以下是Python爬虫涉及的相关库  
  
请求库，解析库，存储库，工具库

1. 请求库：urllib/re/requests
2. 解析库：bs4 beautifulsoup

find\_all()  
搜索当前tag的所有tag子节点，并判断是否符合过滤器的条件。返回值类型是bs4.element.ResultSet。  
完整的语法：  
find\_all( name , attrs , recursive , string , \*\*kwargs )

正则表达式

|  |  |
| --- | --- |
| **模式** | **描述** |
| ^ | 匹配字符串的开头 |
| $ | 匹配字符串的末尾。 |
| . | 匹配任意字符，除了换行符，当re.DOTALL标记被指定时，则可以匹配包括换行符的任意字符。 |
| [...] | 用来表示一组字符,单独列出：[amk] 匹配 'a'，'m'或'k' |
| [^...] | 不在[]中的字符：[^abc] 匹配除了a,b,c之外的字符。 |
| re\* | 匹配0个或多个的表达式。 |
| re+ | 匹配1个或多个的表达式。 |
| re? | 匹配0个或1个由前面的正则表达式定义的片段，非贪婪方式 |
| re{ n} | 匹配n个前面表达式。例如，"o{2}"不能匹配"Bob"中的"o"，但是能匹配"food"中的两个o。 |
| re{ n,} | 精确匹配n个前面表达式。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\*"。 |
| re{ n, m} | 匹配 n 到 m 次由前面的正则表达式定义的片段，贪婪方式 |
| a| b | 匹配a或b |
| (re) | 匹配括号内的表达式，也表示一个组 |
| (?imx) | 正则表达式包含三种可选标志：i, m, 或 x 。只影响括号中的区域。 |
| (?-imx) | 正则表达式关闭 i, m, 或 x 可选标志。只影响括号中的区域。 |
| (?: re) | 类似 (...), 但是不表示一个组 |
| (?imx: re) | 在括号中使用i, m, 或 x 可选标志 |
| (?-imx: re) | 在括号中不使用i, m, 或 x 可选标志 |
| (?#...) | 注释. |
| (?= re) | 前向肯定界定符。如果所含正则表达式，以 ... 表示，在当前位置成功匹配时成功，否则失败。但一旦所含表达式已经尝试，匹配引擎根本没有提高；模式的剩余部分还要尝试界定符的右边。 |
| (?! re) | 前向否定界定符。与肯定界定符相反；当所含表达式不能在字符串当前位置匹配时成功。 |
| (?> re) | 匹配的独立模式，省去回溯。 |
| \w | 匹配数字字母下划线 |
| \W | 匹配非数字字母下划线 |
| \s | 匹配任意空 |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于 [0-9]。 |
| \D | 匹配任意非数字 |
| \A | 匹配字符串开始 |
| \Z | 匹配字符串结束，如果是存在换行，只匹配到换行前的结束字符串。 |
| \z | 匹配字符串结束 |
| \G | 匹配最后匹配完成的位置。 |
| \b | 匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。 |
| \B | 匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。 |
| \n, \t, 等。 | 匹配一个换行符。匹配一个制表符, 等 |
| \1...\9 | 匹配第n个分组的内容。 |
| \10 | 匹配第n个分组的内容，如果它经匹配。否则指的是八进制字符码的表达式。 |

## 正则表达式实例

#### 字符匹配

|  |  |
| --- | --- |
| **实例** | **描述** |
| python | 匹配 "python". |

#### 字符类

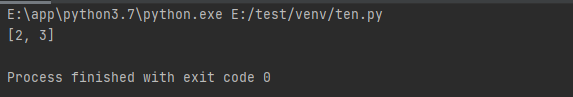
|  |  |
| --- | --- |
| **实例** | **描述** |
| [Pp]ython | 匹配 "Python" 或 "python" |
| rub[ye] | 匹配 "ruby" 或 "rube" |
| [aeiou] | 匹配中括号内的任意一个字母 |
| [0-9] | 匹配任何数字。类似于 [0123456789] |
| [a-z] | 匹配任何小写字母 |
| [A-Z] | 匹配任何大写字母 |
| [a-zA-Z0-9] | 匹配任何字母及数字 |
| [^aeiou] | 除了aeiou字母以外的所有字符 |
| [^0-9] | 匹配除了数字外的字符 |

#### 特殊字符类

|  |  |
| --- | --- |
| **实例** | **描述** |
| . | 匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。 |
| \d | 匹配一个数字字符。等价于 [0-9]。 |
| \D | 匹配一个非数字字符。等价于 [^0-9]。 |
| \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]。 |
| \w | 匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9\_]'。 |
| \W | 匹配任何非单词字符。等价于 '[^A-Za-z0-9\_]'。 |

给一个列表的每个元素加1

lili=[1,2]  
nnn=[1+x for x in lili]  
print(nnn)



### **类的专有方法：**

* **\_\_init\_\_ :** 构造函数，在生成对象时调用
* **\_\_del\_\_ :** 析构函数，释放对象时使用
* **\_\_repr\_\_ :** 打印，转换
* **\_\_setitem\_\_ :** 按照索引赋值
* **\_\_getitem\_\_:** 按照索引获取值
* **\_\_len\_\_:** 获得长度
* **\_\_cmp\_\_:** 比较运算
* **\_\_call\_\_:** 函数调用
* **\_\_add\_\_:** 加运算
* **\_\_sub\_\_:** 减运算
* **\_\_mul\_\_:** 乘运算
* **\_\_truediv\_\_:** 除运算
* **\_\_mod\_\_:** 求余运算
* **\_\_pow\_\_:** 乘方

BIF

E:\app\python3.7\python.exe E:/test/venv/ten.py

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '\_\_build\_class\_\_', '\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

Process finished with exit code 0

列表：

一般用复数命名，如letters、digits或names

用方括号（[]）来表示列表，并用逗号来分隔其中的元素

**访问列表元素**：列表名[索引]，索引从0而不是1开始，通过将索引指定为-1，可让Python返回最后一个列表元素，索引-2返回倒数第二个列表元素，索引-3返回倒数第三个列表元素，以此类推

**使用列表中的各个值：**

names = ["Tom", "Bob", "Jack"]

for i in names:

print(i)

修改、添加和删除元素：

列表名[索引]=修改后的元素

方法append()将元素'XX'添加到了列表末尾：列表名.append(‘XX’)

使用前先建一个空表：表名 = []

在列表中插入元素:方法insert()可在列表的任何位置添加新元素,列表名.append(索引,‘XX’)

从列表中删除元素：使用del语句删除元素:del 列表名[索引]

：使用方法pop()删除元素，方法pop()可删除列表末尾的元素，并让你能够接着使用它。术语弹出（pop）源自这样的类比：列表就像一个栈，而删除列表末尾的元素相当于弹出栈顶元素。

关于获取网页，本书主要介绍了Python的两个第三方模块，一个是requests，另一个是爬虫框架Scrapy。关于解析网页内容，本书主要介绍了3种方式——正则表达式、XPath和BeautifulSoup。两种网页获取方式和3种网页解析方式可以自由搭配，随意使用。

常规的反爬虫措施包括但不限于访问频率检查、验证码、登录验证、行为检测。本书对这些反爬虫策略都会进行一一破解。

简单网络爬虫的架构：

1. URL管理器：管理将要爬取的URL，防止重复抓取和循环抓取。
2. 网页下载器：这是下载网页的组件，用来将互联网上URL对应的网页下载到本地，是爬虫的核心部分之一。在Python中实现HTTP请求比较常用的主要有两个库，一是Urllib库，二是Requests库。
3. 网页解析器：这是解析网页的组件，用来从网页中提取有价值的数据，是爬虫的另一个核心部分。用来解析HTML网页的工具，Python中解析网页主要用3种工具，一是正则表达式，二是Lxml库（使用的是XPath语法），三是Beautiful Soup。
4. 输出管理器：这是保存信息的组件，用来把解析出来的内容输出到文件或数据库中。

在Python中，爬虫框架很多，常见的Python爬虫框架主要有Scrapy框架、Pyspider框架、Cola框架等。

HTTP请求信息由请求方法、请求头部和请求正文3个部分组成

主要关注请求方法和请求头部，这是与编写爬虫息息相关的。

请求方法：（1）GET方法，请求指定的页面信息，并返回实体主体。（2）POST方法，向指定资源提交数据进行处理请求（如提交表单或者上传文件），数据被包含在请求体中。POST请求可能会导致新的资源的建立或已有资源的修改。

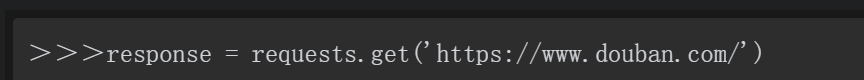
平时打开一个网站一般使用的是GET方法，也就是请求了一个页面；如果涉及向网站提交数据（如登录），就用到了POST方法。还有一些其他的请求方法，如HEAD、PUT、DELETE、CONNECT、OPTIONS、TRACE等，实际编写爬虫中较少用到。

请求头部：请求头部包含许多有关客户端环境和请求正文的有用信息。例如，请求头部可以声明浏览器所用的语言、请求正文的长度等。一般网站服务器最常见的反爬虫措施就是通过读取请求头部的用户代理（User Agent）信息，来判断这个请求是来自正常的浏览器还是爬虫，为了应对服务器的这种反爬虫策略，人们在编写爬虫时经常需要构造请求头部，来伪装成一个正常的浏览器。

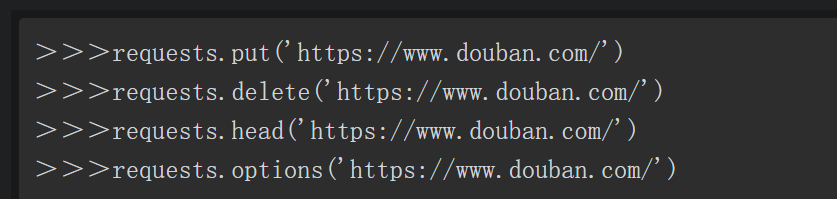
**Requests库**

1. Requests库的请求方法

这样就使用Requests的GET方法获取到了豆瓣网站的首页，并把获取到的响应对象赋值给了response。人们通过Requests的方法名称，很容易判断出它所使用的HTTP请求方法是GET方法。当然，也可以使用Requests的POST方法。

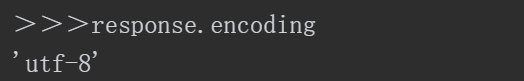






1. Requests库的响应对象

使用Requests请求方法后，系统会返回一个响应（response）对象，其存储了服务器响应的内容。现在回到最开始使用GET方法获取到的响应response，可以使用response.text获取文本方式的响应体内容，Requests会自动解码来自服务器的内容，大多数unicode字符集都能被无缝地解码。请求发出后，Requests会基于HTTP头部对响应的编码做出有根据的推测。当访问response.text时，Requests会使用其推测的文本编码。可以使用如下命令找出Requests使用了什么编码。



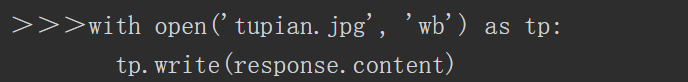
也可以修改response.encoding属性来让response.text使用其他编码进行解码，示例如下。



如果改变了编码，每次访问response.text，Requests都将会使用response.encoding的新值。

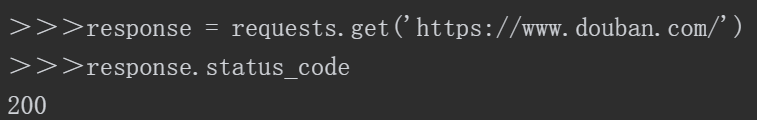
大多数情况下，Requests会自动解码来自服务器的内容，所以并不需要改变response.encoding，只有在使用特殊逻辑计算出文本编码的情况下，才需要修改编码。例如HTTP和XML自身可以指定编码，这样的话，可以使用response.content查看网页源码来找到其编码方式，然后设置response.encoding为相应的编码。这样就能使用正确的编码解析response.text了。

刚才提到了response.content，response.content以字节的方式访问请求响应体，假如获取的是一张图片，可以直接保存请求返回的response.content二进制数据。



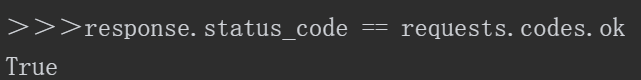
1. 响应状态码

响应状态码（HTTP Status Code）是用以表示网页服务器HTTP响应状态的3位数字代码。可以使用response.status\_code查看响应状态码，示例如下。



下面是常见的HTTP状态码及其含义：200——请求成功；301——资源（网页等）被永久转移到其他URL；404——请求的资源（网页等）不存在；500——内部服务器错误。

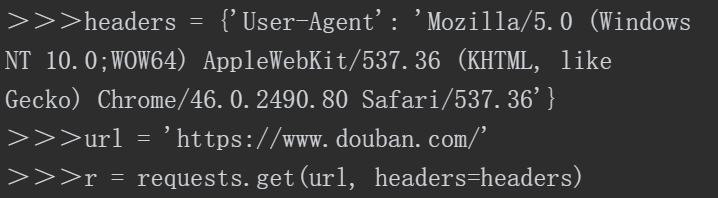
为方便引用，Requests还附带了一个内置的状态码查询对象。



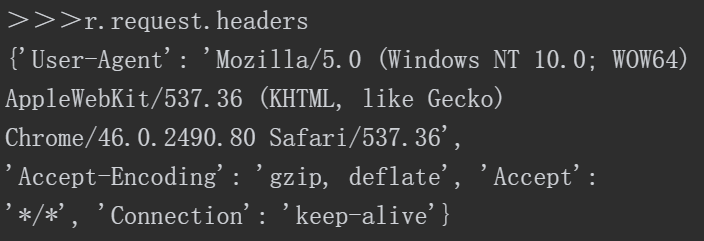
通过判断响应状态码是否与requests.codes.ok相等，可以帮助人们在编写爬虫时判断是否正确地获取到了资源。

1. 定制请求头部

服务器通过读取请求头部的用户代理（User Agent）信息，来判断这个请求是正常的浏览器还是爬虫。因此，可以为请求添加HTTP头部来伪装成正常的浏览器。只要简单地传递一个用户代理的dict给headers参数就可以了，示例如下。



可以通过r.request.headers查看设置的请求头部，示例如下。



可以看到里面User-Agent这一项的值正是所设置的用户代理。也可以使用以下代码查看以一个Python字典形式展示的服务响应头。

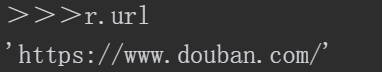


1. 重定向与超时

默认情况下，除了HEAD请求方法，Requests会自动处理所有重定向。可以使用响应对象的History方法来追踪重定向。例如访问http://www.douban.com/，会被重定向到https://www.douban.com/。



注意r.history返回的是一个list，它是一个response对象的列表，为了完成请求而创建了这些对象。这个对象列表按照从最旧到最新的请求进行排序。可以使用r.url查看实际请求的URL。



有时在爬取网页的过程中，服务器没有响应，为了应对这种情况，可以告诉Requests在经过以timeout参数设定的秒数时间之后停止等待响应。基本上所有的生产代码都应该使用这一参数。如果不使用，程序可能会一直等待响应。下面这条代码的意思是如果在3秒内收不到豆瓣服务器的响应，就会抛出一个timeout异常。

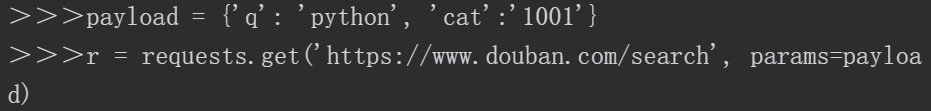


1. 传递URL参数

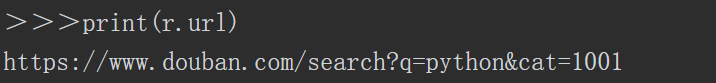
很多时候，网站会通过URL的查询字符串（query string）传递某种数据。例如，在豆瓣的搜索框中搜索关键字python，会发现跳转的URL变为如下形式。



这就是一个通过URL传递查询参数的例子，查询的数据会以键/值对的形式置于URL中，跟在一个问号的后面。Requests允许使用params关键字参数，以一个字符串字典来提供这些参数。举例来说，如果想在豆瓣的书籍栏目中查询与python相关的内容，可以传递q=python和cat=1001到https://www.douban.com/search，使用如下代码。



通过打印输出该请求URL，能看到URL已被正确编码。



**Urllib库基础**

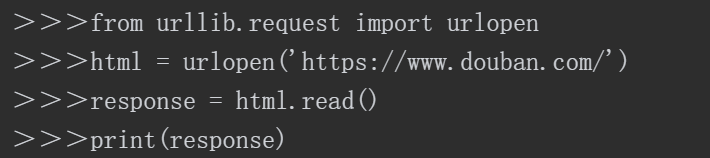
Urllib库是Python的标准库，提供了一系列用于操作URL的功能，其大部分功能与Requests库类似，也有一些特别的用法。它的API并不像Requests那样直观简洁。

Urllib库含有4个模块，主要作用如下：urllib.request——打开和读取URL；urllib.error——包含Urllib.request各种错误的模块；urlib.parse——解析URL；urllib.robotparse——解析网站robots.txt文件。

urllib.request模块用得最多，这里将着重讲解。urllib.parse模块拥有与Requests库不同的、特别的用处，这里也简单介绍一下。

1. 发送GET请求

下面来看如何使用Urllib库发送GET请求。例如，要抓取豆瓣的首页并打印响应内容。



这里首先从urllib.request模块引入urlopen函数，然后使用这个函数向豆瓣首页发送请求，它会返回一个二进制的对象html，对这个html对象进行read()操作，可以得到一个包含网页内容的二进制响应（response），最后打印出这个response。也可以对返回响应解码（decode），打印响应的文本内容。



这里对response使用utf-8解码，获取到了它的文本内容。

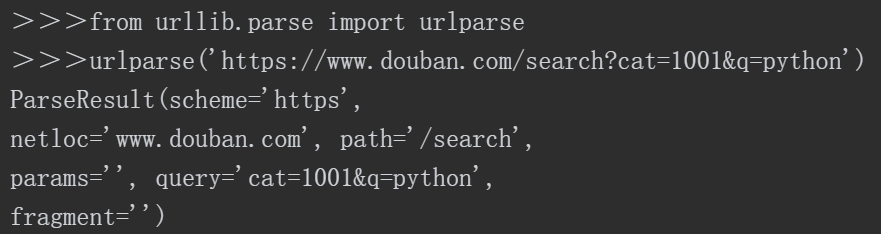
也可以像Requests库那样传递URL参数，例如，还是要在豆瓣的书籍栏目中查询与Python相关的内容，可以使用如下代码。

1. 模拟浏览器发送GET请求
2. URL解析

Urllib.parse提供了几个可以用来为爬虫提供URL解析的工具，这几个工具是Requests库所没有的。

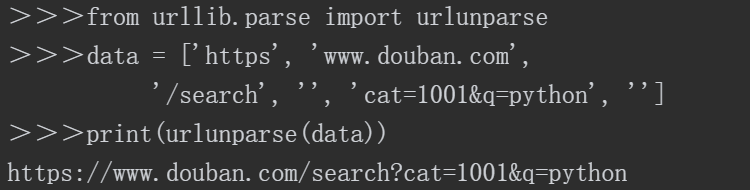
1. urlparse：拆分URL

urlparse模块会将一个普通的URL解析为6个部分，返回的数据类型都是元组。返回的6个部分分别是scheme（机制）、netloc（网络位置）、path（路径）、params（路径段参数）、query（查询）、fragment（片段）。



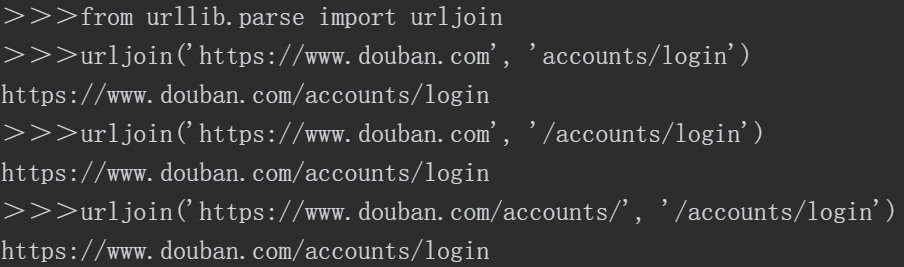
1. urlunparse：拼接URL

urlunparse可以拼接URL，为urlparse的反向操作，它可以将已经分解后的URL再组合成一个URL地址，示例如下。



1. urljoin：拼接两个URL

urljoin这个函数可以很方便地拼接URL，示例如下。



可以看到，urljoin这个函数会自行判断两个URL之间是否有/或重复的路径，从而拼接成正确的URL。在构造爬虫URL时，有时候会用到urljoin这个函数构造爬虫URL的完整路径。

**网页解析**

1. 常用网页解析工具

重点讲解使用Lxml库和其使用的XPath语法，因为XPath基本上能满足从HTML网页中提取信息的绝大部分需求，而且简单、易于掌握，解析速度也快。本章也会简单讲解Beautiful Soup和正则表达式

1. HTML源码简介

作为一种标记语言，基本上只要明白各种标记的用法就算学会了HTML。

HTML的语法格式分为嵌套与非嵌套两类，嵌套格式为<标记＞…</标记＞，非嵌套仅有<标记＞。此外，根据标记的不同，有的标记附带有属性参数，则表示为



1. <html＞</html＞标签

<html＞</html＞这一对标签之间的内容描述整个网页。

1. HTML文档的头部

位于<head＞</head＞中的内容就是HTML文档的头部，其中除了网页标题<title＞Example website</title＞显示在浏览器顶端之外，其他内容并不通过浏览器直接显示给用户，而是有其他的作用。

1. 网页主体内容

位于<body＞</body＞之间的所有内容是一个网页的主体，也就是浏览器窗口中可以出现的所有信息。

1. 链接

<a href="网址"＞链接名称</a＞创建了一个指向“网址”的超链接，其显示为“链接名称”。

1. <div＞</div＞标签

<div＞</div＞这一对标签用来排版大块HTML段落。

1. </br＞

</br＞为断行的标签。

1. XPath语法基础
2. Lxml库的安装

Lxml库是效率非常高且简单易学的网页解析库，Lxml库不是Python标准库，需要自行安装。可以在命令行里面用pip命令来安装Lxml库。

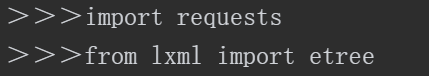


pip list：查看pip已经装好的库

1. XPath语法基础——通过路径查找元素

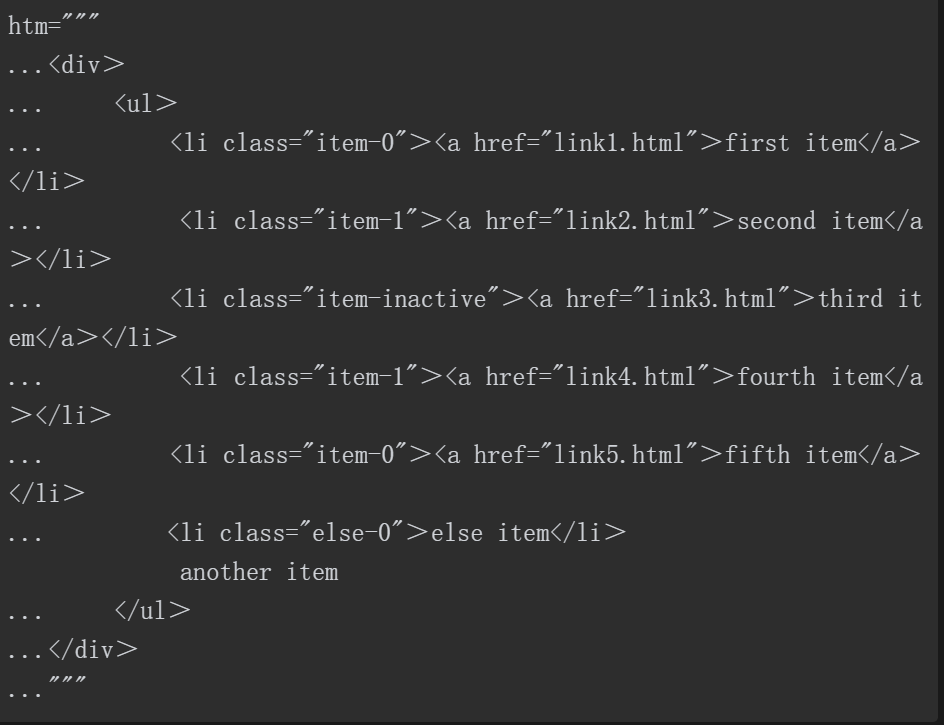
XPath即为XML路径语言，它是一种用来确定XML文档中某部分位置的语言。XPath基于XML的树状结构，有不同类型的节点，包括元素节点、属性节点和文本节点，提供在数据结构树中寻找节点的能力。XPath通常被开发者用来当作小型查询语言。下面用例子来演示XPath的用法。

首先导入Requests和lxml.etree这两个模块。



Lxml大部分功能都存在于lxml.etree下，我们主要使用etree这个模块，所以这里从Lxml导入etree。

然后定义一个HTML源码作为例子，示例如下。



这就是本段HTML源码的结构，它的这种结构是一层一层的，如果想找第一个<li＞标签，可以从头开始找：先找到<div＞→然后找<ul＞→最后找第一个<li＞。

XPath语法实际上就是使用这种层级的路径来找到相应元素的，它非常类似于人们日常使用的地址，它们都是从大的范围一直缩小到具体的某个地址上。当然了，如果要找的某个元素或地址是独一无二的，可以直接指明这个地址，不需要用这种层级关系来一层一层地定位。XPath对这种独一无二的元素可以直接定位。下面具体用代码来实现元素的查找和定位。首先需要使用HTML源码初始化etree。



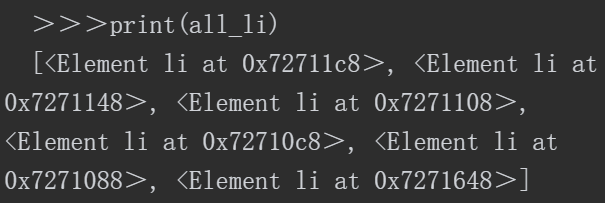
这样就得到了一个名字叫作selector的Element对象，可以对这个Element对象使用XPath筛选，系统会返回一个筛选的结果列表。

先查找所有的<li＞标签。XPath使用//来表示从根节点开始查找。



上面的代码对selector这个Element对象使用XPath方法，方法的参数就是XPath路径，这个路径是一个字符串的形式，上面代码中'//div/ul/li'这个XPath路径可以这样理解：从根节点开始查找<div＞，然后寻找<div＞下面的<ul＞标签，最后在<ul＞下面找到所有的<li＞标签。

这样就查找到了所有的<li＞标签，可以打印一下查看。

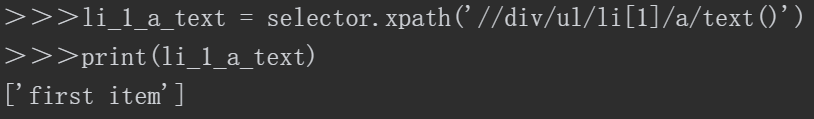


这是一个所有<li＞标签的列表。如果想定位第一个<li＞标签，可以编写如下代码。



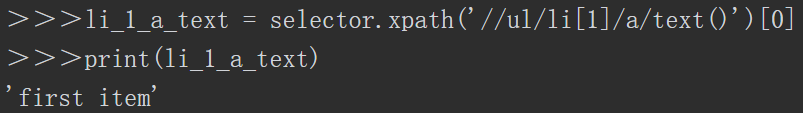
这里还是使用了路径查找，只是指定了<li＞标签的序号，使用li[1]这种形式，说明要提取的是第一个<li＞，要特别注意，XPath语法中的序号是从1开始的，与Python切片语法中从0开始切片是不一样的。还要注意，因为<div＞标签和<ul＞标签都只有一个，所以在查找和定位的时候没有使用序号。

前面已经找到了第一个<li＞标签，现在要提取第一个<li＞标签下的<a＞标签里面的文本，可以在路径中使用text()这种方法提取文本信息。



上面的代码还是按照路径一步步找到<a＞，然后使用text()提取出其中的文本信息，提取出来的信息是只有一个元素的列表，可以对这个列表切片，直接取出其中的文本内容。读者一定要学会使用text()提取文本信息。

读者可能会注意到，既然在这段源码里面<ul＞标签是唯一的，可以直接从根节点去定位<ul＞标签，这样不会有任何歧义，因此提取第一个<li＞标签里面<a＞标签的文本也可以按如下形式编写。



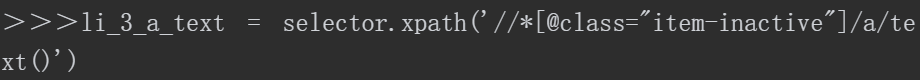
1. 通过属性查找元素

在源码中有<li class="item-0"＞这种形式的代码，这里的class就是一个<li＞标签的属性，同样道理，<a href="link1.html"＞里的href是<a＞标签的一个属性。可以通过属性定位元素。

例如要找第三个<li＞标签，第三个<li＞元素的属性是class="item-inactive"，可以编写如下代码来定位它。



利用属性来定位元素要使用类似[@class="item-inactive"]这种语法形式。再看一下HTML源码，这个属性在整个HTML源码里是唯一的，因此可以从根目录直接根据这个属性定位到第三个<li＞标签。



既可以在单引号里面使用双引号，也可以在双引号里面使用单引号，但是同时使用单引号和双引号会出现问题。上面使用了<li＞的class属性来定位元素，同样可以使用<a＞标签的href属性来定位，效果是完全相同的。



1. 提取属性值

提取属性值就是说，想要提取的内容是某个标签里的某个属性的值，例如要提取第三个<li＞里的<a＞标签的href属性值，可以编写如下代码。



这里通过@href这种语法形式提取到了<a＞标签的href属性值

最后编写如下代码取出所有<li＞标签的class属性。



这里从根节点开始查找<li＞标签，但是有很多个<li＞标签，路径中没有使用序号指明是哪一个<li＞标签，那就代表要提取的是全部的<li＞标签，然后使用@class取出所有<li＞标签的class属性值。

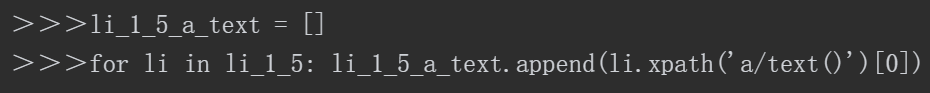
1. XPath的高级用法

HTML源码共有6个<li＞标签，现在假设要提取前5个<li＞标签，前5个<li＞标签的特点：虽然它们的class属性不尽相同，但都是以item-开头的。可以利用这个特点，提取出前5个<li＞标签。



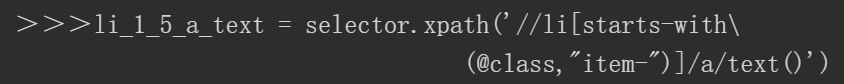
这里使用starts-with这样一种语法形式，成功地定位到了所有class属性以item-开头的<li＞标签。

如果提取出来的元素里面包含着子元素或者说提取出来的是一个代码段，可以对它继续使用XPath查找。例如上面提取了前5个<li＞标签，实质上是5个代码段，因为这些<li＞标签里还都包含着<a＞标签，可以对提取出来的<li＞标签继续使用XPath方法，查找里面的文本内容。



这里使用了一个for循环，对提取出来的前5个<li＞标签的HTML代码分别继续使用XPath方法，通过相对路径a/text()查找出了它们包含的<a＞标签里面的文本信息。也可以使用 .//a/text()这样的相对路径来查找，.// 表示以当前元素为根节点向下查找。

上面是为了说明对提取出来的代码段可以继续使用XPath方法，其实如果提取前5个<li＞标签下<a＞标签里的文本，没必要使用for循环，可以直接编写如下代码。



结果与上面是完全相同的。

来看最后一个例子：如何从代码段中提取出所有的文本。现在假设想要提取<ul＞里各层级下全部的文本。这段HTML源码的特点：<a＞标签里、最后一个<li＞标签里，甚至<ul＞标签里都包含有文本信息。如果使用如下代码只能提取到<ul＞这一层本层的文本。

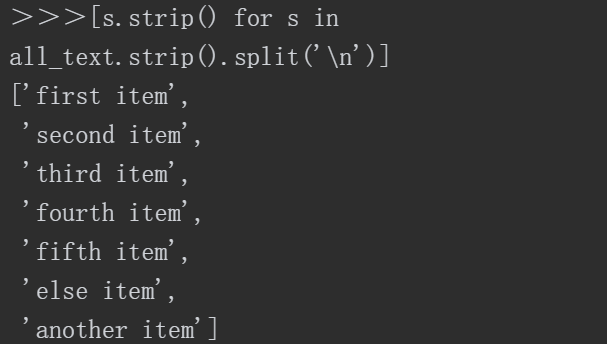


要提取<ul＞里各层级下全部的文本，可以使用下面的代码。



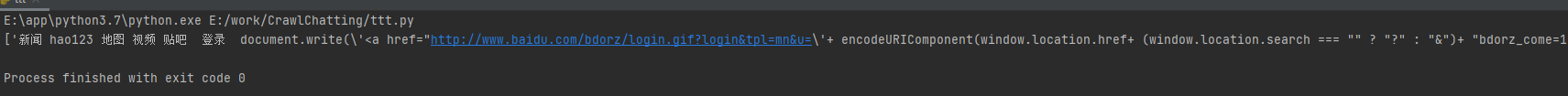
这里在路径的

？strip()参数为空时表示  删除空白符（包括'\n', '\r',  '\t',  ' ')



我的实际代码：

import requests  
import string  
from lxml import etree  
hh = requests.get('https://www.baidu.com/')  
hh.encoding = 'utf-8'  
se = etree.HTML(hh.text)  
aa = se.xpath('string(//\*[@id="u1"])')  
print([s.strip() for s in aa.strip().split('\n')])



1. Beautiful Soup库和正则表达式

除了Lxml库，网页信息的提取还有另外两种常用工具——BeautifulSoup和正则表达式。由于Lxml库的功能基本可以满足提取信息的需求，提取效率高，学习成本低，使用也非常方便，因此这一节只对Beautiful Soup和正则表达式做简单介绍，读者以后在遇到这方面问题时，可以查询其他文档。

1. Beautiful Soup简介

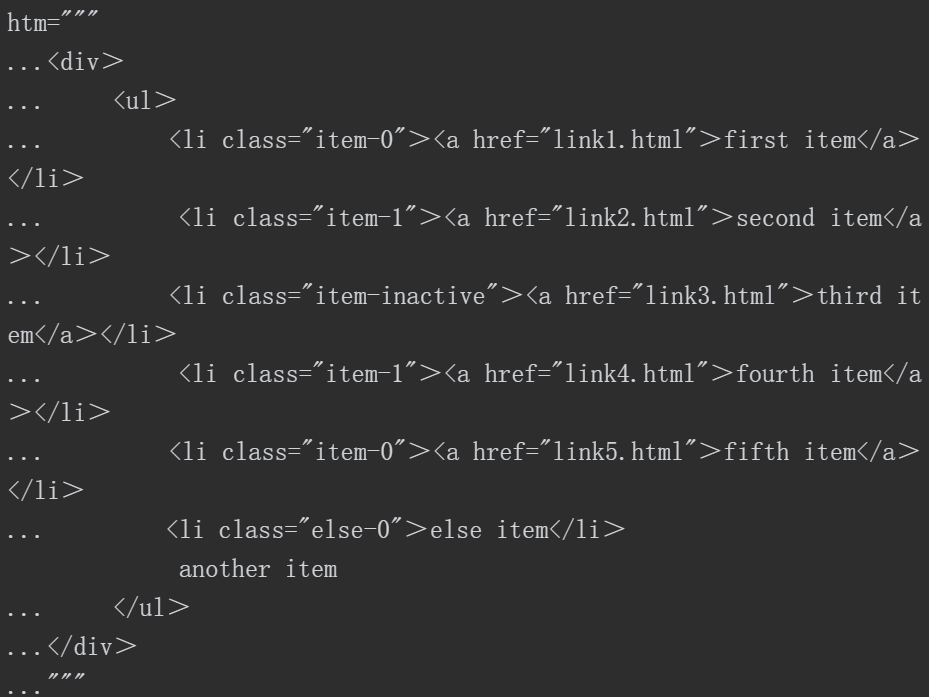
Beautiful Soup是一个灵活又方便的网页解析Python库，处理效率较高，支持多种解析器，很多爬虫初学者都是从使用Beautiful Soup开始学习网页解析的。Beautiful Soup不是Python标准库，需要安装。可以使用如下命令安装BeautifulSoup。



注意这里是beautifulsoup4。安装完成后，需要在Python解释器中引入。



下面还是使用讲解XPath语法的HTML源码做例子。



首先将定义的htm传入Beautiful Soup的构造方法，得到一个文档对象soup。



Beautiful Soup的构造方法的第一个参数是要解析的HTML文档，第二个参数是Beautiful Soup的解析器。Beautiful Soup支持Python标准库中的HTML解析器、Lxml解析器，还支持一些其他第三方解析器，如果不安装这些第三方的解析器，Beautiful Soup会使用Python默认的解析器。Lxml解析器更加强大，速度更快，推荐读者安装使用。上面的构造方法就使用了Lxml解析器。

1. Beautiful Soup基本用法

使用soup.x（x代表标签名）这种形式可以获得标签的内容，如图3-6所示。

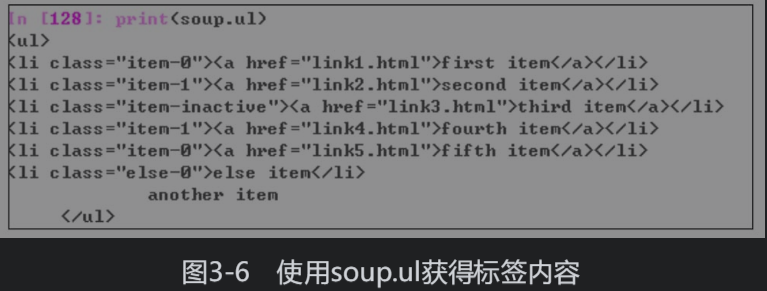
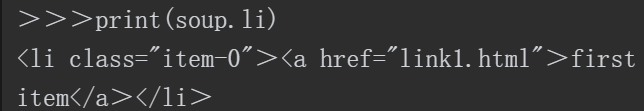
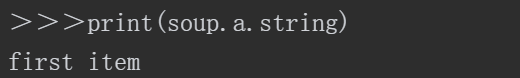




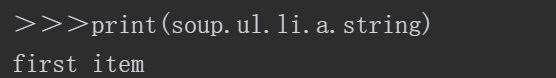
图3-6中打印出了<ul＞标签的全部内容。如果文档中有多个同样的标签，返回的结果是第一个标签的内容，示例如下。



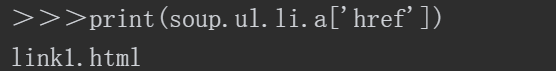
在标签后面加上.string，就可获取标签内的文本，例如加上.string获取第一个<a＞标签的文本内容。



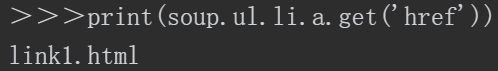
当然也可以通过嵌套的方式获取。



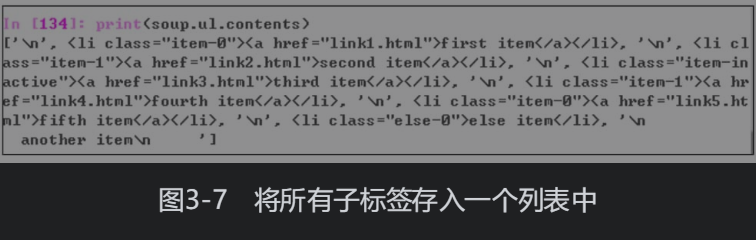
可以使用“标签[属性]”这种形式来获取属性值，如编写如下代码获取第一个<a＞标签的href属性。



也可以使用GET方法获取属性。



可以使用.contents形式将所有子标签存入一个列表中，如图3-7所示



这样打印出的是一个列表。如果使用soup.ul.children，得到的内容与soup.ul.contents完全相同，但它是一个迭代器的形式。

我的代码：

import requests  
import string  
from lxml import etree  
from bs4 import BeautifulSoup  
hh = requests.get('https://www.baidu.com/')  
hh.encoding = 'utf-8'  
se = etree.HTML(hh.text)  
aa = se.xpath('string(//\*[@id="u1"])')  
bb = BeautifulSoup(hh.text,'lxml')  
ii = bb.head.children  
while True:  
 try:  
 print(next(ii))  
 except StopIteration:  
 break

1. Beautiful Soup标准选择器

**#1 find\_all方法**

find\_all方法搜索当前标签的所有子节点，并判断是否符合过滤器的条件。find\_all方法是Beautiful Soup最常用的方法。

例如，要获取所有的<a＞标签，应编写如下代码。

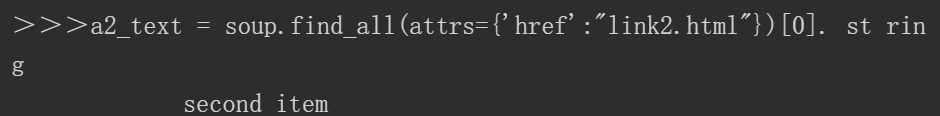


注：这里的soup(a)应该是soup(‘a’)

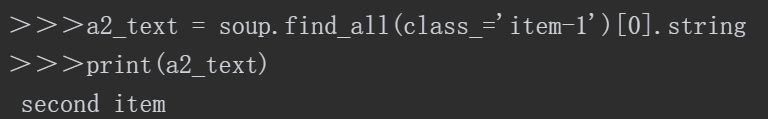
获取第二个<a＞标签的文本信息，应编写如下代码。



注意列表的切片是从0开始的。也可以使用属性定位第二个<a＞标签。



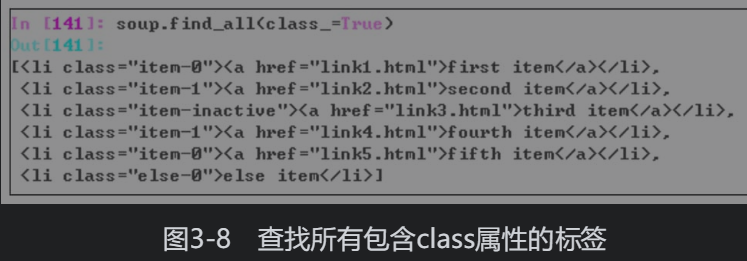
上面使用了attrs参数定义一个字典参数，以搜索包含特殊属性的标签。也可以选择所有class属性等于item-1的标签，然后取出第一个。



这里要注意，class\_是带有下画线的，因为class是Python的语法关键词，如果没有下画线，会出现语法错误。Beautiful Soup也可按id搜索：如果包含一个名字为id的参数，搜索时会把该参数当作指定名字标签的属性来搜索。

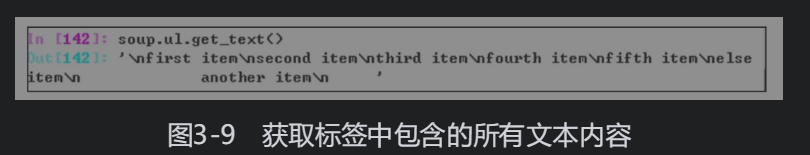


图3-8中的例子查找所有包含class属性的标签，无论class属性的值是什么，这个方法同样可以应用在其他属性上。



**#2 get\_text()**

如果只想得到标签中包含的文本内容，可以用get\_text()方法，这个方法获取到标签中包含的所有文本内容（包括子孙节点中的文本内容），并将结果字符串返回，如图3-9所示。



这样就提取了<ul＞下（包括子孙节点）的全部文本内容。读者可以与XPath使用string()这种提取方法的结果对比，看看是否相同。

1. 正则表达式

正则表达式是用于处理字符串的强大工具，其他编程语言中也有正则表达式的概念，区别只在于不同的编程语言实现支持的语法数量不同。正则表达式拥有自己独特的语法及一个独立的处理引擎，在提供了正则表达式的语言里，正则表达式的语法都是一样的。正则表达式无论是编写还是阅读，都相对复杂，这里只做简单的入门介绍。

Python标准库中的re模块提供正则表达式的全部功能，可以直接引入。



使用re的一般步骤如下。

第一步将正则表达式的字符串形式编译为Pattern实例。

第二步正则表达式处理函数使用Pattern实例处理文本并获得匹配结果。

下面首先介绍Python中常用的正则表达式处理函数，然后给出构造正则表达式Pattern的语法规则。

**#1常用的正则表达式处理函数**

**@1 re.match函数**

re.match尝试从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功，re.match就返回None。



这里pattern代表匹配的正则表达式；string是要匹配的字符串；flags是可选参数，用于指定匹配模式。

**@2 re.search方法**

re.search扫描整个字符串并返回第一个成功的匹配。以下为re.search函数的语法。



**@3 re.split**

re.split按照能够匹配的子串将string分割后返回列表。以下为re.split函数的语法。



maxsplit用于指定最大分割次数，不指定时将全部分割。

**@4 re.findall**

re.findall以列表形式返回全部能匹配的子串。以下为re.findall函数的语法。



**@5 re.sub**

re.sub用于替换每一个匹配的子串并返回替换后的字符串。以下为re.sub函数的语法。

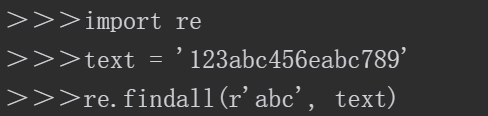


re.sub使用repl替换string匹配的部分，count用于指定最多替换次数，不指定时将全部替换。

**#2 正则表达式Pattern的语法规则**

前面简单介绍了正则表达式的处理函数，现在来看Python正则表达式的构造语法，也就是前面介绍的函数中pattern的写法。

Python正则表达式的基本语法规则就是指定一个字符序列，如要在一个字符串s='123abc456eabc789'中查找字符串'abc'，可以这样编写如下代码。



上述代码返回结果如下。



代码中r'abc'是模式字符串，模式字符串使用特殊的语法来表示一个正则表达式。

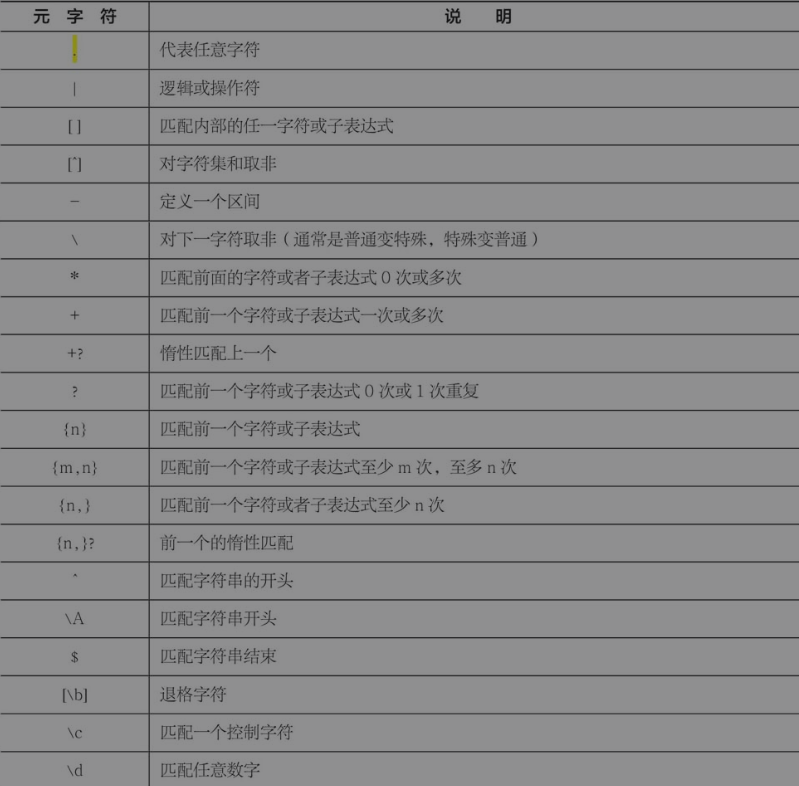
$1字母和数字表示它们自身。一个正则表达式模式中的字母和数字匹配同样的字符串。

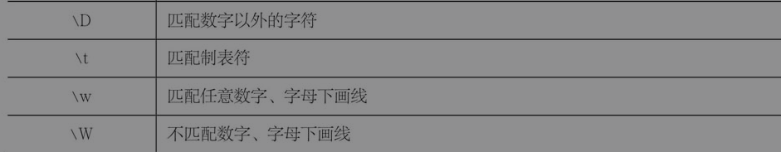
$2多数字母和数字前加一个反斜杠时会有不同的含义。

$3标点符号只有被转义时才匹配自身，否则它们表示特殊的含义。一般使用反斜杠转义。

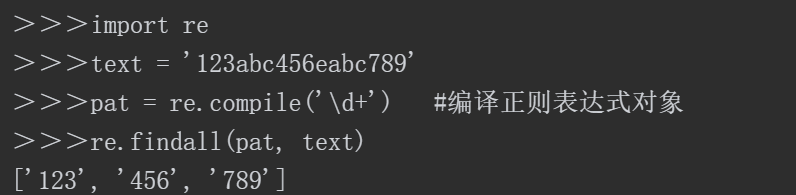
$4反斜杠本身需要使用反斜杠转义。

$5模式元素（如r'\t'等价于 '\\t'）匹配相应的特殊字符。





最后看一下re.compile方法。这个方法是Pattern类的工厂方法，用于将正则表达式pattern编译为正则表达式对象，当单个程序中的表达式被多次使用时，使用re.compile()和保存生成的正则表达式对象进行重用会更有效率。下面是一个例子。



首先使用re.compile编译了匹配模式\d+，根据表3-1，\d表示匹配任意数字，后面跟上 + 代表匹配一次或多次，这样使用re.findall就把文本中的数字全部提取出来了。

**基础爬虫实例**

1. Q房网爬虫实例
2. 网站页面分析

本节爬取的目标网站是深圳Q房网（http://shenzhen.qfang.com/），要爬取二手房房源的小区、户型、面积、区域、总价及房源特色，并把这些信息保存到计算机上。

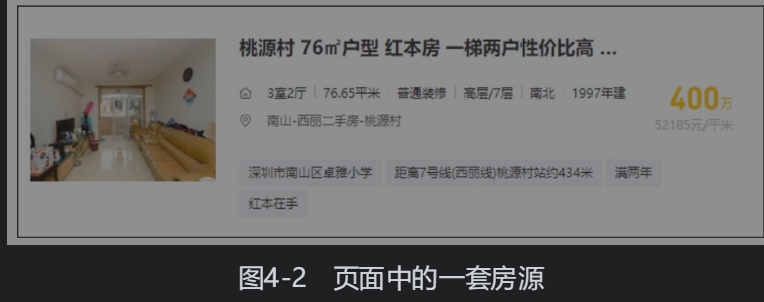
要爬取一个网站，首先要仔细分析它的页面特点和URL构造规律。图4-1所示为Q房网（深圳）的二手房页面。



可以看到每页上有30套房源，单击底部的翻页跳转到房源第二页，URL变为http://shenzhen.qfang.com/sale/f2。继续单击第三页，看到URL变为http://shenzhen.qfang.com/sale/f3。读者应该看得出来二手房页面URL的构造特点，那就是跟在f后面的是页码。如果想验证这个规律，可以直接将URL中的页码改为某一个数字，例如改为54，看到打开的页面的确是第54页。那么第一页是否也满足这样的规律呢？打开http://shenzhen.qfang.com/sale/f1。

可以看到这正是二手房房源的首页。至此，已经找到了URL的构造规律，可以利用这个规律对多个页面进行爬取。这次为了简单，可以爬取10页，也就是300条房源信息。

要爬取的数据包括二手房房源的小区、户型、面积、区域、总价及房源特色，下面分析页面，看看这些数据分布在页面的什么位置。图4-2所示的是页面中的一套房源。

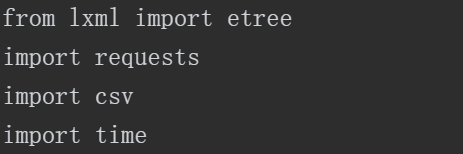


从这一条房源信息很容易看到，房源题目包含小区名称和房源特色介绍，下面一行包含户型和面积，单价和总价在靠右边的位置，板块在更下一行。从这里可以看出，房源列表页面已经包含所需要爬取的所有数据，因此爬虫只需要爬取房源列表页面即可，不需要爬取房源详情页面。

页面分析到此结束，下面可以开始编写代码了。

1. 编写Q房网二手房房源爬虫代码

首先导入Requests库和Lxml库。这里准备把数据保存为csv格式，因此还需要导入csv模块。为了控制爬行速度，还需要导入time模块，控制爬行速度的目的主要是防止被Q房网的服务器反爬虫禁止。



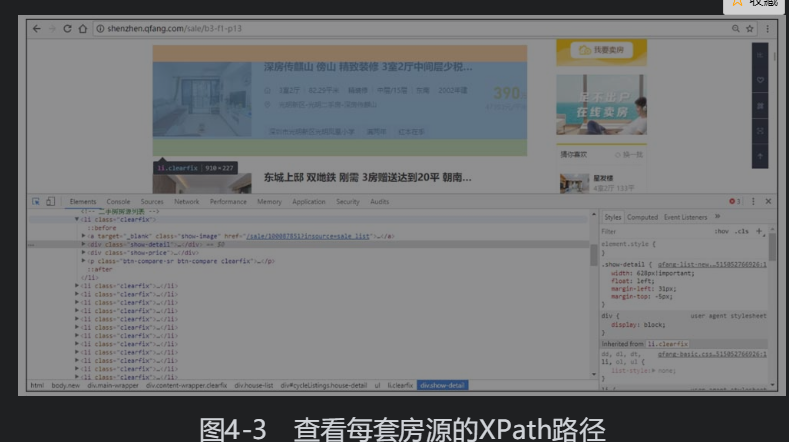
然后定义抓取函数。这里专门定义一个爬取和解析数据的函数。

在第2章爬虫基础中曾讲过，服务器会通过读取请求头部的用户代理（User Agent）信息，来判断这个请求是正常的浏览器还是爬虫。为防止被服务器反爬虫禁止，函数里还要定义一个头部。



到这里已经爬取了房源列表页，每一个房源列表页上有30套（也可能少于30套，如最后一页）房源，每一套房源上都包含一组所需要的信息。提取出这些房源数据有一个常用技巧：先提取每套房源整体的代码段，然后从中解析出每套房源的具体信息。也就是说，应该先获得每一套房源的HTML源码，然后从这一段HTML源码里面解析出来这套房源的详细信息。

先来获取每一套房源的代码段。可以借助Chrome浏览器的“检查”功能，看看如何获取每一套房源的代码段。在页面空白处单击右键，在弹出的菜单中选择“检查”，使用左上角的选择箭头选中一套房源，如图4-3所示。

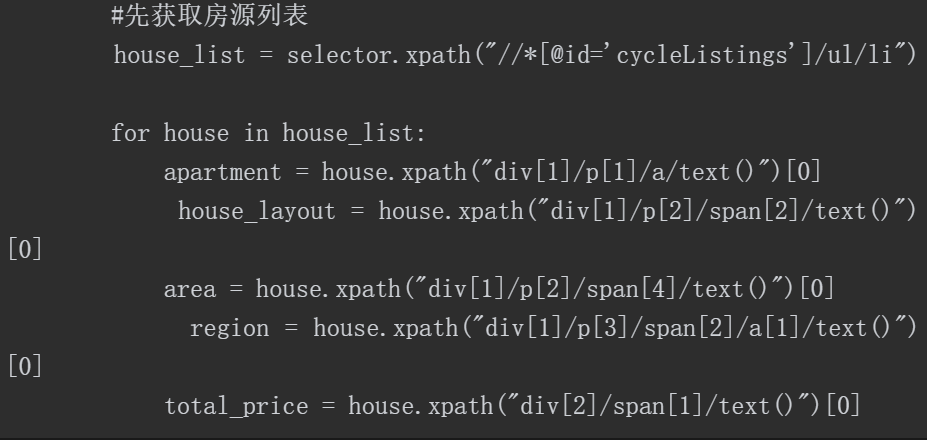


很明显，这里有很多<li＞标签，把鼠标移动到第二个<li＞标签，就会发现第二套房源信息变为蓝色，这样就知道每一个<li＞标签里就是一套房源的代码。使用Chrome浏览器复制第一个<li＞标签的XPath，就会得到类似下面的XPath路径。



如果把路径<li＞后面的序号去掉，就得到了<ul＞标签下所有<li＞标签的路径，可以在HTML代码中查看确认一下<ul＞标签下的所有<li＞标签是否均是房源、没有其他内容的<li＞标签。经过确认，可以得到房源列表的XPath路径为//\*[@id="cycleListings"]/ul/li。

使用这个XPath路径提取出来的，就是这一页中每套房源的代码段组成的一个列表，然后对这个列表做一个循环，在每一套房源的代码片段上再次使用Xpath，取出每套房源的具体信息，按这个思路继续写代码。



这里要注意，在for循环里使用的XPath路径是相对路径，也就是取出的<li＞标签下的路径。要写出正确的相对路径有一个技巧：从Chrome浏览器里面复制的第一套房源代码段的XPath路径如下。

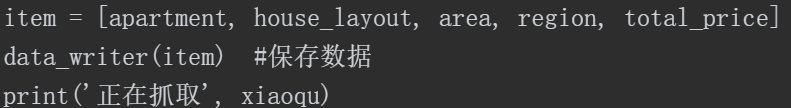


复制出的这套房源户型的XPath路径如下。



通过比对这两个路径，很容易知道户型在房源码中的XPath相对路径是div[1]/p[2]/span[2]。

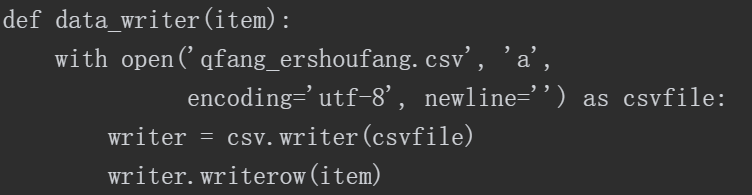
继续把爬取的数据构造成一个list，然后使用data\_writer这个保存函数写入计算机中。



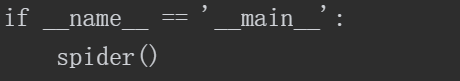
函数最后使用print打印出正在抓取的小区名称，作为爬虫的运行提示。

1. 保存爬取到的信息

可以把获取到的每一套房源信息组成一个list，保存到csv文件里，为了方便复用，这里直接写一个保存函数data\_writer，这个函数接受一个列表，然后使用csv的writerow方法写入一条记录。



首先以追加（a）的方式打开一个csv文件（如果没有，系统会新建一个），设置编码方式为encoding='utf-8'，为了防止csv文件在每次打开添加数据的时候插入空行，设置newline=''。最后编写主函数。



以上就是一个非常简单的爬虫，读者可以在PyCharm中运行这段爬虫代码，查看一下爬取的结果。通过这个例子的学习，读者应该对编写简单爬虫有了一个大概的思路，可以尝试爬取其他相似的栏目做练习，如爬取Q房网的租房栏目的房源信息等。最后要提醒读者的是，在爬取一个网站的时候，应该尽量克制爬取速度，一方面是防止被服务器发现，另一方面，爬取行为不可以占用太多服务器资源而影响网站正常用户的访问。因此，可以在代码中使用time.sleep()进行等待时间控制，人为地控制爬虫的爬取速度。

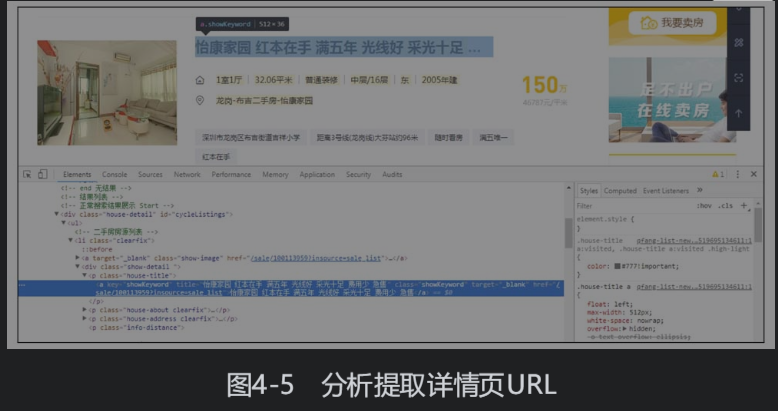
1. 多层页面的爬取
2. 爬取详情页面分析

上一节编写的Q房网二手房房源爬虫，仅仅爬取到了房源列表页面显示的房源信息，如果需要的房源信息只能在房源详情页面中看到，就需要爬虫除了能够爬取房源列表页面，还要能够从房源列表页面中提取出房源详情页面的URL，并爬取房源详情页面，以获取相关数据。

下面分析一下页面。这次希望在4.1节爬取的数据基础上增加房屋年限、抵押信息等数据。这些数据只有在房源详情页面中才能看到，图4-4所示为一套房源的详情页面。



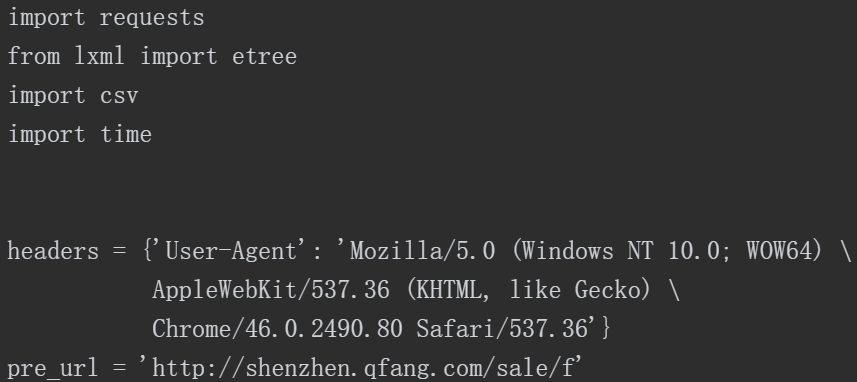
可以看到“交易属性”这个栏目包含了房屋年限和抵押信息。只有让爬虫爬取房源详情页面的URL，GET详情页面后，才能抓取到这些信息。需要在房源列表页面中提取房源详情页面的URL，先使用Chrome浏览器的“检查”功能分析一下，如图4-5所示。



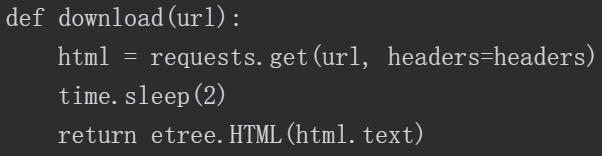
我们看到房源标题所在的<a＞标签里的href属性就是房源详情页面的部分URL，即/sale/100113959?insource=sale\_list。可以在前面加上http://shenzhen.qfang.com，从而构造出完整的房源详情页URL，根据这个思路开始写代码。

1. 编写爬取详情页面的代码

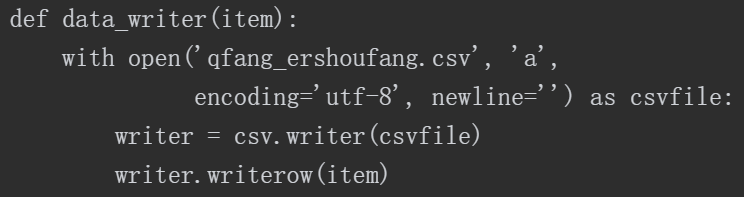
首先，导入需要的包，定义用户代理及网址前缀等常量。



因为爬虫要爬取房源列表和房源详情两个页面，为了实现代码复用，这里定义一个专门的下载函数，这个下载函数主要就是使用Requests下载页面，并返回一个页面信息提取器。



其次，定义保存函数。

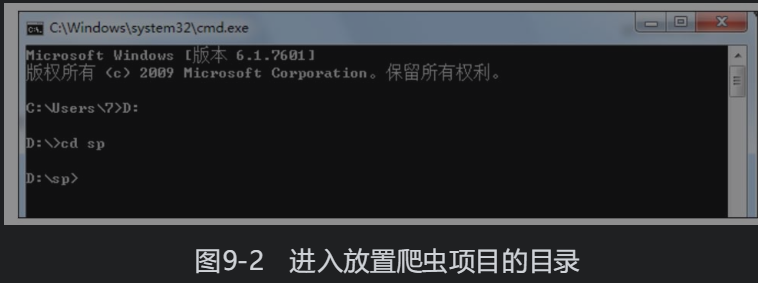


下面定义最主要的爬取函数，它能从房源列表页面中解析出房源详情页的URL，然后打开这个URL，从中提取出房源年限信息和抵押信息。

**Scrapy目录结构和简单爬虫实例**

1. Scrapy目录结构

现在假如要使用Scrapy爬取百度首页右上角的栏目名称和其URL，首先从命令行进入准备放置爬虫项目的目录。例如，爬虫项目准备放在D盘sp这个文件夹下，进入命令行，输入D:并按回车键，然后输入cd sp并按回车键，进入sp这个文件夹，如图9-2所示。

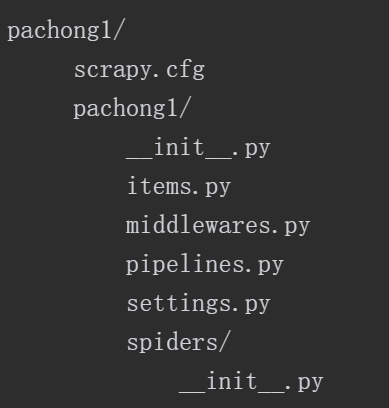


现在可以使用下面的命令创建第一个Scrapy项目，如图9-3所示。



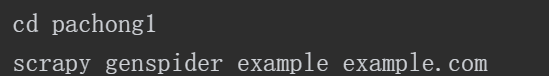


这个命令就在sp文件夹下创建了一个名为pachong1的Scrapy爬虫项目。它的目录结构如下。

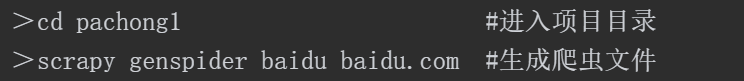


1. scrapy.cfg是项目的配置文件，一般不用设置。scrapy.cfg所在的目录就是项目的根目录。
2. items.py文件是保存爬取到数据的容器。要爬取什么数据，就要在这里面定义。
3. pipelines.py处理已经爬取到的数据。例如，要把爬取的item去重或者保存到数据库，就要在这个文件里面定义。
4. middlewares.py是中间件文件，主要用来对所有发出的请求、收到的响应或者spider做全局性的自定义设置，后面章节会有专门的介绍。
5. settings.py是Scrapy爬虫框架的设置文件。
6. spiders文件夹用于存放编写的爬虫代码，也就是说，爬虫主要逻辑就是在这里面定义的，可以在这个文件夹里定义多个爬虫，目前这个文件夹里面还没有爬虫文件，因为还没有生成或编写爬虫文件。

刚才使用scrapy startproject pachong1生成了爬虫项目，可以看到命令行有如下提示。

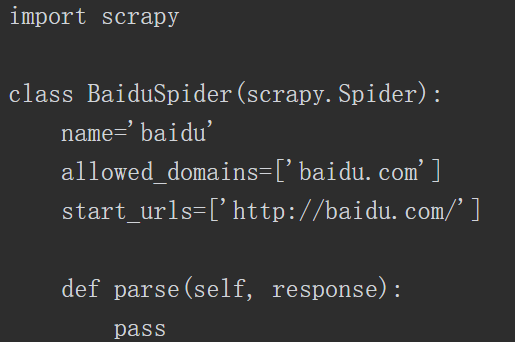


这个提示告诉我们使用Scrapy的模板生成爬虫文件的方法，按照提示执行生成爬虫文件的这两条命令。



上面的两条命令在项目的spiders文件夹下面生成了一个名称为baidu.py的爬虫文件。scrapy genspider baidu baidu.com这条生成爬虫文件的命令中，baidu是为这个爬虫起的名字，baidu.com是这个爬虫要爬取的网址。

用Notepad++编辑器打开baidu.py文件，将看到如下代码。



第一行引入Scrapy，然后定义一个BaiduSpider的类，这个类继承自scrapy.Spider这个类。name属性定义的是爬虫的名字，也就是刚才生成爬虫文件命令中的爬虫名字baidu。在一个爬虫项目中，可以定义多个爬虫，但爬虫的name必须是唯一的。allowed\_domains定义过滤爬取的域名，不在此允许范围内的域名会被过滤而不会进行爬取。start\_urls定义爬虫启动时默认爬取的网址，在通常情况下，爬虫默认从start\_urls开始爬取。

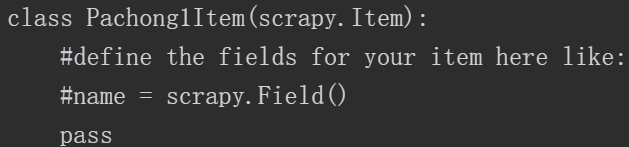
最后定义了一个parse方法，这个方法使用Scrapy爬取start\_urls后得到的响应（response）作为参数。这个方法就是解析爬取到网页的方法，在这里可以定义如何解析爬取到的网页，从response中提取出需要的信息。

以上逐行介绍了Scrapy爬虫框架生成的爬虫文件模板，可以在这个模板基础上增加具体的提取规则、解析方法等。

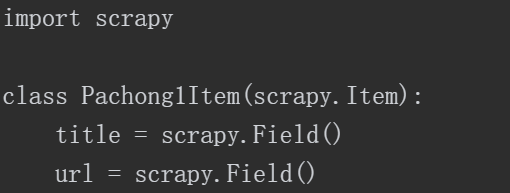
1. 百度爬虫实现

前面已经把百度爬虫文件生成好了，下面具体编写百度爬虫。

第一步，定义要爬取的数据。item.py是用于定义爬取数据的容器，它定义要爬取哪些字段的数据。打开items.py文件，能看到Scrapy爬虫框架默认生成了如下代码。



这是项目创建时自动生成的代码，它给了人们很好的提示，也就是使用类似name=scrapy.Field()来定义要提取的字段，非常简洁。这里提取两个字段，一个是栏目（tilte），另一个是栏目对应的URL。把代码改成如下的形式。



这里按照项目生成的item默认模板定义了一个继承自scrapy.Item的Pachong1Item类，然后定义了两个要提取的字段——title和url。

第二步，编写爬虫文件。打开spiders文件夹下的baidu.py爬虫文件，这个文件中已经生成了BaiduSpider这个爬虫类，start\_urls正是要爬取的百度首页，Scrapy启动后默认会从start\_urls开始爬取，然后把得到的响应（response）传递给parse解析方法。因此，可以直接在parse方法中编写百度首页的解析规则，提取数据到item及生成要进一步处理的URL。这里仅需提取出需要的栏目名称和URL即可。下面简单看一下Scrapy爬虫框架如何提取数据、解析网页。

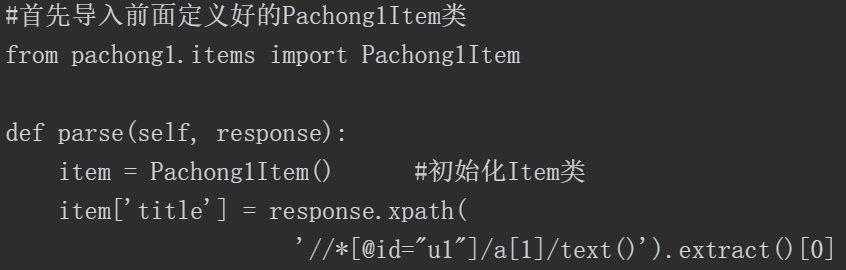
1. Scrapy选择器

Scrapy爬虫框架在Lxml库基础上构建了提取数据的一套机制，它们被称作选择器（seletors），因为它们通过特定的XPath或者CSS表达式来“选择”HTML文件中的某个部分。Scrapy选择器在速度和解析准确性上与Lxml库非常相似。

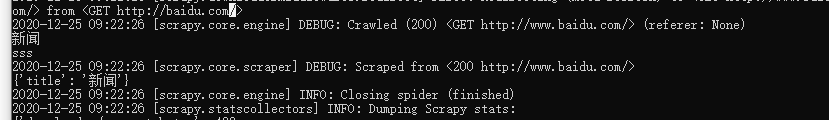
通过前面章节的学习，读者对XPath语法已经有比较深入的了解，同时由于XPath提供了比CSS选择器更强大的功能，因此在后面需要提取数据时，推荐继续使用XPath语法。

具体如何使用选择器呢？

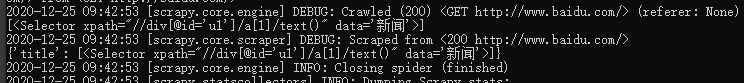
Scrapy爬虫框架提供了选择器的快捷方式——response.xpath()及response.css()，让用户可以直接使用选择器，本书主要使用response.xpath()这个选择器，其参数就是XPath路径。例如要提取出百度首页“新闻”栏目的名称，可以在parse方法中编写以下代码。



加上extract（）[0]



不加extract（）[0]

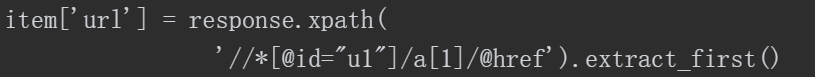


Parse的参数response和requests返回的response类型不一样：

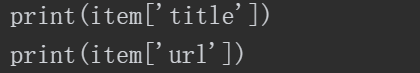
help(scrapy.http.response.html.HtmlResponse)这是parse的

help(requests.models.Response)这是requests是的

上面的代码首先把定义好数据字段的items类导进来，然后使用item =Pachong1Item()初始化这个items数据容器，items数据容器的使用语法类似Python字典，这里为其增加了title这个键和对应提取出来的值。请注意，Scrapy爬虫框架为了提取真实的原文数据，需要调用extract()方法序列化提取节点为unicode字符串，这样提取出来的是全部匹配元素的列表。如果要提取第一个匹配到的元素，既可以使用代码中写的列表切片方法，也可以调用extract\_first()提取。下面继续提取对应的URL。



这里使用了extract\_first()提取第一个数据，这种方法比前面提取title所使用的extract()[0]方法更好，因为如果没有匹配的元素，extract\_first方法会返回None，而extract()[0]则会直接报错。为了让读者看到提取的结果，下面把提取的数据打印出来。



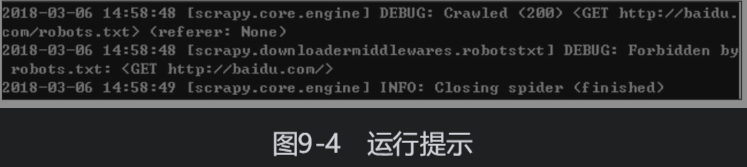
最后返回item。

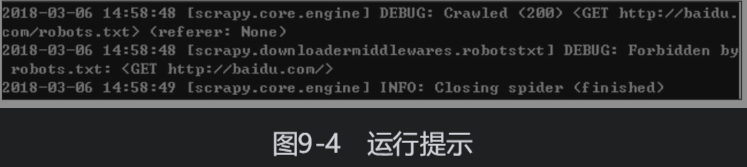


这里暂时不对提取的数据做保存或其他操作，至此百度Scrapy爬虫框架就编写完成了。在命令行中定位到项目根目录下，使用如下命令运行这个爬虫。



这样就运行了Scrapy百度爬虫，但是读者可能没有在屏幕上看到打印结果，而是发现类似Forbidden by robots.txt这样的提示，如图9-4所示。





这一节从爬取百度首页这个例子入手，讲解了Scrapy目录结构、数据容器items、选择器及小型爬虫的编写方法。通过这个小例子，读者应该初步学习到使用Scrapy爬虫框架编写爬虫的方法。后面的章节将详细讲解Scrapy爬虫框架的技术细节。

1. Scrapy命令行工具、选择器、数据容器
2. Scrapy常用命令行工具

#1 scrapy startproject myproject

这条命令用来创建一个Scrapy爬虫项目，myproject代表项目名称。

#2 scrapy genspider mydomain mydomain.com

这条命令在项目中使用BasicSpider模板生成一个爬虫文件，所以此命令需要在项目目录中运行。

#3 scrapy crawl myspider

在项目中启动爬虫命令，myspider代表爬虫名称。

以上3个命令上一节都使用过了。Scrapy还有几个经常使用的命令行工具，可以在命令行中运行scrapy -h，查看所有可用的命令，Scrapy提供了两种类型的命令，一种必须在Scrapy项目中运行，另一种属于全局命令，不需要项目。读者学习使用命令行工具时，要注意其是否需要项目。下面简单介绍最常用的其他几个命令行工具。

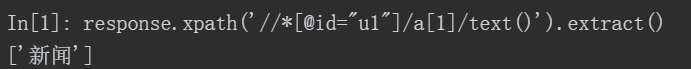
#4 scrapy shell [url]

这条命令是全局命令。以给定的URL（如果给出）或者空（没有给出URL）启动Scrapy shell。这个命令经常用来在交互模式下检查XPath语法提取数据是否正确，例如执行如下命令，就启动了Python交互模式。如果安装了Ipython，默认进入Ipython交互界面，如图9-6所示。

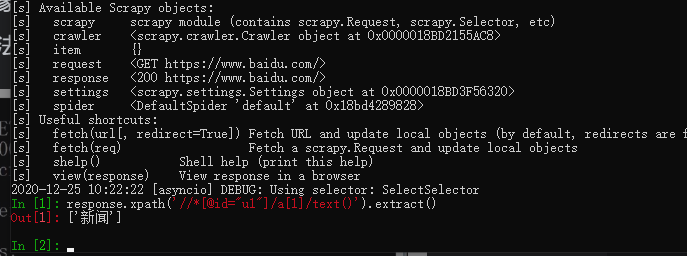


图9-6显示了运行这条命令后的交互终端界面。可以看到Scrapy已经准备好了许多可用的Scrapy对象，如请求对象request，返回响应对象response等，因此可以在response上直接使用XPath语法，查看解析结果。





我的



可以看到系统正确地返回了所要提取的信息。注意Ipython解析器提示符为类似In[1]:这样的运行序号。

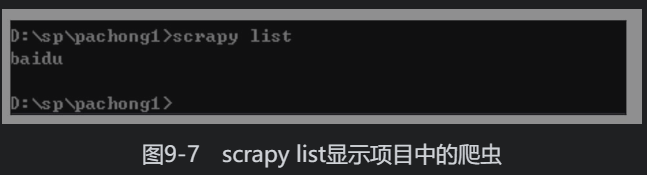
#5 scrapy view [url]

这条命令是全局命令。它会在浏览器中打开给定的URL，并以Scrapy spider获取到的形式展现。有些时候spider获取到的页面和普通用户看到的并不相同。因此该命令可以用来检查spider所获取到的页面，并确认这是用户所期望的。例如执行如下命令，可以看到spider下载的网页，与原来的网页看上去基本相同。



# 6 scrapy list

这条命令需要在项目中运行，它会列出当前项目中所有可用的spider（每行输出一个spider）。运行这条命令还可以帮助用户检查错误，因为它可以提示spider的语法错误。例如在pachong1项目中执行这个命令，结果显示pachong1项目中只有baidu这一个爬虫，如图9-7所示。



#7 scrapy parse <url＞ [options]

这条命令需要在项目中运行，它获取给定的URL并使用相应的spider分析处理。如果我们提供--callback选项，则使用spider的对应方法处理，否则使用parse解析。这条命令也可以帮助人们检查解析函数是否正确。

以上就是比较常用Scrapy命令行工具，Scrapy还有很多其他的命令行工具，读者可以关注Scrapy文档。

1. Scrapy选择器高级应用

上一节简单讲解了Scrapy选择器的用法，下面再看一下Scrapy选择器的其他用法。

#1 嵌套选择器

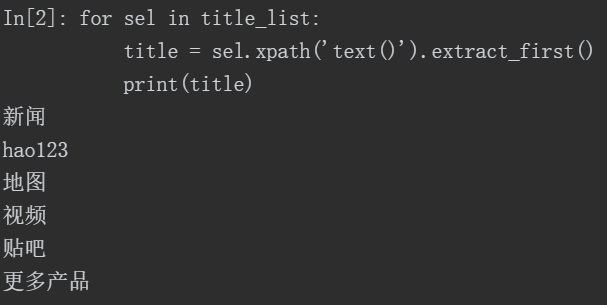
选择器方法（.xpath()或.css()）返回相同类型的选择器列表，因此可以对这些选择器继续调用选择器方法。还是用爬取百度首页作为例子，在命令行中执行如下命令。



系统进入交互界面，然后运行如下代码。



这行代码帮我们获得了百度页面右上面角的栏目列表代码。对这个列表迭代，继续执行XPath，提取栏目名称。



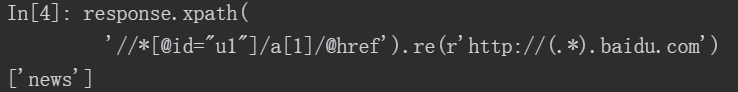
读者应该注意，在for循环里的XPath路径要填写相对路径，也可以用下面的形式（其中的。/代表当前路径）。



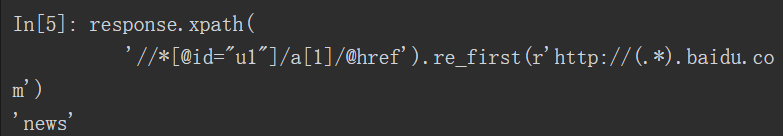
#2 结合正则表达式使用选择器

Scrapy的选择器有一个re()方法，用来通过正则表达式提取数据。不同于使用xpath()或者css()方法，re()方法返回unicode字符串的列表。因此，无法构造嵌套式的re()调用。

还是用上面百度的例子，现在想提取“新闻”栏目网址中的英文名称，可编写如下代码。



另外还有一个糅合了extract\_first()与re()的函数re\_first()。使用该函数可以提取第一个匹配到的字符串。



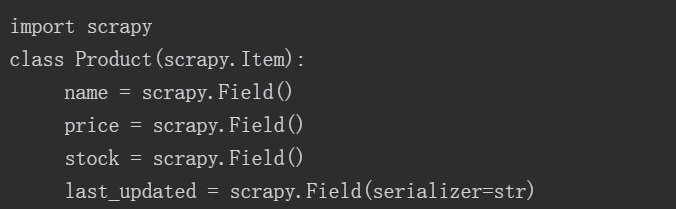
1. Scrapy数据容器

为了方便存储结构性数据，Scrapy提供了Item类。Item对象是一种简单的容器，保存了爬取到的数据。其提供了类似于字典（dictionary-like）的API，以及用于声明可用字段的简单语法。

许多Scrapy组件使用了Item提供的额外信息，将数据导出的exporter类根据Item声明的字段来导出数据，序列化可以通过Item字段的元数据（metadata）来定义等，因此在编写爬虫时，首先要声明Item。

#1 声明Item

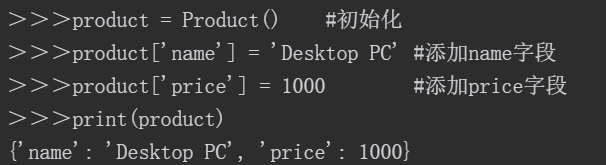
正如上一节百度爬虫例子中演示的那样，Item使用简单的class定义语法及Field对象来声明，示例如下。



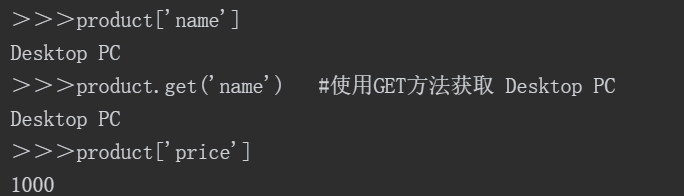
#2 Item字段

Field对象指明了每个字段的元数据（metadata），例如上面的例子中，last\_updated中指明了该字段的序列化函数。Item的API和dict API非常相似，使用起来就像Python字典。

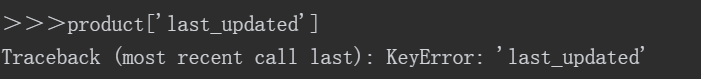
@1 创建item



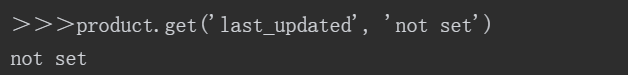
@2 获取字段的值



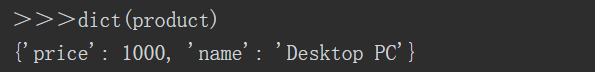
@3 获取不存在的key会报错



@4 可以使用类似字典的GET方法，避免报错



@5 根据item创建字典（dict）



1. 本章小结及要求

本章简单介绍了爬虫框架的概念、Scrapy爬虫框架的安装、Scrapy爬虫目录，并使用Scrapy爬虫框架编写了一个简单的爬虫，最后还介绍了Scrapy的常用命令行工具，深入地讲解了Scrapy选择器和Item的使用方法，这些内容参考了Scrapy官方文档，都是编写爬虫过程中经常使用的基础知识。本章要求读者安装好Scrapy爬虫框架，并熟悉Scrapy的常用命令行工具，学会定义爬虫数据容器Item及使用Scrapy选择器。

**第6章 认识和应对反爬虫**

6.1 常用的网站反爬虫策略及应对措施

前面已经多次提到反爬虫这个概念，这一节介绍常用的网站反爬虫策略，以便有针对性地采取一些措施，防止爬虫被网站服务器识别和禁止。

* + 1. 常用的网站反爬虫策略

1. 通过Headers反爬虫

通过识别用户请求的Headers反爬虫是最常用的网站服务器反爬虫策略。很多网站都会对HTTP请求头部的User Agent进行检测，还有一部分网站会对Referer进行检测（一些资源网站的防盗链就是检测Referer）。

1. 基于用户行为反爬虫

一种常用的反爬虫策略是通过检测用户行为来判断请求是否来自爬虫。例如同一IP地址短时间内多次访问，或者同一账户短时间内多次进行相同操作，都有可能使网站采取反爬虫措施。

1. 采用动态页面反爬虫

有一部分网站的页面与静态页面不同，需要爬取的数据是通过JavaScript生成的，也就是通过AJAX请求得到数据，这样就会对爬虫的爬取造成一些困难。

以上就是网站常用的反爬虫策略，另外，还有少数网站会使用Cookies反爬虫。登录或者下载时需要填写验证码这种措施，也可以被归类为反爬虫的一种策略。

* + 1. 应对网站反爬虫的措施

针对上面分析的几种常用的服务器反爬虫策略，可以考虑以下应对措施。

1. 为爬虫设置Headers

如果遇到通过监测Headers反爬虫的服务器，可以直接在请求中添加User Agent等头部信息。前面的代码曾经使用过这种方法，就是将浏览器的User Agent复制到爬虫的Headers中，或者将Referer值修改为目标网站域名等。对于检测Headers的反爬虫，在爬虫中修改或者添加Headers就能很好地绕过，这是最常用的应对反爬虫措施。一般来说，在爬取网站时都需要添加Headers，模拟成真实的浏览器。

1. 使用IP代理或加大请求间隔时间

针对网站监测IP访问的情况，可以使用IP代理或加大请求间隔时间解决。可以使用代理IP来发起请求，每请求几次更换一个IP，这样就能很容易地绕过检测用户浏览行为的反爬虫策略。下一节将为读者具体演示如何在Requests中使用代理IP。

在需要登录才能爬取的时候，针对检测同一账户浏览行为等反爬虫方式，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次→退出登录→重新登录→继续请求的方法，绕过同一账号短时间内不能多次进行相同请求的限制。

1. 使用Selenium框架

对于动态页面，可以首先用浏览器对网络请求进行分析。如果能够找到AJAX请求，分析出具体参数和响应的具体含义，就可以直接利用Requests模拟AJAX请求，对返回的响应进行分析，得到需要的数据。

有时候有些网站会把AJAX请求的参数全部加密，用户根本没办法构造自己所需要的请求。这时可以使用Selenium框架调用Chrome浏览器或者PhantomJS浏览器，利用浏览器执行JS来模拟人的操作和触发页面中的JS脚本。第7章将详细讲解使用Selenium框架做网站爬虫的方法。

以上就是常用的网站反爬虫策略及其应对措施，当爬虫遭遇网站反爬虫禁止的时候，应该仔细分析它的反爬虫策略，从而采取针对性的措施。还有一点值得读者参考：一般网站的移动页面反爬虫措施较少，并且页面结构简单，特别适合爬取，对于有移动页面的网站，可以优先考虑爬取其移动页面。

* 1. 使用IP代理的方法

6.2.1 Requests中使用代理IP

如前所述，针对网站监测IP访问的反爬虫策略，可以使用代理IP。下面看一下如何在Requests中使用代理IP。

所谓代理IP，就是代理用户去取得网络信息的IP地址。代理IP可以帮助爬虫掩藏真实身份，突破IP访问限制，隐藏爬虫的真实IP，从而避免被网站的反爬虫禁止。

先简单介绍一下Requests中使用IP代理的方法。Requests使用IP代理非常简单，可以通过为任意请求方法提供proxies参数来配置单个请求，下面是Requests文档提供的一个例子。



可以看到，Requests使用代理IP非常方便，只需要构造一个代理IP的字典，然后在发起请求的时候，使用proxies添加这个代理IP的字典即可。

官方文档这个例子其实并不是很明确，我们知道Python字典的key必须是唯一的（如果有重复的key，实际只有最后的value是有效的），如果需要使用多个代理IP，应该如何处理呢？这里要首先明确：proxies字典的key代表代理IP所使用的HTTP请求协议，而字典的value是代理IP地址。当需要使用多个代理IP时，建议读者还是将所有代理IP字典构造成列表，然后从列表中随机选取代理IP。下一节就使用这种方法爬取微信文章。

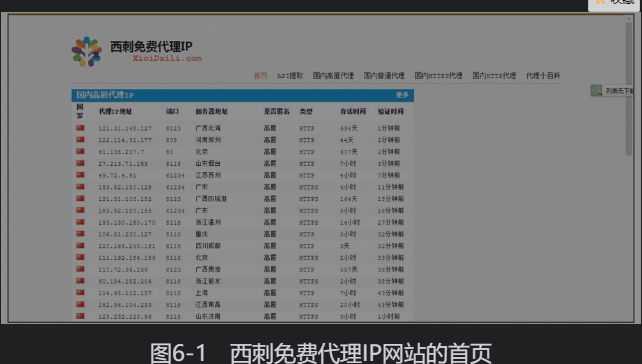
如何获得代理IP呢？一般有如下两个途径。

一是从网上寻找免费的代理IP。有很多提供免费代理IP的网站，可以使用这些网站提供的免费代理IP。使用免费代理的缺点很多，从实用性、稳定性及安全性来考虑，不推荐读者使用免费的代理IP。网上公布的代理IP不一定是可用的，用户很可能在使用过程中发现代理IP不可用或者已失效。

二是购买代理IP，有很多提供这方面服务的网站，读者可以搜索一下。一般购买的代理IP可用性和稳定性都相对较好，缺点是需要一定的成本。在练习写爬虫的时候，可以使用免费代理；在生产环境中，免费代理IP很难满足需求，最好是购买高质量的代理IP。

6.2.2 获取免费代理IP

下一节将使用代理IP爬取微信文章，在爬取之前，需要获取几个可用的代理IP。这里首先从西刺代理网站上寻找几个代理IP，然后检查可用性，为下一节爬取微信文章做好准备。这里要提醒读者，在你看到这里的时候，很可能本书寻找的这几个代理IP已经不可用了，这时可以自己尝试重新寻找可用的免费代理IP。打开西刺免费代理IP网站，图6-1所示为西刺免费代理IP网站的首页。



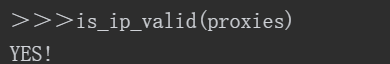
西刺免费代理首页提供了国内高匿代理IP，当用户使用的时候，除了IP地址，还要注意端口和类型，不要弄错。假如现在想使用网页中显示的第一个IP地址作为代理IP，可以按如下代码构造proxies参数。



在使用之前，最好验证一下代理IP的可用性，毕竟很多免费代理IP根本不可用。可以编写一个验证有效性的函数，使用这个代理IP爬取网易首页，然后判断是否成功，从而验证代理IP的可用性。



使用这个函数检测这里找到的代理IP的有效性。



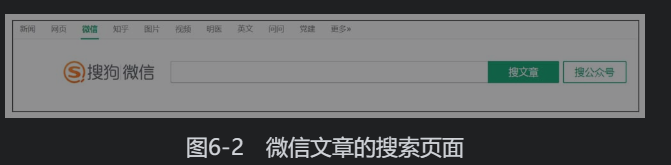
经过测试，这个代理IP是可用的代理IP，下一节就可以使用这个代理IP去爬取微信文章了。以上讲解了Requests使用代理IP的方法，以及如何找到免费的代理IP。在读者实际编写爬虫的时候，如果不想花钱购买稳定的代理IP，可以尝试专门写一个爬虫，爬取免费的代理IP并验证其可用性，然后再应用于实际的爬虫项目中。

* 1. 使用IP代理爬取微信文章

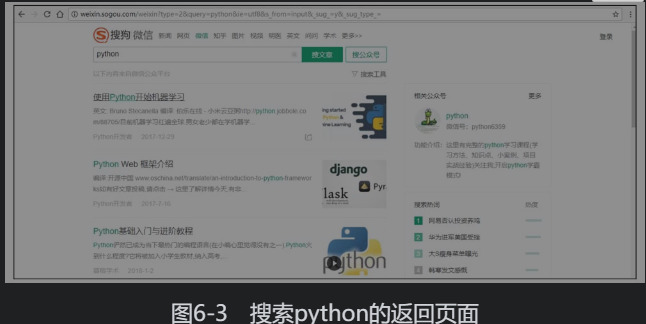
6.3.1 分析微信文章的搜索页面及其URL的构造特点

为了增加趣味性，希望爬虫程序可以按照指定搜索的内容及指定的页数下载对应的微信文章。例如，要搜索Python相关的文章并希望下载前10页，那么爬虫应该可以在运行时接收指令并下载相应页数的文章。下面分析一下微信文章的搜索页面及其URL的构造特点。

打开微信文章搜索的页面，如图6-2所示



页面上部有一个搜索框，搜索关键字python，系统返回图6-3所示的页面，注意看它的URL是如何构造的。

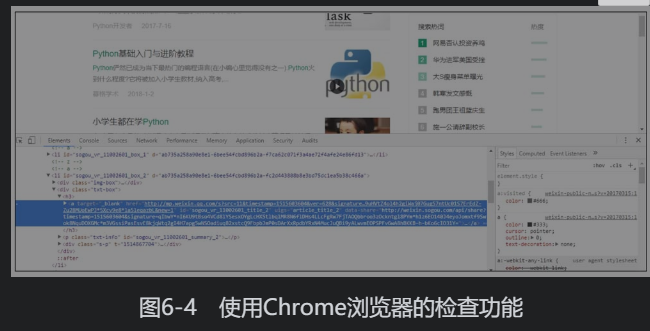


搜索python这个关键字的文章列表页面URL为http://weixin.sogou.com/weixin?type=2&query=python&ie=utf8。很显然，搜索结果页面URL在query=后面部分即是搜索的关键词，也就是说，可以在构造URL时候，在这里使用不同的搜索关键词，来改变搜索内容。

下面看一下翻页的URL是如何构造的。单击下一页，它的URL变为http://weixin.sogou.com/weixin?query=python&type=2&page=2&ie=utf8。也很容易看得出来，这里增加了page这一项，用来指定请求的页码，可以换成page=1，看看第一页是否也满足这个规律。很幸运，第一页满足这一规律。

至此我们已经知道了微信文章搜索的URL构造规律，可以通过更改URL中query项和page项的值来构造不同请求内容的不同页面。

因为下载的目标是微信文章，需要进一步分析搜索结果页面中文章的URL如何提取出来。在搜索结果页面中，右击一篇文章的题目，在弹出的菜单中选择“检查”，使用Chrome浏览器的检查功能，如图6-4所示。



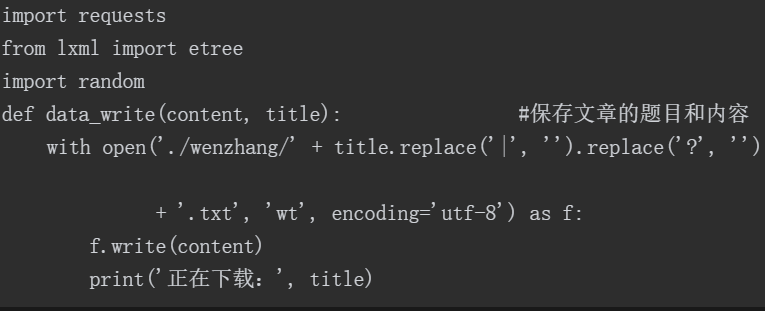
从图中可以看出，文章的URL就是图中蓝色被选中的这行源码的href属性，它的id等于sogou\_vr\_11002601\_title\_2。再查看其他文章题目所在的源码，会发现它们的id有一个共同的特点，那就是都是以sogou\_vr\_11002601\_title\_开头，后面的数字是文章的序号。因此，我们可以通过XPath查询以sogou\_vr\_11002601\_title\_开头的元素，然后取其href属性，就得到了文章URL。最后，直接请求文章URL并下载文章内容就可以了。

以上就是对微信文章搜索页面及其URL构造特点的分析。

6.3.2 编写爬虫代码

根据上面的分析，爬虫应该包含4个部分，也就是说4个函数：下载函数——用于下载网页；保存函数——保存下载的文章；文章URL解析函数——构造并请求搜索结果网址，然后解析出文章URL；文章页内容解析函数。

下面从最简单的开始，导入需要的库，编写保存文章的函数。



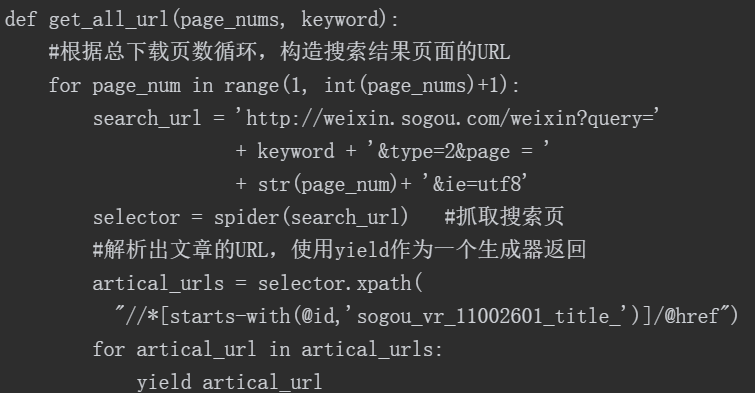
这里准备把每一篇文章保存为一个单独的txt文件，这个保存函数有两个参数——文章内容和文章标题。用文章标题作为保存文件的文件名，因为文件名中不能包含|和？等符号，为防止个别文章题目中包含这些符号而引起错误，把title中的这些符号替换掉。

然后定义一个专门爬取网页、构造选择器的下载函数，这是因为无论是爬取搜索结果页还是下载文章，都要用到爬取网页的动作，所以这里定义一个函数，它的参数就是要爬取的URL，函数返回页面内容的选择器。

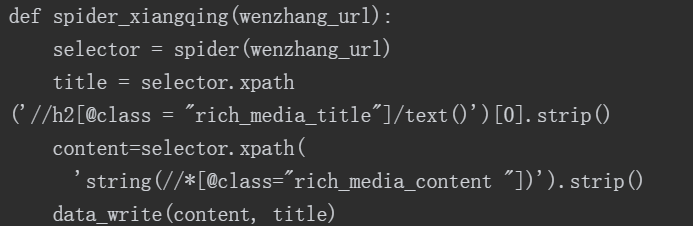


这里为了随机使用不同的代理IP，定义了一个代理IP的列表，使用random.choice方法，每次请求时随机地取出一个IP作为代理。

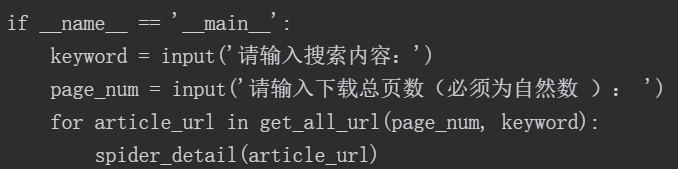
下面编写构造搜索网址及解析所有文章URL的函数get\_all\_url。这个函数有两个参数——搜索关键字和希望下载的总页数。



接下来编写文章内容解析函数。这里作为演示，只简单下载文章的题目和内容。



这里首先获取了文章页面，然后解析出题目和内容，最后使用保存函数保存起来。最后是主程序，主要就是接收用户的输入内容，然后交给上面定义的函数处理。



到这里就完成了整个爬虫的代码编写。为了防止被搜狗反爬虫禁止，这里使用了模拟浏览器的User Agent和随机的代理IP。为了演示，这里只用了4个代理IP。

parse(self, response)

如果Task\_urls非空下发指定贴吧

Task\_urls = [  
 "https://tieba.baidu.com/f?kw=聊天&ie=utf-8&pn=0",  
 "https://tieba.baidu.com/f?kw=李毅&ie=utf-8&pn=0",  
]

yield scrapy.Request(url, callback=self.get\_page\_task\_response, priority=4)

获取所有的下一页

get\_page\_task\_response(self, response):

yield scrapy.Request(url=task\_url, callback=self.get\_info\_page\_response, priority=6)