爬虫入门教程

404 student

零、写在前面

爬虫是个好东西，它可以用来搜集数据、快速操作、重复操作等等。简单来说，爬虫相当于一只在浏览器上面爬的虫子，把本来应该人手干的事情全干完了。

最简单的爬虫是**数据获取型爬虫**，不需要做什么操作，只用获取网页中的内容。复杂一点的就需要爬虫来做点操作，比如登录啥的。

本教程源自于大一下《程序设计实习》课程笔记，我觉得爬虫还是有点意思，想着复习一下，去爬一爬别的网站。大家千万不要爬我的网站啊（哭）

前置要求：有python即可。

一、基本思路

首先我们抛开所有算法，只谈思路。

第一步是找出我们需要的URL（网址），第二步是访问到URL对应的HTML源代码。找到源代码以后，第三步是找到我们需要的元素遵循什么样的格式，然后用正则表达式或者其他方法把这些元素挑出来。

接下来我们就一个一个讲解它们是怎么实现的。

二、找出URL

找出URL，目标是找到我们需要的那个**网址**。这一步是最简单的，比如说你要爬我的网站，只需要知道我的网站的网址即可。有的网站的网址就很复杂，而且经常会随机性变化。比如我们要从某搜索引擎中爬一些图片下来，就需要先进入搜索引擎，搜索目标关键词，然后进入图片栏，把这段网址copy下来即可。

举个例子，我想从BING搜索引擎爬几张好看的小猫图片，只需在BING中搜一下“小猫”，在进入图片专区，找到对应的网址，我找到的是 `https://cn.bing.com/images/search?q=小猫&form=HDRSC2&first=1` 。那如果想要爬别的图片，只要把网址中的“小猫”换成别的关键词就行了。

三、访问URL，获取源代码

爬虫的第一步，就是要让我们的程序被浏览器误认为是真人，不然它不会让我们爬。为了成功访问URL，就是需要写一个这样的函数： `def getHtml(url)` ，参数是指定的网址，返回的是该网址的网页源代码。有几种方法实现：

1. requests库（不推荐）

requests库是python中用于发送HTTP请求的第三方库，可以通过 `pip install requests` 安装，然后用 `import requests` 导入。

这个库有个关键的函数： `requests.get(url, headers)` ，其中url是网址；headers是**请求头**，是HTTP请求和响应的重要部分，没有请求头，就没法访问了；返回的是一个 `Response` 对象，包括响应头、响应状态码、响应内容等等。稍微了解一下请求头：它以**键值对**的形式存在，传递了这个HTTP的一些关键信息。

我们作为爬虫，自然是需要捏造一个请求头。我直接给出一个请求头（其实是我不知道别的）：

```
fakeHeaders = {'User-Agent':
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64)  \
    AppleWebKit/537.36 (KHTML, like Gecko) \
    Chrome/81.0.4044.138 Safari/537.36 Edg/81.0.416.77',
    'Accept': 'text/html, application/xhtml+xml, */*'}
```

仔细分析一下这个请求头： `User-Agent` 是告诉服务器我们使用的浏览器和操作系统的信息。至于值：

值	含义
`Mozilla/5.0`	历史遗留字段，现代浏览器都有这个标识
`Windows NT 10.0; Win64; x64`	运行系统是 Windows 10 64位系统
`AppleWebKit/537.36`	浏览器引擎（Chrome/Edge/Safari都使用它）
`Chrome/81.0.4044.138`	基于 Chromium 的浏览器（如Chrome或Edge）
`Safari/537.36`	兼容 Safari 的渲染引擎
`Edg/81.0.416.77`	表示这是 Microsoft Edge 浏览器（版本 81）

`Accept` 是告诉服务器，客户端接受哪些类型的数据。 `text/html` 在最前面，表示优先接受html格式的数据； `application/xhtml+xml` 表示也可以接受xhtml格式； `*/*` 表示如果没有前两种，任何数据我们都能接受。

虚拟一个请求头，主要有三点好处：

- 避免被**反爬虫**拦截：某些网站会检查 `User-Agent` ，如果是Python的默认UA，会拒绝响应。
- 模拟**真实用户**访问：有些网站依赖 `Accept` 类型返回不同内容。
- **兼容性**：部分网站依赖 `User-Agent` 判断浏览器类型，如果UA不正确，可能导致加载异常。

不知道这些都没关系，直接用就好了。

然后便可以用这个请求头访问了，使用一个变量来接住返回的对象： `r = requests.get(url, fakeHeaders)` ，并且确保网页编码 `r.encoding` 正确，然后就可以获取源代码了。

当然，最好加一个错误处理：如果无法访问，输出报错信息。最后的整体代码长这样：

```
def getHtml(url):
    import requests
    fakeHeaders = {'User-Agent':
        'Mozilla/5.0 (Windows NT 10.0; Win64; x64)  \
        AppleWebKit/537.36 (KHTML, like Gecko) \
        Chrome/81.0.4044.138 Safari/537.36 Edg/81.0.416.77',
        'Accept': 'text/html, application/xhtml+xml, */*'}
    try:
        r = requests.get(url, headers = fakeHeaders)
        r.encoding = r.apparent_encoding
        return r.text  # 返回了整个html源代码
    except Exception as e:
        print(e)  # 输出报错信息
        return ""
```

requests库速度快，安装简单，分发容易，但是**容易被反爬**，并且不能获取包含javascript生成的动态网页。还好我们还有两种方法。

2. selenium库（不推荐）

这个方法又慢又容易反爬，看看得了。

先 `pip install selenium` ，然后下载chrome浏览器或firefox浏览器，此外还需要 `chromedriver.exe` 或 `geckodriver.exe` ，这里是下载方法。

然后我就直接挂代码了。

```
def getHtml(url):
    from selenium import webdriver
    from selenium.webdriver.chrome.options import Options
    from selenium.webdriver.chrome.service import Service
    options = Options()  # 浏览器选项
    # 等价于 options = webdriver.chrome.options.Options()
    options.add_argument('—headless')  # 规定chrome浏览器隐身模式运行
    options.add_argument('--disable-gpu')
    # 禁止chrome使用gpu加速，能快点
    service = Service(executable_path='./chromedriver.exe')
    driver = webdriver.Chrome( service=service, options=options )
    # driver就是个chrome浏览器，需要下载安装chrome驱动器chromedriver.exe
    driver.get(url)  # 浏览器装入网页
    html = driver.page_source
    driver.close()
    driver.quit()
    return html
```

3. pyppeteer库（好用！）

讲半天终于讲重点了。

谷歌公司推出了一款编程工具叫puppeteer，用于控制Chrome浏览器。一位日本工程师以此为基础，推出了Python版本，叫pyppeteer，这里是pyppeteer的官网。

它的工作原理和selenium一样，是这样的：先启动一个浏览器Chromium，装入网页；这时用浏览器可以获取网页源代码，甚至可以向浏览器发送命令，实现键盘输入、鼠标点击等操作。

要使用pyppeteer，首先要 `pip install pyppeteer` ，注意python版本不低于3.6。然后必须下载特殊版本的谷歌浏览器Chromium，并**记住它的位置**。

在具体实现之前，首先要介绍一个很重要的知识：**协程**。

协程可以在单个线程内实现并发执行，并且切换由**程序**控制，而非操作系统调度。它有如下性质：

- **非抢占式**：通过 `await` 让出执行权，而不是强制中断。
- **协作式调度**：协程之间要显示切换。
- 适合**I/O密集型**任务：比如网络请求，文件读写等。

关于协程的实现，有几个重要的关键字和函数：

- `async` ：声明一个函数为**协程函数**（异步函数）： `async def func()` 。
- `await` ：**挂起当前协程**，等待另一个协程操作完成。

- asyncio.run(一个协程函数)：创建**事件循环**，负责调度协程的运行。

pyppeteer中所有函数都是协程函数，调用时都要在前面加 await ，这也是新手**最容易错的地方**。忘记加协程会这样报错：

```
Runtime Warning: coroutine 'XXX' was never awaited.
```

OK，了解了pyppeteer和协程，就要用它来获取网页了。基本的思路是：先创建一个事件循环，然后让获取网页的函数一直在这个循环中运行即可。也就是：

```
def getHtml(url):
    import asyncio
    import pyppeteer as pyp
    async def asGetHtml():  # 稍后实现

    loop = asyncio.new_event_loop()  # 创建新事件循环
    asyncio.set_event_loop(loop)  # 把这个循环设置为当前线程
    html = loop.run_untio.complete(asGetHtml(url))
    return html
```

现在的任务就是把实际获取网页源代码的函数写出来。基本的思路是：先打开浏览器，启动对应页面，设置好反反爬措施，然后爬取。因为很多函数大家不了解，我先把代码给出来，再解释一下它在干什么：

```
async def asGetHtml(url):
    browser = await pyp.launch(headless = False, executablePath = "D:\chromium\chrome-win\chrome
    page = await browser.newPage()
    await page.setUserAgent("Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
    await page.evaluateOnNewDocument("()=>{Object.defineProperties(navigator,{webdriver:{get:()=
    await page.goto(url)
    text = await page.content()
    await browser.close()
    return text
```

是不是很抽象？我一行一行解释：

首先是 launch 函数，它的作用就是打开浏览器，参数 headless 表示要不要用无痕模式打开， executablePath 是之前下载的Chromium浏览器的位置，还有一个参数 userdataDir 是指定用于存放临时文件的文件夹位置，用于防止莫名其妙的错误，当然也可以没有。

变量 page 就是浏览器打开一个新界面。

`setUserAgent` 之前讲过了，就是创建虚拟头。

`navigator.webdriver` 是浏览器暴露的一个属性，一般用这些奇奇怪怪的库访问浏览器时，这个值会为True，网站可以用这种方法反爬。而我们技高一筹，使用 `Object.defineProperties` 重新定义上述属性，让它始终为False。

后面的代码就很好理解了，先登入指定url，再获取网页源代码的字符串 `page.content()` ，最后关闭浏览器。至于完整代码就不给了，两端拼起来就好了。

这种方法大概是 `requests.get()` 速度的五分之一，但是有反反爬技术，就比较稳定。

四、分析并提取网页内容

既然我们获得了整个网页的html文件，现在就只剩最后一步了：找到我们需要的内容。依然有三种方式：

1. 正则表达式（速度最快）

我的网站里暂时没有html教程，虽然我写了一个入门教程，但是我觉得写的太烂了，等我学更深入一些再发。现在我必须简单介绍一下html：

html中的元素，一般是有两个对应的tag包裹起来一个元素（也有可能只有一个tag），前面的tag会有一些属性，形如： `<X attr1 = "xxx" attr2 = "yyy" ...>正文</X>` 。同时，tag可以嵌套。

正则表达式很好理解，只要我知道我想要的东西，它对应的字符串长啥样，就可以用正则表达式拿出来。比如：我用下面这个正则表达式：

```
pt = r'<td class="n">([^<]*)</td><td><span><span class[^<*]> ([^<]*)</span></span></td>'
```

就可以提出这些括号里的内容，如果看不懂的话，我以后介绍一下正则表达式（因为我现在也不会）。简单来说， `[^<]*` 可以匹配字符串中的任何字符，括号括起来的部分会被变量pt接收到。

问题来了：我怎么知道我要的部分对应的字符串长啥样呢？这也简单，打开浏览器，右键你需要的元素，点击“检查”，浏览器就会进入开发者模式，能看到网页源代码。

2. BeautifulSoup库（最推荐）

这个方法虽然速度不及正则表达式，但是架不住它简单好用，不用记正则表达式那些繁琐的规则。先安装 `pip install beautifulsoup4` ，用 `import bs4` 导入。

BeautifulSoup的使用逻辑是：一个字符串可以变成BeautifulSoup对象X，这个对象就会有 `find` ， `find_all` 等方法，可以找到对应的tag对象；还会有 `text` 属性，对应这个对象的正文（就是

两个tag包裹的内容）。

举个例子吧，比如对于下面这个**字符串**：

```
str = '''
    <div id="siteHeader" class="wrapper">
        <h1 class="logo">
        <div id="topsearch">
            <ul id="userMenu">
            <li><a href="http://openjudge.cn/">首页</a></li>
        </div>
    </div>
'''
soup = bs4.BeautifulSoup(str, "html.parser")
print(soup.find("li").text)  # 首页
```

对于html文件，也能这样做：

```
soup = bs4.BeautifulSoup(open("文件位置", "r", encoding="utf-8"), "html.parser")
```

接下来介绍一下两个find方法怎么用： `goals = soup.find("tag名字", attrs={···})` ，有两个参数，前者是你要找的tag，后者是要求这个tag有某种特殊属性。比
如： `diva = soup.find("div", attrs={"id":"synoid"})` ，就是寻找一
个 `<div id="synoid">中间内容</div>` 的字符串。 `find` 函数是找第一个这种元素，而 `find_all` 是找所有这种元素，返回的是一个字符串组成的列表（可能不是列表，我不确定）。

最后介绍soup的属性： `soup.text` 是正文， `soup["属性"]` 是这个soup对应的属性值。

如果文档中存在一些内容，字符串形式和目标很像，但是它不是目标，可以先锁定一个比较大的区域，再在这个区域中锁定目标。

3. pyppeteer实现模拟用户操作

有时我们不止需要一个网址，可能需要点击网页中的某些内容，还可能需要登录，这都需要手动操作。还好爬虫也可以安排上这些功能，使用pyppeteer库即可。

i. 自动登录

首先，为了保证能找到目标的输入框和按钮，需要设定好界面的宽度和高度，在打开浏览器时需要新加这样一个参数： `pyp.launch(args = [f'--window-size={width},{height}'])` ，其中 `width` 和 `height` 是设置好的变量。打开一个新界面以后，也需要设置宽
高： `await page.setViewport({'width': width, 'height': height})` 。

然后进入登录界面： `await page.goto(loginUrl)` 。**寻找元素**用这个函数： `element = await page.querySelector("tag名，.类名，或#id名")` ，其中参数是对应元素的属性，用 `tag名` ， `.类名` 或 `#id名` 都能索引到，也可以递归寻找： `#main > form > div.user-login > ···` ，从高处向下找。输入内容用这个函数： `await element.type("输入内容")` ，点击按钮用这个函数： `await element.click()` 。这就实现了登录。

ii. 等待网页元素出现

有的时候网页加载比较慢，需要等待加载。等待完全加载使用这个函数： `await page.waitForNavigation()` ，等待某个元素加载用这个函数： `await page.waitForSelector("#id名，或.类名", timeout=~)` 。

iii. 获取元素

大家也发现了，这个pyppeteer完全可以承担提取网页内容的责任，只要用上文提到的 `querySelector()` 就好了。有些朋友会问：怎么**寻找一个元素的selector**？很简单，进入检查模式以后，**右键目标元素-复制-复制Selector**即可。

还有一个函数，用于获取正文： `element.getProperty("innerText")` ，这个函数的返回值是一个对象，含有 `.jsonValue()` 这个属性，可以获取正文。

顺便讲一下，获取当前页面的网址可以用 `page.url` ，或许能用于debug。

五、最后的反反爬技巧

为了当一个伪人，需要保证自己有反应的时间。引入时间库 `import time` ，在关键的时候可以 `time.sleep(...)` 来缓冲，避免由于访问频率过高而被逮捕。

六、尾声

我也是复习完了，需要去练习一下了，如果学到了什么好东西还会补充在本教程。

最后忠告各位，爬虫有风险，一定不要乱爬啊！被抓了也不要说看过我的教程（doge）