第16课--青山不改，绿水长流

课程目标

- 爬虫的流程
- Network的使用
- BeautifulSoup解析数据
- json形式的数据解析
- 带参数的请求
- 数据存储
- 协程的使用
- scrapy的使用
- selenium的使用
- 邮件发送
- 定时模块

课程难点

- 爬虫数据解析
- 协程的使用
- scrapy框架的使用

课程内容重现

1、requests使用

1.发起请求

```
import requests
res = requests.get(url)
```

2.使用selenium

```
from selenium import webdriver
driver = webdriver.Chrome()
url = 'www.xiaoke.kaikeba.com'
driver.get(url)
driver.close()
```

2、requests请求后的数据解析

1.解析为文本字符串

```
import requests
url = ''
res = requests.get(url)
html = res.text
print(html)
```

2.解析为bytes数据

```
import requests
url = ''
res = requests.get(url)
html = res.content
print(html)

```

3.文本数据提取信息的方式

利用BeautifulSoup将返回的字符串解析为bs对象

```
import requests
url = ''
res = requests.get(url)
bs = BeautifulSoup(res.text, 'html.parser')
```

使用bs对象里面的方法进行数据的提取,find返回的是一个tag对象，find_all返回的是一个列表，元素全部为tag对象

```
res1 = bs.find('div',class_='pl2')
res2 = bs.find('a')
res3 = bs.find_all('div',class_='pl2')
```

tag对象提取详细信息

```
text = res1.text  #获取文本字符串
href = res2['href'] #获取a标签内的href属性
```

如果是列表的则需要遍历在提取详细信息
ps1:提取属性值必须是定位到当前标签的Tag对象才行（比如：提取a标签的href属性，必须定位到a标签）
ps2:为了与定义类的class做区分，在使用class定位的时候要加个下划线（class_）

3、selenium请求后的数据解析

1、使用方法

1.根据id定位

```
driver.find_element_by_id('teacher')
```

2.根据class定位

```
driver.find_element_by_class_name('teacher')
```

3.根据标签名

```
driver.find_element_by_tag_name('button')
```

4.根据属性name定位

```
driver.find_element_by_name('assist')
```

5.定位到input标签的话可以用send_keys填入内容

```
teacher = driver.find_element_by_id('teacher')
teacher.send_keys('开课吧')
```

6.定位到点击按钮可以用click模拟点击

```
button = driver.find_element_by_tag_name('button')
button.click()
```

7.获取网页源代码

```
driver.page_source
```

2、提取元素

1.使用 text获取文本内容

```
label = driver.find_element_by_tag_name('label')
res = label.text
```

2.使用get_attribute获取属性值

```
label = driver.find_element_by_class_name('form-teacher')
print(label.get_attribute('type')) # 获取type这个属性的值
```

3、更厉害的请求（带参数）

1.requests.get里面的url携带参数

随着学习的深入，我们知道请求可以有多个参数。

params，可以让我们带着参数来请求数据，例如我们想要跳转到第几页、每次请求多少个数据等等。

headers，请求头 一般携带user-agent 用来模拟浏览器发起请求

```
import requests
params = {
  '':''
}
headers={
  'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36'
}
res = requests.get(url,params=params,headers=headers)

```

2.有时候还可以使用post请求，与get方式的区别是get是明文显示参数，post是非明文显示参数

在post请求里，又多了两个参数：data和cookies。我们使用来data传递参数，其用法和params非常相像

```
import requests
data = {
  '':''
}
headers={
 'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36'
}
res = requests.post(url,data=data,headers=headers)
```

利用cookie参数

```
import requests
url_1 = 'https://…'
headers = {'user-agent':''}
data = {}
login_in = requests.post(url,headers=headers,data=data)
cookies = login_in.cookies
# 完成登录，获取cookies
url_2 = 'https://…'
params = {}
# 定义url和params
response = requests.get(url,headers=headers,params=params,cookies=cookies)
```

4、数据存储

1.利用csv读写

写入csv文件

```
import csv
# 需要写入的数据
score1 = ['math', 95]
score2 = ['english', 90]
# 打开文件，追加a，newline=""，可以删掉行与行之间的空格
file= open("score.csv", "a", newline="")
# 设定写入模式
csv_write = csv.writer(file)
# 写入具体内容
csv_write.writerow(score1)
csv_write.writerow(score2)
```

读取csv文件

```
import csv
# 打开csv文件
```

```
file = open("score.csv")
# 读取文件内容，构造csv.reader对象
reader = csv.reader(file)
# 打印reader中的内容
for item in reader:
    print(item)
```

2.利用openpyxl

写入excel

```
import openpyxl
# 引用openpyxl
wb = openpyxl.Workbook()
# 利用openpyxl.Workbook()函数创建新的workbook（工作薄）对象，就是创建新的空的Excel文件。
sheet = wb.active
# wb.active就是获取这个工作薄的活动表，通常就是第一个工作簿，也就是我们在上面的图片中看到的sheet1。
sheet.title = 'kaikeba'
# 可以用.title给工作表重命名。现在第一个工作表的名称就会由原来默认的“sheet1”改为"kaikeba"。
sheet['A1'] = 'kaikeba'
# 向单个单元格写入数据
score1 = ['math', 95]
sheet.append(score1)
# 写入整行的数据，变量类型是一个列表
wb.save('score.xlsx')
# 保存修改的Excel
wb.close()
# 关闭Excel
```

读取excel

```
import openpyxl
wb = openpyxl.load_workbook('score.xlsx')
# 打开的指定的工作簿
sheet = wb['kaikeba']
# 指定读取的工作表的名称
A1_value = sheet['A1'].value
# 打印值
print(A1_value)

```

5、协程的使用

```
from gevent import monkey
```

```
#从gevent库里导入monkey模块
import gevent
import time
import requests
from gevent.queue import Queue
#从gevent库里导入queue模块
monkey.patch_all()
#monkey.patch_all()能把程序变成协作式运行，就是可以帮助程序实现异步。
start = time.time()
url_list = ['https://www.baidu.com/',
'https://www.sina.com.cn/',
'http://www.sohu.com/',
'https://www.qq.com/',
'https://www.163.com/',
'http://www.iqiyi.com/',
'https://www.tmall.com/',
'http://www.ifeng.com/']
work = Queue()
#创建队列对象，并赋值给work
for url in url_list:
#遍历url_list
    work.put_nowait(url)
    #用put_nowait()函数可以把网址都放进队列里
def crawler():
    while not work.empty():
    #当队列不是空的时候，就执行下面的程序
        url = work.get_nowait()
        #用get_nowait()函数可以把队列里的网址都取出
        r = requests.get(url)
        #用requests.get()函数抓取网址
        print(url,work.qsize(),r.status_code)
        #打印网址、队列长度、抓取请求的状态码
tasks_list  = [ ]
#创建空的任务列表
for x in range(2):
#相当于创建了2个爬虫
    task = gevent.spawn(crawler)
    #用gevent.spawn()函数创建执行crawler()函数的任务
    tasks_list.append(task)
    #往任务列表添加任务。
gevent.joinall(tasks_list)
#用gevent.joinall方法，执行任务列表里的所有任务，就是让爬虫开始爬取网站
end = time.time()
print(end-start)
```

6、框架爬虫（Scrapy）

1.创建Scrapy项目

scrapy startproject douban，douban就是我们项目的名字

Scrapy项目里每个文件都有它对应的具体功能。例如settings.py 是scrapy里的各种设置、items.py是用来定义数据的、pipelines.py是用来处理数据的，它们对应的就是Scrapy的结构中的Item Pipeline（数据管道）。

spiders是放置爬虫的目录。我们可以在spiders这个文件夹里创建爬虫文件。我们来把这个文件，命名为Book_douban_Top250。后面的大部分代码都需要在这个Book_douban_Top250.py文件里编写。

先在Book_douban_Top250.py文件里导入我们需要的模块

```
import scrapy
import bs4
```

导入BeautifulSoup用于解析和提取数据;导入scrapy是待会我们要用创建类的方式写这个爬虫，我们所创建的类将直接继承scrapy中的scrapy.Spider类。这样，有许多好用属性和方法，就能够直接使用。

2.编辑爬虫

爬虫代码

```
import scrapy
import bs4
from ..items import BookDoubanItem
#定义一个类DoubanSpider，它继承自scrapy.Spider
class DoubanSpider(scrapy.Spider):
    #定义爬虫的名字为book_douban。
    name = 'book_douban'
    #定义爬虫爬取网址的域名。
    allowed_domains = ['book.douban.com']
    #定义爬虫爬取网址的域名。
    start_urls = ['https://book.douban.com/top250?start=0']
    #定义起始网址。
    for x in range(3):
        url = 'https://book.douban.com/top250?start={num}'.format(x * 25)
        #添加新的url到sart_urls中
        start_urls.append(url)
    # parse是默认处理response的方法
    def parse(self, response):
        # 用BeautifulSoup解析response
        bs = bs4.BeautifulSoup(response.text, 'html.parser')
        #用find_all提取<tr class="item">元素，这个元素里含有书籍信息
        datas = bs.find_all('tr', class_="item")
        # 遍历data
        for data in datas:
            # 实例化DoubanItem这个类
            item = BookDoubanItem()
            # 提取出书名，并把这个数据放回DoubanItem类的title属性里
item['title'] = data.find_all('a')[1]['title']
```

```
            # 提取出出版信息，并把这个数据放回DoubanItem类的publish里
            item['publish'] = data.find('p', class_='pl').text
            # 提取出评分，并把这个数据放回DoubanItem类的score属性里。
            item['score'] = data.find('span', class_='rating_nums').text
            # 打印书名
            print(item['title'])
            # yield item是把获得的item传递给引擎
            yield item
```

3.设置

Scrapy里的默认设置没被修改。比如我们需要修改请求头（User-Agent需要设置）。点击settings.py文件，你能在里面找到如下的默认设置代码：

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'class13 (+http://www.yourdomain.com)'
# Obey robots.txt rules
ROBOTSTXT_OBEY = True
```

把USER _AGENT的注释取消（删除#），然后替换掉user-agent的内容，就是修改了请求头。

因为Scrapy是遵守robots协议的，如果是robots协议禁止爬取的内容，Scrapy也会默认不去爬取，所以我们还得修改Scrapy中的默认设置。

把ROBOTSTXT_OBEY=True改成ROBOTSTXT_OBEY=False，就是把遵守robots协议换成无需遵从robots协议，这样Scrapy就能不受限制地运行。

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
USER_AGENT = 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36'
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

4.运行

想要运行Scrapy有两种方法：

一种是在本地电脑的终端跳转到scrapy项目的文件夹（跳转方法：cd+文件夹的路径名），然后输入命令行：scrapy crawl book_douban（book_douban就是我们爬虫的名字）

另一种运行方式需要我们在最外层的大文件夹里新建一个main.py文件（与scrapy.cfg同级）。在这个main.py文件里，输入以下代码，点击运行，Scrapy的程序就会启动。

```
from scrapy import cmdline
#导入cmdline模块,可以实现控制终端命令行
cmdline.execute(['scrapy','crawl','book_douban'])
#用execute（）方法，输入运行scrapy的命令
```

7、邮件发送

```
import smtplib
from email.mime.text import MIMEText
from email.header import Header
#引入smtplib、MIMETex和Header
mailhost='smtp.qq.com'
#把qq邮箱的服务器地址赋值到变量mailhost上，地址应为字符串格式
qqmail = smtplib.SMTP()
#实例化一个smtplib模块里的SMTP类的对象，这样就可以调用SMTP对象的方法和属性了
qqmail.connect(mailhost,25)
#连接服务器，第一个参数是服务器地址，第二个参数是SMTP端口号。
#以上，皆为连接服务器。
sender = input('请输入你的邮箱：')
#获取邮箱账号，为字符串格式
password = input('请输入你的密码：')
#获取邮箱密码，为字符串格式
qqmail.login(sender,password)
#登录邮箱，第一个参数为邮箱账号，第二个参数为邮箱密码
#以上，皆为登录邮箱。
receiver=input('请输入收件人的邮箱：')
#获取收件人的邮箱。
content=input('请输入邮件正文：')
#输入你的邮件正文，为字符串格式
message = MIMEText(content, 'plain', 'utf-8')
#实例化一个MIMEText邮件对象，该对象需要写进三个参数，分别是邮件正文，文本格式和编码
subject = input('请输入你的邮件主题：')
#输入你的邮件主题，为字符串格式
message['Subject'] = Header(subject, 'utf-8')
#在等号的右边是实例化了一个Header邮件头对象，该对象需要写入两个参数，分别是邮件主题和编码，然后赋值给等号左边的变量message['Subject']。
#以上，为填写主题和正文。
try:
    qqmail.sendmail(sender, receiver, message.as_string())
    print ('邮件发送成功')
except:
    print ('邮件发送失败')
qqmail.quit()
#以上为发送邮件和退出邮箱
```

8、定时模块

```
import schedule
import time
#引入schedule和time
def job():
    print("Working in progress...")
    #定义一个叫job的函数，函数的功能是打印'Working in progress...'
#部署情况
schedule.every(10).minutes.do(job) #部署每10分钟执行一次job()函数的任务
```

```
schedule.every().hour.do(job) #部署每x小时执行一次job()函数的任务
schedule.every().day.at("10:30").do(job) #部署在每天的10:30执行job()函数的任务
schedule.every().monday.do(job) #部署每个星期一执行job()函数的任务
schedule.every().wednesday.at("13:15").do(job)#部署每周三的13：15执行函数的任务
while True:
    schedule.run_pending()
    time.sleep(1)
    #检查部署的情况，如果任务准备就绪，就开始执行任务
```

9、爬虫进阶路线指引

1. 解析与提取

除了我们已学过的提取数据的方式，还有一些其他的方式可以用来提取数据

xpath/lxml/正则

2. 存储

存储数据的方式也不仅仅限制在csv和excel当中，我们还可以使用MySQL和MongoDB等强大的数据库来存储数据

3. 数据分析与可视化

为了对获取到的数据进行分析来实现数据的价值，我们还会用到数据分析相关的模块，那么基础的数据分析模块有 numpy pandas matplotlib等

4. 更多的爬虫

我们已经知道，协程在本质上只用到CPU的一个核。而多进程（multiprocessing库）爬虫允许你使用CPU的多个核，所以你可以使用多进程，或者是多进程与多协程结合的方式进一步优化爬虫。理论上来说，只要CPU允许，开多少个进程，就能让你的爬虫速度提高多少倍。但这种方式会因受到CPU核数的限制，要实现这个突破，那我们就要用到分布式爬虫了：即用多个设备，去跑同一个项目

5. 更强大的爬虫——框架

爬虫框架也不仅仅只有scrapy，还有PySpider等也是一些优秀的爬虫框架

6. 反爬虫应对策略汇总

- 我们可以填写user-agent声明自己的身份，以此来应对网站对请求头的限制，即Request Headers，有时还要去填写origin和referer声明请求的来源。
- 我们可以用cookies和session的知识去模拟登录，来应对网站对登录的限制。

- 有的网站会做一些复杂的交互，比如设置“验证码”。解决方案也各有不同，例如：我们用Selenium去手动输入验证码；我们用一些图像处理的库自动识别验证码（tesserocr/pytesserart/pillow）。
- 有的网站则会限制IP。使用搜索引擎搜索“IP”，你也能看到自己的IP地址。解决方案使用time.sleep()来对爬虫的速度进行限制；建立IP代理池（你可以在网络上搜索可用的IP代理），一个IP不能用了就换一个用