百度爬虫广告过滤系统

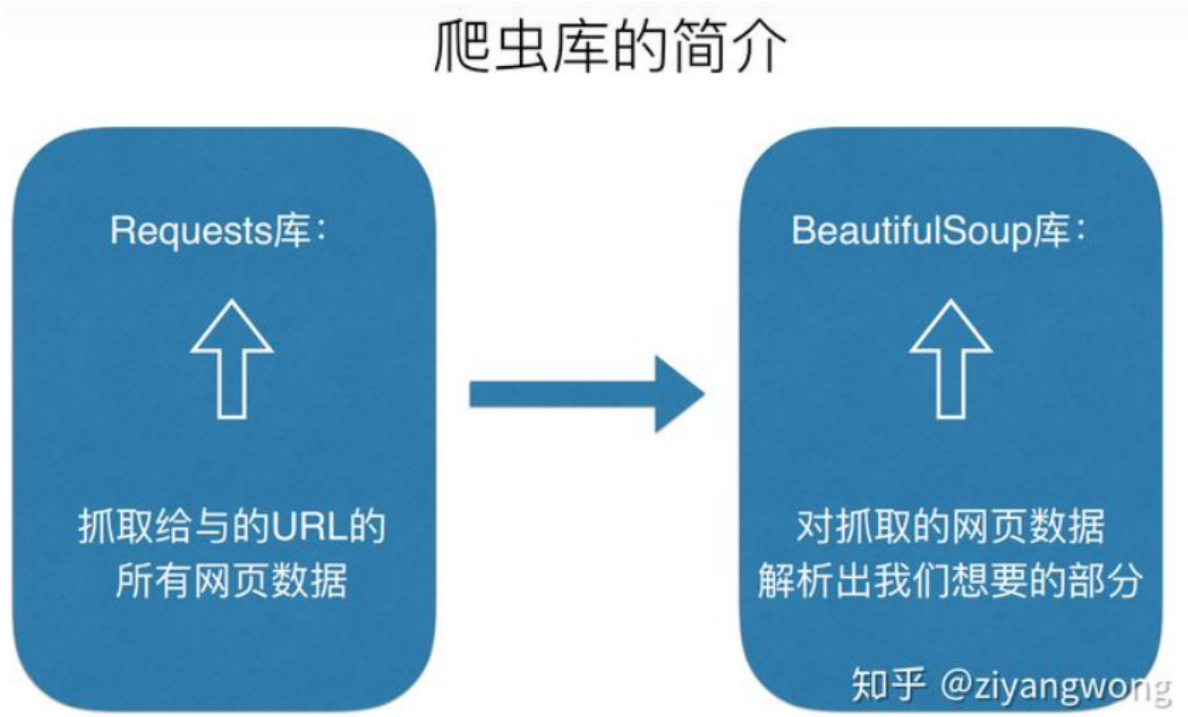
一、普通的百度爬虫：

仅满足爬取指定URL的百度搜索结果的爬取，不保存、不筛选、小量爬取

1.1 导入所需要的库

```
import requests
from bs4 import BeautifulSoup
```

首先我们必须要知道requests库和beautifulSoup库到底有些什么作用?



而对于BeautifulSoup库的使用我们首先掌握soup.find()和soup.find_all()的用法：

find和find_all函数都可根据多个条件从html文本中查找标签对象，只不过find的返回对象类型为bs4.element.Tag，为查找到的第一个满足条件的Tag；而find_all的返回对象为bs4.element.ResultSet（实际上就是Tag列表）,这里主要介绍find函数，find_all函数类似。

- `find(name=None, attrs={}, recursive=True, text=None, **kwargs)` 注：其中name、attrs、text的值都支持正则匹配。
- `find_all(name=None, attrs={}, recursive=True, text=None, limit=None, **kwargs)`注：其中name、attrs、text的值都支持正则匹配。

1.2 使用requests进行请求

```
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36"
}
#requests.get()命令中的第一个headers指的是浏览器头部（强调的是参数属性）
#第二个headers则为人为伪装的浏览器头部（强调的是参数内容）
response = requests.get('https://www.baidu.com/s?wd=粮食', headers=headers)
```

为了提高程序的可扩充性和适用性，我们并不赞同将keyword直接写入URL之中，因此：

```
key_words = '新疆棉'
# 伪装浏览器头部
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
}
response = requests.get('https://www.baidu.com/s?wd='+keyword,
headers=headers)
```

考虑到百度搜索结果的**一页结果之中只有10条搜索结果**，这是远远不足以满足数据分析需要的，因此，我们往往需要**加大页面深度**，规定好需要爬取的信息条数。

因此当搜索结果需要翻页时，我们只需用修改URL中的“&pn”的取值即可。通过观察百度网页源码可以发现，第一页的pn取值为0，第二页取值为10，第三页取值为20，以此类推。

```
key_words = '新疆棉'
# 页面深度
depth = 2
# 伪装浏览器头部
kv = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
}
# 获得每页搜索结果
for i in range(depth):#从0到depth给i赋值，i的取值不包括depth
    url = 'https://www.baidu.com/s?wd=' + key_words + '&pn=' + str(i * 10)
    #print(url)
    try:
        r = requests.get(url, headers=kv)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        html = r.text
    except:
        print("Error1")
```

这里值得注意的就是 r.raise_for_status()和r.encoding = r.apparent_encoding，下面我们一一对此进行解释：

- raise_for_status()，这个方法是专门与**异常**打交道的方法，该方法有这样一个有趣的功能，它能够**判断返回的Response类型状态是不是200。**如果是200，他将表示返回的内容是正确的，如果不是200，他就会产生一个HttpError的异常。
- r.encoding是它在**返回头**字段中推断的这个页面的可能的编码方式，但是有很大的概率是错误的，从而**出现大量乱码。**所以我们利用**r.apparent_encoding这个从返回内容中推断的，可靠性很高的编码方式**替换掉初始的编码方式。这样方便我们在输出的时候观察我们获取的页面信息。

代码创造到这个时候，我们已经能够获取一定数目的网页搜索结果的爬取并保存下来了，但是令人并不满意的地方是：

1. 爬取下来的内容并不能长久保存，当重新允许程序或运行另外的程序时，之前所爬取下来的内容就会丢失；
2. 由于百度搜索的商业模式，因此常常有大量的广告内容，而这些东西往往对于数据处理和分析是无用的，因此在爬取数据的时候就需要对内容有所筛选、甄别。

二、对于爬虫的改进

2.1 能够长久保存爬取内容的爬虫

一般情况下，我们选择将爬取到的内容写入一个文件，这往往需要对文件进行open、write和close操作，Python语言刚好有配套的代码：

```
with open('新疆棉m.txt','a', encoding='utf-8' ) as f:
    f.write('Hello World!'+'\n')
```

这样我们就将字符串”Hello World！\n“以utf-8的格式写入了文件新疆棉.txt

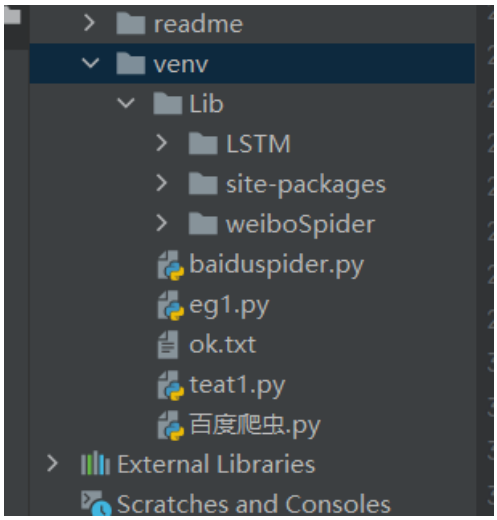
在此基础之上，利用**BeautifulSoup方法**对当前搜索页进行过滤，针对每一个h3标签，分别**获取a标签的文字信息和链接。**

基于这一原理，我们对上述程序进行微调：

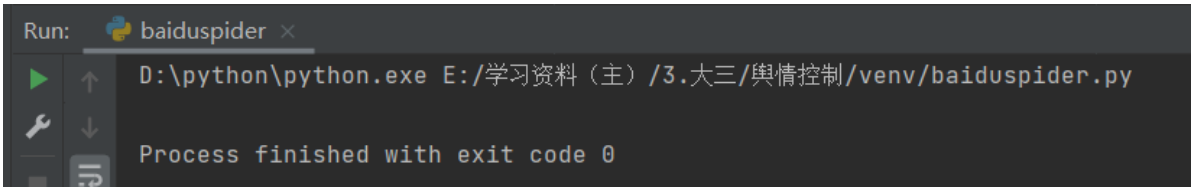
```
import requests
from bs4 import BeautifulSoup

key_words = '新疆棉'
# 页面深度
depth = 2
# 伪装浏览器头部
kv = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36"
}
# 获得每页搜索结果
for i in range(depth):
    url = 'https://www.baidu.com/s?wd=' + key_words + '&pn=' + str(i * 10)
    try:
        r = requests.get(url, headers=kv)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        html = r.text
    except:
        print("Error1")
    # 获得链接及非属性字符串
    # 通过字符串里面的源代码构建BeautifulSoup类对象,html-HTML文档字符串，
html.parser-HTML解析器， from_encoding-HTML文档的编码
    soup = BeautifulSoup(html, 'html.parser')
    h3 = soup.find_all('h3')#获得了该页面的所有h3标签

    for i in h3:
        #a对应于h3标签下的a标签
        a = i.a
        try:
            with open('新疆棉m.txt','a', encoding='utf-8' ) as f:
                #获取a标签中的链接
                href = a.attrs['href']
                # 获取a标签中的文字，并写入文件之中
                f.write(a.text+'\n')
```

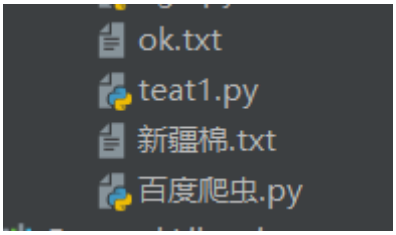
```
        except:
            print('Error2')
```

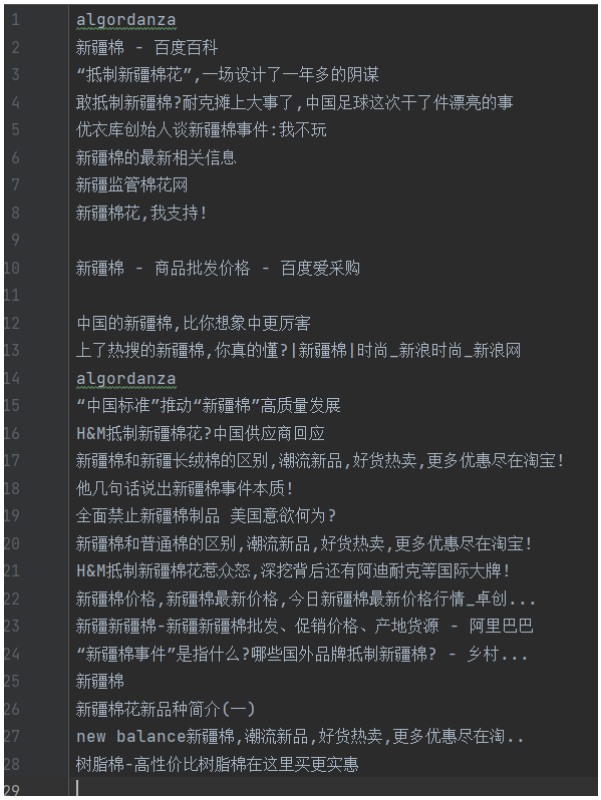
执行代码程序之前：



执行程序之后：







我们可以发现：所需要的文字信息都已经被存储在了**新疆棉.txt**文件之中了。

2.2 对百度搜索结果进行过滤

2.2.1 对HTML文档溯源

通过对百度网页的源码进行读取:

我们可以发现以下规律:

- class为`'result c-container'`的为非百度，非广告的内容(我们需要的内容)
- class为`'result-op c-container xpath-log'`的为百度自家的内容(可以按需筛选)
- class为其它的都为广告

因此我们可以根据标签中的**class属性**不同来筛选我们需要的搜索结果和链接。

2.2.2 `Xpath`技术与`BeautifulSoup`库的比较

由于Python中支持`Xpath`技术，同时具有`BeautifulSoup`库，因此我们需要对比一下这两者，从而确定我们使用哪一种方法来解决我们的广告过滤问题。

首先，我们必须要明确的是：`BeautifulSoup`**是一个库，而**`XPath`**是一种技术。**python中**最常用的**`XPath`**库是**`lxml`，因此，这里就拿`lxml`来和`BeautifulSoup`做比较吧

- **性能** `lxml` >> `BeautifulSoup`

`BeautifulSoup`和`lxml`的原理不一样，`BeautifulSoup`是基于DOM的，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多。而`lxml`只会局部遍历，另外`lxml`**是用c写的，而**`BeautifulSoup`**是用python写的**，因此性能方面自然会差很多。

- **易用性** `BeautifulSoup` >> `lxml`

`BeautifulSoup`用起来比较简单，`API`非常人性化，支持`css`选择器。`lxml`的`XPath`写起来麻烦，开发效率不如`BeautifulSoup`。

```
title = soup.select('.content div.title h3')
```

同样的代码用`Xpath`写起来会更麻烦

```
title = tree.xpath("//*[@class='content']/div[@class='content']/h3")
```

总结

需求比较确定，**要求性能的场合用`lxml`，快速开发用`BeautifulSoup`**

ps: `BeautifulSoup4`可以使用`lxml`作为parser了

2.2.3 基于`Xpath`的百度广告过滤系统

根据2.2.1中的内容，我们只需对Xpath中的class属性进行特定约束即可，因此：

```
html.xpath('//div[@class="result c-container new-pmd"]')
```

此时，我们得到的是非百度、非广告的内容，但是这个时候仅仅只是将需要的内容的源码筛选了出来，我们需要对内容进行处理，使其成为我们所需要的文字和链接信息。可以利用text()方法和string(.)的方法，这二者之间的区别就在于：**text()方法可以提取当前元素的信息，但是某些元素下包含很多嵌套元素，我们想一并的提取出来，这时候就用到了string(.)方法。**则有：

```
r11 = r1[i].xpath('string(.)')
```

到现在为止，我们已经成功提取出所需要的元素及其嵌套元素了。接下来我们需要考虑保存的问题：

- 保存于什么格式的文件之中？用什么函数实现保存操作？
- 在进行保存操作时是否需要对字串内容的编码方式进行规定？

首先我们考虑将内容保存在.txt文件中，此时我们利用**open()方法**打开文件，使用**write()函数**对文件进行写操作；为了简化open方法的使用，我们利用with方法进行简化。具体操作如下：

```
with open('新疆棉.txt', 'a', encoding='utf-8') as f:
    # 获取a标签中的文字，并写入文件之中
    f.write(a.text + '\n')
```

这个时候，我们已经能够建立起一个比较完善的百度广告过滤系统了。但是，这还远远不够，我们需要获取的信息有很多，比如：标题、链接、简介、内容、时间、来源等等，因此仅仅只是筛选掉无用的广告信息是远远不够的，我们需要对此有着更细化的分析：

```
 #对爬取的内容进行过滤筛选
    r1 = text_html.xpath('//div[@class="result c-container new-pmd"]//h3')#
获取文字标题
    r2 = text_html.xpath('//*[@class="c-abstract"]')#获取内容简介
    r3 = text_html.xpath('//*[@class="t"]/a/@href')#获取内容链接
    r4 = text_html.xpath('//*[@class="c-abstract"]/span')
```

综合以上所有，我们将最终的源码放在下面：

```
import requests
import json
from lxml import etree
#基于XPath的百度爬虫广告过滤系统

key_words = '新疆棉'
# 页面深度
```

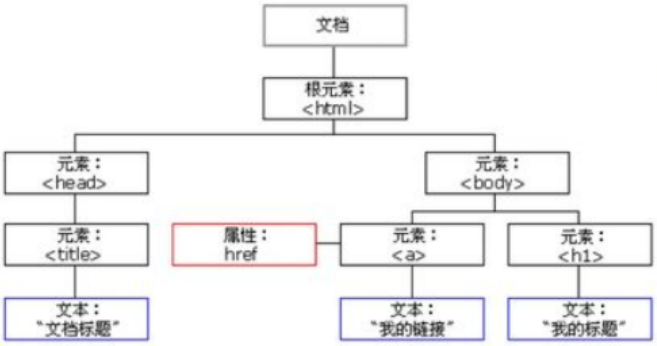
```
depth = 1
# 伪装浏览器头部
kv = {
    "User-Agent":  "Mozilla / 5.0(Windows NT 10.0; Win64; x64) AppleWebKit
/ 537.36(KHTML, like Gecko) Chrome / 80.0.3987.122  Safari / 537.36"
}

for i in range(depth):
    url = 'https://www.baidu.com/s?wd=' + key_words + '&pn=' + str(i * 10)
    try:
        r = requests.get(url, headers=kv)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        text=r.text
    except:
        print('Error2')
    # etree.HTML()可以用来解析字符串格式的HTML文档对象，将传进去的字符串转变成
_Element对象。
    # 作为_Element对象，可以方便的使用xpath()方法。
    text_html = etree.HTML(text, etree.HTMLParser())

    # 对爬取的内容进行过滤筛选
    r1 = text_html.xpath('//div[@class="result c-container new-pmd"]//h3')
 # 获取文字标题
    r2 = text_html.xpath('//*[@class="c-abstract"]')  # 获取内容简介
    r3 = text_html.xpath('//*[@class="t"]/a/@href')  # 获取内容链接
    r4 = text_html.xpath('//*[@class="c-abstract"]/span') #获取发布时间

    a = [len(r1),len(r2),len(r3),len(r4)]
    a = min(a)

    '''print(len(r1))
    print(len(r2))
    print(len(r3))
    print(len(r4))
    print(a)'''

    for i in range(a):
        # 我们在爬取网站使用Xpath提取数据的时候，最常使用的就是Xpath的text()方
法，该方法可以提取当前元素的信息
        # 但是某些元素下包含很多嵌套元素，我们想一并的提取出来，这时候就用到了
string(.)方法
        r11 = r1[i].xpath('string(.)')
        r22 = r2[i].xpath('string(.)')
        r33 = r3[i]
        r44 = r4[i].xpath('string(.)')
        # 将爬取到的内容写入‘新疆棉.txt’文件中
        with open('新疆棉.txt', 'a', encoding='utf-8') as c:
            # json.dumps 序列化时对中文默认使用的ascii编码.想输出真正的中文需
要指定ensure_ascii=False
            c.write(json.dumps(r11, ensure_ascii=False) + '\n')
            c.write(json.dumps(r22, ensure_ascii=False) + '\n')
```

```
            c.write(json.dumps(r33, ensure_ascii=False) + '\n' )
            c.write(json.dumps(r44, ensure_ascii=False) + '\n'+ '\n')
        '''print(r11, end='\n')
        print(r22, end='\n')
        print(r33, end='\n')
        print(r44)
        print('----------------------------')'''
```

2.2.4 基于`BeautifulSoup`库的百度广告过滤系统

```
import requests
from bs4 import BeautifulSoup

key_words = '新疆棉'
# 页面深度
depth = 27
# 伪装浏览器头部
kv = {
    "User-Agent": "Mozilla / 5.0(Windows NT 10.0; Win64; x64) AppleWebKit / 537.36(KHTML, like Gecko) Chrome / 80.0.3987.122  Safari / 537.36"
}
for i in range(depth):
    url = 'https://www.baidu.com/s?rtt=1&bsst=1&cl=2&tn=news&ie=utf-8&word=' + key_words + '&pn=' + str(i * 10)
    try:
        r = requests.get(url, headers=kv)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        html=r.text
    except:
        print('Error2')

    # 获得链接及非属性字符串
    # 通过字符串里面的源代码构建BeautifulSoup类对象,html-HTML文档字符串，html.parser-HTML解析器，from_encoding-HTML文档的编码
    soup = BeautifulSoup(html, 'html.parser')
    h3 = soup.find_all('h3')

    for i in h3:
        a = i.a
        try:
            with open('新疆棉2.txt', 'a', encoding='utf-8') as f:
                href = a.attrs['href']  # 每一条搜索结果对应的链接
                # 获取a标签中的文字，并写入文件之中
                f.write(a.text + '\n')
        except:
            print('Error2')
```

附录一：网页的构成



我们的网页其实就像是一棵树，
树根 > 大枝干 > 小枝干 > 枝丫 >

例：如果我们要找到"我的标题"
HTML > body > h1