### 目标

利用 Selenium 抓取科学网生命科学分类下的论文以及新闻部分，并用正则表达式通过解析索引页文章标题以及链接，得到文章的具体内容信息，并保存。

### 分析

通过requests模拟post请求的话比较繁琐，难以分析参数，所以选择使用python的selenium模块以及Phantomjs模拟浏览器请求，获取真正的相应页面。

#### 索引页分析

捉取入口链接为：

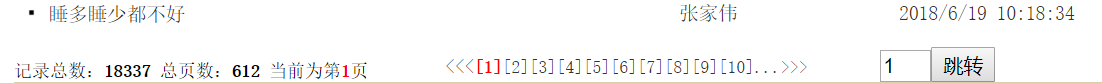
# 论文链接

#url = 'http://paper.sciencenet.cn/paper/fieldlist.aspx?id=2'

# 领域新闻链接

url = 'http://news.sciencenet.cn/fieldlist.aspx?id=3'

  
对于该页面，不管你点击第几页，它的url都是不变的。如果想要分页的话，不过在页面下方有一个分页导航，包括前 10页的链接，也包括下一页的链接，同时还有一个输入任意页码跳转的链接，如下图：



可以看到，一共有612页，我们要获取每一页的内容，只需要将页码从 1 到612页顺次遍历即可，页码数是确定的。所以在这里我们可以直接在页面跳转文本框中输入要跳转的页码，然后点击确定按钮跳转即可到达页码对应的页面。  
当我们成功加载出某一页时，利用 Selenium 即可获取页面源代码，然后我们再用lxml解析库解析,获取文章的详细链接。

#### 索引页获取

http://news.sciencenet.cn/fieldlist.aspx?id=3

用 Selenium 进行抓取上面url，通过以下方法抓取索引页：

# -\*- coding:utf-8 -\*-

from selenium import webdriver

from selenium.common.exceptions import TimeoutException, NoSuchElementException

from selenium.webdriver.common.by import By

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.wait import WebDriverWait

from lxml import etree

import json

import time

import requests

from requests.exceptions import ConnectionError

import re

#import os

# 构造一个 WebDriver 对象，调用phantomjs

browser = webdriver.PhantomJS(executable\_path=r'C:/Users/CRAB/Desktop/mybooks/execute/phantomjs-2.1.1-windows/bin/phantomjs.exe')

# 设置等待时长

wait = WebDriverWait(browser, 10)

def index\_page(page, judge):

"""

抓取索引页

:param page: 页码

:param judge: 用于判断上次爬取位置

"""

print('正在爬取第', page, '页索引页')

try:

#url\_list = ['http://paper.sciencenet.cn/paper/fieldlist.aspx?id=2','http://news.sciencenet.cn/fieldlist.aspx?id=3']

#for url in url\_list:

# 论文链接

url = 'http://paper.sciencenet.cn/paper/fieldlist.aspx?id=2'

# 领域新闻链接

# url = 'http://news.sciencenet.cn/fieldlist.aspx?id=3'

browser.get(url)

# 判断页码

if page > 1:

input = wait.until(

# 定位页码输入位置

EC.presence\_of\_element\_located((By.NAME, 'AspNetPager1\_input')))

submit = wait.until(

# 定位页码跳转框

EC.element\_to\_be\_clickable((By.NAME, 'AspNetPager1')))

# 清楚页码输入框数据

input.clear()

# 填入page参数

input.send\_keys(page)

# 点击跳转

submit.click()

# 判断当前高亮页是否为传递过去的参数page

wait.until(EC.text\_to\_be\_present\_in\_element\_value((By.NAME, 'AspNetPager1\_input'), str(page)))

# 判断是否爬取到上次爬取位置，是的话返回1

return get\_pages(page, judge)

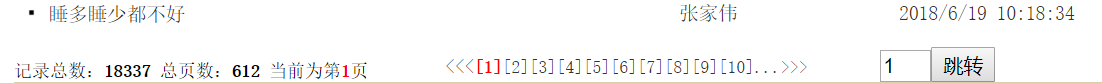
except TimeoutException:

print('爬取第', page, '页索引页time out,尝试重新爬取!')

# 如果timeout，尝试重新获取页面

index\_page(page, judge)

通过构造一个 WebDriver 对象，调用无界浏览器 phantomjs，然后定义一个 get\_index() 方法，用于抓取索引页。  
在该方法里我们首先访问了这个链接，然后判断了当前的页码，如果大于 1 ，那就进行跳页操作，然后等待页面跳转完成。  
等待加载指定为最长 10 秒。如果在这个时间内成功匹配了等待条件，那就立即返回相应结果并继续向下执行，否则到了最大等待时间还没有加载出来就直接抛出超时异常。  
判断是否加载以及跳页完成，可以通过判断当前高亮的页码数是当前的页码数即可，如下：



所以在这里使用等待条件:

text\_to\_be\_present\_in\_element\_value((By.NAME, 'AspNetPager1\_input'), str(page))

通过判断高亮页码节点对应的页码数是否与我们传递过去的参数相等，如果是，那就证明页面成功跳转到了这一页，页面跳转成功。  
然后通过 get\_pages() 方法进行页面解析。

#### 解析索引页文章url以及保存文章

接下来获取页面源代码，然后用 lxml 进行解析。  
通过调用 page\_source 属性获取了页码的源代码，然后构造 lxml 解析对象，通过

items = result.xpath('//\*[@id="DataGrid1"]/tbody//tbody')

获取具体文章的url所在的list,用 for 循环提取每个文章列表节点内部的url，通过组合得到每个文章的真正url。并提取url中的数字，作为保存的数据文件名。通过requests请求得到回应内容，然后利用正则表达式提取文章内容。

html = browser.page\_source

result = etree.HTML(html)

# 获取索引页文章列表内容

items = result.xpath('//\*[@id="DataGrid1"]/tbody//tbody')

# 提取每一个文章的url

for item in items:

# 领域新闻为 @width = "60%" ，

# kw = item.xpath('.//td[@width = "60%"]/a/@href')[0]

# 论文为 @width = "70%"

kw = item.xpath('.//td[@width = "70%"]/a/@href')[0]

# 组合url

url = 'http://paper.sciencenet.cn' + kw

#/htmlpaper/201861510484322846535.shtm

# 提取url中的数字作为文件名保存

filename\_pattern = re.compile(r'[a-zA-Z:\.\/\-\\_]')

filename = filename\_pattern.sub('', kw)

headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_11\_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.116 Safari/537.36'

}

response = requests.get(url, headers = headers)

response.encoding = 'utf-8'

time.sleep(2)

# 提取文章中的内容

detail\_pattern = re.compile(r'<table id="content".\*<!-- JiaThis Button END -->.\*?</table>',re.S)

detail\_search = detail\_pattern.search(response.text)

detail\_result = detail\_search.group()

#### 获取文章中的图片

利用正则表达式匹配图片url链接，然后通过

img\_sour\_pattern = re.compile(r'http')

判断url是否需要组合，获取图片的真正链接，然后调用requests请求保存图片到文章保存位置的下级文件夹img。

# 匹配图片url链接

img\_pattern = re.compile(r'<img(.\*?)\ssrc="(.\*?)"', re.S)

# 获取文章中所有的图片url链接

img\_findall = img\_pattern.findall(detail\_result)

for detail in img\_findall:

try:

# 判断提取的url是否需要组合

img\_sour\_pattern = re.compile(r'http')

img\_judge = img\_sour\_pattern.search(detail[1])

if img\_judge is None:

img\_url = 'http://news.sciencenet.cn' + detail[1]

else:

img\_url = detail[1]

# 获取图片

img\_response = requests.get(img\_url, headers = headers).content

img\_save\_name = filename\_pattern.sub('', detail[1])

# 保存图片

save\_img(img\_response, img\_save\_name)

except ConnectionError:

print('图片网址有误:' + url)

def save\_img(result, filename):

"""

保存文章中的图片

:param result: 图片文件

:param filename: 保存的图片名

"""

img\_save\_full\_name = './sciencenet\_spider/sciencenet\_spider\_result/img/' + filename + '.jpg'

with open(img\_save\_full\_name, 'wb') as f:

f.write(result)

然后通过正则表达式，把文章中图片的链接，替换为我们本地图片的链接，利用re.sub的repl参数，如下：

def img\_url\_name(match):

"""

匹配文章内容中的图片url，替换为本地url

"""

img\_url\_pattern = re.compile(r'[a-zA-Z:/\.\-\\_]')

img\_url\_detail = img\_url\_pattern.sub('', match.group(2))

img\_url\_detail\_add = img\_url\_pattern.search(match.group(1))

img\_name = '<img src="./img/' + img\_url\_detail + '.jpg"'

return img\_name

# 匹配文章内容中的图片url，替换为本地图片url

real\_result = img\_pattern.sub(img\_url\_name, detail\_result)

# 保存文章内容

save\_page(real\_result, filename)

然后得到修改过的文章源码

real\_result = img\_pattern.sub(img\_url\_name, detail\_result)

通过save\_page方法保存内容。

def save\_page(result, filename):

"""

保存到文件

:param result: 结果

:param filename: 保存的文件名

"""

full\_filename = './sciencenet\_spider/sciencenet\_spider\_result/' + filename + '.html'

with open(full\_filename, 'w', encoding = 'utf-8') as f:

f.write(result.group())

#print('文件' + full\_filename + '保存成功')

#### 判断上次爬取位置

判断爬取的文章是否已进行过爬取，通过引入一个值进行对比，每次脚本进行爬取时，都会把当前爬取的第一个文章的url写入文件next\_judge。

if page == 1:

# 领域新闻为 @width = "60%" ，

# next\_judge = items[0].xpath('.//td[@width = "60%"]/a/@href')[0]

# 论文为 @width = "70%"

next\_judge = items[0].xpath('.//td[@width = "70%"]/a/@href')[0]

with open('sciencenet\_spider/judge.txt', 'w', encoding = 'utf-8') as f:

print("next\_judge:\t" + next\_judge)

f.write(next\_judge)

然后在下一次脚本爬取时，先读取这个文件，获得一个字符串，把这个字符串与每次获取的文章url进行对比，如果相等，就break，退出循环，结束爬取。  
首先在爬取开始时读取上次爬取保存的字符串：

with open('sciencenet\_spider/judge.text', 'r', encoding = 'utf-8') as f:

judge = f.read()

for i in range(1, 4):

params = index\_page(i, judge)

if params == 1:

break

通过传递参数judge,在get\_pages方法中进行判断，如果相等，就退出循环，返回1,爬取脚本判断params == 1,脚本退出。

if kw == judge:

print("爬取到上次爬取位置，结束爬取！")

return 1

break