个重要的属性：name和attrs。

soup.标签名.name/attrs

name 即为 [document]，对于其他内部标签，输出的值便为标签本身的名称。

attrs 标签的所有属性打印输出，得到的类型是一个字典。

如果需要单独获得某个属性可以使用字典的打印方法或者get(‘key’)

NavigableString：获取标签内部的文字 soup.标签名.string

BeautifulSoup：表示的是一个文档的全部内容

Comment：特殊类型的NavigableString对象，首先判断的类型是否为 Comment 类型，然后再进行其他操作，如打印输出

if type(soup.li.string) == element.Comment:

print(soup.li.string)

遍历文档树

直接子节点(不包含孙节点)

tag的content属性可以将tag的子节点以列表的方式输出：soup.标签名.contents

可以用列表索引来获取它的某一个元素: soup.标签名.contents[1]

children是一个 list 生成器对象,返回的不是一个 list, 可以通过遍历获取所有子节点

for child in soup. 标签名.children:

print(child)

搜索文档树: find\_all(name, attrs, recursive, text, limit, \*\*kwargs)：

find\_all() 方法搜索当前tag的所有tag子节点,并判断是否符合过滤器的条件

name参数：查找所有名字为name的tag,字符串对象会被自动忽略掉

传递字符：搜索方法中传入一个字符串参数，Beautiful Soup会查找与字符串完整匹配的内容

传递正则表达式：通过正则表达式的 match() 来匹配内容

传递列表：将与列表中任一元素匹配的内容返回

传递True：可以匹配任何值，查找到所有的tag

attrs参数：定义一个字典参数来搜索包含特殊属性的tag

print(soup.find\_all(attrs={"class":"title"}))

recursive参数：调用tag的 find\_all() 方法时,Beautiful Soup会检索当前tag的所有子孙节点,如果只想搜索tag的直接子节点,可以使用参数 recursive=False。

text参数：搜索文档中的字符串，接受字符串，正则表达式 ，列表，True

limit参数：find\_all() 方法返回全部的搜索结构，使用 limit 参数限制返回结果的数量

kwargs参数：如果传入 class 参数,Beautiful Soup 会搜索每个 class 属性为 title 的 tag 。kwargs 接收字符串，正则表达式

Selenium

#模拟提交搜索  
browser = webdriver.Firefox()  
#打开请求的URL，webdriver会等到页面全部加载完毕后返回  
browser.get('http://www.baidu.com/')  
assert '百度一下，你就知道' in browser.title  
#find\_element\_by\_\*方法寻找网页元素  
elem=browser.find\_element\_by\_name('wd')  
#模拟键盘输入  
elem.send\_keys('python')  
#模拟回车  
elem.send\_keys(Keys.RETURN)  
#打印提交后页面信息  
print(browser.page\_source)

元素选取

单个元素选取（多个元素选取，将element变为elements）

find\_element\_by\_id

find\_element\_by\_name

find\_element\_by\_xpath

find\_element\_by\_link\_text

find\_element\_by\_partial\_link\_text

find\_element\_by\_tag\_name

find\_element\_by\_class\_name

find\_element\_by\_css\_selector

利用 By 类来确定哪种选择方式

from selenium.webdriver.common.by import By

find\_element(By.XPATH, '//button[text()="Some text"]')

界面交互

通过元素选取，能够找到元素的位置，根据这个元素的位置进行相应的事件操作，例如输入文本框内容、鼠标单击、填充表单、元素拖拽等等

elem = driver.find\_element\_by\_xpath("//a[@data-fun='next']")

elem.click()

点击这个按键之前，窗口最好移动到那里，因为如果这个按键被其他元素遮挡，click()就触发异常。

将窗口滑动到page这个位置

page = driver.find\_elements\_by\_xpath("//div[@class='page']")

driver.execute\_script('arguments[0].scrollIntoView();', page[-1]) #拖动到可见的元素去

Xpath（XML Path）：元素查找方式，几乎可以定位到页面上的任意元素

xpath路径以//开头时，则表示让xpath引擎从文档的任意符合的元素节点开始进行解析，/出现在xpath路径中时，则表示寻找父节点的直接子节点，当//出现在xpath路径中时，表示寻找父节点下任意符合条件的子节点，用\*号省略具体的标签名称，但元素的层级关系必须体现出来

构造合理http请求头