引言

- 网络爬虫被应用于许多领域，用于收集不太容易以其他格式获取的数据。
- 你可能是正在撰写新闻报道的记者，也可能是正在抽取新数据集的数据科学家。即使你是临时的开发人员，网络爬虫也是非常有用的工具。

内容提要

- 网络爬虫何时有用
- 网络爬虫是否合法
- 对目标网站进行背景调研
- 编写第一个网络爬虫
- 为爬虫添加一些高级功能
- 使用requests库

1.1 网络爬虫何时有用

- Case 1: 网店商品价格对比
  - 假设我有一个鞋店，并且想要即时了解竞争对手的价格。我可以每天访问他们的网站，与我店铺中的鞋子的价格进行对比。但是，如果我店铺中的鞋子品种繁多，或者是希望能够更加频繁地查看价格变化，就需要话费大量的时间，甚至难以实现。
- Case 2: 等待商品促销
  - 假设我看中了一双鞋，想等到它促销时再购买。我可能需要每天访问这家鞋垫的网站来查看这双鞋是否降价，也许需要等待几个月的时间，我才能如愿盼到这双促销。

1.1 网络爬虫何时有用

- Case 3: 分析新冠疫情动态
  - 假设我们要对新冠疫情动态做数据分析，则可能需要每天都从卫生部门的官网上获取最新发布的相关数据。我们可以手工选取数据、复制数据、保存到本地等。
- 这几个重复性的手工流程，都可以利用网络爬虫技术实现自动化处理。

1.1 网络爬虫何时有用

- 在理想状态下，网络爬虫并不是必需品，每个网站都应该提供API，以结构化的格式共享他们的数据。然而在现实情况中，虽然一些网站提供了这种API，但是他们通常会限制可以抓取的数据，以及访问这些数据的频率。另外，网站开发人员可能会变更、移除或限制其后端API。总之，不能仅仅依赖于API去访问我们所需的在线数据，而是应该学习一些网络爬虫技术的相关知识。

1.2 网络爬虫是否合法

- 网络爬虫使用时法律所允许的内容仍然处于建设中。
- 如果被抓取的数据用于个人用途，且在合理使用版权法的情况下，通常没有问题。
- 但是，如果这些数据会被重新发布，并且爬取行为的攻击性过强导致网站宕机，或者其内容受版权保护，抓取行为违反了其他服务条款，那么就有可能面临法律审判。

1.2 网络爬虫是否合法

- 当抓取的数据是现实生活中真实的公共数据(例如，营业地址、电话清单)时，在遵守合理的使用规则的情况下是允许转载的。但是，如果是原创数据(例如意见和评论或用户隐私数据)，通常就会受到版权限制而不能转载。
- 无论如何，当你抓取某个网站的数据时，请记住自己时该网站的访客，应当约束自己的抓取行为，否则他们可能会封禁你的IP，甚至才采取更进一步的法律行为。这就要求下载请求的速度需要限定在一个合理值之内，并且还需要设定一个专属的用户代理来标识自己的爬虫
- 你还应该设法查看网站的服务条款，确保你所获取的数据不是私有或受版权保护的内容。

1.3 对目标网站进行背景调研

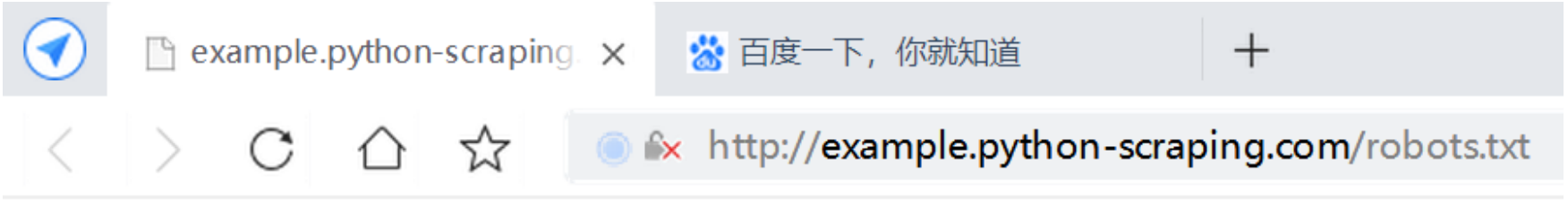
- 在深入讨论爬取一个网站之前，首先需要对目标站点的规模和结构进行一定程度的了解。
- 网站自身的robots.txt和Sitemap文件都可以为我们提供一定的帮助。
- 此外，还有一些能够提供更详细信息的外部工具，比如Google搜索。

1.3.1 检查robots.txt文件

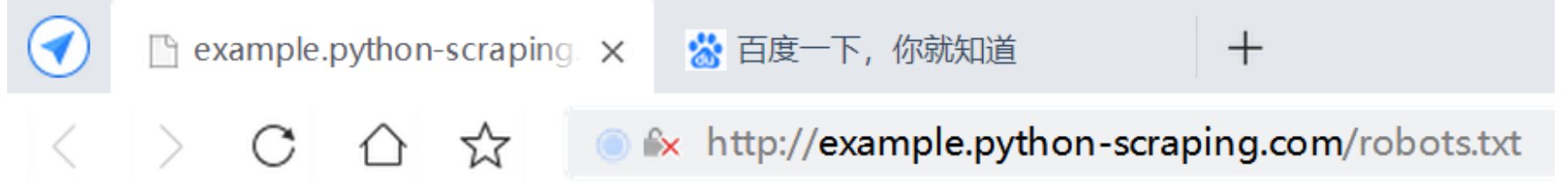
- 大多数网站都会定义robots.txt文件，这样可以让爬虫了解爬取该网站时存在哪些限制。这些限制虽然是仅仅作为建议给出，但是良好的网络公民都应当遵守这些限制。
- 在爬取之前，检查robots.txt文件这一宝贵资源可以将爬虫被封禁的可能性降至最低，而且还能发现和网站结构相关的线索。

1.3.1 检查robots.txt文件

- 下面的代码是示例文件robots.txt中的内容，可以访问http://example.python-scraping.com/robots.txt获取。

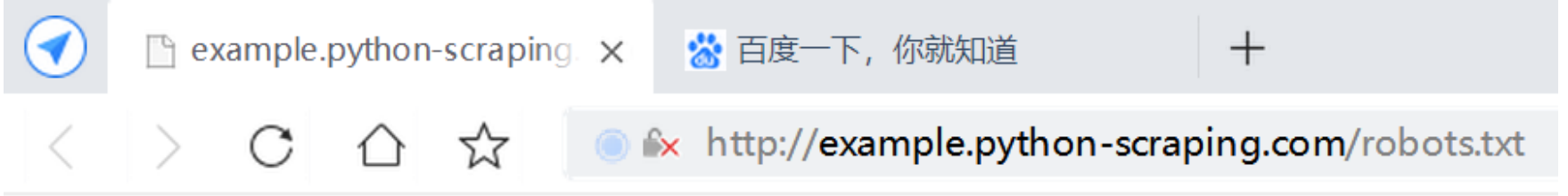



```
# section 1
User-agent: BadCrawler
Disallow: /

# section 2
User-agent: *
Disallow: /trap
Crawl-delay: 5

# section 3
Sitemap: http://example.python-scraping.com/sitemap.xml
```




```
# section 1
User-agent: BadCrawler
Disallow: /

# section 2
User-agent: *
Disallow: /trap
Crawl-delay: 5

# section 3
Sitemap: http://example.python-scraping.com/sitemap.xml
```

- 在section 1中，robots.txt文件禁止用户代理为BadCrawler的爬虫爬取该网站。
- 不过这种写法可能无法到应有的作用，因为恶意爬虫根本不会遵从robots.txt的要求。
- 我们可以让爬虫自动遵守robots.txt的要求。



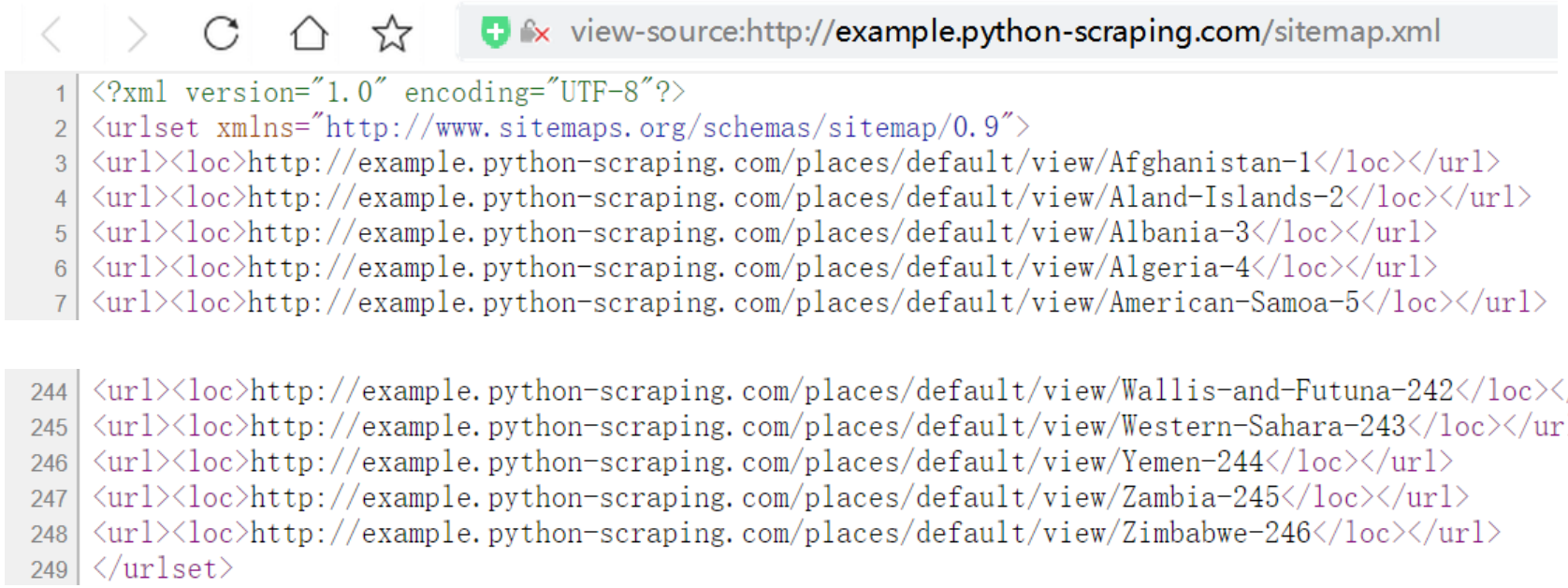

```
# section 1
User-agent: BadCrawler
Disallow: /

# section 2
User-agent: *
Disallow: /trap
Crawl-delay: 5

# section 3
Sitemap: http://example.python-scraping.com/sitemap.xml
```

- 在**section 2**中，无论使用哪种用户代理，都应该在两次下载请求之间给出**5**秒的爬取延迟，我们应遵从该建议以避免服务器过载。
- **/trap**链接用于封不允许访问的链接的恶意爬虫。如果你访问的这个链接，服务器就会封禁你的**IP**一分钟！一个真实的网站可能会对你的**IP**封禁更长时间，甚至是永久封禁。

- 在section 3中，定义了网站地图文件Sitemap.xml的所在位置的URL。

1.3.2 查看网站地图

- 网站提供的Sitemap.xml文件（即网站地图）可以帮助我们定位网站最新的内容，而无需爬取每一个网页。许多网站发布平台都有自动生成网站地图的能力。
- 下面robots.txt文件中的定位到的Sitemap文件的内容。



1.3.2 查看网站地图

- 网站地图提供了所有网页的链接，我们将会使用这些信息。用于创建第一个爬虫。虽然Sitemap文件提供了一种爬取网站的有效方式，但是仍需对其谨慎处理，因为该文件可能存在缺失、过期或不完整的问题。

view-source:http://example.python-scraping.com/sitemap.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
3  <url><loc>http://example.python-scraping.com/places/default/view/Afghanistan-1</loc></url>
4  <url><loc>http://example.python-scraping.com/places/default/view/Aland-Islands-2</loc></url>
5  <url><loc>http://example.python-scraping.com/places/default/view/Albania-3</loc></url>
6  <url><loc>http://example.python-scraping.com/places/default/view/Algeria-4</loc></url>
7  <url><loc>http://example.python-scraping.com/places/default/view/American-Samoa-5</loc></url>

244 <url><loc>http://example.python-scraping.com/places/default/view/Wallis-and-Futuna-242</loc></url>
245 <url><loc>http://example.python-scraping.com/places/default/view/Western-Sahara-243</loc></url>
246 <url><loc>http://example.python-scraping.com/places/default/view/Yemen-244</loc></url>
247 <url><loc>http://example.python-scraping.com/places/default/view/Zambia-245</loc></url>
248 <url><loc>http://example.python-scraping.com/places/default/view/Zimbabwe-246</loc></url>
249 </urlset>
```

1.3.3 估算网站大小

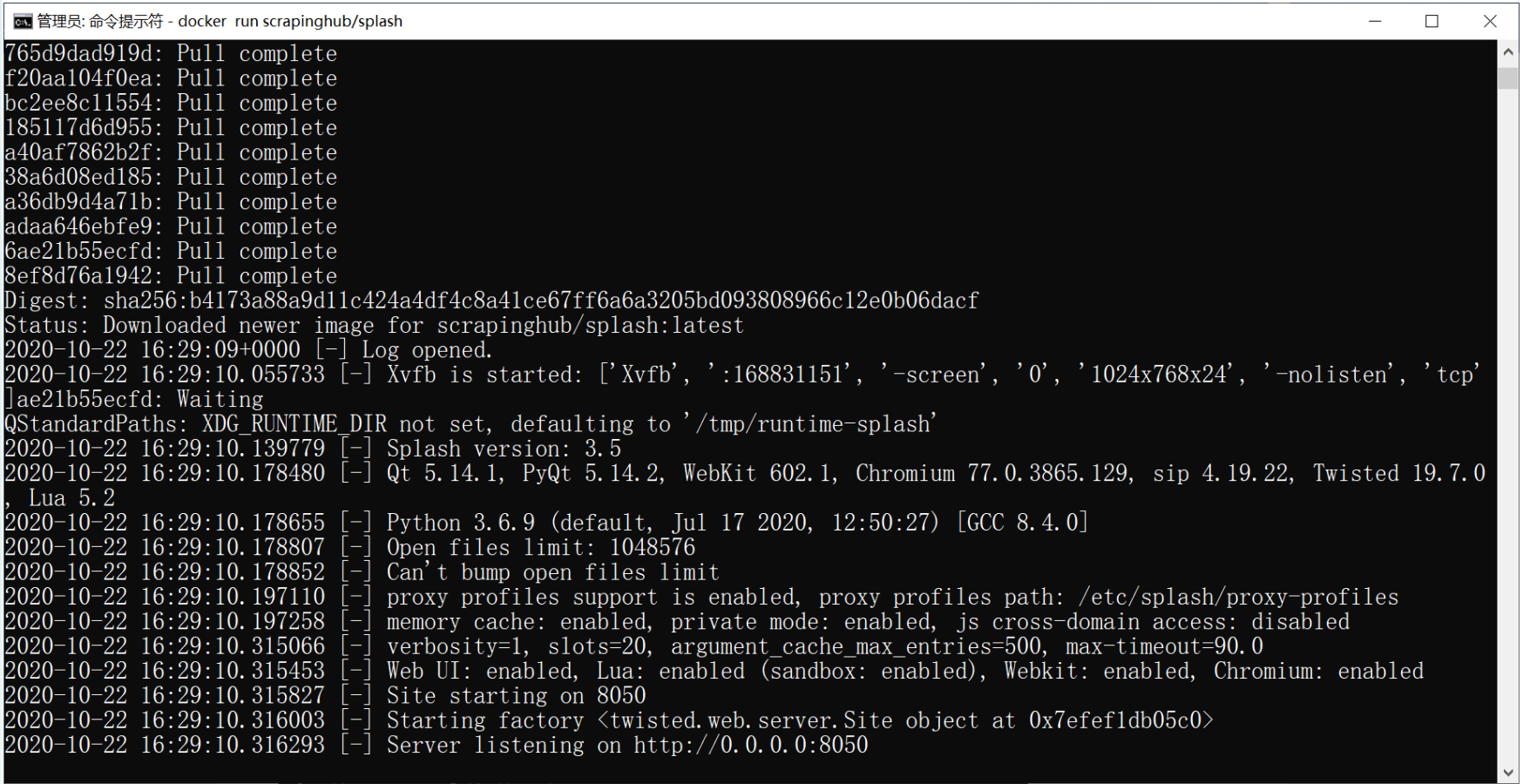
- 目标网站的大小会影响我们如何进行爬取。如果是像我们的示例站点这样只有几百个URL网站，效率并没有那么重要；但如果是拥有数百万个网页的站点，使用串行下载可能需要持续数月才能完成，这时就需要使用分布式下载来解决了。
- 估算网站大小的一个简便方法是检查Google爬虫的结果，因为Google很可能已经爬取过我们感兴趣的网站。可以通过Google搜索的site关键词过滤域名结果，从而获取该信息。
- 在域名后面添加URL路径，可以对结果进行过滤，仅显示网站的某些部分。因为通常情况下，只希望爬取网站中包含有用数据的部分，而不是爬取所有页面。

1.3.4 识别网站所用技术

- 构建网站所使用的技术类型也会对我们如何爬取产生影响。有一个十分有用的工具可以检查网站构建的技术类型——detectem模块，该模块需要Python 3.5+环境以及Docker。如果你还没有安装Docker，可以遵照https://www.docker.com/products/overview中你使用的操作系统所对应的说明操作。
- 当Docker安装好后，你可以在命令行中运行如下命令：
  - C:\Windows\system32>docker pull scrapinghub/splash
  - C:\Windows\system32> pip install detectem
- 上述操作将从ScrapingHub拉取最新的Docker镜像，并通过pip安装该库。

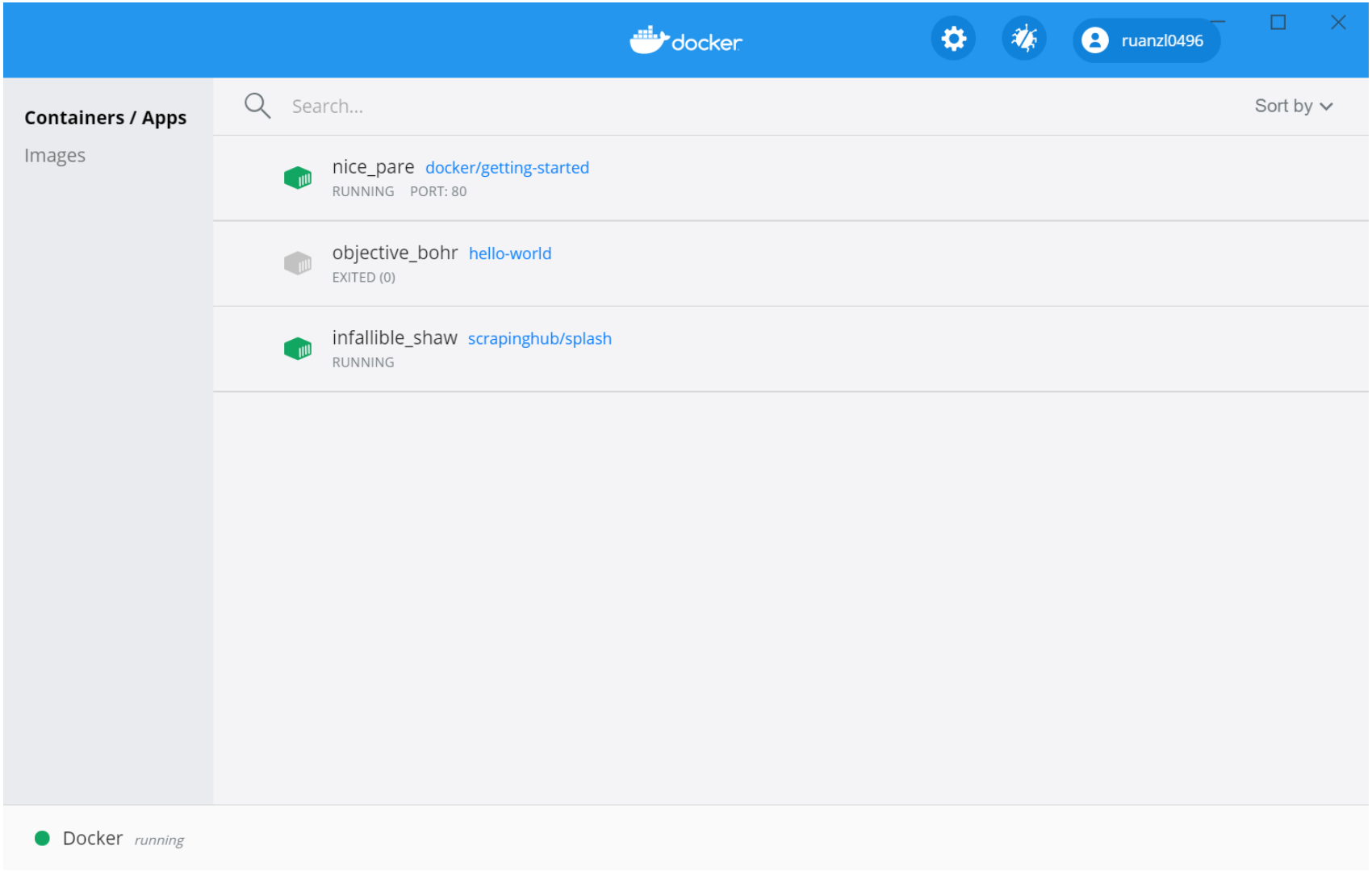
1.3.4 识别网站所用技术

- 拉取并运行ScrapingHub/splash的Docker镜像：
  - C:\Windows\system32>docker pull scrapinghub/splash
  - C:\Windows\system32>docker run scrapinghub/splash



1.3.4 识别网站所用技术

- Docker容器中的scrapinghub/splash



1.3.4 识别网站所用技术

- 安装dectem模块：
  - C:\Windows\system32>pip install detectem

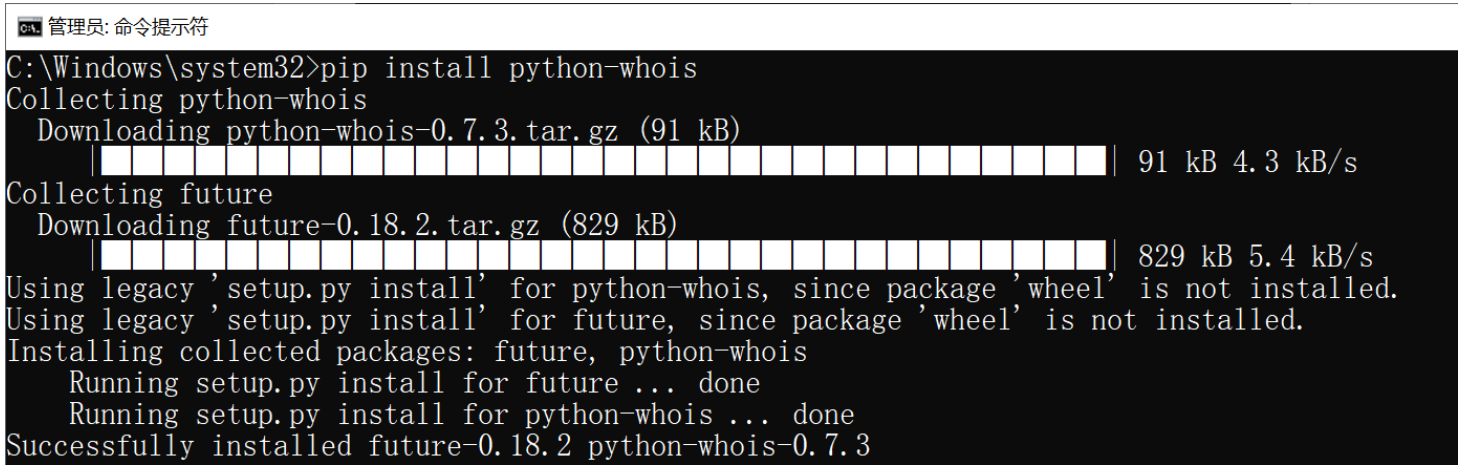
1.3.4 识别网站所用技术

- detectem模块基于许多扩展模块，使用一系列请求和响应，来探测网站使用的技术。它使用了Splash，这是由ScrapingHub开发的一个脚本化浏览器。要想运行该模块，只需在命令行使用det命令即可。
  - C:\Windows\system32>det http://example.python-scraping.com

    输出： [('jquery', '1.11.0')]
- 可以看到示例网站使用了通用的JavaScript库，因此其内容很可能嵌入在HTML当中，相对来说应该比较容易抓取。

1.3.5 寻找网站所有者

- 对于一些网站，我们可能会关心其所有者是谁。比如，如果已知网站的所有者会封禁网络爬虫，那最好把下载速度控制得更加保守一些。
- 为了找到网站的所有者，可以使用WHOIS协议查询域名的注册者是谁。Python中有一个针对该协议的封装库，其文档地址为https://pypi.python.org/pypi/python-whois，可以通过pip进行安装。
  - C:\Windows\system32>pip install python-whois



1.3.5 寻找网站所有者

- 下面是使用该模块对appspot.com这个域名进行WHOIS查询时返回结果的核心部分。

```
>>> import whois
>>> import pprint
>>> pprint.pprint(whois.whois('appspot.com'))
{'address': None,
 'city': None,
 'country': 'US',
 'creation_date': [datetime.datetime(2005, 3, 10, 2, 27, 55), ……],
 'dnssec': 'unsigned',
 'domain_name': ['APPSPOT.COM', 'appspot.com'],
'emails': ['abusecomplaints@markmonitor.com',
            'whoisrequest@markmonitor.com'],
 ......
```

1.3.5 寻找网站所有者

```
.....
 'name_servers': [
                  'NS1.GOOGLE.COM',
                  'NS2.GOOGLE.COM',
                  'NS3.GOOGLE.COM',
                  'NS4.GOOGLE.COM',
                  'ns2.google.com',
                  'ns4.google.com',
                  'ns1.google.com',
                  'ns3.google.com'],
 'org': 'Google LLC',
 'referral_url': None,
 'registrar': 'MarkMonitor, Inc.',
 'state': 'CA',
.....
}
```

- 从结果中可以看，该域名归属于Google，实际上也确实如此。该域名是用于Google App Engine服务的。
- Google会经常会阻断网络爬虫，尽管实际上其自身就是一个网络爬虫业务。
- 爬取该域名时需要十分小心，因为Google经常会阻断抓取其服务过快的IP。

1.4 第一个网络爬虫

- 为了抓取网站，首先需要下载包含有感兴趣数据的网页，该过程一般称为爬取（crawling）。爬取一个网站有很多种方法，而选用哪种方法更加合适，则取决于目标网站的结构。
- 本章中，首先会探讨如何安全地下载网页，然后会介绍如下3种爬取网站的常见方法：
  - 爬取网站地图；
  - 使用数据库ID遍历每个网页；
  - 跟踪网页链接。

1.4.1 抓取与爬取的对比

- 网络抓取通常针对特定网站，并在这些站点上获取指定信息。网络抓取用于访问这些特定的页面，如果站点发生变化或者站点中的信息位置发生变化的话，则需要进行修改。例如，你可能想要通过网络抓取查看你喜欢的当地餐厅的每日特色菜，为了实现该目的，你需要抓取其网站中日常更新该信息的部分。
- 网络爬取通常是以通用的方式构建的，其目标是一系列顶级域名的网站或是整个网络。爬取网络的任务就是从许多不同的站点或页面中获取小而通用的信息，然后跟踪链接到其他页面中。

1.4.2 下载网页

- 要想抓取网页，首先需要将其下载下来。
- 示例：脚本使用Python的urllib库的request模块下载URL。

```
import urllib.request as request

def download(url: str):
    """
    下载网页。

    :param url: URL地址
    :return: 网页的HTML代码（字节数组）
    """
    response = request.urlopen(url)  # type: http.client.HTTPResponse
                                     # Base class for buffered IO objects
    return response.read() #  这里返回的是字节数组

url = "http://www.upc.edu.cn"
print(download(url=url))
```

1.4.2 下载网页

- 要想抓取网页，首先需要将其下载下来。
- 示例：脚本使用Python的urllib库的request模块下载URL

1.4.2 下载网页

- 保存下载页面至文件：字节数组直接写入文件即可。

```
import urllib.request as request

def download(url: str):
    response = request.urlopen(url,
                               timeout=30) # type: http.client.HTTPResponse
                                           # Base class for buffered IO objects
    return response.read() #  这里返回的是字节数组

url = "http://www.upc.edu.cn"
html = download(url=url)

#  保存到文件
filename = 'upc2.html'
with open(file=filename, mode='bw') as file:
    file.write(html)
```

1.4.2 下载网页

- 缓存字节流按UTF-8解码：通过io.TextIOWrapper对象实现，将将缓冲字节流转换为字符流。

```
import urllib.request as request
import io

def download(url: str):
    response = request.urlopen(url,
                               timeout=30) # type: http.client.HTTPResponse
                                           # Base class for buffered IO objects
    # response.read()读取的是原始字节
    textIOWrapper = io.TextIOWrapper(buffer=response, encoding="UTF-8")  #
将缓冲字节流转换为字符流
    html = textIOWrapper.read()  # 读取所有字符
    return html

url = "http://www.upc.edu.cn"
html = download(url=url)
print(html)
```

1.4.2 下载网页

- 缓存字节流按UTF-8解码

Run: demo2_downloadPage_encoding

```
<!DOCTYPE html><html><head>
<title>中国石油大学</title><META Name="keywords" Content="中国石油大学" />




<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
<link href="css/same.css" rel="stylesheet">
<link href="css/my.css" rel="stylesheet">
<link href="css/slick.css" rel="stylesheet">
 <link href="css/base.css" rel="stylesheet">
<link href="css/menu20160330.css" rel="stylesheet">
<script src="js/bdtxk.js"></script><script src="js/slide.js"></script>
<script src="js/comm2.js"></script>
<script src="js/slick.js"></script>
```

4: Run TODO 6: 问题 终端 Python 控制台 事件日志

1.4.2 下载网页

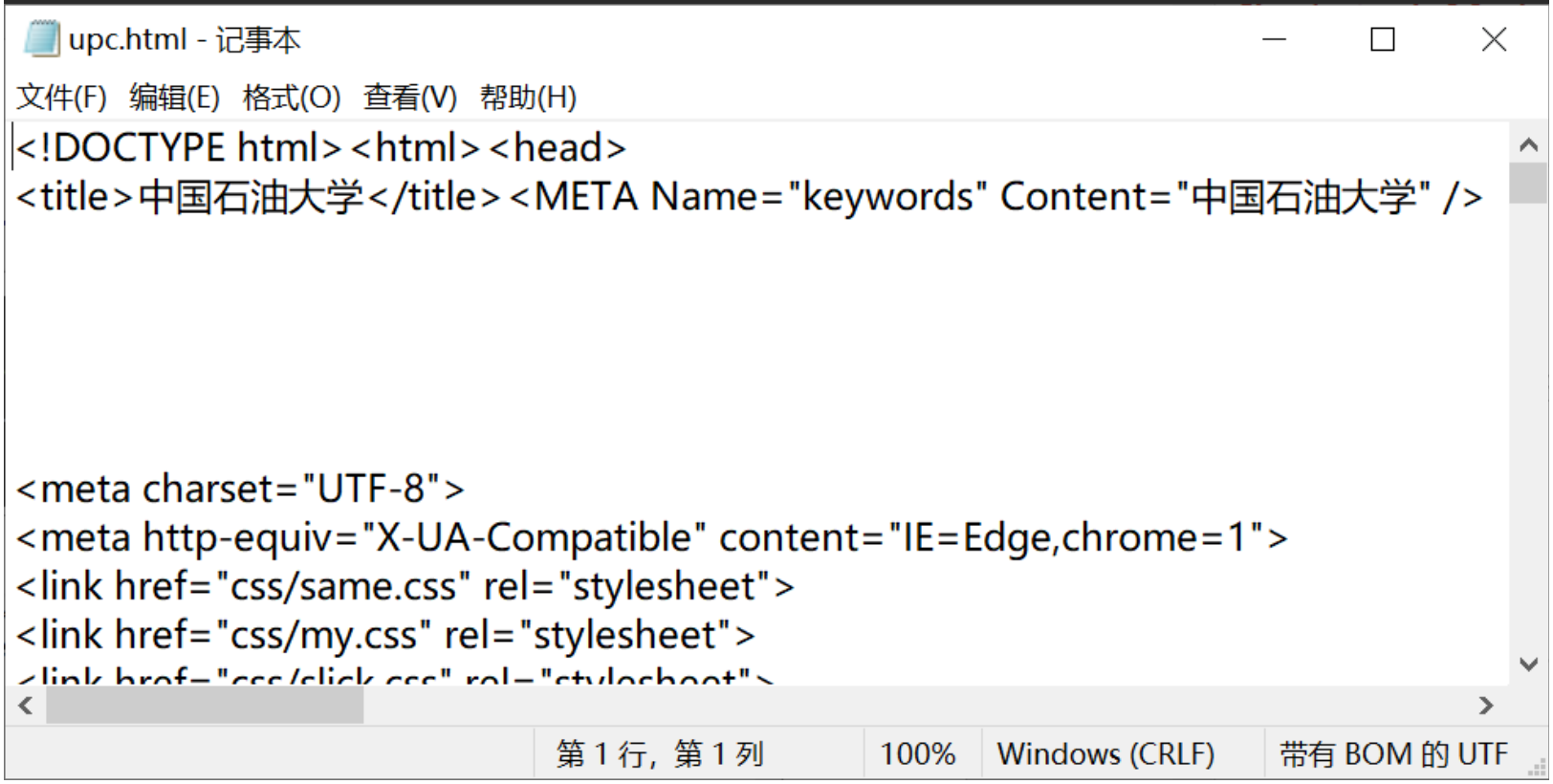
- 缓存字节流按UTF-8解码：也可以从缓冲字节流读取字节，然后对字节流解码。

```
import urllib.request as request

def download(url: str):
    response = request.urlopen(url,
                               timeout=30) # type: http.client.HTTPResponse
                                           # Base class for buffered IO objects
    # 读取字节序列
    byte_array=response.read()  # type: bytearray
    html = byte_array.decode(encoding='UTF-8')  # 按UTF-8字符集，字节序列解码为文本
    return html

url = "http://www.upc.edu.cn"
html = download(url=url)
print(html)
```

1.4.2 下载网页

- 将下载的网页保存到文件(即HTML源代码写入指定文件)

```
# 保存到文件
filename = 'upc.html'
with open(file=filename, mode='tw', encoding='UTF-8') as file:
    file.write(html)
```



1.4.2 下载网页

- 当下载网页时，可能会遇到一些无法控制的错误，比如请求的页面不存在。此时，urllib会抛出异常，然后退出脚本。为安全起见，应给出一个更稳建的版本，可以捕获这些异常。

```
import urllib.request as request
from urllib.error import *
def download(url: str):
    print('Downloading:', url)
    try:
        response = request.urlopen(url, timeout=30)
        byte_array = response.read()  # type: bytearray
        html = byte_array.decode(encoding='UTF-8')
    except(URLError, HTTPError, ContentTooShortError) as e:
        print('Download error:', e.reason)
        html = None
    return html
```

现在，当出现下载或URL错误时，该函数能够捕获到异常，然后返回None。

1.4.2 下载网页

```
urls = ["http://www.upc.edu.cn", "http://www.upc1.edu.cn"]
filenames = ['a.html', 'b.html']

for url, filename in zip(urls, filenames):
    html = download(url=url)
    # 保存到文件
    if html is not None:
        with open(file=filename, mode='tw', encoding='UTF-8') as file:
            file.write(html)
```

```
Run:  demo4_downloadPage_handleException2
Downloading: http://www.upc.edu.cn
Downloading: http://www.upc1.edu.cn
Download error: [Errno 11001] getaddrinfo failed
```

1.4.2 下载网页

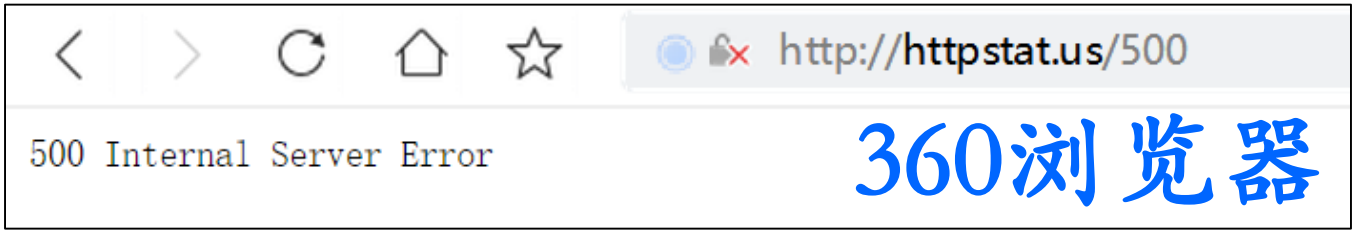
- 重试下载
  - 下载时遇到的错误经常是临时性的，比如服务器过载时返回的503 Service Unavailable错误。对于此类错误，可以在短暂等待后尝试重新下载，因为这个服务器问题现在可能已经解决。不过，不需要对所有错误都尝试重新下载。如果服务器返回的是404 Not Found这种错误，则说明该网页目前并不存在，再次尝试同样的请求一般也不会出现不同的结果。
  - 互联网工程任务组（Internet Engineering Task Force）定义了HTTP错误的完整列表，从中可以了解到4xx错误发生在请求存在问题时，而5xx错误则发生在服务端存在问题时。所以，只需要确保download函数在发生5xx错误时重试下载即可。
  - 下面是支持重试下载功能的download函数新版本代码：

1.4.2 下载网页

```
def download(url: str, num_retries=2):
    print('Downloading:', url)
    try:
        response = request.urlopen(url, timeout=30)
        byte_array = response.read()  # type: bytearray
        html = byte_array.decode(encoding='UTF-8')
    except(URLError, HTTPError, ContentTooShortError) as e:
        print('Download error:', e.reason)
        if hasattr(e, 'code'):
            print('Error code:', e.code)
        html = None
        if num_retries > 0:
            if hasattr(e, 'code') and (500 <= e.code < 600):
                print('Retry to download.')
                html = download(url, num_retries=num_retries - 1)  # 重试下载
    return html
```

1.4.2 下载网页

- 重试下载
  - 现在，当download函数遇到5xx错误码时，将会递归调用函数自身进行重试。此外，该函数还增加了一个参数，用于设定重试下载的次数，其默认值为2次。限制网页下载的尝试次数，是因为服务器错误可能暂时还没有恢复。
  - 想要测试该函数，可以尝试下载http://httpstat.us/500，该网址会始终返回500错误码。




```
url = "http://httpstat.us"
html = download(url=url, num_retries=2)
```

```
Downloading: http://httpstat.us
Download error: Forbidden
Error code: 403
```

1.4.2 下载网页

- 重试下载
  - 有的服务器要求两次下载请求之间要有一定的延迟，从而避免服务器过载。

```
……
import time
def download(url: str, num_retries=2):
    ……
        if num_retries > 0:
            if hasattr(e, 'code') and (500 <= e.code < 600):
                delay=10  # 延迟发送请求的秒数
                print(f'Pause for {delay} seconds.')
                time.sleep(delay)  # 暂停执行若干秒
                print('Retry to download.')
                html = download(url, num_retries=num_retries - 1)  # 重试下载
        return html
```

1.4.2 下载网页

- 设置用户代理（User Agent, UA）
  - User Agent中文名为用户代理，是Http协议中的一部分，属于首部字段的组成部分。
  - 它是一个特殊字符串头，是一种向访问网站提供你所使用的浏览器类型及版本、操作系统及版本、浏览器内核等信息的标识。
  - 通过这个标识，用户所访问的网站可以显示不同的排版从而为用户提供更好的体验或者进行信息统计。例如用手机访问谷歌和电脑访问是不一样的，这些是谷歌根据访问者的UA来判断的。

1.4.2 下载网页

- 设置用户代理（User Agent, UA）
  - 默认情况下，urllib使用Python-urllib/3.x作为用户代理下载网页内容，其中3.x是环境当前所用Python的版本号。如果能使用可辨识的用户代理则更好，这样可以避免网络爬虫碰到一些问题。此外，也许是因为曾经历过质量不佳的Python网络爬虫造成的服务器过载，一些网站还会封禁这个默认的用户代理。
  - 如果使用默认用户代理，一些服务器可能会禁止其下载请求，因此，为了使下载网站更加可靠，需要控制用户代理的设定。
  - 为此，在发送http请求（即执行urlopen( )）之前，需要先设置请求首部（header），其中包括了用户代理。
  - 下面的代码对download函数进行了修改，设定了一个默认的用户代理Porter（搬运工、小红帽）。

```python
def download(url: str, ua='Porter', num_retries=2):
    print('Downloading:', url)
    rq = request.Request(url)  # http请求
    # 指定用户代理
    rq.add_header(key='User-agent', val=ua)
    try:
        response = request.urlopen(rq, timeout=30)
        byte_array = response.read()  # type: bytearray
        html = byte_array.decode(encoding='UTF-8')
    except(URLError, HTTPError, ContentTooShortError) as e:
        print('Download error:', e.reason)
        if hasattr(e, 'code'):
            print('Error code:', e.code)
        html = None
        if num_retries > 0:
            if hasattr(e, 'code') and (500 <= e.code < 600):
                print('Retry to download.')
                html = download(url, ua=ua, num_retries=num_retries - 1)
    return html
```

也可以这样指定用户代理

```python
headers = {'User-agent': ua}
rq = request.Request(url, headers=headers)
```

1.4.2 下载网页

- 设置用户代理（User-agent）

```
url = "http://httpstat.us/500"
html = download(url=url, num_retries=2)

if html is not None:
    print(html)
```

```
Downloading: http://httpstat.us/500
Download error: Internal Server Error
Error code: 500
Downloading: http://httpstat.us/500
Download error: Internal Server Error
Error code: 500
Downloading: http://httpstat.us/500
Download error: Internal Server Error
Error code: 500
```

- 从返回结果可以看出，设置用户代理后，download函数的行为和预期一致，先尝试下载网页，在接收到500错误后，又进行了两次重试才放弃。

1.4.2 下载网页

- 设置用户代理（User Agent, UA）
  - UA也可以是常见网页浏览器的用户代理，爬虫程序可以伪装成浏览器用户访问网址。常见代理的信息如下：

PC端windows系统：

- **Safari**

Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50

- **MS Edge**

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36 Edg/110.0.1587.41

- **Google Chrome**

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36

- **Firefox**

Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1

- **360浏览器**

Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; 360SE)

1.4.2 下载网页

- 设置用户代理（User Agent, UA）
  - UA也可以是常见网页浏览器的用户代理，爬虫程序可以伪装成浏览器用户访问网址。常见用户代理的信息如下：

移动端：

- **safari iOS 4.33 – iPhone**

Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5

- **Android Opera Mobile**

Opera/9.80 (Android 2.3.4; Linux; Opera Mobi/build-1107180945; U; en-GB) Presto/2.8.149 Version/11.10

- **Windows Phone Mango**

Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0; HTC; Titan)

- **UC Opera**

Mozilla/4.0 (compatible; MSIE 6.0;) Opera/UCWEB7.0.2.37/28/999

1.4.2 下载网页

- 设置用户代理（User Agent, UA）
  - 示例1：伪装成PC端 Firefox 4.0.1 – Windows 浏览器用户访问网址

```
url = "http://www.upc.edu.cn"
ua = 'Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101
Firefox/4.0.1'
html = download(url=url, ua=ua,num_retries=2)
```

  - 示例2：伪装成移动端UC标准浏览器用户访问网址

```
url = "http://www.upc.edu.cn"
ua = 'NOKIA5700/ UCWEB7.0.2.37/28/999'
html = download(url=url, ua=ua,num_retries=2)
```

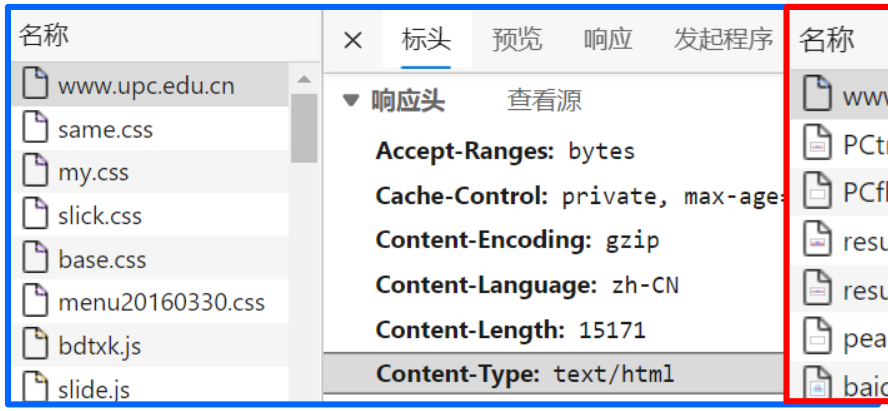
注意，伪装成不同的浏览器，返回的html代码可能是有所不同的，这与网站提供的功能有关，例如这两个示例返回的结果就有所不同。

1.4.3 尝试获取网页字符集

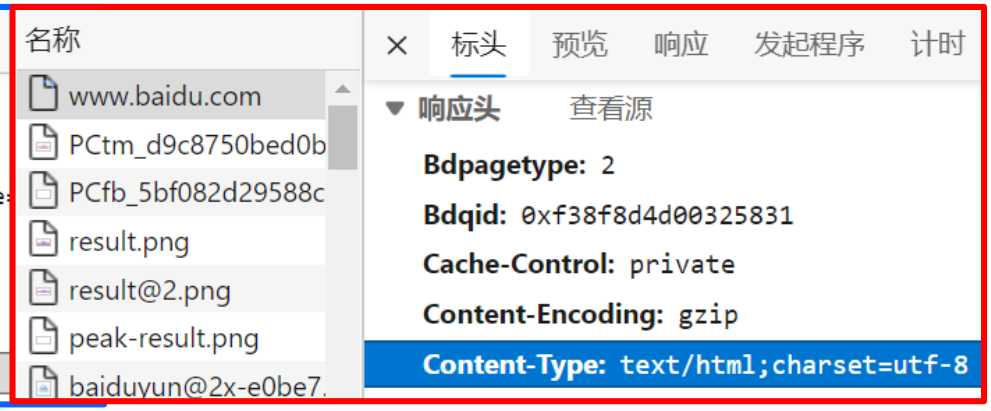
- 前面都假定网页的字符集为UTF-8，更恰当的做法是尝试从httpResponse的headers中获取网页字符集

```
def download(url: str, ua='Porter', num_retries=2, charset='UTF-8'):
    print('Downloading:', url)
    rq = request.Request(url)  # http请求
    rq.add_header(key='User-agent', val=ua)
    try:
        response = request.urlopen(rq , timeout=30)
        cs = response.headers.get_content_charset()  # 获取网页字符集
        if not cs:  # 从headers中未获取到字符集
            print('使用默认字符集：', charset)
            cs = charset
        else:
            print('网页字符集为：', cs)
        byte_array = response.read()  # type: bytearray
        html = byte_array.decode(encoding=cs)  # 将字节序列解码为文本
    except(URLError, HTTPError, ContentTooShortError) as e:
        ……
    return html
```

1.4.3 尝试获取网页字符集



响应头中不包含字符集



响应头中包含字符集

- 正如上面代码中的download方法所示，必须更新字符编码才能利用正则表达式处理网站响应。
- read方法返回字节，而正则表达式期望的则是字符串。

```
re.findall('<loc>(.*?)</loc>', sitemap_xml)
TypeError: cannot use a string pattern on a bytes-like object
```

- 当前的代码依赖于网站维护者在响应头中包含适当的字符编码。若未返回字符编码头部，则设置为默认值UTF-8，并抱有最大希望。
- 当然，如果返回头中的编码不正确，或是编码没有设置并且也不是UTF-8，则会抛出错误。
- 还有一些更复杂的方式用于猜测编码，例如Chardet。

1.4.3 尝试获取网页字符集

- **Chardet: The Universal Character Encoding Detector**

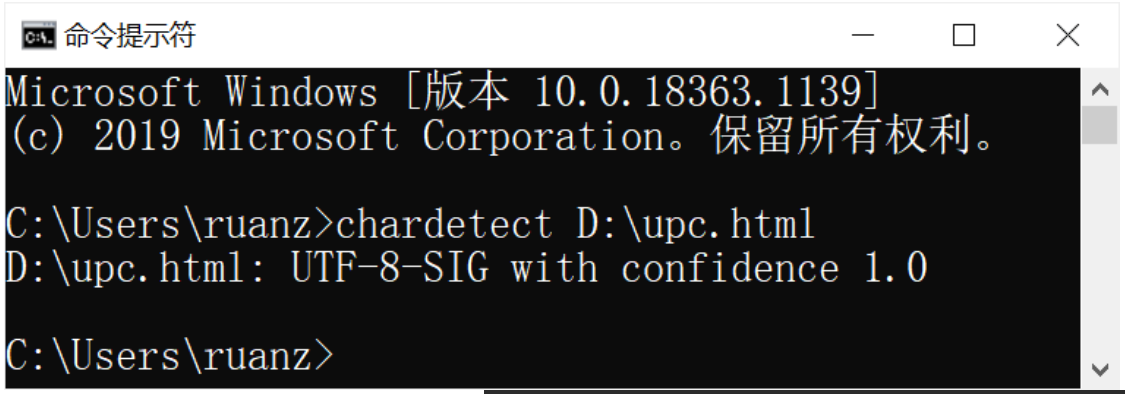
  说明详见 https://pypi.org/project/chardet/

Detects

- ASCII, UTF-8, UTF-16 (2 variants), UTF-32 (4 variants)
- Big5, GB2312, EUC-TW, HZ-GB-2312, ISO-2022-CN (Traditional and Simplified Chinese)
- EUC-JP, SHIFT_JIS, CP932, ISO-2022-JP (Japanese)
- EUC-KR, ISO-2022-KR (Korean)
- KOI8-R, MacCyrillic, IBM855, IBM866, ISO-8859-5, windows-1251 (Cyrillic)
- ISO-8859-5, windows-1251 (Bulgarian)
- ISO-8859-1, windows-1252 (Western European languages)
- ISO-8859-7, windows-1253 (Greek)
- ISO-8859-8, windows-1255 (Visual and Logical Hebrew)
- TIS-620 (Thai)

1.4.3 尝试获取网页字符集

- 或者使用某些字符集检测工具，例如Chardet。
  - 安装：pip install chardet
  - 使用命令行程序chardetect检测

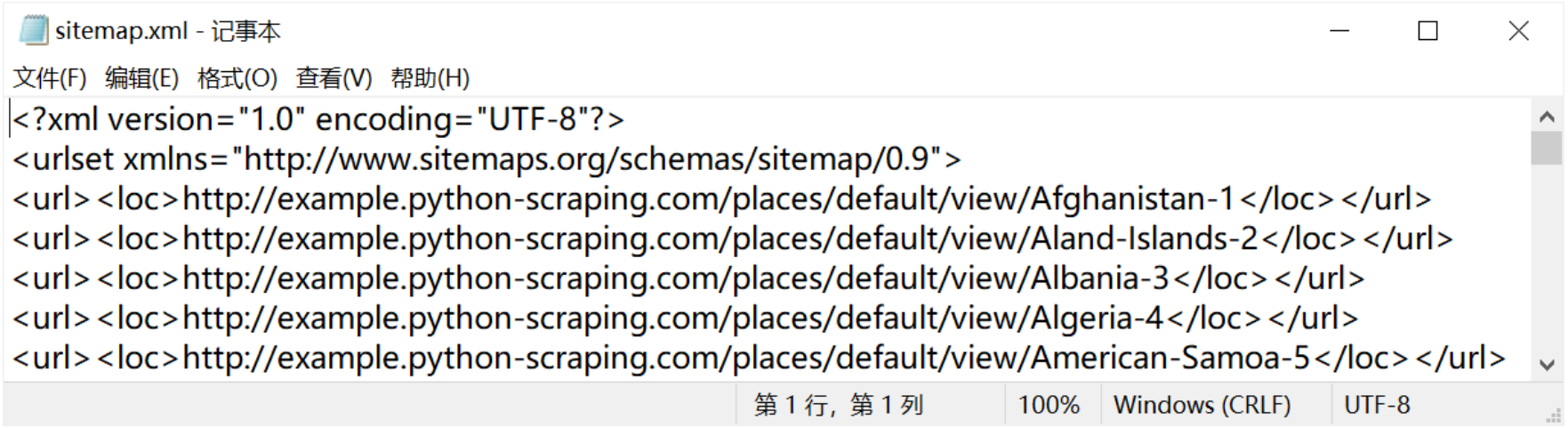


保存页面至文件，则直接保存原始直接即可，而不必关心字符集。

```
def download(url: str):
        response = request.urlopen(url , timeout=30)
        html=response.read()
        return html

html = download(url="http://www.upc.edu.cn")
with open(file='upc.html', mode='bw') as file:
    file.write(html)
```

1.4.4 网站地图爬虫

- 现在使用示例网站robots.txt文件中发现的网站地图来下载所有网页。为了解析网站地图，可以使用一个简单的正则表达式，从<loc>标签中提取出URL。




```
regex = '<loc>(.*?)</loc>'
```

```
……
def crawl_sitemap(url):
    # download the sitemap file
    sitemap_xml = download(url)
    print(sitemap_xml)
    if sitemap_xml is None:
        print('Fail to download', url)
        return
    # extract the sitemap links
    links = re.findall('<loc>(.*?)</loc>', sitemap_xml)
    pprint.pprint(links)
    print('请输入回车键继续...')
    enter = sys.stdin.read(1)

    for link in links:  # download each link
        html = download(link)
        print(html)
        time.sleep(2)  # 暂停一会儿
        # scrape or save html here

url = "http://example.python-scraping.com/sitemap.xml"
crawl_sitemap(url)
```

1.4.4 网站地图爬虫

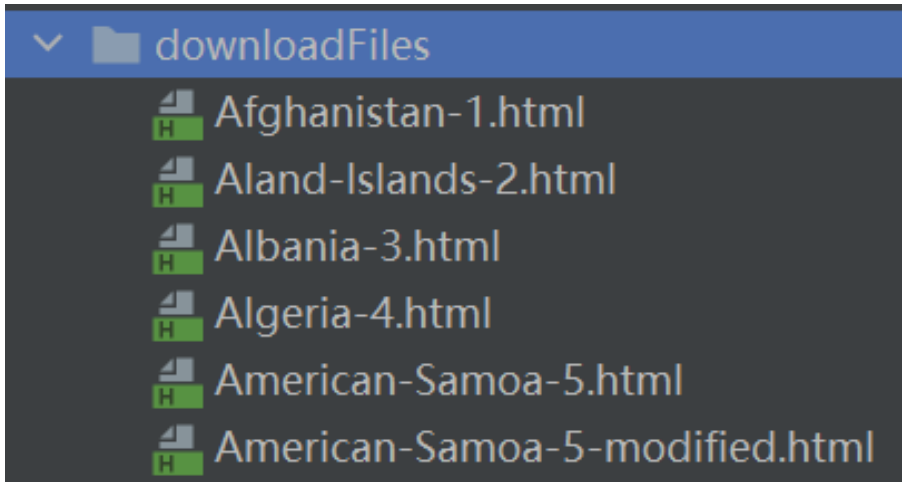
- 保存下载的网页至磁盘文件，以link中最后一个斜杆后的字符串作为网页下载后所保存的html文件名。

sitemap.xml - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url><loc>http://example.python-scraping.com/places/default/view/Afghanistan-1</loc></url>
<url><loc>http://example.python-scraping.com/places/default/view/Aland-Islands-2</loc></url>
<url><loc>http://example.python-scraping.com/places/default/view/Albania-3</loc></url>
<url><loc>http://example.python-scraping.com/places/default/view/Algeria-4</loc></url>
<url><loc>http://example.python-scraping.com/places/default/view/American-Samoa-5</loc></url>
```

第 1 行，第 1 列 100% Windows (CRLF) UTF-8



1.4.4 网站地图爬虫

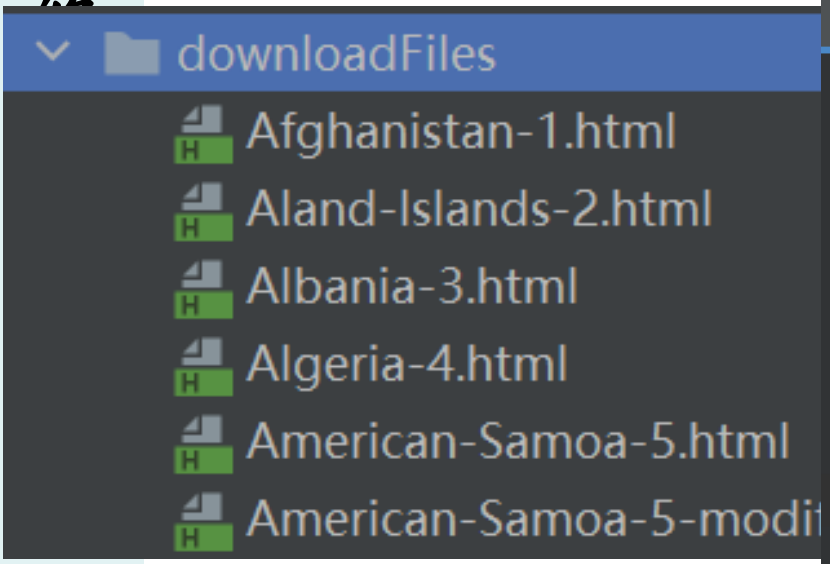
- 保存下载的网页至磁盘文件，以link中最后一个斜杆后的字符串作为网页下载后所保存的html文件名。

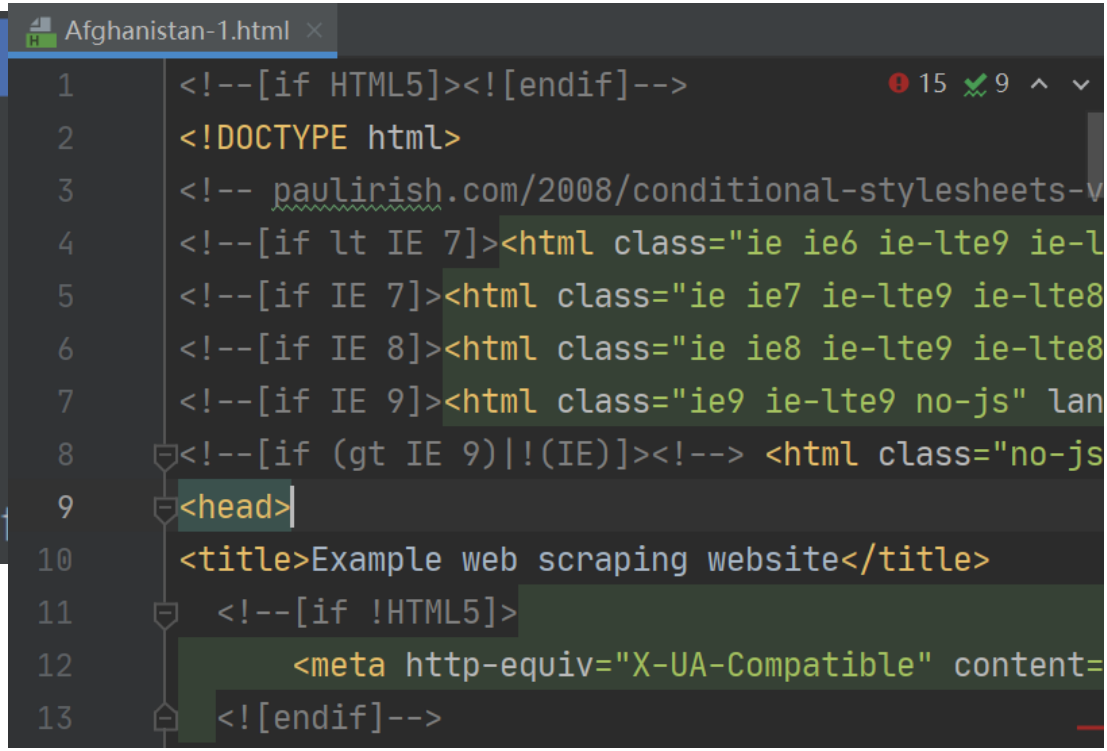
```
……
def saveToFile(text: str, filename: str):
    with open(file=filename, mode='tw', encoding='utf-8') as file:
        file.write(text)
def crawl_sitemap(url):
    ……
    for link in links:   # download each link
        filename = link[link.rindex('/') + 1:] + '.html'
        html = download(link)
        print(html)
        if html is None:
            continue
        saveToFile(html, 'downloadFiles/' + filename)
    # scape html here
url = "http://example.python-scraping.com/sitemap.xml"

crawl_sitemap(url)
```

1.4.4 网站地图爬虫

- 运行网站地图爬虫，从示例网站中下载所有国家或地区页面

```
Downloading: http://example.python-scraping.com/sitemap.xml
Downloading: http://example.python-scraping.com/view/Afghanistan-1
Downloading: http://example.python-scraping.com/view/Aland-Islands-2
Downloading: http://example.python-scraping.com/view/Albania-3
```

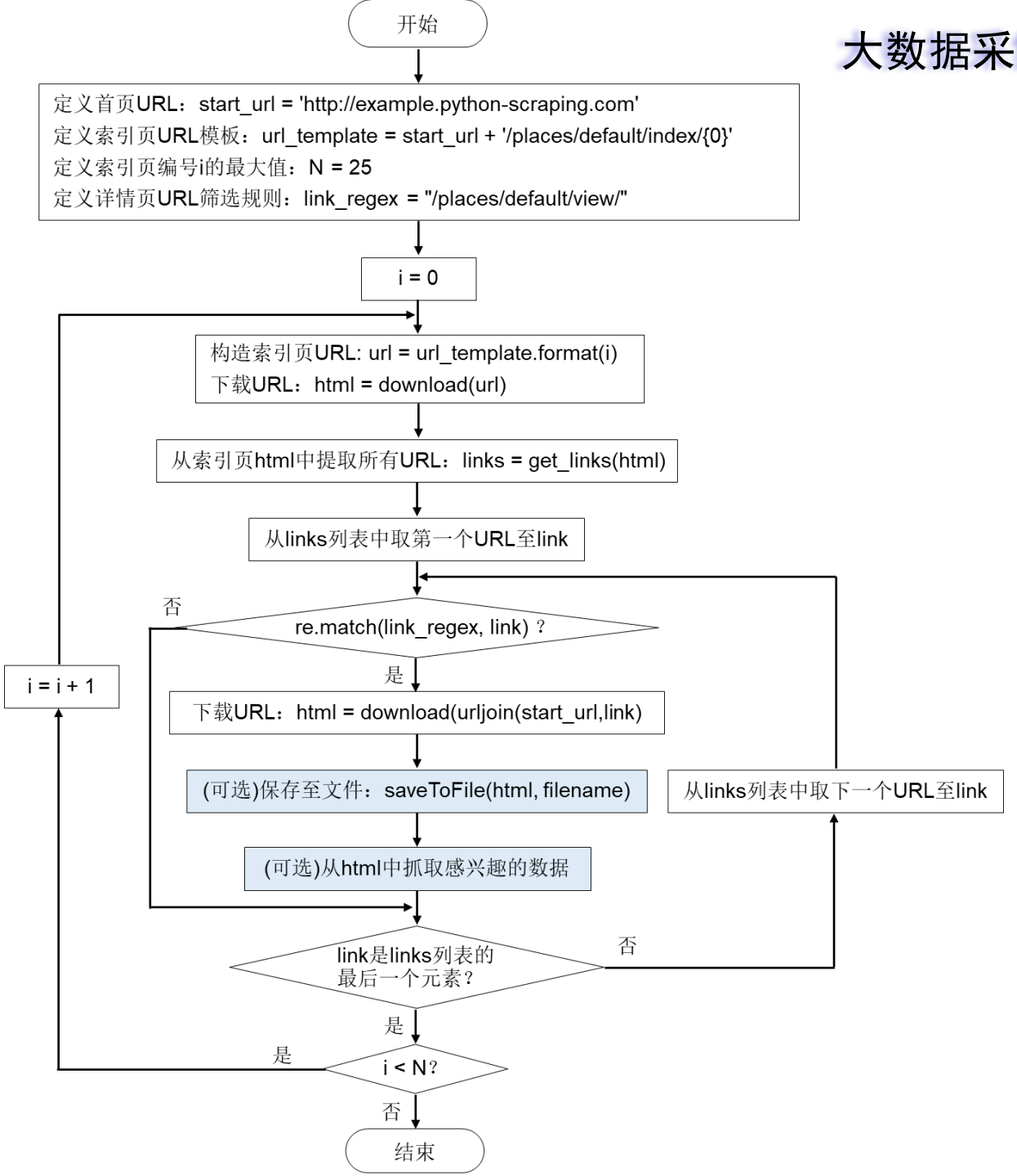




1.4.5 链接模板爬虫

- 有些网站的URL比较有规律，或者说这些URL有一个共同的模板，尤其是一些分页显示的页面的URL。这些URL中往往带有查询参数，例如第几页，不同的页面参数不同。因此只要根据URL模板循环构造各页面URL，便可下载所有页面。如果这些页面是索引页面，则需要再将索引页面中的详情页面URL找出，并下载它们。
- 示例网站中，索引页面的URL的路径是有规律的，两个URL只有最后一级路径的数字不同，即其模板为：

```
url_template = 'http://example.python-scraping.com/places/default/index/{0}'
```


开始
定义首页URL：start_url = 'http://example.python-scraping.com'
定义索引页URL模板：url_template = start_url + '/places/default/index/{0}'
定义索引页编号i的最大值：N = 25
定义详情页URL筛选规则：link_regex = "/places/default/view/"
i = 0
构造索引页URL: url = url_template.format(i)
下载URL：html = download(url)
从索引页html中提取所有URL：links = get_links(html)
从links列表中取第一个URL至link
否
re.match(link_regex, link) ？
是
下载URL：html = download(urljoin(start_url,link)
(可选)保存至文件：saveToFile(html, filename)
(可选)从html中抓取感兴趣的数据
从links列表中取下一个URL至link
否
link是links列表的
最后一个元素？
是
i = i + 1
是
i < N?
否
结束

1.4.5 链接模板爬虫

- 定义提取一个页面中所有链接的函数


```
import re
……
def get_links(html):
    """ Return a list of links from html """
    # a regular expression to extract all links from the webpage
    webpage_regex = re.compile("""<a[^>]+href=["'](.*?)["']""", re.IGNORECASE)
   # list of all links from webpage
    links = webpage_regex.findall(html)
    # 有些相对链接可能是空串(<a href=''>,例如sina.com.cn首页)，因此筛除.
    return list(  filter(lambda link: len(link.strip()) > 0, links)  )
```

1.4.5 链接模板爬虫

- 定义链接模板爬虫函数

```
def link_template_crawler(start_url: str, link_regex: str):
    N = 25  # 索引页面数量
    info_page_count = 0  # 详情页面计数
    url_template = start_url + '/places/default/index/{0}' # 索引页URL模板
    # 循环处理各索引页面
    for i in range(N):
        print(f'下载第{i+1}个索引页面')
        url = url_template.format(i)  # 构造URL
        html = download(url)  # 下载索引页面
        links = get_links(html)  # 提取所有URL
        # 循环下载详情页面
        for link in links:
            if not re.match(link_regex, link):
                continue
            html = download(urljoin(start_url, link))
            info_page_count += 1
            filename = link[link.rindex('/') + 1:] + '.html'  # 文件名
            saveToFile(html, 'link_template_crawler/' + filename)  # 保存页面至文件
            time.sleep(2)  # 延迟2秒
    print(f'总共下载{info_page_count}个详情页面')
```

urljoin函数返回start_url中的域名部分+link, 斜杆自动补全或移除。
例如：start_url = 'http://180.201.178.103/xyz'
link = 'places/default/view/a.html'
则urljoin返回：http://180.201.178.103/places/default/view/a.html

1.4.5 链接模板爬虫

- 执行链接模板爬虫

```
if __name__ == "__main__":
    start_url = "http://example.python-scraping.com"
    link_regex = "/places/default/view/"
    link_template_crawler(start_url, link_regex)  # 执行链接模板爬虫
```

```
下载第1个索引页面
Downloading: http://example.python-scraping.com/places/default/index/0
Downloading: http://example.python-scraping.com/places/default/view/Afghanistan-1
Downloading: http://example.python-scraping.com/places/default/view/Aland-Islands-2
……
下载第2个索引页面
Downloading: http://example.python-scraping.com/places/default/index/1
Downloading: …
 …..
下载第25个索引页面
Downloading: http://example.python-scraping.com/places/default/index/24
……
Downloading: http://example.python-scraping.com/places/default/view/Zambia-245
Downloading: http://example.python-scraping.com/places/default/view/Zimbabwe-246
总共下载246个详情页面
```

1.4.6 链接爬虫

- 到目前为止，网站地图爬虫已经符合预期。不过，我们无法依靠Sitemap文件提供每个网页的链接，因为该文件可能存在缺失、过期或不完整的问题。
- 通常，需要让爬虫表现得更像普通用户，跟踪链接，访问感兴趣的内容。通过跟踪每个链接的方式，可以很容易地下载整个网站的页面。
- 但是，这种方法可能会下载很多并不需要的网页。例如，想要从一个在线论坛中抓取用户账号详情页，那么此时只需要下载账号页，而不需要下载讨论贴的页面。
- 本章使用的链接爬虫将使用正则表达式来确定应当下载哪些页面。

1.4.6 链接爬虫

- 链接爬虫示例，其中使用正则表达式来确定当前页面中的哪些链接对应的页面应当下载

```
import re
from urllib.parse import urljoin
…
def download(url: str, ua='Porter', num_retries=2, charset='UTF-8'):
      ……
def saveToFile(text: str, filename: str):
     ……
def get_links(html):
    """ Return a list of links from html """
    # a regular expression to extract all links from the webpage
    webpage_regex = re.compile("""<a[^>]+href=["'](.*?)["']""", re.IGNORECASE)
   # list of all links from webpage
    links = webpage_regex.findall(html)
    # 有些相对链接可能是空串(<a href=''>,例如sina.com.cn首页)，因此筛除.
    return list(  filter(lambda link: len(link.strip()) > 0, links)  )
```

```
def link_crawler(start_url, link_regex):
    """ Crawl from given start URL following links matched by link_regex """
    crawl_queue = [start_url]
    while crawl_queue:
        url = crawl_queue.pop()  #弹出栈顶元素(即列最后一个元素)
        html = download(url=url)  #  下载页面
        if html is None:
            continue
        pos = url.rindex('/')  #  url中最后一个/所在位置
        if url == start_url or pos == len(url) - 1:
            filename = 'index.html'
        else:
            filename = url[pos + 1:] + '.html'
        saveToFile(html, 'link_crawler/' + filename)  #  保存至文件
        for link in get_links(html): # filter for links matching our regular expression
            if re.match(link_regex, link):
                abs_link = urljoin(start_url, link)
                crawl_queue.append(abs_link)  #  绝对链接加入队列末尾（压栈）
        time.sleep(2)  #  暂停
```

urllib定位网页需要使用绝对链接，以便包含定位网页的所有细节。Python的urllib库中的parse模块的urljoin方法来创建绝对路径。

```
url = 'http://example.python-scraping.com'
regex = '/places/default/(index|view)/'

link_crawler(url, regex)
```

注意，请事先在程序所在目录中建立link_crawler文件夹

http://example.python-scraping.com/

Example web scraping website

 Afghanistan
 Aland Islands
 Albania
 Algeria
 American Samoa
 Andorra
 Angola
 Anguilla
 Antarctica
 Antigua and Barbuda

```
url = 'http://example.python-scraping.com'
regex = '/places/default/(index|view)/'
link_crawler(url, regex)
```

< Previous | Next >

Elements Console Sources Network Performance Memory

```
<table>
  <tbody>
    <tr>
      <td>
        <div>
          <a href="/places/default/view/Afghanistan-1">…</a>
        </div>
      </td>
      <td>
        <div>
          <a href="/places/default/view/Aland-Islands-2">…</a>
        </div>
      </td>
    </tr>
```

```
<div id="pagination">
  "
    < Previous
  |
    "
  <a href="/places/default/index/1">Next ></a>
</div>
</div>
```

http://example.python-scraping.com/

Example web scraping website

下载完首页时的crawl_queue

Afghanistan | Aland Islands
Albania | Algeria
American Samoa | Andorra
Angola | Anguilla
Antarctica | Antigua and Ba

< Previous | Next >

```
crawl_queue = {list: 11} ['http://example.python-scraping.com/places/default/view/Afghanistan-1', 'h
 00 = {str} 'http://example.python-scraping.com/places/default/view/Afghanistan-1'
 01 = {str} 'http://example.python-scraping.com/places/default/view/Aland-Islands-2'
 02 = {str} 'http://example.python-scraping.com/places/default/view/Albania-3'
 03 = {str} 'http://example.python-scraping.com/places/default/view/Algeria-4'
 04 = {str} 'http://example.python-scraping.com/places/default/view/American-Samoa-5'
 05 = {str} 'http://example.python-scraping.com/places/default/view/Andorra-6'
 06 = {str} 'http://example.python-scraping.com/places/default/view/Angola-7'
 07 = {str} 'http://example.python-scraping.com/places/default/view/Anguilla-8'
 08 = {str} 'http://example.python-scraping.com/places/default/view/Antarctica-9'
 09 = {str} 'http://example.python-scraping.com/places/default/view/Antigua-and-Barbuda-10'
 10 = {str} 'http://example.python-scraping.com/places/default/index/1'
 __len__ = {int} 11
```

Elements | Console | Sources | Network | Performance | Memory

```
<table>
  <tbody>
    <tr>
      <td>
        <div>
          <a href="/places/default/view/Afghanistan-1">…</a>
        </div>
      </td>
      <td>
        <div>
          <a href="/places/default/view/Aland-Islands-2">…</a>
        </div>
      </td>
    </tr>
```

```
<div id="pagination">
  "
    < Previous
  |
    "
  <a href="/places/default/index/1">Next ></a>
</div>
</div>
```

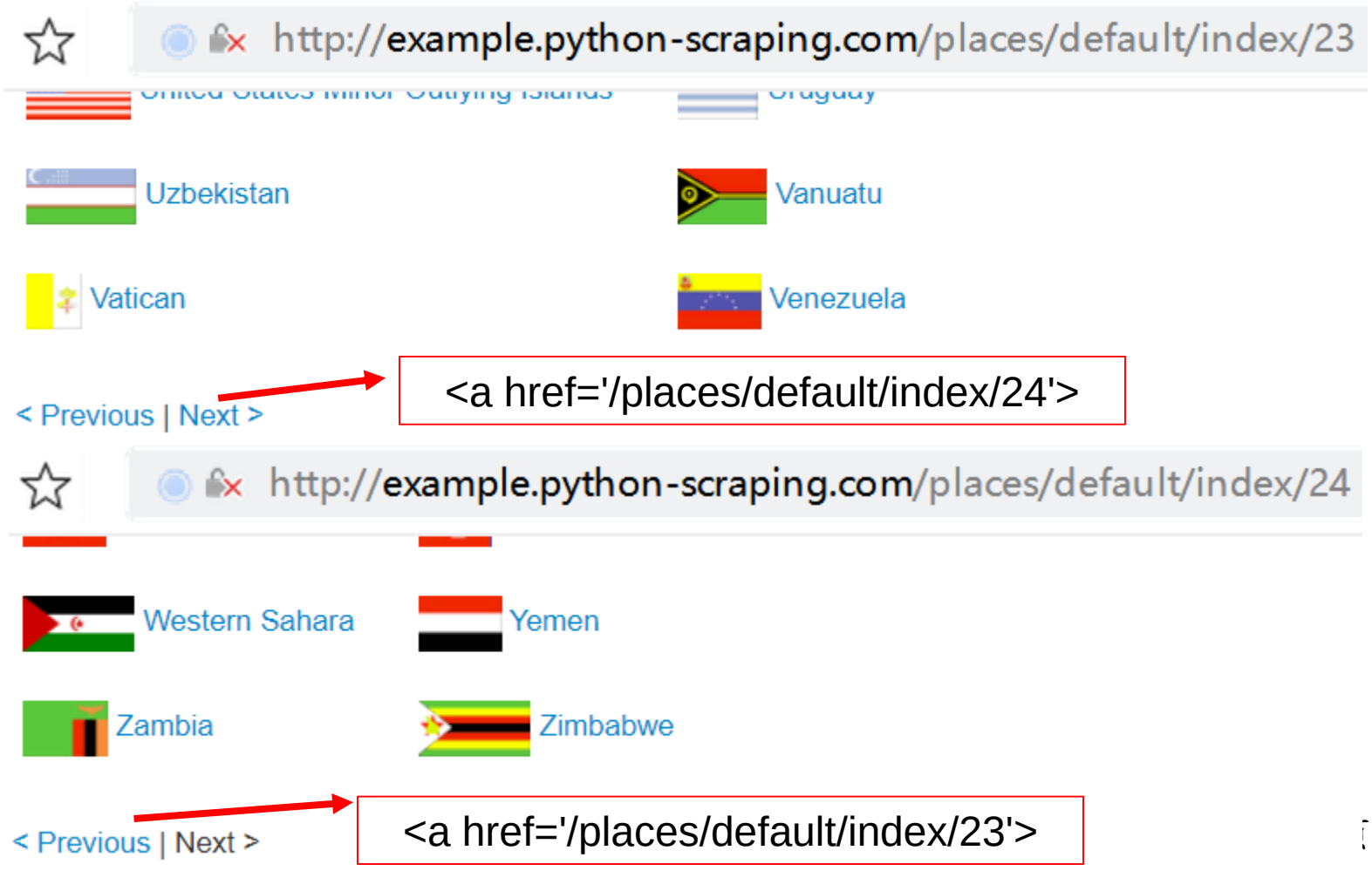
1.4.6 链接爬虫

- 在下载完1.html~24.html后，爬虫为何又不断反复循环下载23.html和24.html页面，且其他详情页面无法下载?

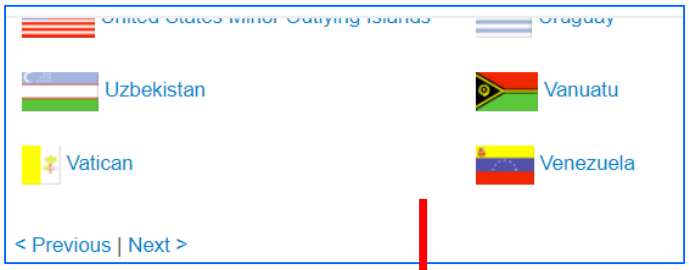
Downloading: http://example.python-scraping.com
Downloading: http://example.python-scraping.com/places/default/index/1
Downloading: http://example.python-scraping.com/places/default/index/2
……
Downloading: http://example.python-scraping.com/places/default/index/22
Downloading: http://example.python-scraping.com/places/default/index/23
Downloading: http://example.python-scraping.com/places/default/index/24
Downloading: http://example.python-scraping.com/places/default/index/23
Downloading: http://example.python-scraping.com/places/default/index/24
……
Downloading: http://example.python-scraping.com/places/default/index/23
Downloading: http://example.python-scraping.com/places/default/index/24
……

1.4.6 链接爬虫

- 在下载完1.html~24.html后，爬虫为何又不断反复循环下载23.html和24.html页面，且其他详情页面无法下载？



假设总共有3个页面



C_{11} C_{12} … C_{1n} P_2

C_{21} C_{22} … C_{2n} P_1 P_3

C_{31} C_{32} … C_{3m} P_2

链接队列变化

获取完Page1中的链接后

C_{11} C_{12} … C_{1n} P_2

获取完Page2中的链接后

C_{11} C_{12} … C_{1n} C_{21} C_{22} … C_{2n} P_1 P_3

获取完Page3中的链接后

C_{11} C_{12} … C_{1n} C_{21} C_{22} … C_{2n} P_1 C_{31} C_{32} … C_{3m} P_2

C_{11} C_{12} … C_{1n} C_{21} C_{22} … C_{2n} P_1 C_{31} C_{32} … C_{3m} C_{21} C_{22} … C_{2n} P_1 P_3

1.4.6 链接爬虫

- 页面相互之间可能存在链接，因此同样的地点总是会被不断地下载。
- 比如，澳大利亚链接到了南极洲，而南极洲又链接回了澳大利亚，此时爬虫就会继续将这些URL放入队列，永远不会到达队列尾部。
- 解决办法：要想避免重复爬取相同的链接，则需要记录哪些链接已经被爬取过。一个可行的办法是添加一个集合，用于存储已经爬取过的链接，从而避免重复爬取相同的链接。

```
def link_crawler(start_url, link_regex):
    crawl_queue = [start_url]
    seen = set(crawl_queue) # keep track which URL's have seen before
    while crawl_queue:
        url = crawl_queue.pop()  # 弹出栈顶元素（即列表最后一个元素）
        html = download(url=url)  # 下载页面
        if html is None:
            continue
        pos = url.rindex('/')  # url中最后一个/所在位置
        if url == start_url or pos == len(url) - 1:
            filename = 'index.html'
        else:
            filename = url[pos + 1:] + '.html'
        saveToFile(html, 'link_crawler2/' + filename)  # 保存至文件
        # filter for links matching our regular expression
        for link in get_links(html):
            if re.match(link_regex, link):
                abs_link = urljoin(start_url, link)
                if abs_link not in seen: # check if have already seen this link
                    seen.add(abs_link)
                    crawl_queue.append(abs_link)  # 绝对链接加入队列末尾（压栈）
        time.sleep(2)  # 暂停
```

当运行该脚本时，它会爬取所有地点，并且能够如期停止。最终，我们得到了一个可用的链接爬虫！

```
Downloading: http://example.python-scraping.com
Downloading: http://example.python-scraping.com/places/default/index/1
Downloading: http://example.python-scraping.com/places/default/index/2
Download error: timed out
Pause for 10 seconds.
Retry to download.
Downloading: http://example.python-scraping.com/places/default/index/2
……
Downloading: http://example.python-scraping.com/places/default/index/23
Downloading: http://example.python-scraping.com/places/default/index/24
Downloading: http://example.python-scraping.com/places/default/view/Zimbabwe-246
Downloading: http://example.python-scraping.com/places/default/view/Zambia-245
Downloading: http://example.python-scraping.com/places/default/view/Yemen-244
……
Downloading: http://example.python-scraping.com/places/default/view/Albania-3
Downloading: http://example.python-scraping.com/places/default/view/Aland-Islands-2
Downloading: http://example.python-scraping.com/places/default/view/Afghanistan-1
```

1.5 为爬虫添加一些高级功能

- 接下来，为链接爬虫添加一些功能，使其在爬取其他网站时更加有用。
  - 解析robots.txt文件以避免下载禁止爬取的URL
  - 使用代理访问某个网站
  - 对爬虫限速
  - 避免爬虫陷阱

1.5.1 解析robots.txt文件

- 首先，需要解析robots.txt文件，以避免下载禁止爬取的URL。使用Python的urllib库中的robotparser模块，就可以轻松完成这项工作，如下面的代码所示。

```
from urllib import robotparser

rp = robotparser.RobotFileParser()  # 创建对象
url = ' http://example.python-scraping.com '
rp.set_url(url + '/robots.txt')
rp.read()  # 读取URL指定的文件

ua = 'BadCrawler'
# can_fetch()函数确定指定的用户代理是否允许访问网页
print(rp.can_fetch(ua, url))
ua = 'Porter'
print(rp.can_fetch(ua, url))
print(rp.site_maps()) # 读取站点地图的URL，返回一个URL列表
```

当用户代理设置为'BadCrawler'时，robotparser模块的返回结果表明无法获取网页，正如在示例网站的robots.txt文件中看到的定义一样。

```
False
True
['http://example.python-scraping.com/robots.txt']
```

1.5.1 解析robots.txt文件

- 为了将robotparser集成到链接爬虫中，首先需要创建一个新函数用于返回RobotParser对象。

```
def get_robots_parse(robots_url):
    """ Return the robots parser object using the robots_url """
    rp = robotparser.RobotFileParser()  # 创建对象
    rp.set_url(robots_url)
    rp.read()  # 读取URL指定的文件
    return rp
```

1.5.1 解析robots.txt文件

- 我们需要可靠地设置robots_url，此时可以通过向函数传递额外的关键词参数的方法实现这一目标。还可以设置一个默认值，防止用户没有传递该变量。假设从网站根目录开始爬取，则可以简单地将robots.txt添加到URL的结尾处。
- 此外还要定义ua，以便传递给can_fetch()函数。

```
def link_crawler(start_url, link_regex, robots_url=None, ua='Porter'):
    """ Crawl from given start URL following links matched by link_regex """
    crawl_queue = [start_url]
    # keep track which URL's have been before
    seen = set(crawl_queue)

    if not robots_url:
        robots_url = '{}/robots.txt'.format(start_url)
    rp = get_robots_parse(robots_url)
    ……
```

1.5.1 解析robots.txt文件

- 最后，在crawl循环中添加解析器检查。

```
def link_crawler(start_url, link_regex, robots_url=None, ua='Porter'):
    ……
    while crawl_queue:
        url = crawl_queue.pop()  # 弹出队列首元素（链接）
        # check url passes robots.txt restrictions
        if rp.can_fetch(ua, url):
            html = download(url=url, ua=ua)  # 下载页面
            ……
        else:
            print('Blocked by robots.txt:', url)
```

- 可以通过使用用户代理BadCrawler字符串来测试这个高级链接爬虫以及robotparser的使用。

```
url = 'http://example.python-scraping.com'
regex = '/places/default/(index|view)/'
ua = 'BadCrawler'
link_crawler(url, regex, ua=ua)
```

输出：**Blocked by robots.txt: http://example.python-scraping.com**

1.5.2 使用代理访问某个网站

- 什么是代理
  - 代理就是第三方代替本体处理相关事务。例如：生活中的代理：代购，中介，微商.....
- 爬虫中为什么需要使用代理
  - 一些网站会有相应的反爬虫措施，例如很多网站会检测某一段时间某个IP的访问次数，如果访问频率太快以至于看起来不像正常访客，它可能就会会禁止这个IP的访问。所以我们需要设置一些代理IP，每隔一段时间换一个代理IP，就算IP被禁止，依然可以换个IP继续爬取。
  - 有时我们需要使用代理访问某个网站。比如，Hulu在美国以外的很多国家被屏蔽，YouTube上的一些视频也是。
- 免费代理ip提供网站
  - http://www.goubanjia.com/，西祠代理，快代理，.....

1.5.2 使用代理访问某个网站

- 使用urllib支持代理并没有想象中那么容易。后面的小节将介绍一个对用户更友好的Python HTTP模块——requests，该模块同样也能够处理代理。

```
def download(url: str, ua='Porter', num_retries=2, charset='UTF-8', proxy=None):
    print('Downloading:', url)
    rq = request.Request(url)  # http请求
    rq.add_header(key='User-agent', val=ua)  # 指定用户代理
    try:
        if proxy:
            proxy_support = request.ProxyHandler(proxy)
            opener = request.build_opener(proxy_support)
            request.install_opener(opener)
        response = request.urlopen(url=rq, timeout=10)
        ......
    except Exception as e:
        ......
        html = download(url, ua = ua,
                        num_retries=num_retries - 1, proxy=proxy)  # 重试下载
    return html
```

1.5.2 使用代理访问某个网站

- 使用urllib支持代理并没有想象中那么容易。我们将在后面的小节介绍一个对用户更友好的Python HTTP模块——requests，该模块同样也能够处理代理。

```
# 不同的代理IP
proxy_list = [
    {"http": "123.56.75.226:3128"} , # 北京联通 2020-10-28测试可行
    {"http": "113.204.164.194:8080"} , #  重庆联通2020-10-28测试可行
    {"http": "163.204.95.51:9999"},
    {"http": "125.123.156.93:3000"},
    {"http": "125.110.115.47:9000"}
]
proxy = random.choice(proxy_list)
url = 'http://example.python-scraping.com'
html = download(url=url, ua=ua, proxies=proxy)
```

1.5.3 对爬虫限速

- 如果爬取网站的速度过快，就会面临被封禁或是造成服务器过载的风险。为了降低这些风险，可以在两次下载之间添加一组延时，从而对爬虫限速。
- 这里，我们将限速功能封装在Throttle类中
  - delay属性—下载延时；字典domains—记录了每个域名上次访问的时间。
  - 构造方法__init__()—初始化delay和domains。
  - wait( )方法—如果当前时间距离上次访问时间小于指定延时delay，则执行睡眠操作，即通过time.sleep( )方法实现睡眠等待若干时长。

1.5.3 对爬虫限速

```
import time
from urllib.parse import *

class Throttle:  # （节流阀） 记录每个域名上次访问的时间
    """    Add a delay between downloads to the same domain    """
    def __init__(self, delay):
        self.delay = delay  # amount of delay between downloads for each domain
        self.domains = {}  # timestamp of when a domain was last accessed
                           # {'example.python-scraping.com': 1601377330.093846}
    def wait(self, url):
        domain = urlparse(url).netloc
        last_accessed = self.domains.get(domain)  # get last accessed time for domain
        if self.delay > 0 and last_accessed is not None:
            sleep_secs = self.delay - (time.time() - last_accessed)
            if sleep_secs > 0:  # domain has been accessed recently,so need to sleep
                time.sleep(sleep_secs)
        self.domains[domain] = time.time() # update the last accessed time
```

可以在每次下载之前调用throttle对爬虫进行限速。

```
……
from webScrapingExamples.ch1.throttle import Throttle
def get_robots_parse(robots_url): ……
def link_crawler(start_url, link_regex, robots_url=None, ua='Porter'):
    ……
    global throttle
    while crawl_queue:
        url = crawl_queue.pop()  #弹出栈顶元素（队列末尾元素）
        if rp.can_fetch(ua, url): # check url passes robots.txt restrictions
            throttle.wait(url)   #   保证与上次访问的时间间隔不少于5秒
            html = download(url=url)  # 下载页面
            ……
        else:
            print('Blocked by robots.txt:', url)
def get_links(html): ……
def download(url: str, ua='Porter', num_retries=2, charset='UTF-8'): ……
def saveToFile(text: str, filename: str): ……

if __name__ == '__main__':
    url = 'http://example.python-scraping.com'
    regex = '/places/default/(index|view)/'
    delay = 5
    throttle = Throttle(delay)  #   保证与上次访问

    link_crawler(url, regex)
```

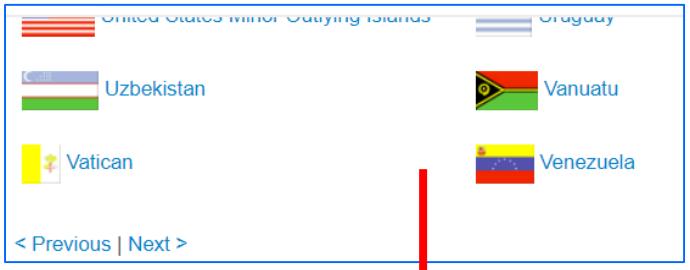
1.5.4 避免爬虫陷阱

- 目前，我们的爬虫会跟踪所有之前没有访问过的链接。但是，一些网站会动态生成页面内容，这样就会出现无限多的网页。比如，网站有一个在线日历功能，提供了可以访问下个月和下一年的链接，那么下个月的页面中同样会包含访问再下个月的链接，这样就会一直持续请求到部件设定的最大时间（可能会是很久之后的时间）。该站点可能还会在简单的分页导航中提供相同的功能，本质上是分页请求不断访问空的搜索结果页，直至达到最大页数。这种情况被称为**爬虫陷阱**。

1.5.4 避免爬虫陷阱

- 想要避免陷入爬虫陷阱，一个简单的方法是记录到达当前网页经过了多少个链接，也就是深度。当到达最大深度时，爬虫就不再向队列中添加该网页中的链接了。
- 要实现最大深度的功能，我们需要修改seen变量。该变量原先只记录访问过的网页链接，现在修改为一个字典，增加了已发现链接的深度记录。

假设总共有3个页面，集合seen改为字典，url：depth为元素



链接队列与字典变化

C_{11}	C_{12}	…	C_{1n}	P_2

C_{21}	C_{22}	…	C_{2n}	P_1	P_3

C_{31}	C_{32}	…	C_{3m}	P_2

初始化 depth = 0

{ } = seen

获取完Page1中的链接后

C_{11}	C_{12}	…	C_{1n}	P_2

depth += 1

{C_{11}:1, C_{12}:1, …, C_{1n}:1, P_2:1 } = seen

获取完Page2中的链接后

C_{11}	C_{12}	…	C_{1n}	C_{21}	C_{22}	…	C_{2n}	P_1	P_3

depth += 1

{C_{11}:1, C_{12}:1, …, C_{1n}:1, P_2:1, C_{21}:2, C_{22}:2 , …,C_{2n}:2, P_1:2, P_3:2} = seen

获取完Page3中的链接后

C_{11}	C_{12}	…	C_{1n}	C_{21}	C_{22}	…	C_{2n}	P_1	C_{31}	C_{32}	…	C_{3m}	P_2

depth += 1

seen =

{C_{11}:1, C_{12}:1, …, C_{1n}:1, P_2:1, C_{21}:2, C_{22}:2 , …,C_{2n}:2, P_1:2, P_3:2 ,C_{31}:3, C_{32}:3 , …,C_{3n}:3}

```
.....
def link_crawler(……, max_depth=4):
    .....
    seen = {}  #  seen = dict()
    while crawl_queue:
        url = crawl_queue.pop()  # 弹出队列首元素（链接）
        if rp.can_fetch(ua, url): # check url passes robots.txt restrictions
            depth = seen.get(url, 0)
            if depth > max_depth:  # 到达最大深度后，不再解析该页面中的链接
                print('Skipping %s due to depth' % url)
                continue
            throttle.wait(url)
            html = download(url=url)  # 下载页面
            .....
            for link in get_links(html):
                if re.match(link_regex, link):
                    abs_link = urljoin(start_url, link)
                    if abs_link not in seen: # check if have already seen this link
                        seen[abs_link] = depth + 1
                        crawl_queue.append(abs_link)  # 绝对链接加入队列末尾（压栈）
        else:
            print('Blocked by robots.txt:', url)
.....
if __name__ == '__main__':
    .....
    link_crawler(url, regex, max_depth=2)
```

如果想要禁用该功能，只需将max_depth设为一个负数即可，此时当前深度永远不会与之相等。

1.5.4 避免爬虫陷阱

- 现在，让我们使用默认的用户代理，并将最大深度设置为1，这样只有主页上的链接才会被下载。

```
link_crawler(url, regex, max_depth=1)
```

```
Downloading: http://example.python-scraping.com
Downloading: http://example.python-scraping.com/places/default/index/1
Skipping http://example.python-scraping.com/places/default/index/2 due to depth
Skipping http://example.python-scraping.com/places/default/index/0 due to depth
Skipping http://example.python-scraping.com/places/default/view/Barbados-20 due to depth
……
Skipping http://example.python-scraping.com/places/default/view/Armenia-12 due to depth
Skipping http://example.python-scraping.com/places/default/view/Argentina-11 due to depth
Downloading: http://example.python-scraping.com/places/default/view/Antigua-and-Barbuda-10
Downloading: http://example.python-scraping.com/places/default/view/Antarctica-9
……
Downloading: http://example.python-scraping.com/places/default/view/Aland-Islands-2
Downloading: http://example.python-scraping.com/places/default/view/Afghanistan-1
```

这些未下载的链接是第二个页面中的

1.6 使用requests库

- 尽管只使用urllib就已经实现了一个相对高级的解析器，不过目前Python编写的主流爬虫一般都会使用requests库来管理复杂的HTTP请求。

Requests is an elegant and simple HTTP library for Python, built for human beings.

- 该项目起初只是以“人类可读”的方式协助封装urllib功能的小库，不过现如今已经发展成为拥有数百名贡献者的庞大项目。可用的一些功能包括内置的编码处理、对SSL和安全的重要更新以及对POST请求、JSON、cookie和代理的简单处理。
- 在大部分情况下，都将使用requests库，因为它足够简单并且易于使用，而且它事实上也是大多数网络爬虫项目的标准。
- 官方文档：https://requests.readthedocs.io/en/latest/

1.6.1 requests库的安装与测试

- 安装requests库

  Windows平台“以管理员身份运行” cmd，执行：

  pip install requests

- 测试

```
>>>import requests
>>>r=requests.get("http://www.baidu.com" , timeout=30)
>>> r.status_code
200
>>>r.text
```

'<!DOCTYPE html>\r\n<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>ç\x99¾å°¦ä¸\x80ä¸\x8bï¼\x8cä½\xa0å° ±ç\x9f¥é\x81\x93</title>…… <img src=//www.baidu.com/img/gs.gif> </p>   </body> </html>\r\n'

1.6.2 requests库的7个主要方法

方法	说明
request()	构造一个HTTP请求
get()	通过GET方式发送HTTP请求，是下载网页的主要方法，对应HTTP的GET命令
head()	获取HTTP请求头，对应于HTTP的HEAD命令
post()	通过POST方式发送HTTP请求，可以携带数据，对应于HTTP的POST命令
put()	向网页提交PUT请求，对应于HTTP的PUT命令
patch()	向网页提交局部修改请求，对应于HTTP的PATCH命令
delete()	向网页提交删除请求，对应于HTTP的DELETE命令

```
>>>r=requests.get("http://www.baidu.com" , timeout=30)    # get请求
```

```
url = "https://fanyi.baidu.com/sug"
data = {'kw': '早上好' }  # 该字典键值对的形式可以通过form data中查询
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.89 Safari/537.36'}

r = requests.post(url, data=data, headers=headers , timeout=30)  # post请求
```

1.6.3 Response对象的主要属性和方法

属性	说明
status_code	HTTP请求返回的状态码。
encoding	从HTTP响应头中的页面字符集。
apparent_encoding	从页面内容中分析出来的页面字符集。
content	HTTP响应内容的二进制表示（bytes类型）。
text	HTTP响应内容的字符串表示，即页面HTML源代码。
url	HTTP响应对应的URL地址。页面跳转时，请求的url地址与真正打开的url地址不同。
headers	HTTP响应头，是一个字典。

- encoding属性是从Response对象的headers中的charset字段中提取的编码方式。若headers中没有charset字段，则encoding默认为ISO-8859-1编码，因此无法解析中文，这是乱码的原因。
- apparent_encoding属性值是通过分析网页内容猜测得到编码，因此，当encoding为默认值ISO-8859-1时，可以选用apparent_encoding的属性值作为编码，即 r.encoding = r. apparent_encoding。
- 当网页出现乱码时，可以把apparent_encoding的属性值赋给encoding。

```
if r.encoding == 'ISO-8859-1': #  若headers中没有charset字段,
    r.encoding = r.apparent_encoding  # 使用猜测的编码
```

1.6.3 Response对象的主要属性和方法

- r.content.decode('utf-8') # 将响应的字节解码为指定字符集的字符串
- r.json( ) # 针对响应为json字符串解码为python字典
- r.raise_for_status( ) #若响应状态码不是200，则引发requests.HTTPError异常。
- 注意：
  - r.text 属性（该属性往往会出现乱码，此时使用r.encoding='utf-8'或者r.encoding=r.apparent_encoding）
  - r.request.url # 该响应对应的发送请求的url地址
  - r.url # 响应的url地址(页面跳转时，请求的url地址与真正打开的url地址是不同的)
  - r.request.headers # 该响应对应的请求头
  - r.headers # 响应头

1.6.3 Response对象的主要属性和方法

```
>>> r = requests.get('https://www.python.org' , timeout=30)
>>> r.status_code   # 获得响应状态码
200
>>> r.status_code == requests.codes.ok
True
>>> b'Python is a programming language' in r.content
True
```

```
r = requests.get('http://httpbin.org/get' , timeout=30)
print(r.headers)  # 获得响应头信息
print(r.headers['Content-Type'])
print(r.headers.get('Content-Length'))
```

```
{'Date': 'Sun, 18 Jul 2021 10:22:23 GMT', 'Content-Type': 'application/json', 'Content-Length': '305', 'Connection': 'keep-alive', 'Server': 'gunicorn/19.9.0', 'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-Credentials': 'true'}
application/json
305
```

1.6.4 requests库的相关异常

异常类型	说明
requests.connectionError	网络连接错误异常，如DNS查询失败、拒绝连接等
requests.HTTPError	HTTP错误异常
requests.URLRequired	URL缺失异常
requests.TooManyRedirects	超过最大重定向次数，产生重定向异常
requests.ConnectTimeout	连接远程服务器超时异常
requests.Timeout	请求URL超时，产生超时异常

```
r = requests.get(url, timeout=30)  # 发送http请求，并返回http响应
r.raise_for_status()  # 如果状态不是200，引发requests.HTTPError异常
if r.encoding == 'ISO-8859-1': # 若headers中没有charset字段,
                                则encoding默认为ISO-8859-1编码
    r.encoding = r.apparent_encoding  # 使用猜测的编码
print(r.text)  # 输出HTML源码
```

1.6.5 使用requests库爬取网页的通用框架

```
import requests  # 导入requests包

def getHTMLText(url):
    try:
        r = requests.get(url, timeout=30)  # 发送http请求，并返回响应
        r.raise_for_status()  # 若状态不是200，引发HTTPError异常
        if r.encoding == 'ISO-8859-1': # 若headers中没有charset字段,
                                       则encoding默认为ISO-8859-1编码
            r.encoding = r.apparent_encoding  # 使用猜测的编码
        return r.text  # 返回HTML源码
    except:
        pass


if __name__ == "__main__":
    url = 'http://www.upc.edu.cn'
    html = getHTMLText(url)
    print(html)
```

1.6.6 HTTP协议

超文本传输协议(Hypertext Transfer Protocol，HTTP)

- HTTP是一个基于“请求与响应”模式的、无状态的应用层协议。
- URL（Uniform Resource Locator，统一资源定位符）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。

✓ 无状态是指服务端对于客户端每次发送的请求都认为它是一个新的请求，上一次会话和下一次会话没有联系。

✓ HTTP 协议这种特性有优点也有缺点，优点在于解放了服务器，每一次请求“点到为止”不会造成不必要连接占用，缺点在于每次请求会传输大量重复的内容信息。

1.6.7 两个完整示例

- 示例1：爬取百度翻译信息。
- 示例2：创建一个使用requests的高级链接爬虫。

示例1：爬取百度翻译信息。

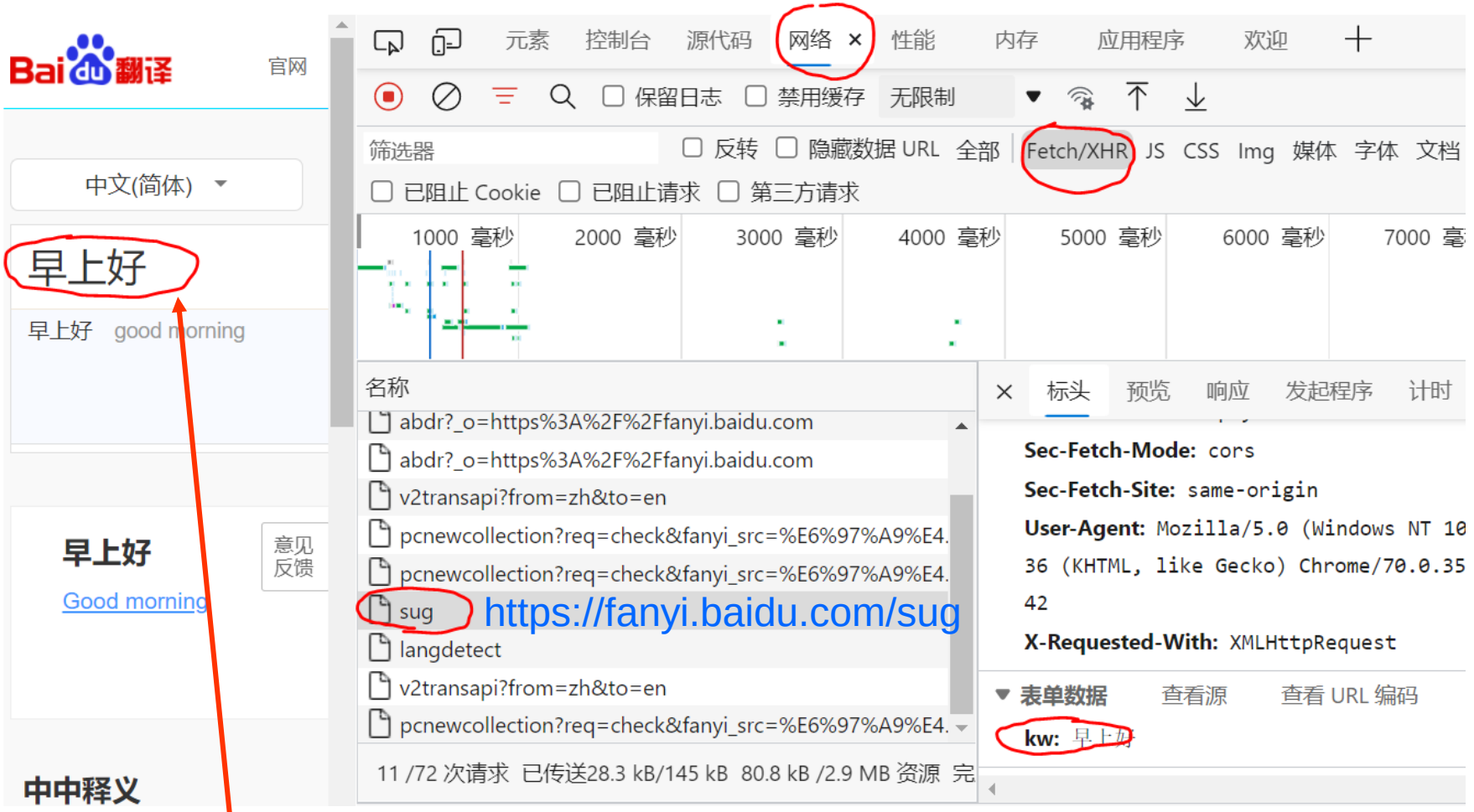
```
import requests
import json

def baidu_spider(keyword: str = '中国石油大学') -> None :
    """
    根据关键字爬取百度翻译信息。

    :param keyword: 需要翻译的文本
    :return: 百度翻译结果
    """
    if keyword is None:
        print("Parameter keyword should not be None!")
        return
```

```
def baidu_spider(keyword: str = '中国石油大学') -> None:
    ......
    # 1.指定链接
    url = "https://fanyi.baidu.com/sug"  # 通过网页逆向工程得到

    # 2.UA伪装
    headers = { 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.141
Safari/537.36' }

    # 3.POST参数设置
    params = {'kw': keyword}
    try:
        # 4.发送请求
        r = requests.post(url=url, data=params, headers=headers , timeout=30)
        r.raise_for_status()  # 如果状态不是200，引发HTTPError异常
        if r.encoding == 'ISO-8859-1': # 若headers中没有charset字段,
                                       则encoding默认为ISO-8859-1编码
            r.encoding = r.apparent_encoding  # 使用猜测的编码
        # 5.获取响应的信息
        dict_obj = r.json()
```

```
def baidu_spider(keyword: str = '中国石油大学') -> None:
        ......

            # 6.持久化存储
             fileName = keyword + '.json'
            fp = open(fileName, 'wt', encoding='utf-8')
            json.dump(dict_obj, fp=fp,  ensure_ascii=False)
            fp.close()
            print("翻译结果: ", dict_obj["data"])
        except:
            pass

if __name__ == '__main__':
    # 输入需要翻译的关键字
        keyword = input("Enter the key words you want to translate: ")
    # 爬取百度翻译信息
        baidu_spider(keyword)
```

```
Enter the key words you want to translate: 中国石油大学
翻译结果:  [{'k': '中国石油大学', 'v': '名. China University of Petroleum'}]
```

1.6.7 两个完整示例



注意，在当前窗口下输入要翻译的中文，不会出现sug链接，而删除一些字符后才会出现。

示例2：创建一个使用requests的高级链接爬虫

与前面的爬虫类似，但是download函数需要修改。

```
.....
import requests
def download(url: str, ua= 'Porter', num_retries=2, proxies=None):
    print('Downloading:', url)
    headers = {'User-agent': ua}  # 指定用户代理
    try:
        r = requests.get(url=url, headers=headers, proxies=proxies , timeout=30)
        if r.encoding == 'ISO-8859-1': # 若headers中没有charset字段
            r.encoding = r.apparent_encoding  # 使用猜测的编码
        html = r.text
        if r.status_code != 200:
            print('Error code:', r.status_code )
            html = None
            if num_retries > 0 and (500 <= r.status_code < 600):
                delay = 10  # 延迟发送请求的秒数
                print(f'Pause for {delay} seconds. ')
                time.sleep(delay)
                print('Retry to download. ')
                html = download(url, ua, num_retries - 1, proxies)
        else:
            print('encoding=', r.encoding)
    except requests.exceptions.RequestException as e:
        print('Download error:', e.reason)
    return html
```

1.6 使用requests库

- status_code属性的使用更加方便，因为每个请求中都包含该属性。
- 另外，不再需要测试字符编码了，因为Response对象的text属性已经为我们自动化实现了该功能。
- 对于无法处理的URL或超时等罕见情况，都可以使用RequestException进行处理，只需一句简单的捕获异常的语句即可。
- 代理处理也已经被考虑进来了，只需传递代理的字典即可。

1.7 本章小结

- 本章介绍了网络爬虫，然后给出了一个能够在后续章节中复用的成熟爬虫。
- 此外，还介绍了一些外部工具和模块的使用方法，用于了解网站、用户代理、网站地图、爬取延时以及各种高级爬取技术。
- 下一章将讨论如何从已爬取到的网页中获取数据。

谢谢大家！