爬虫程序核心是对网页进行解析，从中提取出自己想要的信息数据。这些数据可能是网址（url、href）、图片（image）、文字（text）、语音（MP3）、视频（mp4、avi……），它们隐藏在网页的html数据中，在各级等级分明的element里面，通常是有迹可循的，否则就没有爬取的必要了。提取的手段主要有三种：xpath、BeautifulSoup、正则表达式（Re）。下面分别进行介绍：

**（一）BeautifulSoup**

从本心来说，我更喜欢用BeautifulSoup。因为它更符合直观语义特性，find（）和find\_all（）函数已经基本上足够提取出任何的信息，对于身份证号、QQ号等特征特别明显的数据，顶多再加上一个正则表达式就完全OK了。

Beautiful Soup提供一些简单的、python式的函数用来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。

Beautiful Soup自动将输入文档转换为Unicode编码，输出文档转换为utf-8编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时，Beautiful Soup就不能自动识别编码方式了。然后，你仅仅需要说明一下原始编码方式就可以了。Beautiful Soup已成为和lxml、html6lib一样出色的python解释器，为用户灵活地提供不同的解析策略或强劲的速度。

pip install beautifulsoup4                 #pip方式安装bs4

pip install lxml                                   #pip方式安装 lxml 解析工具

具体介绍参见：

[BeautifulSoup 的结构和用法简介](https://link.jianshu.com?t=http:/cuiqingcai.com/1319.html)

静下心来看完这篇博文才了解到原来BeautifulSoup有这么多的用法，特别是find\_all（）和find（），下面是一些实例：

#实例

import requests

from bs4 import BeautifulSoup

headers = {'User-Agent': "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/22.0.1207.1 Safari/537.1"}

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

soup = BeautifulSoup(html.text, 'lxml')  #以上是网络获取html

soup=BeautifulSoup(open('index.html')) # 读取本地的html，加个open函数即可

print（soup.prettify()）  # 用标准html 显示方法打印html

#soup.find\_all()方法介绍 ，soup.find()与之基本类似，只是返回的是第一个值

find\_all( name , attrs , recursive , text , \*\*kwargs )

soup.find\_all('b')  #查找所有的b标签，返回列表

soup.find\_all(re.compile("^b")) # 正则表达式

soup.find\_all(["a", "b"])  #传入列表参数，找到所有的a标签和b标签

soup.find\_all(id='link2')  #传入id是link2的参数,Beautiful Soup会搜索每个tag的”id”属性

soup.find\_all(href=re.compile("elsie")) #传入正则表达式，查找所有的href标签内容中含有 elsie 的内容

soup.find\_all(href=re.compile("elsie"), id='link1') # 多层过滤，除了href进行限定之外，对id标签的内容也做了限定

soup.find\_all("div", class\_="sister") #最常用的查找技巧，这里之所以加‘\_=’是因为‘class’不仅是html中的tag，也是python语法的关键词，其他的不用加下划线

data\_soup.find\_all(attrs={"data-foo": "value"}) # 针对html5里面的data- 进行的专项查找

soup.find\_all(text="Elsie") # 对text内容进行查找

soup.find\_all(text=["Tillie", "Elsie", "Lacie"]) # 列表形式进行查找，与上面name类似

soup.find\_all(text=re.compile("Dormouse")) # 正则表达式形式，与上面类似

soup.find\_all("a", limit=2) # 找到前两个a标签， limit用来限定次数(

还有一个select()函数比较有用，基本用法如下：

# 我们在写 CSS 时，标签名不加任何修饰，类名前加点，id名前加 #，在这里我们也可以利用类似的方法来筛选元素，用到的方法是**soup.select()，**返回类型是**list**

**（1）通过标签名查找**

soup.select('title')

**（2）通过类名查找**

soup.select('.sister')

**（3）通过 id 名查找**

soup.select('#link1')

**（4）组合查找**

组合查找即和写 class 文件时，标签名与类名、id名进行的组合原理是一样的，例如查找 p 标签中，id 等于 link1的内容，二者需要用空格分开

soup.select('p #link1')

**（5）属性查找**

查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。

soup.select('a[class="sister"]')

soup.select('a[href="http://example.com/elsie"]')

get\_text()方法可以用来获取内容，请看下面代码：

soup = BeautifulSoup(html.text, 'lxml')

print (type(soup.select('title')))

print (soup.select('title')[0].get\_text())  # 获取第一个title标签的对应内容

for title in soup.select('title'):

           print (title.get\_text()) # 获取列表中的title对应内容

好了，BeautifulSoup的用法基本介绍到这里，除了速度上比较鸡肋之外，BeautifulSoup的查找方法做到了堪称人性化，给人以非常直观的语义理解。

**（二）Xpath的介绍和用法**

XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。结构关系包括 父、子、兄弟、先辈、后代等。

这里主要参考这篇博文：

[Xpath 语法和lxml的用法简介](https://link.jianshu.com?t=http:/www.cnblogs.com/BigFishFly/p/6380016.html)

表达式                         功能描述

nodename       选取此节点的所有子节点。

/                        从根节点选取。

//                       从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。

.                        选取当前节点。

..                      选取当前节点的父节点。

@                     选取属性。

**实例**

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式                                                     结果

bookstore                            选取 bookstore 元素的所有子节点。

/bookstore                           选取根元素 bookstore。注释：假如路径起始于正斜杠( / )，则此路径始                                                终代表到某元素的绝对路径！

bookstore/book                  选取属于 bookstore 的子元素的所有 book 元素。

//book                                  选取所有 book 子元素，而不管它们在文档中的位置。

bookstore//book                选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于                                                      bookstore 之下的什么位置。

//@lang                               选取名为 lang 的所有属性。

**谓语（Predicates）**

谓语用来查找某个特定的节点或者包含某个指定的值的节点。谓语被嵌在 **方括号** 中。

**实例**

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式                                                                        结  果

/bookstore/book[1]                       选取属于 bookstore 子元素的第一个 book 元素。

/bookstore/book[last()]                 选取属于 bookstore 子元素的最后一个 book 元素。

/bookstore/book[last()-1]             选取属于 bookstore 子元素的倒数第二个 book 元素。

/bookstore/book[position()<3]     选取最前面的两个属于 bookstore 元素的子元素的 book 元素。

**//title[@lang]**                                  选取所有拥有名为 lang 的属性的 title 元素。

**//title[@lang=’eng’]**                        选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。

/bookstore/book[price>35.00]     选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值                                                            须大于 35.00。

/bookstore/book[price>35.00]/title     选取 bookstore 元素中的 book 元素的所有 title 元素，且其中                                                                   的 price 元素的值须大于 35.00。

路径表达式                                    结果

/bookstore/\*                   选取 bookstore 元素的所有子元素。

//\*                                    选取文档中的所有元素。

//title[@\*]                        选取所有带有属性的 title 元素。

**选取若干路径**

通过在路径表达式中使用“|”运算符，您可以选取若干个路径。

**实例**

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式                                                                        结果

//book/title | //book/price                        选取 book 元素的所有 title 和 price 元素。

//title | //price                                            选取文档中的所有 title 和 price 元素。

/bookstore/book/title | //price                 选取属于 bookstore 元素的 book 元素的所有 title 元素，以及                                                                  文档中所有的 price 元素。

用法简介：

from lxml import etree

html = etree.parse(html.text)

result = etree.tostring(html, pretty\_print=True)

print(result)

（1）获取所有的标签

result = html.xpath('//li')

（2）获取标签的所有 class

result = html.xpath('//li/@class')

（3）获取标签下 href 为 link1.html 的[标签](" \t "_blank)

result = html.xpath('//li/a[@href="link1.html"]')

（4）获取[标签下的所有标签](" \t "_blank)

result = html.xpath('//li//span') #*因为span 只是在li 标签下面，并不是父子关系*

（5）获取所有li标签下的a标签中的所有 class的值，

result = html.xpath('//li/a//@class')

（6）获取最后一个li标签[的](" \t "_blank)a标签中[的 href](" \t "_blank)

result = html.xpath('//li[last()]/a/@href')

（7）获取倒数第二个元素的内容

result = html.xpath('//li[last()-1]/a')

print result[0].text

（8）获取 class 为 bold 的标签名

result = html.xpath('//\*[@class="bold"]')

print result[0].tag

通过以上实例的练习，相信大家对 XPath 的基本用法有了基本的了解。

也可以利用 text 方法来获取元素的内容。

下面是一些在程序中见到的表达式，大家有兴趣的话可以解析一下：

response.xpath('//div[@class="moco-course-wrap"]/a[@target="\_blank"]')

找到所有 div 标签下 class 名为moco-course-wrap的类，并且具有target="\_blank"的子标签 a标签

box.xpath('.//@href')

找到当前节点下，选取href属性

box.xpath('.//img/@alt').extract()[0]

找到当前节点下，所有的img标签（不一定是子标签），选取alt属性，提取文本内容

box.xpath('.//@src').extract()[0]

找到当前节点下，所有的src属性，提取内容

box.xpath('.//span/text()').extract()[0].strip()[:-3]

找到当前节点下，所有的span标签下的文本内容

box.xpath('.//p/text()').extract()[0].strip()

当前节点下，p标签的文本内容

url = response.xpath("//a[contains(text(),'下一页')]/@href").extract()

所有包含有 “下一页” 这个文本内容的a标签，提取这个a标签中的href属性值

爬取糗百CR版的名称和图片地址，果然非常好用。

nodes\_title = page.xpath('//div[@class="mtitle"]//text()')

print(len(nodes\_title))

#8

print(nodes\_title[0])

#老板，出个价吧

nodes\_imgurl = page.xpath('//div[@style="text-align: center;"]//img/@src')

#http://wx2.sinaimg.cn/mw690/8903e1f3gy1ffq18eskojg20b4069e81.gif

**（三）正则表达式**

最为头痛，最不直观的正则表达式，下次再写吧。

参考博文：

[Python爬虫入门（7）：正则表达式](https://link.jianshu.com?t=http:/python.jobbole.com/81346/)

[在线正则表达式测试](https://link.jianshu.com?t=http:/tool.oschina.net/regex/)

今天又鼓捣了几个小时的正则表达式，从基础到应用都看了半天，哎，正则表达式，能少用尽量少用吧，容错率太低了，一点点错了位置，可能都获取不到正确的数据。先来看看正则表达式的基础吧：

正则表达式(regular expression)描述了一种字符串匹配的模式（pattern），可以用来检查一个串是否含有某种子串、将匹配的子串替换或者从某个串中取出符合某个条件的子串等。

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与运算符可以将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

正则表达式是由普通字符（例如字符 a 到 z）以及特殊字符（称为"元字符"）组成的文字模式。模式描述在搜索文本时要匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

**普通字符**

普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。这包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

**非打印字符**

非打印字符也可以是正则表达式的组成部分。下表列出了表示非打印字符的转义序列：

字符                                               描  述

\cx         匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或                 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。

\f            匹配一个换页符。等价于 \x0c 和 \cL。

\n           匹配一个换行符。等价于 \x0a 和 \cJ。

\r            匹配一个回车符。等价于 \x0d 和 \cM。

\s            匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [ \f\n\r\t\v]。

\S            匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。

\t              匹配一个制表符。等价于 \x09 和 \cI。

\v             匹配一个垂直制表符。等价于 \x0b 和 \cK。

**特殊字符**

所谓特殊字符，就是一些有特殊含义的字符。若要匹配这些特殊字符，必须首先使字符"转义"，即，将反斜杠字符\放在它们前面。下表列出了正则表达式中的特殊字符：

特别字符描述

$                      匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 $ 也匹配 '\n' 或 '\r'。要匹配 $ 字符本身，请使用 \$。

( )                     标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \( 和 \)。

\*                        匹配前面的子表达式零次或多次。要匹配 \* 字符，请使用 \\*。

+                       匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。

.                         匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。

[                         标记一个中括号表达式的开始。要匹配 [，请使用 \[。

?                        匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使                           用 \?。

\                          将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例                            如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。

^                          匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字                            符集合。要匹配 ^ 字符本身，请使用 \^。

{                          标记限定符表达式的开始。要匹配 {，请使用 \{。

|                           指明两项之间的一个选择。要匹配 |，请使用 \|。

**限定符**

限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。有\*或+或?或{n}或{n,}或{n,m}共6种。

正则表达式的限定符有：

字符                                                                          描述

\*                        匹配前面的子表达式零次或多次。例如，zo\* 能匹配 "z" 以及 "zoo"。\* 等价于{0,}。

+                       匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。

?                        匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于 {0,1}。

{n}                      n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。

{n,}                     n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o\*'。

{n,m}                  m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例                 如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

**定位符**

定位符使您能够将正则表达式固定到行首或行尾。它们还使您能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

定位符用来描述字符串或单词的边界，^和$分别指字符串的开始与结束，span class="marked">\b 描述单词的前或后边界，span class="marked">\B 表示非单词边界。

正则表达式的限定符有：

字符描述

^                         匹配输入字符串开始的位置。如果设置了 RegExp 对象的 Multiline 属性，^ 还会与                              \n 或 \r 之后的位置匹配。

$                        匹配输入字符串结尾的位置。如果设置了 RegExp 对象的 Multiline 属性，$ 还会与                              \n 或 \r 之前的位置匹配。

\b                        匹配一个字边界，即字与空格间的位置。

\B                        非字边界匹配。

**反义字符**

有时需要查找不属于某个能简单定义的字符类的字符。比如想查找除了数字以外，其它任意字符都行的情况，这时需要用到反义：

常用的反义代码

代码/语法                           说明

\W                        匹配任意不是字母，数字，下划线，汉字的字符

\S                         匹配任意不是空白符的字符

\D                         匹配任意非数字的字符

\B                          匹配不是单词开头或结束的位置

[^x]                        匹配除了x以外的任意字符

[^aeiou]                匹配除了aeiou这几个字母以外的任意字符

例子：\S+匹配不包含空白符的字符串。

]+>匹配用尖括号括起来的以a开头的字符串。

举个例子，学习一下：

var str = "https://www.runoob.com:80/html/html-tutorial.html";

var patt1 = /(\w+):\/\/([^/:]+)(:\d\*)?([^# ]\*)/;

arr = str.match(patt1);

for (var i = 0; i < arr.length ; i++)

{    document.write(arr[i]);

              document.write("  br  ");

}

第一个括号子表达式捕获 Web 地址的协议部分。该子表达式匹配在冒号和两个正斜杠前面的任何单词。

第二个括号子表达式捕获地址的域地址部分。子表达式匹配 / 和 : 之外的一个或多个字符。

第三个括号子表达式捕获端口号（如果指定了的话）。该子表达式匹配冒号后面的零个或多个数字。只能重复一次该子表达式。

最后，第四个括号子表达式捕获 Web 地址指定的路径和 / 或页信息。该子表达式能匹配不包括 # 或空格字符的任何字符序列。

将正则表达式应用到上面的 URI，各子匹配项包含下面的内容：

第一个括号子表达式包含"http"

第二个括号子表达式包含"www.runoob.com"

第三个括号子表达式包含":80"

第四个括号子表达式包含"/html/html-tutorial.html"

在html中查找，在每一行的结尾，都要加上一个\s\*进行匹配，\s匹配非空白字符，从上一行末尾到下一行的起始中间，有可能是换行符，有可能是制表符，有可能一个或者多个，都可以用\s\*进行匹配。

来上几个例子吧，大家可以试着破解一下，我觉得尽量还是不要用正则，或者是配合着BeautifulSoup用正则，的确是比较坑。

这个是好不容易测试成功的第一个自己写的正则表达式，发现用正则的时候，不敢越级查找，哪怕子节点里面没有我们需要的数据，也要用.\*?来把这个内容表示出来，上面这个式子这么一大堆，其实只是提取了我们需要的两个数据而已， 发帖作者和图片地址，如果用soup去查找，那是相当清楚明白。

后来又试了一下，好像也不要，关键是后面的re.S，连同‘\n’这个换行字符考虑进去，要不然在html.text里面，各种换行符，确实是无从查找。

上面这句话是用来提取每个糗事的内容和图片相对地址的，与上面的正则表达式比较，进步的地方是，不需要再一层一层地写了，可以越级了

result 本身是list类型的，而它的分量 result[0]是元组（tuple）类型的，即不可更改的列表。这些提取出来的信息，被保存或者重组，然后进行下一步的处理。或者item代替result比较好，因为更加贴近scrapy的用法。(item是scrapy里面的一个类，用来保存提取出的信息，结构类型为字典）

参考文献：

[1][正则表达式30分钟入门教程](https://link.jianshu.com?t=https:/deerchao.net/tutorials/regex/regex.htm)

[2][正则表达式 -语法](https://link.jianshu.com?t=http:/www.runoob.com/regexp/regexp-syntax.html)

作者：八神苍月  
链接：https://www.jianshu.com/p/41d06a4ed896  
来源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。