**第1章 Requests库入门**

The Website is API...

未来所有信息通过Website提供，Website对于爬虫而言是API。

[Requests中文官网](http://docs.python-requests.org/zh_CN/latest/user/quickstart.html)

**🕮1.1 requests库安装**

|  |
| --- |
| pip install requests |

|  |
| --- |
| import requests  headers = {"User-Agent": "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;Trident/5.0;"}  url = 'https://www.baidu.com'  res = requests.get(url, *headers*=headers)  print(res.status\_code) # 200  res.encoding = 'utf-8'  print(res.text) |

**Requests库7个主要方法**

|  |  |
| --- | --- |
| **方法** | **说明** |
| request() | 构造一个请求，支撑以下各方法的基础方法 |
| get() | 获取HTML页面的主要方法，对应于HTTP的GET |
| head() | 获取HTML页面头信息的方法，对应于HTTP的HEAD |
| post() | 向HTML页面提交POST请求的方法，对应于HTTP的POST |
| put() | 提交PUT请求，对应于PUT |
| patch() | 提交局部修改请求，对应于PATCH |
| delete() | 提交删除请求，对应于DELETE |

**🕮1.2 requests.get()方法**

|  |
| --- |
| *def* get(*url*, *params*=None, \*\**kwargs*):  *r*"""Sends a GET request.  :param url: URL for the new :class:`Request` object.  :param params: (optional) Dictionary or bytes to be sent in the query string for the :class:`Request`.  :param \\*\\*kwargs: Optional arguments that ``request`` takes.  :return: :class:`Response <Response>` object  :rtype: requests.Response  """  kwargs.setdefault('allow\_redirects', True)  return request('get', url, *params*=params, \*\*kwargs) |

参数说明：

|  |
| --- |
| 1) url：要获取页面的URL链接  2) params：可选，URL的额外参数，查询字符串，字典或字节流格式  3) \*\*kwargs：12个控制访问的参数 |

可以看到get()方法又调用了request()方法。

get()方法返回Response对象，包含爬虫返回的内容。Response对象包含服务器返回的所有信息，也包含请求的Request信息

**Response对象常用5个属性**

|  |  |
| --- | --- |
| **属性** | **说明** |
| status\_code | HTTP请求返回状态码 |
| text | HTTP响应内容字符串形式，即url对应页面内容 |
| encoding | 从HTTP header中猜测的响应内容编码方式 |
| apparent\_encoding | 从内容中分析出的响应内容编码方式(备选编码方式) |
| content | HTTP响应内容的二进制形式 |

关于两个编码：

|  |
| --- |
| res = requests.get('https://www.baidu.com')  print(res.status\_code) # 200  print(res.encoding) # ISO-8859-1  print(res.apparent\_encoding) # utf-8 |

encoding：如果header中不存在charset，则认为编码是ISO-8859-1

apparent\_encoding：根据网页内容分析出的编码方式；原则上比encoding更准确

**🕮1.3 爬取网页的通用代码框架**

get()发送请求访问网页，但不一定每次都成功，异常处理很重要。

**Requests库6种常用异常**

|  |  |
| --- | --- |
| **异常** | **说明** |
| requests.ConnectionError | 网络连接错误异常，如DNS查询失败、拒绝连接等 |
| HTTPError | HTTP错误异常 |
| URLRequired | URL缺失异常 |
| TooManyRedirects | 超过最大重定向次数，产生重定向异常 |
| ConnectTimeout | 连接远程服务器超时异常 |
| Timeout | 请求URL超时(发送请求到获得响应整个过程超时) |

Response对象的raise\_for\_status()方法：

如果状态码不是200，产生异常requests.HTTPError

res.raise\_for\_status()在方法内部判断res.status\_code是否等于200，不需要增加额外的if语句，该方法便于利用try‐except进行异常处理。

将get()发送请求返回HTML响应封装成函数：

|  |
| --- |
| import requests  headers = {"User-Agent": "..."}  *def* get\_html(*url*):  try:  res = requests.get(url, *headers*=headers, *timeout*=30)  res.raise\_for\_status() # 如果状态码不是200,产生HTTPError异常  res.encoding = res.apparent\_encoding  return res.text  except:  return '发生异常!'  if \_\_name\_\_ == '\_\_main\_\_':  url = 'https://www.baidu.com'  print(get\_html(url)) |

**🕮1.4 HTTP协议及Requests库方法**

HTTP：HyperText Transfer Protocol，超文本传输协议

HTTP是一个基于"请求与响应"模式的、无状态的应用层协议。

HTTP协议采用URL作为定位网络资源的标识。URL格式：

|  |
| --- |
| http(s)://host[:port][path] |

1) host：合法的Internet主机域名或IP地址

2) port：端口号，缺省端口为80

3) path：请求资源的路径

HTTP URL的理解：

URL是通过HTTP协议存取资源的Internet路径，一个URL对应一个数据资源。如同电脑中一个文件的路径，只不过是Internet上的文件。

**HTTP协议对资源的操作**

|  |  |
| --- | --- |
| **方法** | **说明** |
| GET | 请求获取URL位置的资源 |
| HEAD | 请求获取URL位置资源的头部信息(响应消息报告) |
| POST | 请求向URL位置的资源后附加新的数据(用户提交) |
| PUT | 请求向URL位置存储一个资源，覆盖原URL位置的资源 |
| PATCH | 请求局部更新URL位置的资源，即改变该处资源的部分内容 |
| DELETE | 请求删除URL位置存储的资源 |



通过URL和命令管理资源，每次操作独立无状态。

网络通道及服务器是黑盒子，只能看见URL链接和相关操作。

HTTP协议及Requests库方法是一一对应的。

理解PATCH和PUT的区别

|  |
| --- |
| 假设URL位置有一组数据UserInfo，包括UserID、UserName等20个字段  需求：用户修改了UserName，其他不变  1) 采用PATCH，仅向URL提交UserName的局部更新请求；  2) 采用PUT，必须将所有20个字段一并提交到URL，未提交字段被删除。  PATCH的最主要好处：节省网络带宽。 |

**🕮1.5 Requests库主要方法解析**

① requests.request()是其他方法的**基础方法**

request(method, url, \*\*kwargs)

method：请求方式，对应GET/POST/.../OPTIONS(用的少)等7种

url：要获取页面的URL链接

\*\*kwargs：控制访问的参数，共13个

|  |
| --- |
| 1) params：字典或字节序列，作为参数增加到URL中(查询字符串)  2) data：字典、字节序列或文件对象，作为Request的内容(form表单)  3) json：JSON格式的数据，作为Request的内容  4) headers：字典，HTTP定制头  5) cookies：字典或CookieJar，Request中的cookie  6) auth：元组，支持HTTP认证功能  7) files：字典类型，传输文件  8) timeout：设定超时时间，秒为单位。如果指定时间没有返回响应，产生Timeout异常  9) proxies：字典类型，设定访问代理服务器，可以增加登录认证。反反爬虫  10) allow\_redirects：True/False，默认为True，重定向开关  11) stream：True/False，默认为True，获取内容立即下载开关  12) verify：True/False，默认为True，认证SSL证书开关  13) cert：本地SSL证书路径 |

② get(url, params=None, \*\*kwargs)

params就是request()的控制访问参数

\*\*kwargs是除了params剩余12个控制访问参数

get()方法是最常用的方法，因为在HTTP中，向URL提交资源的功能在服务器严格受控，这主要考虑到网络安全的问题。

③ head(url, \*\*kwargs)

④ post(url, data=None, json=None, \*\*kwargs)

⑤ put(url, data=None, \*\*kwargs)

⑥ patch(url, data=None, \*\*kwargs)

⑦ delete(url, \*\*kwargs)

**第2章 网络爬虫的"盗亦有道"**

**🕮2.1 网络爬虫引发的问题**

**网络爬虫的尺寸**

|  |  |  |  |
| --- | --- | --- | --- |
| **分类** | **规模** | **爬取速度** | **常用库** |
| 爬取网页、玩转网页 | 小规模，数据量小 | 不敏感 | Requests库 |
| 爬取网站、爬取系列网站 | 中规模，数据规模较大 | 敏感 | Scrapy库 |
| 爬取全网 | 大规模，搜索引擎 | 关键 | 定制开发 |

网络爬虫引发的问题：

**① 性能骚扰**

Web服务器默认接收人类访问，网络爬虫如同人们生活中的骚扰电话一般。

受限于编写水平和目的，网络爬虫将会为Web服务器带来巨大的资源开销。

**② 法律风险**

服务器上的数据有产权归属。如果网络爬虫获取数据后牟利将带来法律风险。

**③ 隐私泄露**

网络爬虫可能具备突破简单访问控制的能力，获得被保护数据，从而泄露个人隐私。

服务器对网络爬虫的限制：

1) 来源审查：判断User-Agent进行限制

检查来访HTTP协议头的User-Agent，只响应浏览器或友好爬虫的访问

2) 发布公告：Robots协议

告知所有爬虫网站的爬取策略，要求爬虫遵守。

**🕮2.2 Robots协议**

全称Robots Exclusion Standard，网络爬虫排除标准

作用：网站告知网络爬虫哪些页面可以抓取，哪些不行。

形式：在网站根目录下的robots.txt文件。

许多大型网站都有Robots协议，如[京东的Robots协议](https://www.jd.com/robots.txt)：

|  |
| --- |
| User-agent: \* # 所有爬虫  Disallow: /?\* # 不允许  Disallow: /pop/\*.html  Disallow: /pinpai/\*.html?\*  User-agent: EtaoSpider # 黑名单爬虫  Disallow: /  User-agent: HuihuiSpider  Disallow: /  User-agent: GwdangSpider  Disallow: /  User-agent: WochachaSpider  Disallow: / |

Robots协议通过基本语法，告知所有爬虫，内部资源可访问的权限。

其他一些网站的Robots协议：

|  |
| --- |
| https://www.baidu.com/robots.txt  https://news.sina.com.cn/robots.txt  https://www.qq.com/robots.txt  https://news.qq.com/robots.txt  http://www.moe.edu.cn/robots.txt (无robots协议) |

**🕮2.3 Robots协议的遵守方式**

网络爬虫：自动或人工识别robots.txt，再进行内容爬取。

约束性：Robots协议是**建议**但**非约束性**，网络爬虫可以不遵守，但存在法律风险。对于访问量很小或如同人的访问，可以遵守或不遵守；对于访问量大或非商业目的，建议遵守；对于商业目的或爬取全网，必须遵守。

**第3章 Requests库网络爬虫实战**

**🕮3.1 百度搜索关键字提交**

百度搜索的关键词接口：https://www.baidu.com/s?wd=keyword

360搜索的关键词接口：https://www.so.com/s?q=keyword

|  |
| --- |
| import requests, re, time  headers = {"User-Agent": "Mozilla/5.0 ..."}  *def* baidu\_search(*keyword*):  url = 'https://www.baidu.com/s'  dct = {'wd': keyword}  try:  res = requests.get(url, *headers*=headers, *timeout*=30, *params*=dct)  print(res.request.url) # 打印实际请求URL地址  res.raise\_for\_status() # 如果状态码不是200,产生HTTPError异常  res.encoding = res.apparent\_encoding  return res.text  except:  print('爬取失败!')  *def* save\_html\_file(*html*):  if html:  p = *r*'<title>(\S+)</title>'  lst = re.findall(p, html)  filename = lst[0] if len(lst) > 0 else *f*'{time.time()*:.0f*}'  with open(filename+'.html', 'w', *encoding*='utf-8') as f:  f.write(html)  if \_\_name\_\_ == '\_\_main\_\_':  keyword = '星空凛'  save\_html\_file(baidu\_search(keyword)) |

**🕮3.2 网络图片的爬取和存储**

|  |
| --- |
| import requests, os  headers = {"User-Agent": "Mozilla/5.0 ..."}  *def* download(*url*, *base\_dir*=None, *filename*=None):  if base\_dir is None or base\_dir == '':  base\_dir = 'E:/pic/'  if filename is None or filename == '':  filename = url.split('/')[-1]  if not os.path.exists(base\_dir): # 如果目录不存在,创建之  os.makedirs(base\_dir)  file = os.path.join(base\_dir, filename)  if os.path.exists(file): # 如果文件已存在,不下载  print(*f*'{filename}已经存在!')  return  try:  res = requests.get(url, *headers*=headers, *timeout*=30)  res.raise\_for\_status()  except:  print('爬取失败')  else:  with open(file, 'wb') as f:  f.write(res.content)  con\_len = eval(res.headers['Content-Length'])  size = os.path.getsize(file)  if size != con\_len:  os.remove(file)  print('下载文件大小有误!')  else:  size /= 1024  print(*f*'{filename}下载成功! 大小为', *end*='')  print(*f*'{size*:.2f*}KB' if size < 1024 else *f*'{size/1024*:.2f*}MB')  if \_\_name\_\_ == '\_\_main\_\_':  url = 'https://card.niconi.co.ni/asset/assets/image/units/b\_rankup\_41005010.png'  download(url) |

**🕮3.3 IP地址归属地的自动查询**

查询接口：http://m.ip138.com/ip.asp?ip=ip\_address

|  |
| --- |
| from bs4 import BeautifulSoup as BS  *def* query\_ip(*ip*):  url = 'http://m.ip138.com/ip.asp'  dct = {'ip': ip}  try:  res = requests.get(url, *headers*=headers, *timeout*=30, *params*=dct)  res.raise\_for\_status() # 如果状态码不是200,产生HTTPError异常  res.encoding = res.apparent\_encoding  soup = BS(res.text, 'lxml')  ret = soup.select('p.result') # 返回list  return '\n'.join([i.text for i in ret])  except:  print('爬取失败!')  if \_\_name\_\_ == '\_\_main\_\_':  print(query\_ip('202.204.80.112')) |

结果：

|  |
| --- |
| 本站主数据：北京市海淀区 北京理工大学 教育网  参考数据一：北京市 北京理工大学 |

**第4章 BeautifulSoup库入门**

**🕮4.1 BeautifulSoup库的安装**

[BeautifulSoup中文官网](https://www.crummy.com/software/BeautifulSoup/bs4/doc.zh/)，俗称美味汤...

安装：

|  |
| --- |
| pip install beautifulsoup4 |

|  |
| --- |
| from bs4 import BeautifulSoup as BS  if \_\_name\_\_ == '\_\_main\_\_':  test\_url = 'https://python123.io/ws/demo.html'  html = get\_html(test\_url)  soup = BS(html, 'html.parser') # 需要提供解析器  print(soup.prettify()) # 格式 |

**🕮4.2 BeautifulSoup库的基本元素**

BeautifulSoup库是解析、遍历、维护"标签树"的功能库。

HTML文档⇔标签树⇔BeautifulSoup类

BeautifulSoup对应一个HTML/XML文档的全部内容

BeautifulSoup库解析器

|  |  |  |
| --- | --- | --- |
| **解析器** | **使用方法** | **条件** |
| bs4的HTML解析器 | BeautifulSoup(html, 'html.parser') | 安装bs4库 |
| lxml的HTML解析器 | BeautifulSoup(html, 'lxml') | pip install lxml |
| lxml的XML解析器 | BeautifulSoup(html, 'xml') | pip install lxml |
| html5lib的解析器 | BeautifulSoup(html, 'html5lib') | pip install html5lib |

BeautifulSoup类的基本元素

|  |  |
| --- | --- |
| **基本元素** | **说明** |
| Tag | 标签，最基本的信息组织单元，soup.<tag> |
| Name | 标签名字，<tag>.name |
| Attributes | 标签属性，字典，<tag>.attrs |
| NavigableString | 标签内非属性字符串，<>...</>中的字符串，<tag>.string |
| Comment | 标签内字符串的注释部分，一种特殊的NavigableString类型 |

① tag和name

|  |
| --- |
| title = soup.title  print(*type*(title)) # <class 'bs4.element.Tag'>  print(title) # <title>This is a python demo page</title>  print(title.name) # title |

② attrs

|  |
| --- |
| a = soup.a # 如果有多个a标签,只获取第1个  print(a.attrs) # 字典类型  print(a.attrs['class']) # ['py1'],class是多值属性返回list  print(a.attrs['href']) # http://www.icourse163.org/course/BIT-268001 |

③ string

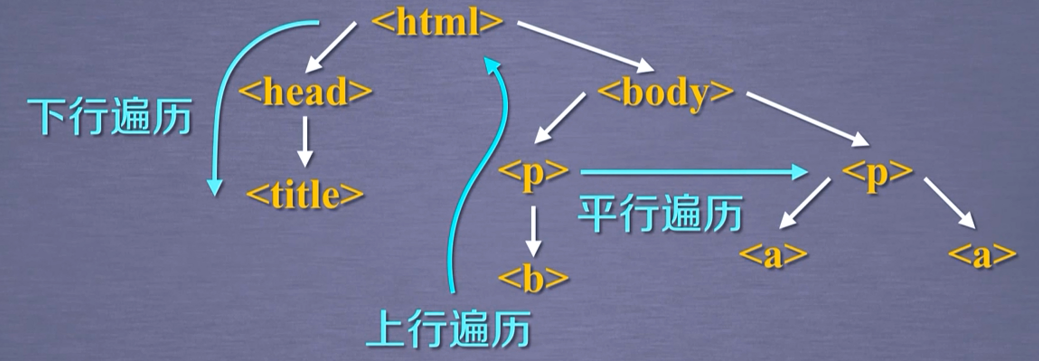
|  |
| --- |
| print(a.string) # Basic Python  p = soup.p  print(p) # <p class="title"><b>The demo python introduces several python courses.</b></p>  print(p.string) # The demo python introduces several python courses.  # NavigableString可以跨越多个层次 |

④ comment (不常用)

|  |
| --- |
| soup = BS('<p><!--rin--></p>', 'html.parser')  p = soup.p.string  print(p) # rin //输出内容不包括注释符号  print(*type*(p)) # <class 'bs4.element.Comment'>  soup = BS('<p>rin</p>', 'html.parser')  p = soup.p.string  print(p) # rin  print(*type*(p)) # <class 'bs4.element.NavigableString'> |

**🕮4.3 基于bs4库的HTML内容遍历方法**

[Demo](https://python123.io/ws/demo.html)的HTML基本格式



**① 标签树的下行遍历**

|  |  |
| --- | --- |
| **属性** | **说明** |
| .contents | 子结点的list |
| .children | 子结点的list\_iterator，与.contents类似，用于遍历子结点 |
| .descendants | 子孙结点的generator，包含所有子孙结点 |

BeautifulSoup类型是标签树的根结点。

标签的子结点包括标签结点和字符串结点，如'\n'。

**② 标签树的上行遍历**

|  |  |
| --- | --- |
| **属性** | **说明** |
| .parent | 结点的父亲标签bs4.element.Tag |
| .parents | 结点的祖先标签的generator |

|  |
| --- |
| soup = BS(open('demo.html'), 'html.parser')  body = soup.body  print(body.parent.parent == soup) # True  print(soup.name) # [document]  print(soup.parent) # None  for p in soup.a.parents: # 遍历第1个a标签所有祖先结点,包括soup  if p is not None:  print(p.name) |

**③ 标签树的平行遍历**

|  |  |
| --- | --- |
| **属性** | **说明** |
| .next\_sibling | 返回下一个平行结点标签 |
| .previous\_sibling | 返回上一个平行结点标签 |
| .next\_siblings | 返回后续所有平行结点标签的generator |
| .previous\_siblings | 返回前续所有平行结点标签的generator |

**注意**：平行遍历发生在同一个父结点下的各结点之间。

|  |
| --- |
| soup = BS(open('demo.html'), 'html.parser')  a = soup.a  print(a.next\_sibling) # ' and ' |

**注意**：尽管标签树中采用标签是形式组织，但标签之间的NavigableString也构成了标签树的结点。

**🕮4.4 基于bs4库的HTML格式输出**

如何让HTML内容更加友好的显示?

soup.prettify()方法为HTML文本<>及其内容增加'\n'

prettify()也可用于标签：<tag>.prettify()

bs4库将任何HTML输入都变成utf-8编码；Python 3.x默认支持编码是utf-8，解析无障碍。

**第5章 信息标记与提取方法**

**🕮5.1 信息标记的三种形式**

**信息的标记**

1) 标记后的信息可形成信息组织结构，增加了信息维度

2) 标记后的信息可用于通信、存储或展示

3) 标记的结构与信息一样具有重要价值

4) 标记后的信息更利于程序理解和运用

HTML是WWW(World Wide Web)的信息组织方式，可以将声音、图像、视频等超文本嵌入到文本中。

**信息标记的三种形式**

① XML (eXtensible Markup Language)

可以认为是能自定义标签、HTML扩展。

② JSON (JavsScript Object Notation)

有类型的键值对：key:value

|  |
| --- |
| {  *"name"*: "hikari",  *"age"*: 25,  *"skills"*: ["python", "javascript", "java"]  } |

对于JavaScript等编程语言，JSON可以直接作为程序的一部分。

③ YAML (YAML Ain’t Markup Language)

无类型键值对 key:value

|  |
| --- |
| info: # 缩进表示所属关系  name: hikari  age: 25  skills: # - 表示并列关系  - python  - javascript  - java  about: | # | 表达整块数据  非常长的一段话  address:  country: china  city: nj |

**🕮5.2 三种信息标记形式的比较**

① XML

|  |
| --- |
| <person>  <firstName>Tian</firstName>  <lastName>Song</lastName>  <address>  <streetAddr>中关村南大街5号</streetAddr>  <city>北京市</city>  <zipcode>100081</zipcode>  </address>  <prof>Computer System</prof><prof>Security</prof>  </person> |

使用标签并嵌套，文本中有效信息所占比例不高。

② JSON

|  |
| --- |
| {  *"firstName"*: "Tian",  *"lastName"*: "Song",  *"address"*: {  *"streetAddr"*: "中关村南大街5号",  *"city"*: "北京市",  *"zipcode"*: "100081"  },  *"prof"*: ["Computer System", "Security"]  } |

③ YAML

|  |
| --- |
| firstName : Tian  lastName : Song  address :  streetAddr : 中关村南大街5号  city : 北京市  zipcode : 100081  prof :  ‐Computer System  ‐Security |

|  |  |  |
| --- | --- | --- |
| XML | 最早的通用信息标记语言，可扩展性好，但繁琐 | Internet上的信息交互与传递 |
| JSON | 信息有类型，适合程序处理(JavaScript)，较XML简洁，但无注释 | 移动应用云端和节点的信息通信 |
| YAML | 信息无类型，文本信息比例最高，有注释可读性好 | 各类系统的配置文件 |

**🕮5.3 信息提取的一般方法**

从标记后的信息中提取所关注的内容

① XML JSON YAML，完整解析信息的标记形式，再提取关键信息，需要标记解析器，例如：bs4库的标签树遍历。

优点：信息解析准确；缺点：提取过程繁琐，速度慢。

② 搜索：无视标记形式，直接搜索关键信息，对信息的文本查找函数即可。

优点：提取过程简洁，速度较快；缺点：提取结果准确性与信息内容相关。

③ 融合方法 (XML JSON YAML 搜索)：

结合形式解析与搜索方法，提取关键信息，需要标记解析器及文本查找函数。

示例：提取HTML中URL链接

搜索所有<a>标签；解析<a>标签格式，提取href的链接内容

|  |
| --- |
| soup = BS(open('demo.html'), 'html.parser')  for link in soup.find\_all('a'):  print(link.get('href')) |

**🕮5.4 基于bs4库的HTML内容查找方法**

|  |
| --- |
| find\_all(*self*, *name*=None, *attrs*={}, *recursive*=True, *text*=None,  *limit*=None, \*\**kwargs*) |

返回匹配的Tag对象的list。

1) name：对标签名称的检索字符串

可以是标签名，标签名的list，正则表达式

|  |
| --- |
| soup = BS(open('demo.html'), 'html.parser')  lst = soup.find\_all('a') # 查找所有a标签  lstab = soup.find\_all(['a', 'b']) # 查找所有a标签和b标签  print(lst)  print(lstab)  # True表示所有子孙标签的list  for tag in soup.find\_all(True):  print(tag.name)  print('-'\*50)  # 正则表达式, 此处返回b开头的标签的list  for tag in soup.find\_all(re.compile('b')):  print(tag.name) |

2) attrs：对标签属性值的检索字符串，可标注属性检索

|  |
| --- |
| lst = soup.find\_all('p', 'course') # 属性值为course的p标签  print(len(lst)) # 1  lst = soup.find\_all(*id*='link1') # id为link1的标签,没有返回[]  print(len(lst)) # 1  # 查找class属性值以'py'开始的标签, class是Python关键字,此处使用class\_  lst = soup.find\_all(*class\_*=re.compile('py'))  print(len(lst)) # 2 |

3) recursive：是否对子孙全部检索，默认True

4) text：<>…</>中字符串区域的检索

|  |
| --- |
| lst = soup.find\_all(*text*='Python')  print(lst) # []  lst = soup.find\_all(*text*=' and ')  print(lst) # [' and ']  print(*type*(lst[0])) # <class 'bs4.element.NavigableString'>  lst = soup.find\_all(*text*=re.compile('python'))  print(lst) # ['This is a python demo page', 'The demo python introduces several python courses.'] |

可以看出，如果不使用正则表达式，需要完整写出NavigableString全部字符串内容；使用正则表达式可以轻松地搜索HTML的字符串内容。

由于find\_all()非常常用，可以简写为：

<tag>(..) ⇔ <tag>.find\_all(..)

soup(..) ⇔ soup.find\_all(..)

find\_all()的7个扩展方法

|  |  |
| --- | --- |
| **方法** | **说明** |
| find() | 搜索只返回一个结果 |
| find\_parents() | 在先辈结点中搜索，返回list |
| find\_parent() | 在先辈结点中搜索，返回一个结果 |
| find\_next\_siblings() | 在后续平行结点中搜索，返回list |
| find\_next\_sibling() | 在后续平行结点中搜索，返回一个结果 |
| find\_previous\_siblings() | 在前面平行结点中搜索，返回list |
| find\_previous\_sibling() | 在前面平行结点中搜索，返回一个结果 |

**第6章 实例1：中国大学排名定向爬虫**

爬取网站：[最好大学网](http://www.zuihaodaxue.cn/zuihaodaxuepaiming2018.html)

技术路线：requests、bs4

定向爬虫：仅对输入URL进行爬取，不扩展爬取

① 从网络上获取大学排名网页内容：get\_html()

② 提取网页内容中信息到合适的数据结构：get\_univ\_rank()

③ 利用数据结构展示并输出结果：print\_univ\_lst()或save2json()

|  |
| --- |
| import requests  import time  from bs4 import BeautifulSoup as BS  from bs4.element import Tag  import json  headers = {"User-Agent": "Mozilla/5.0 ..."}  *def* get\_html(*url*):  try:  res = requests.get(url, *headers*=headers, *timeout*=30)  res.raise\_for\_status() # 如果状态码不是200,产生HTTPError异常  res.encoding = res.apparent\_encoding  return res.text  except:  return ''  *def* get\_univ\_rank(*url*):  html = get\_html(url)  soup = BS(html, 'html.parser')  tbody = soup.find('tbody', *class\_*='hidden\_zhpm')  if tbody:  for tr in tbody.children:  if isinstance(tr, Tag): # 排除NavigableString  tds = tr('td') # 查找tr中的td  rank, univ, province, score = [x.string for x in tds[:4]]  yield {'rank': rank,  'university': univ,  'province': province,  'score': score}  *def* save2json(*gen*, *n*, *filename*=None): # 前n项保存到json文件  n = n if n >= 0 else 0  if filename is None:  filename = *f*'{time.time()*:.0f*}'  ret = []  for i in range(n):  try: # 每次取出一个序列化, ensure\_ascii的作用就是为了False  ret.append(json.dumps(next(gen), *ensure\_ascii*=False))  except:  break # 没货了就结束  with open(filename+'.json', 'w', *encoding*='utf-8') as f:  f.write('[\n')  f.write(',\n'.join(*f*'\t{i}' for i in ret))  f.write('\n]')  if \_\_name\_\_ == '\_\_main\_\_':  url = 'http://www.zuihaodaxue.cn/zuihaodaxuepaiming2018.html'  save2json(get\_univ\_rank(url), 100, 'ChinaUniversityRank') |

json文件(只取前10)：

|  |
| --- |
| [      {*"rank"*: "1", *"university"*: "清华大学", *"province"*: "北京", *"score"*: "95.3"},      {*"rank"*: "2", *"university"*: "北京大学", *"province"*: "北京", *"score"*: "78.6"},      {*"rank"*: "3", *"university"*: "浙江大学", *"province"*: "浙江", *"score"*: "73.9"},      {*"rank"*: "4", *"university"*: "上海交通大学", *"province"*: "上海", *"score"*: "73.1"},      {*"rank"*: "5", *"university"*: "复旦大学", *"province"*: "上海", *"score"*: "66.0"},      {*"rank"*: "6", *"university"*: "中国科学技术大学", *"province"*: "安徽", *"score"*: "61.9"},      {*"rank"*: "7", *"university"*: "南京大学", *"province"*: "江苏", *"score"*: "59.8"},      {*"rank"*: "8", *"university"*: "华中科技大学", *"province"*: "湖北", *"score"*: "59.1"},      {*"rank"*: "9", *"university"*: "中山大学", *"province"*: "广东", *"score"*: "58.6"},      {*"rank"*: "10", *"university"*: "哈尔滨工业大学", *"province"*: "黑龙江", *"score"*: "57.4"}  ] |

格式化字符串format()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| : | <填充> | <对齐> | <宽度> | , | <.精度> | <类型> |
| 引导符号 | 用于填充的单个字符 | <左对齐  >右对齐  ^居中对齐 | 槽的设定输出宽度 | 数字的千位分隔符适用于整数和浮点数 | 浮点数小数部分的精度或字符串的最大输出长度 | 整数类型b,c,d,o,x,X  浮点数类型e,E,f,% |

中文对齐问题：当中文字符宽度不够时，采用西文字符填充；而中西文字符占用宽度不同。解决方法是采用中文字符的空格chr(12288)填充

|  |
| --- |
| *def* print\_univ\_lst(*gen*, *n*): # 打印大学排名  tpl = "{0*:^10*}\t{1*:*{3}^10}\t{2*:^10*}" # 模板字符串; chr(12288)是中文空格  print(tpl.format("排名", "学校名称", "总分", chr(12288)))  n = n if n > 0 else 1  for i in range(n):  try:  item = next(gen)  print(tpl.format(  item['rank'], item['university'], item['score'], chr(12288)))  except:  break |

结果：

|  |
| --- |
| 排名 　　　学校名称　　　 总分  1 　　　清华大学　　　 95.3  2 　　　北京大学　　　 78.6  3 　　　浙江大学　　　 73.9  4 　　上海交通大学　　 73.1  5 　　　复旦大学　　　 66.0  6 　中国科学技术大学　 61.9  7 　　　南京大学　　　 59.8  8 　　华中科技大学　　 59.1  9 　　　中山大学　　　 58.6  10 　哈尔滨工业大学　　 57.4 |

**第7章 re库入门**

**🕮7.1 正则表达式的概念**

正则表达式(regular expression)

1) 是用来简洁表达一组字符串的表达式。

2) 是一种通用的字符串表达框架

3) 是一种针对字符串表达"简洁"和"特征"思想的工具

4) 可以用来判断某字符串的特征归属

正则表达式在文本处理中十分常用：

1) 表达文本类型的特征(病毒、入侵等)

2) 同时查找或替换一组字符串

3) 匹配字符串的全部或部分...

正则表达式主要应用在字符串匹配中。

编译：将符合正则表达式语法的字符串转换成正则表达式特征

|  |
| --- |
| regex = *r*'P(Y|YT|YTH|YTHO)?N'  p = re.compile(regex) |

**🕮7.2 正则表达式的语法**

正则表达式语法由字符和操作符构成

经典正则表达式实例：

|  |  |
| --- | --- |
| ^[A-Za-z]+$ | 大小写字母组成的字符串 |
| ^[A-Za-z0-9]+$ | 字母和数字组成的字符串 |
| ^-?\d+$ | 整数字符串 |
| ^[0-9]\*[1-9][0-9]\*$ | 正整数字符串 |
| [1-9]\d{5} | 6位正整数, 中国的邮编 |
| [\u4e00-\u9fa5] | 中文字符 |
| \d{3}-\d{8}|\d{4}-\d{7} | 国内电话号码, 如010-68913536 |

IPv4地址字符串形式的正则表达式

|  |  |
| --- | --- |
| 0-99 | [1-9]?\d |
| 100-199 | 1\d\d |
| 200-249 | 2[0-4]\d |
| 250-255 | 25[0-5] |
| 0-255 | ([1-9]?\d|1\d\d|2[0-4]\d|25[0-5]) |

IP地址的正则表达式：

(([1-9]?\d|1\d\d|2[0-4]\d|25[0-5])\.){3}([1-9]?\d|1\d\d|2[0-4]\d|25[0-5])

**🕮7.3 re库的基本使用**

re库使用raw string(原生字符串)类型表示正则表达式，raw string是不包含对转义符\再次转义的字符串。

如果使用string类型，更加繁琐，需要转义\，如r'\d'需要写成'\\d'。

re库主要功能函数

|  |  |
| --- | --- |
| **函数** | **说明** |
| re.search() | 搜索匹配的第1个位置，返回match对象 |
| re.match() | 从字符串开始位置起匹配，返回match对象 |
| re.findall() | 搜索字符串，返回全部匹配子串的list |
| re.split() | 将字符串按正则匹配结果分割，返回list |
| re.finditer() | 搜索字符串，返回匹配结果的迭代类型，每个迭代元素是match对象 |
| re.sub() | 替换所有匹配的子串，返回替换后的字符串 |

① search(*pattern*, *string*, *flags*=0)

pattern：正则表达式的字符串或原生字符串表示

string：待匹配字符串

flags：正则表达式使用时的控制标记

|  |  |
| --- | --- |
| **常用标记** | **说明** |
| re.I re.IGNORECASE | 忽略正则表达式的大小写 |
| re.M re.MULTILINE | ^能够将给定字符串的每行当作匹配开始 |
| re.S re.DOTALL | .能够匹配所有字符，默认匹配除换行外的所有字符 |

|  |
| --- |
| s = 'my email is not hikari@qq.com'  p = *r*'\S{3,16}@\w{2,16}\.(com|net|edn\.cn)'  match = re.search(p, s)  if match:  print(match.group(0)) # hikari@qq.com  print(match.group(1)) # com |

② match(*pattern*, *string*, *flags*=0)

从头匹配，用的少：

|  |
| --- |
| match = re.match(p, s)  print(match) # None |

③ findall(*pattern*, *string*, *flags*=0)

//个人最常用的re函数...

④ split(*pattern*, *string*, *maxsplit*=0, *flags*=0)

maxsplit：最大分割数，剩余部分作为最后一个元素输出

⑤ finditer(*pattern*, *string*, *flags*=0)

返回callable\_iterator迭代器，每个元素是match对象

|  |
| --- |
| s = 'honoka eli kotori umi rin maki nozomi hanayo nico'  p = *r*'\b\w{3}\b' # \b单词边界  g = re.finditer(p, s)  print(*type*(g)) # <class 'callable\_iterator'>  for i in g:  print(i.group(0)) |

⑥ sub(*pattern*, *repl*, *string*, *count*=0, *flags*=0)

repl：替换匹配字符串的字符串或callable对象(函数或可调用的对象)

count：匹配的最大替换次数

|  |
| --- |
| s = 'honoka eli kotori umi rin maki nozomi hanayo nico'  p = *r*'\b\w{3}\b' # \b单词边界  s = re.sub(p, *lambda* *x*: x[0].upper(), s) # 将匹配的子串大写  print(s) # honoka ELI kotori UMI RIN maki nozomi hanayo nico |

re库的两种用法

1) 函数式用法：一次性操作

2) 面向对象用法：编译后的多次操作

|  |
| --- |
| p = re.compile(*r*'\b\w{3}\b') # pattern对象  s = 'honoka eli kotori umi rin maki nozomi hanayo nico'  s2 = 'chika riko kanan dia you yohane hanamaru mari ruby'  print(p.findall(s)) # ['eli', 'umi', 'rin']  print(p.findall(s2)) # ['dia', 'you'] |

好处：一次编译，多次匹配，提高效率。

compile()函数：compile(*pattern*, *flags*=0)

**注意**：字符串或原生字符串不是正则表达式，compile()编译后得到的pattern对象才是正则表达式。pattern对象同样具有和上面相同的方法，只不过不需要pattern字符串参数。

**🕮7.4 Match对象**

Match对象是一次匹配的结果，包含匹配的很多信息

Match对象属性

|  |  |
| --- | --- |
| **属性** | **说明** |
| .string | 待匹配的文本 |
| .re | 匹配时使用的patter对象(正则表达式) |
| .pos | 正则表达式搜索文本的开始位置 |
| .endpos | 正则表达式搜索文本的结束位置 |

Match对象方法

|  |  |
| --- | --- |
| **方法** | **说明** |
| .group(0) | 获得匹配后的字符串 |
| .start() | 匹配字符串在原始字符串的开始位置 |
| .end() | 匹配字符串在原始字符串的结束位置 |
| .span() | 返回tuple (.start(), .end()) |

|  |
| --- |
| s = 'my email is not hikari@qq.com'  p = *r*'\S{3,16}@\w{2,16}\.(com|net|edn\.cn)'  match = re.search(p, s)  if match:  print(*type*(match)) # <class '\_sre.SRE\_Match'>  print(match.re) # re.compile('\\S{3,16}@\\w{2,16}\\.(com|net|edn\\.cn)')  print(match.group(0)) # hikari@qq.com  print(match.span()) # (16, 29) |

**🕮7.5 贪婪匹配和最小匹配**

Re库默认采用贪婪匹配，即输出匹配最长的子串：

|  |
| --- |
| html = '<p>rin</p><p>maki</p><p>nozomi</p>'  p = *r*'<p>(\S+)</p>'  print(re.findall(p, html)) # ['rin</p><p>maki</p><p>nozomi'] |

只要长度输出可能不同，都可以通过在操作符后增加?变成最小匹配

如\*?, +?, ??, {m,n}?：

|  |
| --- |
| html = '<p>rin</p><p>maki</p><p>nozomi</p>'  p = *r*'<p>(\S+?)</p>'  print(re.findall(p, html)) # ['rin', 'maki', 'nozomi'] |

**第8章 实例2：淘宝商品比价定向爬虫**

目标：获取淘宝搜索页面的信息，提取其中的商品名称和价格

理解：获取淘宝的搜索接口；翻页的处理

技术路线：requests-re

淘宝搜索页面的[robots协议](https://s.taobao.com/robots.txt)不允许任何爬虫爬取。但是仅仅模拟人的操作，应该是可以的，爬取需要稍加限制。

① 提交商品搜索请求，循环获取页面

② 对于每个页面，提取商品名称和价格信息

③ 将信息输出到屏幕上

// 淘宝现在需要登录才能浏览商品，个人觉得还是得用Selenium + headless browser

**第9章 实例3：股票数据定向爬虫**

目标：获取上交所和深交所所有股票的名称和交易信息

输出：保存到文件中

技术路线：requests-bs4-re

候选网站：[新浪股票](https://finance.sina.com.cn/stock/)、[百度股票](https://gupiao.baidu.com/stock/)、[东方财富网](http://quote.eastmoney.com/stocklist.html)

选取原则：股票信息静态存在于HTML页面中；非js代码生成；没有Robots协议限制。

选取方法：浏览器F12，源代码查看等

选取心态：不要纠结于某个网站，多找信息源尝试

① 从东方财富网获取股票列表

② 根据股票列表逐个到百度股票获取个股信息

③ 将结果存储到文件

//只能爬一个，后面全被百度限制了，使用User-Agent池也不行，难道要用代理IP池?

|  |
| --- |
| import requests  import re  import time  from bs4 import BeautifulSoup as BS  import traceback  import random  USER\_AGENT\_LIST = ["Mozilla/5.0..."] # User-Agent池  headers = {"User-Agent": random.choice(USER\_AGENT\_LIST)}  *def* get\_html(*url*, *try\_max*=3, *encoding*='utf-8'):  while try\_max > 0:  try:  res = requests.get(url, *headers*=headers, *timeout*=30)  res.raise\_for\_status() # 如果状态码不是200,产生HTTPError异常  res.encoding = encoding  return res.text  except:  print(*f*'发生异常! - {try\_max}')  try\_max -= 1  time.sleep(1)  *def* get\_stock\_lst(*url*): # 根据东方财经网返回股票代号的list  html = get\_html(url, *encoding*='GB2312')  p = *r*'<li><a target="\_blank" href="http://quote.eastmoney.com/(s[hz]\d{6}).html">\S+\(\d{6}\)</a></li>'  lst = re.findall(p, html)  return *list*(*set*(lst))  *def* get\_stock\_info(*url*, *stock\_lst*): # 根据每个股票代号,去爬百度股票  cnt, badurl, noneurl = 1, 1, 1  for stock in stock\_lst:  url += stock+'.html'  html = get\_html(url)  if not html:  print(*f*'bad: {badurl}: {url}')  badurl += 1  continue  soup = BS(html, 'html.parser')  info = soup.find('div', *attrs*={'class': 'stock-bets'})  # 从第1次后, 返回的页面被百度限制,info为None  if info is None:  print(*f*'none: {noneurl}')  noneurl += 1  continue  name = info.find\_all(*attrs*={'class', 'bets-name'})[0]  dct = {'股票名称': name.text.split()[0]}  key\_lst = info.find\_all('dt')  val\_lst = info.find\_all('dd')  for i in range(len(key\_lst)):  key, val = key\_lst[i].text, val\_lst[i].text  dct[key] = val.strip()  print(*f*'cnt:{cnt}')  cnt += 1  yield dct  if \_\_name\_\_ == '\_\_main\_\_':  stock\_lst\_url = 'http://quote.eastmoney.com/stocklist.html'  stock\_info\_url = 'https://gupiao.baidu.com/stock/'  stock\_lst = get\_stock\_lst(stock\_lst\_url)  lst = get\_stock\_info(stock\_info\_url, stock\_lst)  save2json(lst, 10, 'baidu\_stock') |