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

🕮 1.1　requests 库安装

```
pip install requests
```

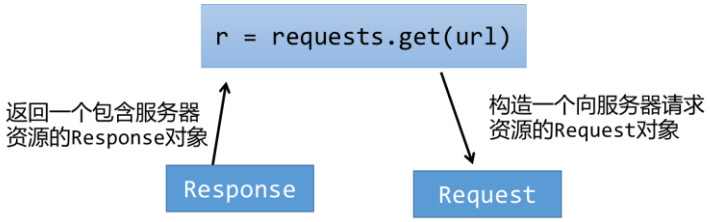
```
import requests

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
```

```python
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：

```
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，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
```

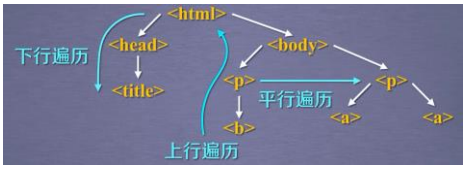
④ comment (不常用)

```
    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 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
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、百度股票、东方财富网
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?

```python
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')
```