Python 爬虫技术

使用 Python 实践 Web 内容抓取技术。主要内容包括：

- 爬虫技术简介
- Web 基础知识
- Python 实现 Web 内容抓取和解析的工具
- 实践 1：静态内容爬取和解析
- 实践 2：动态内容的爬取和解析
- 实践 3：爬与反爬
- 总结：
  - 抓取页面内容解析
  - 抓取 json 数据解析
  - 传参（get 传参和 post 传参）

1、爬虫技术简介

数据时代，数据获取的方式多种多样。为什么要使用爬虫？到底什么是爬虫？

1.1 数据获取的方式

分析所采用数据的来源有哪些？

1. 交易数据。
   如信用卡刷卡数据、电子商务数据、互联网点击数据、订单数据等。
2. 移动通讯数据。
   通过移动设备上的软件追踪和沟通无数事件产生的交易数据(如搜索产品的记录事件)等。
3. 机器和传感器数据。
   来自感应器、量表和其他设施的数据、定位/GPS 系统数据等。
4. 互联网上的“开放数据”来源。
   如政府机构,非营利组织和企业免费提供的数据等。
5. 人为数据即通过人类行为产生的数据。
   例如电子邮件、文档、图片、视频，以及通过微信、博客等社交媒体产生的数据流。

在当前的大数据时代，我们有不同的方式来获取数据：

1. 企业生产的用户数据: 大型互联网公司有海量用户, 所以他们积累数据有天然优势. 有数据意识的中小型企业, 也开始积累数据.
2. 数据管理咨询公司: 通常这样的公司有很庞大的数据采集团队, 一般会通过市场调研, 和各行各业

的公司进行合作，专家对话

3. 政府/第三方机构提供的公开数据：地方政府，通过各地政府统计上报的数据进行整合；机构都是权威的第三方网站.
4. 第三方数据平台购买：通过各个数据交易平台购买各行各业的数据，根据获取难度定价.
5. 爬虫爬取数据：如果市场上没有我们需要的数据，或者价格太高.那我们就可以招一个爬虫工程师，从互联网上定向获取数据.

1.2 什么是爬虫？

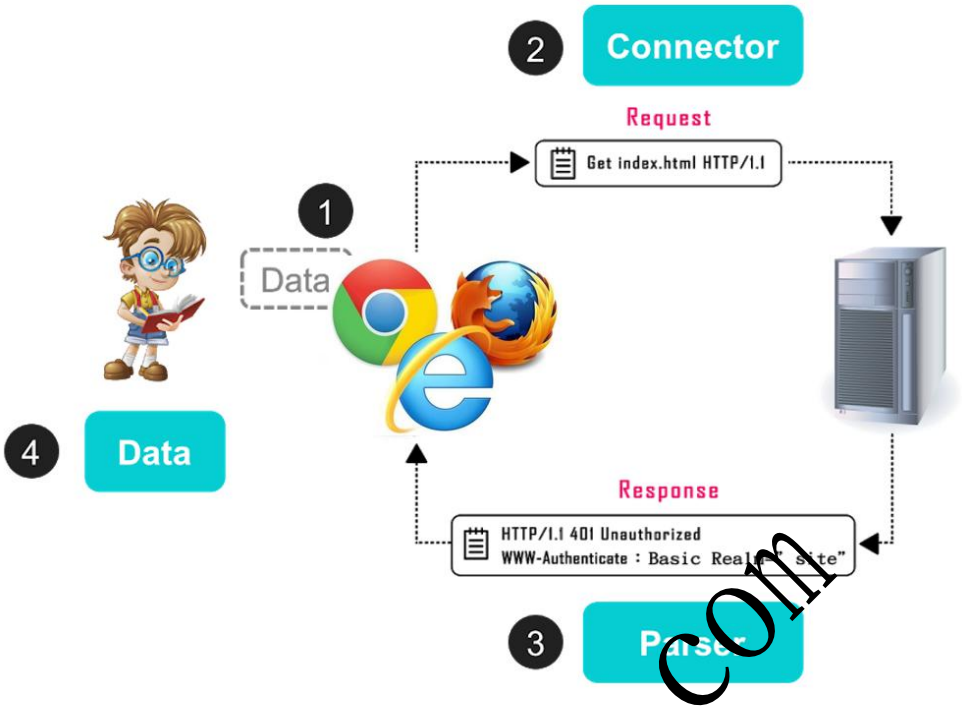
网络爬虫，是一种按照一定的规则，自动地抓取万维网信息的程序或者脚本。爬虫就是程序，或者进一步来说，是 Web 客户端程序。通过不断请求互联网上的网页内容，抓取自己想要的数据。

我们通过程序，模拟浏览器向服务器发送请求、获取信息、分析信息并储存我们想要的内容。

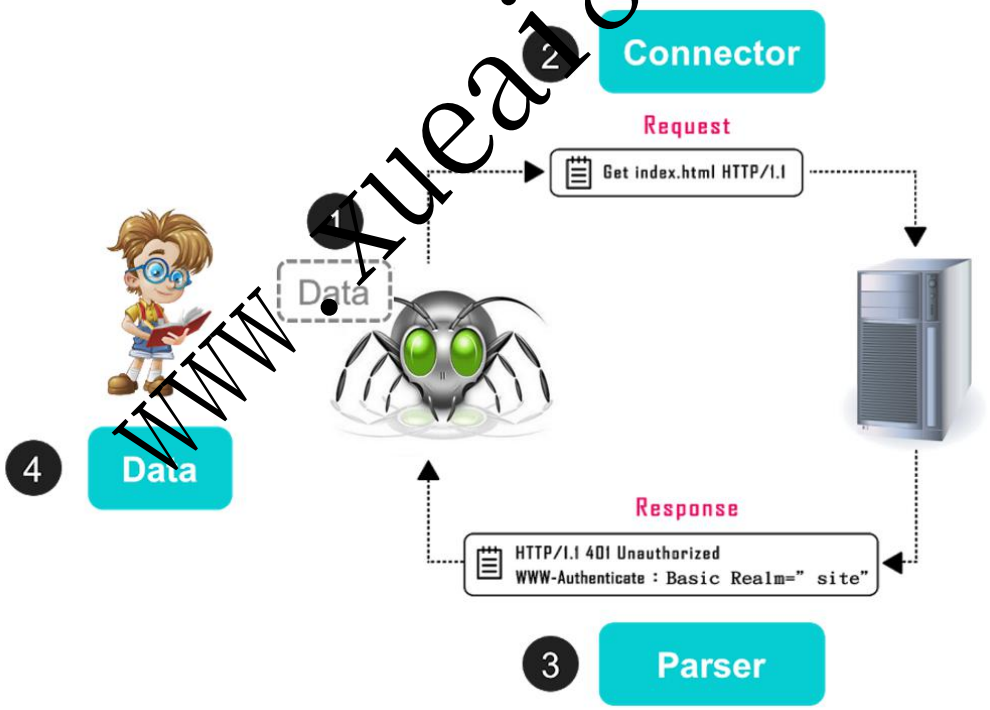
百度/google/搜狗等搜索引擎就是用采用爬虫的方法，定期搜索互联网上的链接并更新其服务器，这样我们才能通过搜索引擎搜到我们想要的信息。

1.3 爬虫的工作原理

浏览器工作原理:

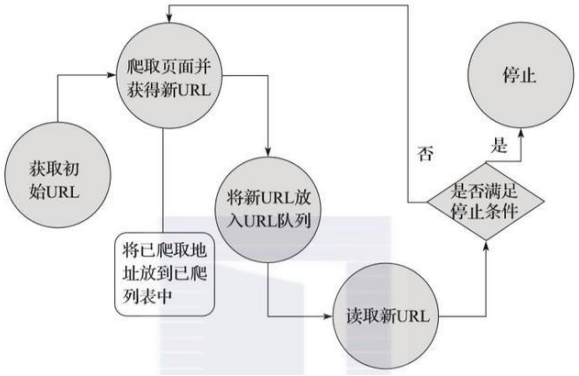


爬虫工作原理：

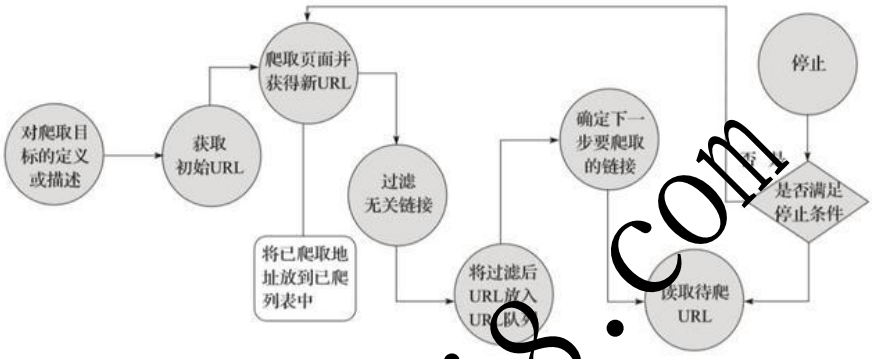


理解“通用网络爬虫”和“聚集网络爬虫”

通用网络爬虫：



聚焦网络爬虫

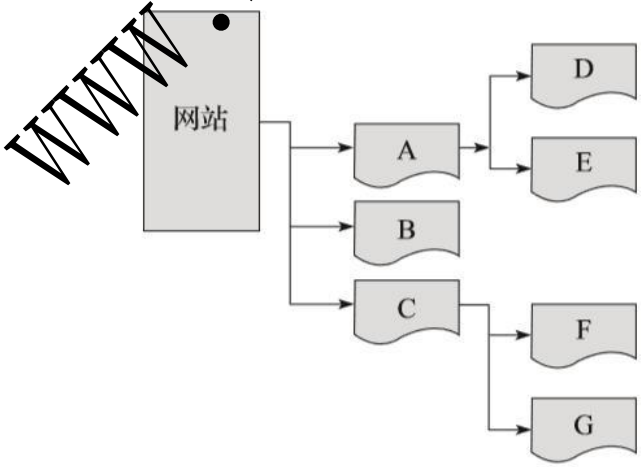


理解爬行策略

在网络爬虫爬取的过程中，在待爬取的 URL 列表中，可能有许多 URL 地址，那么这些 URL 地址，爬虫应该先爬取哪个，后爬取哪个呢？

在通用网络爬虫中，爬取的顺序并不是那么重要。但在聚焦网络爬虫中，爬取的顺序非常重要。而爬取的顺序，一般由爬行策略决定。

常见的爬行策略主要有：深度优先策略、广度优先策略、大站策略等



深度优先爬行策略，爬行顺序为：A -> D -> E -> B -> C -> F -> G。

广度优先爬行策略，爬行顺序为：A -> B -> C -> D -> E -> F -> G。

开发网络爬虫的语言有很多，常见的语言有：Python、Java、PHP、NodeJS、C、Go、Scala 等。这里我们使用 Python 语言。

要使用 Python 进行 web 抓取来提取数据，需要遵循以下基本步骤：

1) 找到要抓取的 URL
2) 检查 HTML 页面
3) 找到要提取的数据
4) 编写代码
5) 运行代码并提取数据
6) 以所需格式存储数据

1.4 技术储备

要能够独立编写爬虫程序，从互联网上抓取数据，需要掌握以下技术：

- Web 基础知识
  - HTTP 协议和“请求-响应模式”和
  - HTML、CSS 和 JavaScript
  - Ajax 原理、json 数据和前后端分离
- Python 编程
  - Python 语言基础
  - Python 常用库
  - Python 爬虫工具库
  - Python 数据库技术
- 数据库和 SQL 语言
  - mysql 数据库
  - SQL 语言

2、Web 基础知识

要使用爬虫从互联网上爬取数据，必须具备一定的 Web 基础知识，这包括：

1) 了解 HTTP 协议和“请求-响应模式”
2) 了解基本的 HTML、CSS 和 JavaScript 知识
3) 理解 Ajax 原理、json 数据和前后端分离

2.1 HTTP 协议

HTTP 协议是 Hyper Text Transfer Protocol（超文本传输协议）的缩写，是用于从万维网（WWW:World Wide Web ）服务器传输超文本到本地浏览器的传送协议。

HTTP 是一个基于 TCP/IP 通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

HTTP 协议工作于客户端-服务端架构为上。浏览器作为 HTTP 客户端通过 URL 向 HTTP 服务端即

WEB 服务器发送所有请求。Web 服务器根据接收到的请求后，向客户端发送响应信息。

HTTP 工作原理

HTTP 协议定义 Web 客户端如何从 Web 服务器请求 Web 页面，以及服务器如何把 Web 页面传送给客户端。

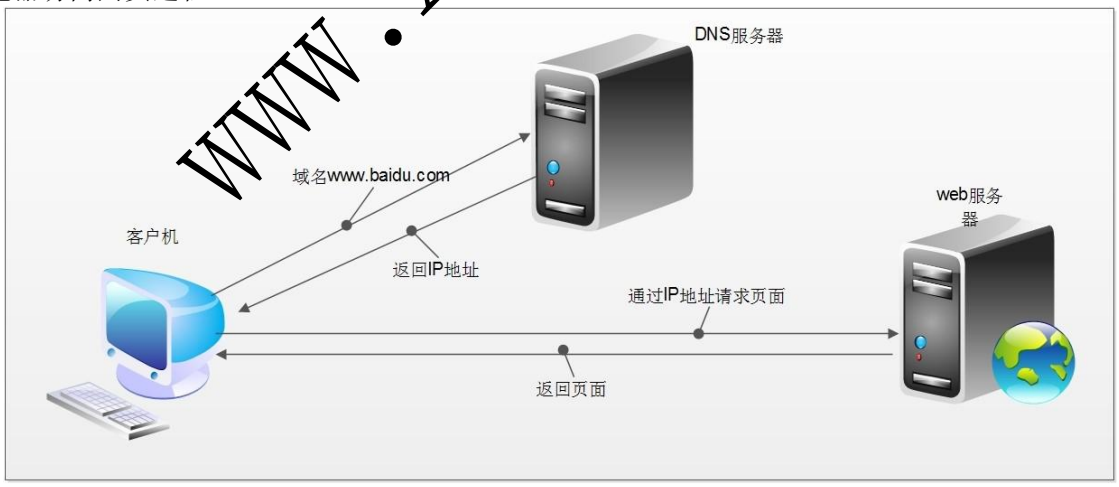
HTTP 协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

请求-响应模式：

以下是 HTTP 请求/响应的步骤：

1) 客户端连接到 Web 服务器
2) 发送 HTTP 请求
3) 服务器接受请求并返回 HTTP 响应
4) 释放连接 TCP 连接
5) 客户端浏览器解析 HTML 内容：客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的 HTML 文档和文档的字符集。客户端浏览器读取响应数据 HTML，根据 HTML 的语法对其进行格式化，并在浏览器窗口中显示。

浏览器访问网页过程：



HTTP 之 URL

HTTP 使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL 是一种特殊类型的 URI，包含了用于查找某个资源的足够的信息。

URL 全称是 UniformResourceLocator, 中文叫统一资源定位符，是互联网上用来标识某一处资源的地址。采用 URL 可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。

URL 一般由三部组成：

①协议(或称为服务方式)

②存有该资源的主机 IP 地址(有时也包括端口号)

③主机资源的具体地址。如目录和文件名等

以下面这个 URL 为例，介绍下普通 URL 的各部分组成：

http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name

从上面的 URL 可以看出，一个完整的 URL 包括以下几部分：

1.协议部分：该 URL 的协议部分为“http：”，这代表网页使用的是 HTTP 协议。在 Internet 中可以使用多种协议，如 HTTP，FTP 等等本例中使用的是 HTTP 协议。在"HTTP"后面的“//”为分隔符

2.域名部分：该 URL 的域名部分为“www.aspxfans.com”。一个 URL 中，也可以使用 IP 地址作为域名使用

3.端口部分：跟在域名后面的是端口，域名和端口之间使用“:”作为分隔符。端口不是一个 URL 必须的部分，如果省略端口部分，将采用默认端口

4.虚拟目录部分：从域名后的第一个“/”开始到最后一个“/”为止，是虚拟目录部分。虚拟目录也不是一个 URL 必须的部分。本例中的虚拟目录是“/news/”

5.文件名部分：从域名后的最后一个“/”开始到“？”为止，是文件名部分，如果没有“?”,则是从域名后的最后一个“/”开始到“#”为止，是文件部分，如果没有“？”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分。本例中的文件名是“index.asp”。文件名部分也不是一个 URL 必须的部分，如果省略该部分，则使用默认的文件名

6.锚部分：从“#”开始到最后，都是锚部分。本例中的锚部分是“name”。锚部分也不是一个 URL 必须的部分

7.参数部分：从“？”开始到“#”为止之间的部分为参数部分，又称搜索部分、查询部分。本例中的参数部分为“boardID=5&ID=24618&page=1”。参数可以允许有多个参数，参数与参数之间用“&”作为分隔符。

HTTP 之请求消息 Request

HTTP requests 和 HTTP responses 使用 RFC 822 普通的消息格式传输数据

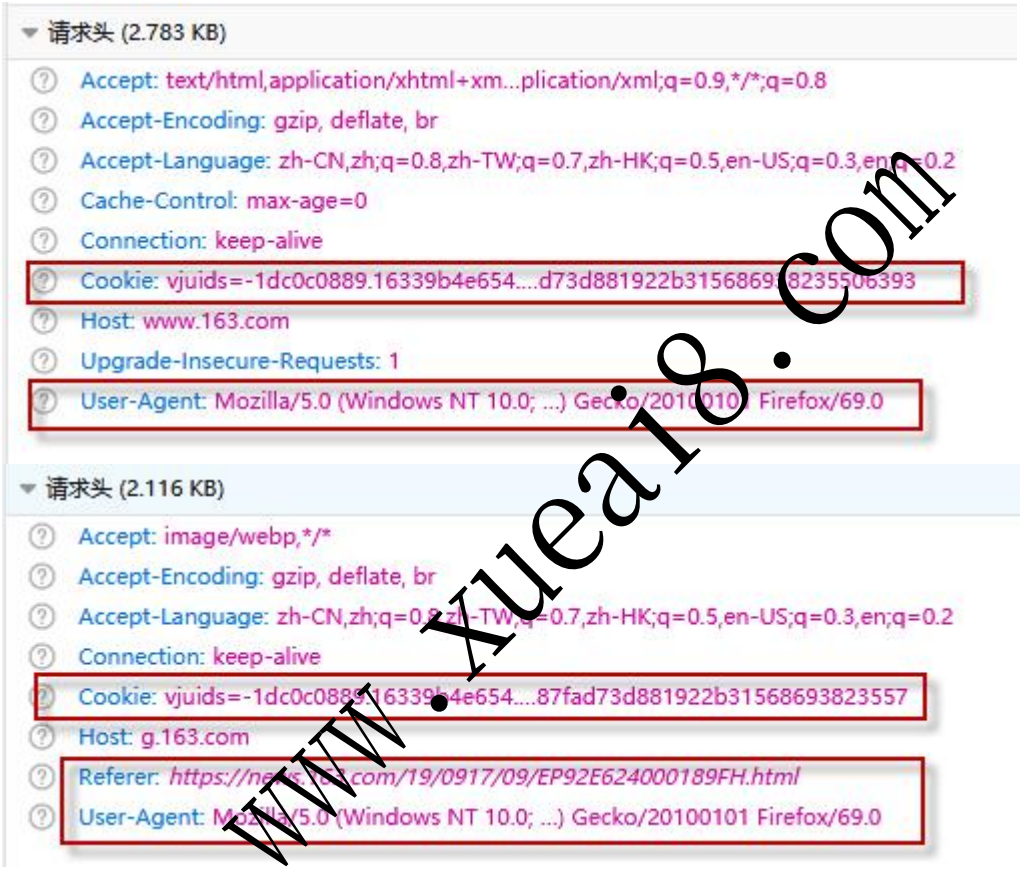
客户端发送一个 HTTP 请求到服务器，该请求消息由四部分组成：

- 一个起始行(请求行)
- 零个或多个头部字段，后跟 CRLF
- 一个空行，标明头部字段的结束
- 可选的一个 message-body（消息体，如查询数据）

下面是一个请求消息：

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.mysite.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

licenseID=string&content=string&/paramsXML=string
```

使用POST方法，向服务器传送表单数据



客户端请求格式说明如下表：

头部信息	描述
Accept	这个头部指定浏览器或其它客户端能处理的 MIME 类型。image/png 或 image/jpeg 可能是其中两个最常见的值
Accept-Charset	这个头部指定浏览器能使用来显示信息的字符集。例如：ISO-8859-1
Accept-Encoding	这个头部指定浏览器知道怎样去处理的编码类型。gzip 或 compress 是其中最有可能常见的两个值
Accept-Language	这个头部指定客户端的首选语言，例如，en,en-us,cn
Authorization	当访问受密码保护的 web 页面时，这个头部由客户端使用以标识其自身
Connection	这个头部说明客户端能否处理持久化 HTTP 连接。持久化连接允许客户端或其它浏览器在单个请求内获得多个文件。值 Keep-Alive 意味着应该使用持久化连接
Content-Length	这个头部只用于 POST 请求，给出 POST 数据的大小（字节）
Cookie	这个头部返回发送给服务器的 cookie，该 cookie 之前由服务器发送给了浏览器
Host	这个头部指定在原始 URL 中给出的主机和端口号
If-Modified-Since	这个头部指出客户端只想要在指定日期后修改过的页面。服务器发送一个代码，304 意味着没有修改过的头部，如果没有新的结果可用的话。
If-Unmodified-Since	这个头部与 If-Modified-Since 相反。它指定只有文档比指定的日期[illegible]操作才成功
Referer	这个头部用来指定引用 web 页面的 URL。例如，如果你在 page1，并单击一个链接到 page2，那么当浏览器请求 page2 时，page1 的 URL 就会包含在 Referer 头部。
User-Agent	这个头部标识发出请求的浏览器或其它客户端。可被用来根据不同类型的浏览器返回不同类型的内容

HTTP 之响应消息 Response

一般情况下，服务器接收并处理客户端发过来的请求后会返回一个 HTTP 的响应消息。

该响应消息也由四部分组成：

- 一个起始行(状态行)
- 零个或多个头部字段（消息报头），后跟 CRLF
- 一个空行，标明头部字段的结束
- 响应正文（消息体，如文件、查询输出）

下面是一个服务器响应消息：

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=utf-8

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
   <title>404 Not Found</title>
</head>
<body>
   <h1>Not Found</h1>
   <p>您请求的URL /t.html没有找到.</p>
</body>
</html>
```

HTTP 响应头说明如下表：

响应头	描述
Allow	这个响应头说明服务器支持的请求方法（GET,POST 等）
Cache-Control	这个响应头说明响应文档能被安全地缓存的情况。它的值可以 public、private 或 no-cache 等等。Public 意味着文档是可缓存的，private 意味着文档只用于单个用户，只能被保存为私有的缓存（不共享），而 no-cache 则意味着文档永远不被缓存
Connection	这个响应头指示浏览器在 HTTP 连接中是否使用持久连接。值 close 指明浏览器不使用持久 HTTP 连接，而值 keep-alive 则意味着使用持久连接
Content-Disposition	这个响应头会让浏览器询问用户以保存响应到磁盘文件中，以给定的名称
Content-Encoding	这个响应头说明在传输过程中页面使用哪种方式编码
Content-Language	这个响应头表示写文档时所使用的语言，例如，en,en-us,cn 等
Content-Length	这个响应头指出响应中的字节数。只有当浏览器使用一个持久 HTTP 连接（keep-alive）时才使用这个信息
Content-Type	这个响应头给出响应文档的 MIME 类型（Multipurpose Internet Mail Extension）
Expires	这个响应头指出内容失效的时间（即不再缓存）
Last-Modified	这个响应头说明文档最后修改的时间。客户端就可以据此缓存文档并在稍后的请求中通过 If-Modified-Since 请求头提供一个日期
Location	这个响应头应该被包含在所有状态码在 300 以内的响应中。这用来通知浏览器文档的地址，浏览器自动地重新连接到这个地址并获得新的文档
Refresh	这个响应头说明间隔多长时间浏览器再次请求一个更新的页面。可以指定一个以秒为单位的时间，过了该时间后一个页面应该被刷新
Retry-After	这个响应头可以与 503 (服务不可用)响应联合使用，以告诉客户端多长时间它可以重复请求
Set-Cookie	这个响应头指定一个与该页面相关联的 cookie

HTTP 之响应状态码

状态代码有三位数字组成，第一个数字定义了响应的类别。共分五种类别:

- 1xx：指示信息--表示请求已接收，继续处理
- 2xx：成功--表示请求已被成功接收、理解、接受
- 3xx：重定向--要完成请求必须进行更进一步的操作
- 4xx：客户端错误--请求有语法错误或请求无法实现
- 5xx：服务器端错误--服务器未能实现合法的请求

常见状态码：

- 200 OK 客户端请求成功
- 400 Bad Request 客户端请求有语法错误，不能被服务器所理解
- 401 Unauthorized 请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
- 403 Forbidden 服务器收到请求，但是拒绝提供服务
- 404 Not Found 请求资源不存在，eg：输入了错误的 URL
- 500 Internal Server Error 服务器发生不可预期的错误
- 503 Server Unavailable 服务器当前不能处理客户端的请求，一段时间后可能恢复正常

服务器端不总是返回.html 网页，它还会返回图片、文件、json 数据、pdf、excel 等。客户端浏览器对不同类型的返回（响应）数据有不同的处理方法。客户端怎么识别从服务器返回的响应数据类型呢？它是根据响应消息 Response 中的”Content-Type”头部信息来识别的。例如：

- Content-Type: text/html
- Content-Type: image/jpeg, image/gif
- Content-Type: application/json

HTTP 请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法：GET, POST 和 HEAD 方法。

HTTP1.1 新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

GET　请求指定的页面信息，并返回实体主体。

HEAD　　类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头

POST　　向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。

PUT　从客户端向服务器传送的数据取代指定的文档的内容。

DELETE　请求服务器删除指定的页面。

CONNECT　HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

OPTIONS　允许客户端查看服务器的性能。

TRACE　　回显服务器收到的请求，主要用于测试或诊断。

GET 和 POST 请求的区别

1、GET 提交，请求的数据会附在 URL 之后（就是把数据放置在 HTTP 协议头中），以?分割 URL 和传输数据，多个参数用&连接；例 如：login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0%E5%A5%BD。如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密，得出如： %E4%BD%A0%E5%A5%BD，其中%XX 中的 XX 为该符号以 16 进制表示的 ASCII。

POST 提交：把提交的数据放置在是 HTTP 包的包体中。

因此，GET 提交的数据会在地址栏中显示出来，而 POST 提交，地址栏不会改变

2、传输数据的大小：首先声明：HTTP 协议没有对传输的数据大小进行限制，HTTP 协议规范也没有对 URL 长度进行限制。

而在实际开发中存在的限制主要有：

GET:特定浏览器和服务器对 URL 长度有限制，例如 IE 对 URL 长度的限制是 2083 字节(2K+35)。对于其他浏览器，如 Netscape、FireFox 等，理论上没有长度限制，其限制取决于操作系统的支持。

因此对于 GET 提交时，传输数据就会受到 URL 长度的限制。

POST:由于不是通过 URL 传值，理论上数据不受限。但实际各个 WEB 服务器会规定对 post 提交数据大小进行限制，Apache、IIS6 都有各自的配置。

3、安全性

POST 的安全性要比 GET 的安全性高。比如：通过 GET 提交数据，用户名和密码将明文出现在 URL 上，因为(1)登录页面有可能被浏览器缓存；(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用 GET 提交数据还可能会造成 Cross-site request forgery 攻击

GET 方法

请注意，查询字符串（名称/值对）是在 GET 请求的 URL 中发送的：

```
/test/demo_form.asp?name1=value1&name2=value2
```

有关 GET 请求的其他一些注释：

- GET 请求可被缓存
- GET 请求保留在浏览器历史记录中
- GET 请求可被收藏为书签
- GET 请求不应在处理敏感数据时使用
- GET 请求有长度限制
- GET 请求只应当用于取回数据

POST 方法

请注意，查询字符串（名称/值对）是在 POST 请求的 HTTP 消息主体中发送的：

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

有关 POST 请求的其他一些注释：

- POST 请求不会被缓存
- POST 请求不会保留在浏览器历史记录中
- POST 不能被收藏为书签
- POST 请求对数据长度没有要求

查看浏览器设置：https://www.whatismybrowser.com/。这个网站可以让服务器测试浏览器的属性。
查看头部信息：https://www.whatismybrowser.com/detect/what-http-headers-is-my-browser-sending

2.2 网站搭建工具

一站式网站搭建工具：xampp
https://www.apachefriends.org/index.html

2.3 网页设计语言 HTML

HTML：超文本标记语言

浏览器开发工具：
火狐：查看源代码 Fn+F12，查看元素：Ctrl+Shift+I
Chrome：查看源代码：Ctrl+U，查看元素：Ctrl+Shift+I

重要的概念：

- 标签
  - html、head、body、title
  - div
  - h1 - h6
  - p
  - span
  - img
  - table、th、tr、td
  - ...
- 超链接
  - a

2.4 级联样式表 CSS

HTML 用来定义文档结构和主义，而 CSS 用来定义文档样式。

重要的概念：

- CSS 选择器
  - id 选择器
  - 标签选择器
  - 类选择器

了解 CSS 选择器的原因：相同的选择器语法可在 Python 中使用，用来从 HTML 页面中快速查找和获取元素。在一些 HTML 元素上单击右键，然后按下“拷贝，拷贝选择器”。

2.5 JavaScript 网页编程语言

也叫做网页脚本语言。
JavaScript 编写的脚本代码嵌入在 HTML 页面中，执行动态操作，实现动态效果。
Ajax 技术是 JavaScript 的一个重要应用。

2.6 Ajax 技术和 JSON 数据格式

Ajax 技术：页面无刷新的技术。使用 JavaScript 实现。
实现 Web 程序的前后端分离。
JSON 数据格式：

```
{
    "name":"张三",
    "age":23,
    "hobby":["抽烟","喝酒","烫头"]
}
```

3、Python3 实现网络爬虫

数据时代，数据获取的方式多种多样。为什么要使用爬虫？到底什么是爬虫？

3.1 爬虫开发工具选择

根据自己的喜好，自由选择开发工具：

- Jupyter notebook
- PyCharm

3.2 Python 常用工具库

以下 Python 库常用到：

- json：用于分析 json 格式的数据
- re：用于进行正则表达式匹配
- pandas：用于分析数据
- pymysql： 关系型数据库

3.3 Python 网络工具库:requests

我们会用到的 Python 网络库有：requests：用于向服务器发送请求并获取数据。Requests 是一个 Python 模块，可以使用它发送各种 HTTP 请求。它是一个易于使用的库，具有许多特性，从在 url 中传递参数到发送自定义头部和 SSL 验证。

使用 requests 可以发送 HTTP/1.1 请求。可以使用简单的 Python 字典添加头、表单数据、multi-part 文件和参数，并以相同的方式访问响应数据。

参考：http://docs.python-requests.org/en/master/
在命令行窗口下，执行安装命令：pip install -U requests

简单示例：

```
import requests

url = "http://www.163.com"
r = requests.get(url)
print(type(r))        # requests.models.Response
# r.encoding = "utf8"   # 设置编码
print(r.text)
```

说明：

- requests.get 方法返回一个 requests.Response Python 对象，包含从服务器返回的 HTTP 响应信息，requests 负责解析
- r.text 包含 HTTP response 内容体，以文本格式
- 除了 requests.get()方法，还有 request.post()方法，以及通用的 requests.request('GET', url)方法

response 对象：

- response.text　返回 headers 中的编码解析的结果，可以通过 r.encoding = 'gbk'来变更解码方式
- response.encoding　　页面编码
- response.content　　返回二进制结果
- response.json()　　返回 JSON 格式，可能抛出异常
- response.status_code　　状态码 200
- response.reason　　状态文本 OK
- response.cookies　　响应头中的 cookie
- response.raw　　返回原始 socket response，需要加参数 stream=True (requests.get(url, stream=True))
- response.headers　　HTTP 响应头
- response.headers["Content-Type"]　　响应头中指定的 key 对应的 value
- response.request　　请求信息作为 Python 对象保存在 response.request 中
- response.request.headers　　HTTP 请求头
- response.request.url　　HTTP 请求 url
- response.is_redirect　　网页是否被重定向了

- response.elapsed　　获取从发送请求到返回响应之间所经过的时间
- response.url　　实际响应的 URL（可能因为重定向而与请求的不同）
- response.history

获取状态码、请求头等信息：

```
"""
简单爬取网页内容：爬取 豆瓣电影 网站的首页内容
查看响应对象 Response
"""
import requests

# 豆瓣电影首页的 url
url = "https://movie.douban.com/"

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url)

# 获取响应的状态码和状态文本
# print(response.status_code)
# print(response.reason)

# 查看响应的头部信息
# print(response.headers)  # {}
# print(response.headers["Content-Type"])

# 查看请求的信息(作为 Python 对象保存在 response.request 中)
# print(response.request.url)
# print(response.request.headers)
# print(response.request.headers["User-Agent"])

# 响应的网页内容
print(response.text)
```


3.4 Python HTML 解析库:Beautiful Soup

Beautiful Soup：这个库用来解析、结构化和组织 HTML，并以易于处理的 Python 结构呈现出来。解析从服务器端返回的 HTML 内容，我们使用"Beautiful Soup"库。

安装：pip install -U beautifulsoup4

参考文档：https://www.crummy.com/software/BeautifulSoup/bs4/doc/

BeautifulSoup 默认支持 Python 的标准 HTML 解析库，但是它也支持一些第三方的解析库：

解析器	使用方法	优势	劣势
Python标准库	`BeautifulSoup(markup, "html.parser")`	• Python的内置标准库 • 执行速度适中 • 文档容错能力强	• Python 2.7.3 or 3.2.2)前的版本中文档容错能力差
lxml HTML 解析器	`BeautifulSoup(markup, "lxml")`	• 速度快 • 文档容错能力强	• 需要安装C语言库
lxml XML 解析器	`BeautifulSoup(markup, ["lxml", "xml"])` `BeautifulSoup(markup, "xml")`	• 速度快 • 唯一支持XML的解析器	• 需要安装C语言库
html5lib	`BeautifulSoup(markup, "html5lib")`	• 最好的容错性 • 以浏览器的方式解析文档 • 生成HTML5格式的文档	• 速度慢 • 不依赖外部扩展

使用 Beautiful Soup 从创建 BeautifulSoup 对象开始

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html)
```

Beautiful Soup 的主要任务是将 HTML 内容转换到一个基于树的表示。一旦创建了一个 BeautifulSoup 对象，就可使用该对象的两个方法来抓取页面中的数据：

- find(name,attrs,recursive,string,**keywords);
- find_all(name,attrs,recursive,string,limit,**keywords)

这两个方法的目的就是在 HTML 树中查找元素，其中各参数的含义如下：

- name：定义想要查找的标签名。可以传入一个字符串，或一组标签。如果这个参数设为空字符串，会选择所有的元素。
- attrs：Python 属性字典，匹配与这些属性相匹配的 HTML 元素。
- recursive：是一个 boolean 参数，控制搜索深度。如果设为 True，这也是默认值，会进行递归查找。如果设为 False，将只在直接子元素查找。
- string：用来执行基于元素文本内容的匹配。
- limit：只用在 find_all 方法中，用来限制获取的元素的数量。注意，find 在功能上等价于调用 limit 为 1 的 find_all 方法。find 方法直接返回获取的元素，而 find_all 总是返回一个 list，即使只包含一个元素。另外，如果 find_all 找不到匹配的内容，会返回一个空 list，而 find 会返回 None。
- **keywords：是一个特殊情况。这部分方法签名指明可以添加任意多个额外的命名参数(可变参数)，然后简单地被用作属性过滤器。

使用 BeautifulSoup 解析网页内容示例：

```
"""
简单爬取网页内容：爬取 豆瓣电影 网站的首页内容
使用 BeautifulSoup 解析返回的网页内容
"""
import requests
from bs4 import BeautifulSoup

# 豆瓣电影首页的 url
```

```
url = "https://movie.douban.com/"

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url)

# 响应的网页内容
html = response.text
# print(html)

# 创建 BeautifulSoup 对象
soup = BeautifulSoup(html, "lxml")

# 格式化输出内容
print(soup.prettify())
# print(soup.prettify(formatter=None))
```

两个方法 find 和 find_all 都返回 Tag 对象。使用这个对象，可以：

- 访问 name 属性获取标签名；
- 访问 contents 属性获取一个 Python list，包含该标签的子元素(直接子标签)作为一个 list。
- children 属性提供了一个子元素迭代；descendants 属性返回一个迭代器，以递归的方式包含所有的标签的子元素。
- 还可以使用 parent 和 parents 属性向上遍历 HTML 树。如果要找兄弟元素，使用 next_sibling、previous_sibling 和 next_siblins、previous_siblings。
- 将该 Tab 对象转 ram 为一个 string，同时显示该标签及其 HTML 内容为字符串。
- 通过 Tag 对象的 attrs 属性访问该元素的属性，为了方便，还可以直接使用该 Tag 对象本身作为 dictionary。
- 使用 text 属性来获得该 Tag 对象的内容(text 部分，不带 HTML 标签)。
- 如果标签只有一个子元素，并且该子元素是简单文本，那么还可以使用 string 属性来获取该文本内容。不过，如果该标签包含其它嵌套的 HTML 标签，string 将返回 None 而 text 会递归地取回所有的文本。

最后，并不是所有的 find 和 find_all 搜索需要从原始的 BeautifulSoup 对象开始。每个 Tab 对象本身可被用作一个新的 root，从这里再开始一个新的搜索。

Tag 对象方法：

- find()
- find_all()
- find_parent()
- find_parents()
- find_next_sibling()
- find_next_siblings()
- find_previous_sibling()
- find_previous_siblings()
- find_previous()
- find_all_previous()
- find_next()

- find_all_next()

3.5 爬取网页内容

3.5.1 案例：爬取“东方财富网”股票吧评论数据

URL 地址：

http://guba.eastmoney.com/

爬取内容：

“热门”贴子

代码实现：

```
"""
东方财富网 股票吧 热门帖子

http://guba.eastmoney.com/
"""
import requests
from bs4 import BeautifulSoup

# url
url = "http://guba.eastmoney.com/"

# headers
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3"
}

# 请求
response = requests.get(url, headers=headers)

# 输出
# print(response.text)

# soup
soup = BeautifulSoup(response.text, "lxml")

# div.balist
div_balist = soup.find(name="div", class_="balist")
# print(div_balist)
```

```
# ul.newlist
ul_newlist = div_balist.find(name="ul", class_="newlist")
# print(ul_newlist)

# li_all
li_all = ul_newlist.find_all(name="li")

print(" 阅读，评论，标题，作者，更新时间 ")
for li in li_all:
    print(li.find_all("cite")[0].text.strip(), end=", ")
    print(li.find_all("cite")[1].text.strip(), end=", ")
    print(li.find("a", class_="balink").text, end=", ")
    print(li.find("a", class_="note").get("title"), end=", ")
    print(li.find("cite", class_="aut").find("font").text, end=", ")
    print(li.find("cite", class_="last").text)
```

执行以上代码，输出结果如下：

```
 阅读，评论，标题，作者，更新时间
12175，23，财经评论吧，习近平：坚定信心埋头苦干奋勇争先 谱写新时代中原更加出彩的[illegible]，财经评论，09-18 20:57
242049，222，财经评论吧，险资权益投资比例提升细则落地在即 长期增量资金有望加速入[illegible]，财经评论，09-18 16:03
249421，151，财经评论吧，今年来境外流入A股资金规模达万亿级 三大因素决定不会“大进大出”，财经评论，09-18 20:29
7834，2，股市实战吧，【股吧日报】沪指震荡有望重拾升势 白酒板块还有多大上涨空间？，股市News，09-18 20:13
25695，193，财经评论吧，北上资金今日净流入39.21亿 净买入贵州茅台4.5[illegible]亿，财经评论，09-18 20:48
14452，22，财经评论吧，官方回应换脸软件“ZAO”涉嫌侵权：已制定多项[illegible]准，财经评论，09-18 20:43
4445，3，股市实战吧，【每日股市吐槽】白酒股涨跌都是利好啊 ，文[illegible]09-18 18:46
34315，70，财经评论吧，国务院：取消内燃机等13类工业产品生产许可证[illegible]理，财经评论，09-18 20:51
65838，145，财经评论吧，今晚油价要涨了！加满一箱油 多花5[illegible]，财经评论，09-18 20:58
4012，6，股市实战吧，【访谈精彩观点】市场迎来最好的[illegible]股吧访谈，09-18 18:46
11660，7，股市实战吧，量能大幅萎缩止跌企稳下跌中继[illegible]里寻宝，09-18 20:04
7635，5，股市实战吧，这些异象短期要关注，皮夹克少年[illegible]182，09-18 17:17
6601，8，股市实战吧，静待大盘稳步收复失地，[illegible]09-18 18:51
22801，36，财经评论吧，抓了1.8万人！涉案金额[illegible]亿！警方铲除百亿“套路贷” 大数据泄露不得不防，财经评论，09-18 20:37
50166，61，财经评论吧，第六届世界互联网大会将[illegible]10月20日至22日在浙江乌镇举行，财经评论，09-18 18:10
```


3.5.2 案例：爬取豆瓣电影网站首页的“一周口碑榜”数据

该数据是网站服务器返回给客户端的静态网页数据。

1）网页结构分析：

一周口碑榜中的数据是以表格形式布局的。每部电影一行（tr）。

2）网页 HTML 代码分析：

网页代码如下：

```
<div class="billboard-bd">
  <table>
    <tbody>
      <tr>
        <td class="order">1</td>
        <td class="title">
          <a onclick="moreurl(this, {from:'mv_rk'})" href="https://movie.douban.com/subject/26709258/">罗小黑战记</a> event
        </td>
      </tr>
      <tr>
        <td class="order">2</td>
        <td class="title">
          <a onclick="moreurl(this, {from:'mv_rk'})" href="https://movie.douban.com/subject/30210691/">极限逃生</a> event
        </td>
      </tr>
      <tr>
        <td class="order">3</td>
        <td class="title">
          <a onclick="moreurl(this, {from:'mv_rk'})" href="https://movie.douban.com/subject/30391300/">我们是小僵尸</a> event
        </td>
      </tr>
      <tr> ... </tr>
      <tr> ... </tr>
      <tr> ... </tr>
      <tr> ... </tr>
      <tr> ... </tr>
      <tr> ... </tr>
      <tr> ... </tr>
    </tbody>
  </table>
</div>
```


3）需要爬取的数据说明：

需要爬取以下字段：

- order：榜单排序；
- title：电影名
- id：电影 id
- detail_url：电影详情页 url

4）代码实现：

实现代码如下所示：

```
"""
爬取豆瓣电影网站首页的“一周口碑榜”数据。
1、实现首页数据爬取（这里是表格结构）
2、实现深度为 2 的嵌套爬取（先爬取口碑电影 URL，再根据每个 URL 爬取电影详情数据）

需要爬取以下字段：
- order：榜单排序；
- title：电影名
- id：电影 id
- detail_url：电影详情页 url
"""
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time, random

# ----------------------------------------------------------------------------------------
# 豆瓣电影首页的 url
url = "https://movie.douban.com"

# 假装自己是浏览器
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36",
    "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exch
ange;v=b3"
}

 # 设置一个有效的代理 IP，隐藏自己的真实 IP
proxies = {'http': '120.83.109.129:9999'}

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url, headers=headers, proxies=proxies)

# 构造 BeautifulSoup
soup = BeautifulSoup(response.text, "html.parser")

# 先找到表格
table = soup.find("div", attrs={"class": "billboard-bd"}).find("table")
# print(table)

# 再找到表格下的所有行(tr)
orders = []
titles = []
detail_urls = []
ids = []

tr_all = table.find_all("tr")
```

```
for tr in tr_all:
    td_all = tr.find_all("td")
    orders.append(td_all[0].get_text())
    titles.append(td_all[1].get_text())
    movie_url = td_all[1].find("a").get("href")
    detail_urls.append(movie_url)
    movie_id = movie_url.split("/")[-2:-1][0]
    ids.append(movie_id)

# 构造 DataFrame
data = {
    "id": ids,
    "title": titles,
    "order": orders,
    "detail_url": detail_urls
}

df = pd.DataFrame(data)
print(df)
```

输出结果如下所示：

	id	title	order	detail_url
0	26709258	罗小黑战记	1	https://movie.douban.com/subject/26709258/
1	30210691	极限逃生	2	https://movie.douban.com/subject/30210691/
2	30391300	我们是小僵尸	3	https://movie.douban.com/subject/30391300/
3	30206464	七个会议	4	https://movie.douban.com/subject/30206464/
4	30191950	飞翔吧！埼玉	5	https://movie.douban.com/subject/30191950/
5	30330264	最大的小小农场	6	https://movie.douban.com/subject/30330264/
6	33403736	巧虎大飞船历险记	7	https://movie.douban.com/subject/33403736/
7	27040109	狂躁节拍	8	https://movie.douban.com/subject/27040109/
8	34781114	爱哭鬼上学记	9	https://movie.douban.com/subject/34781114/
9	27114963	奇妙的家族	10	https://movie.douban.com/subject/27114963/

3.5.3 案例：爬取豆瓣电影网站首页的“正在热映|即将上映”的电影数据

通过分析，这也是静态数据。但因为网页开发人员自身的问题，返回的是不规范的网页。因此需要额外处理。

1）网页结构分析：

2）网页 HTML 代码分析：

3）需要爬取的数据：

- ❑ movie_id：25779217
- ❑ movie_title："诛仙 I"
- ❑ movie_release：2019
- ❑ movie_rate：5.3
- ❑ movie_star：30
- ❑ movie_duration："101 分钟"
- ❑ movie_region："中国大陆"
- ❑ movie_director："程小东"
- ❑ movie_actors："肖战 / 李沁 / 孟美岐"
- ❑ movie_rater：203691
- ❑ movie_detail：https://movie.douban.com/subject/25779217/?from=showing
- ❑ movie_poster：https://img3.doubanio.com/view/photo/s_ratio_poster/public/p2567346094.jpg

4）代码实现：

爬取豆瓣电影首页的“正在热映 | 即将上映”电影数据。在爬取过程中注意以下几个坑：

- 这个网页本身 HTML 写得有问题，包装每部电影的 li 标签，结束标签
不成对，都在最后，造成连环嵌套；（这需要仔细分析网页的 HTML 结构才能发现）
- 这样获取的 li 有很多是空标签，在解析时需要排除掉；
- 很多爬取的数据是在 li 标签的属性中的，例如，data-title="诛仙 I"。所以要使用 BeautifulSoup 的 get("属性名")来取属性值。

参考实现：

```
"""
爬取豆瓣电影首页的“正在热映 | 即将上映”电影数据。

需要爬取的数据：
movie_id：25779217
movie_title："诛仙 I"
movie_release：2019
movie_rate：5.3
movie_star：30
movie_duration："101 分钟"
movie_region："中国大陆"
movie_director："程小东"
movie_actors："肖战 / 李沁 / 孟美岐"
movie_rater：203691
movie_detail：https://movie.douban.com/subject/25779217/?from=showing
movie_poster：https://img3.doubanio.com/view/photo/s_ratio_poster/public/p2567346094.jpg

注：通过分析网页结构可知，最后两个数据来自于嵌套的标签，其余的数据来自于属性值
"""
import requests
from bs4 import BeautifulSoup
import pandas as pd

pd.set_option('display.max_columns', None)  # 显示所有列
pd.set_option('display.max_rows', None)     # 显示所有行
pd.set_option('display.width', 160)         # 行宽，默认 80
pd.set_option('max_colwidth', 160)          # 设置 value 的显示长度为 80，默认为 50

# 豆瓣电影首页的 url
url = "https://movie.douban.com/"

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url)

# 响应的网页内容
html = response.text
# print(html)

# 创建 BeautifulSoup 对象
soup = BeautifulSoup(html, "html.parser")

# 解析返回的网页内容（注：这里的 html 网页写的有问题，li 嵌套不正确）
# 先找到最外围的标签
```

```
div_screening = soup.find(attrs={"id": "screening"})
# print(div_screening)

# 再找里面的每个电影项，并保存到各个数组中，以便构造 DataFrame
movie_ids = []
movie_titles = []
movie_releases = []
movie_rates = []
movie_stars = []
movie_durations = []
movie_regions = []
movie_directors = []
movie_actorses = []
movie_raters = []
movie_details = []
movie_posters = []

li_items = div_screening.find_all(name="li", attrs={"class": "ui-slide-item"})
for item in li_items:
    if item.get("data-ticket"):    # get 方法，取当前 Tag 的属性
        data_ticket_url = item.get("data-ticket")
        movie_id = data_ticket_url.split("=")[1]
        movie_ids.append(movie_id)
        movie_titles.append(item.get("data-title"))
        movie_releases.append(item.get("data-release"))
        movie_rates.append(item.get("data-rate"))
        movie_stars.append(item.get("data-star"))
        movie_durations.append(item.get("data-duration"))
        movie_regions.append(item.get("data-region"))
        movie_directors.append(item.get("data-director"))
        movie_actorses.append(item.get("data-actors"))
        movie_raters.append(item.get("data-rater"))
        # 详情链接
        movie_href = "https://movie.douban.com/subject/" + movie_id + "/?from=showing"
        movie_details.append(movie_href)
        # 海报
        poster = item.find(name="img")
        movie_posters.append(poster.get("src"))


# 构造 DataFrame
data = {"id": movie_ids,
        "title": movie_titles,
        "release": movie_releases,
        "rate": movie_rates,
        "star": movie_stars,
        "duration": movie_durations,
        "region": movie_regions,
        "director": movie_directors,
        "actors": movie_actorses,
```

```
            "rater": movie_raters,
            "detail": movie_details,
            "poster": movie_posters
            }
movie_df = pd.DataFrame(data)

# 输出
print(movie_df)
```

3.5.4 作业：爬取维基百科“权利的游戏”电视剧集列表

https://zh.wikipedia.org/wiki/权力的游戏剧集列表

各季列表 [编辑]

第一季（2011年） [编辑]

主条目：权力的游戏 (第一季)

总集数	集数	标题	导演	编剧	首播日期	美国收视人数（百万）
1	1	凛冬将至 Winter Is Coming	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月17日	2.22[8]
2	2	国王大道 The Kingsroad	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月24日	2.20[9]
3	3	雪诺大人 Lord Snow	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月1日	2.44[10]
4	4	跛子、杂种和废物 Cripples, Bastards, and Broken Things	布莱恩·寇克	布莱恩·克格曼	2011年5月8日	2.45[11]
5	5	狮狼之争 The Wolf and the Lion	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月15日	2.58[12]
6	6	黄金之冕 A Golden Crown	丹尼尔·米纳汉	故事：大卫·贝尼奥夫 & D·B·魏斯 剧本：珍·伊斯塔森 & 大卫·贝尼奥夫 & D·B·魏斯	2011年5月22日	2.44[13]
7	7	成王败亡 You Win or You Die	丹尼尔·米纳汉	大卫·贝尼奥夫 & D·B·魏斯	2011年5月29日	2.40[14]
8	8	彼剑之尖 The Pointy End	丹尼尔·米纳汉	乔治·R·R·马丁	2011年6月5日	2.72[15]
9	9	贝勒圣堂 Baelor	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月12日	2.66[16]
10	10	血火同源 Fire and Blood	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月19日	3.04[17]

URL 地址：

https://zh.wikipedia.org/wiki/权利的游戏剧集列表

要求：

编写 Python 爬虫程序，将《权利的游戏》所有剧集爬取下来，并分别存入.csv 文件和数据库中。

3.6 爬取多级网页内容

网站中的网页是通过超链接联系起来的。通常先爬取首页，提取 URL；然后再遍历首页中的 URL，依次请求每个 URL，爬取二级页面；然后同样方式爬取三级页面，...。

3.6.1 案例：爬取豆瓣电影网站首页的“一周口碑榜”电影及其详情

在首页中的“一周口碑榜”只是列出了电影的基本信息和 url 连接。我们需要爬取这些 url 以获得二级页面的电影详情页。

1）分析网页内容

2）分析网页 HTML 结构

在首页爬取每部电影的 URL：

再请求详情页，爬取每部电影的详细数据以及豆瓣评分、星级和评分人数：

3）需要爬取的字段

需要爬取以下字段：

- title：电影名
- id：电影 id
- director：导演
- writer：编剧
- actors：主演
- genre：类型
- region：制片国家/地区
- language：语言
- release：上映日期
- duration：片长
- alias：别名
- score：豆瓣评分
- stars：星级
- rater：评分人数

4）代码实现

以下为实现的 Python 代码：

```
"""
爬取豆瓣电影网站首页的“一周口碑榜”数据。
1、实现首页数据爬取（这里是表格结构）
2、实现深度为 2 的嵌套爬取（先爬取口碑电影 URL，再根据每个 URL 爬取电影详情数据）
```

```
"""
import requests
from bs4 import BeautifulSoup
import pandas as pd
import time
import random
import re

# ------------------------------------------------------------------------------------
pd.set_option('display.max_columns', None)   # 显示所有列
pd.set_option('display.max_rows', None)      # 显示所有行
pd.set_option('display.width', 200)          # 行宽，默认 80
pd.set_option('max_colwidth', 80)            # 设置 value 的显示长度为 80，默认为 50

# 高匿代理 IP 池
proxies_pool = [
        {"http": "120.83.106.21:9999"},
        {"http": "58.253.157.72:9999"},
        {"http": "120.83.109.129:9999"},
        {"http": "114.239.148.184:808"},
        {"http": "1.197.204.179:9999"},
        {"http": "114.239.147.75:808"},
        {"http": "112.85.168.67:9999"},
        {"http": "115.53.33.147:9999"},
        {"http": "1.198.73.64:9999"},
        {"http": "123.163.96.94:9999"}
    ]

# 随机获取一个有效的代理 IP
def get_one_proxies(proxies_pool):
    index = random.randint(0, len(proxies_pool)-1)
    valided_proxies = proxies_pool[index]
    print("正在尝试:", end=" ")
    print(valided_proxies)
    while not valid_proxies(valided_proxies):
        index = random.randint(0, len(proxies_pool)-1)
        valided_proxies = proxies_pool[index]
        print("正在尝试:", end=" ")
        print(valided_proxies)
        time.sleep(random.randint(1,9))
    print("有效的代理 IP：", end=" ")
    print(valided_proxies)
    return valided_proxies

# 检验代理是否有效
def valid_proxies(proxies):
    my_proxies = proxies
    # 假装自己是浏览器
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

```
Chrome/74.0.3729.169 Safari/537.36",
            "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exch
ange;v=b3"
        }
        try:
            # 使用代理 IP 请求百度首页。如果返回状态码是 200，说明这是一个有效的代理 IP
            r = requests.get("http://www.baidu.com", proxies=my_proxies, headers=headers)
            return r.status_code == 200
        except Exception as e:
            return False

# ----------------------------------------------------------------------------------------
# 豆瓣电影首页的 url
url = "https://movie.douban.com"

# 假装自己是浏览器
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36",
    "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exch
ange;v=b3"
}

 # 设置一个有效的代理 IP，隐藏自己的真实 IP
# proxies = get_one_proxies(proxies_pool)
proxies = {'http': '120.83.109.129:9999'}

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url, headers=headers, proxies=proxies)

# 构造 BeautifulSoup
soup = BeautifulSoup(response.text, "html.parser")

# 准备存放数据的列表
orders = []
titles = []
ids = []
detail_urls = []

# 先找到表格
table = soup.find("div", attrs={"class": "billboard-bd"}).find("table")
# print(table)

# 再找到表格下的所有行(tr) -  一级页面
tr_all = table.find_all("tr")
for tr in tr_all:
    td_all = tr.find_all("td")
    # order
```

```
        order = td_all[0].get_text()
        orders.append(order)
        # title
        title = td_all[1].get_text()
        titles.append(title)
        # 电影详情 url 链接
        movie_url = td_all[1].find("a").get("href")
        detail_urls.append(movie_url)
        # 电影 id
        movie_id = movie_url.split("/")[-2:-1][0]
        ids.append(movie_id)

# 请求每部电影的详情 url - 二级页面
# 准备存放数据的列表
directors = []
writers = []
actorses = []
genres = []
regions = []
languages = []
releases = []
durations = []
aliases = []
scores = []
starses = []
raters = []

for detail_url in detail_urls:
        # 向服务器发起请求，并保存返回的响应对象
        response2 = requests.get(detail_url, headers=headers, proxies=proxies)

        # 构造 BeautifulSoup
        soup2 = BeautifulSoup(response2.text, "html.parser")

        # 找到电影详细外围 div
        movie_info_div = soup2.find("div", attrs={"id": "info"})
        # print(movie_info_div)
        # 导演
        director = movie_info_div.find("span").find_all("span")[1].text
        directors.append(director)
        # 编剧（注意：有的电影没有编剧）
        writer = movie_info_div.find("span", text="编剧")
        if writer:
                writer = writer.next_sibling.next_sibling.text
        writers.append(writer)
        # 主演
        actor = movie_info_div.find("span", {"class": "actor"}).find("span", {"class": "attrs"}).text
        actorses.append(actor)
        # 类型
        genre = movie_info_div.find("span", property="v:genre").text
```

```
        genres.append(genre)
        # 地区（注意：去除空格）
        region = movie_info_div.find("span", text="制片国家/地区:").next_sibling.strip()
        regions.append(region)
        # 语言
        language = movie_info_div.find("span", text="语言:").next_sibling.strip()
        languages.append(language)
        # 上映日期
        release = movie_info_div.find("span", property="v:initialReleaseDate").text
        releases.append(release)
        # 片长
        duration = movie_info_div.find("span", property="v:runtime").text
        durations.append(duration)
        # 又名
        alias = movie_info_div.find("span", text="又名:").next_sibling.strip()
        aliases.append(alias)

        # 找到评分评级外围 div
        movie_interest_div = soup2.find("div", attrs={"id": "interest_sectl"})
        rating_div = movie_interest_div.find("div", class_="rating_self", typeof="v:Rating")
        # 豆瓣评分
        score = rating_div.find("strong", class_="rating_num").text
        scores.append(score)
        # 星级
        bigstar = rating_div.find("div", class_="bigstar").get("class")[2]
        star = re.findall(r"\d{2}", bigstar)[0]
        starses.append(star)
        # 评价人数
        rater = rating_div.find("span", property="v:votes").text
        raters.append(rater)


# 构造 DataFrame
data = {
        "id": ids,
        "title": titles,
        "order": orders,
        "director": directors,
        "writer": writers,
        "actors": actorses,
        "genre": genres,
        "region": regions,
        "language": languages,
        "release": releases,
        "duration": durations,
        "alias": aliases,
        "score": scores,
        "star": starses,
        "rater": raters
}
```

```
df = pd.DataFrame(data)
# print(df)

# 只显示指定的列
columns = ["id", "title", "order", "genre", "region", "language", "score", "star", "rater"]
df_simple = df.loc[:, columns]
print(df_simple)
```

执行以上代码，输出结果如下：

```
          id      title order genre region language score star     rater
0   26709258      罗小黑战记     1     动作    中国大陆     汉语普通话     8.2    40    138049
1   30210691       极限逃生     2     喜剧      韩国       韩语     7.7    40     25376
2   30391300     我们是小僵尸     3     剧情      日本       日语     7.8    40      3296
3   30206464       七个会议     4     剧情      日本       日语     7.8    40      1447
4   30191950     飞翔吧！埼玉     5     喜剧      日本       日语     7.5    40      1087
5   30330264    最大的小小农场     6    纪录片      美国       英语     9.2    45       697
6   33403736   巧虎大飞船历险记     7     动画    中国大陆     汉语普通话     7.6    40       670
7   27040109       狂躁节拍     8     剧情      英国       英语     8.0    40       329
8   34781114     爱哭鬼上学记     9     恐怖      美国       英语     7.4    35       536
9   27114963      奇妙的家族    10     喜剧      韩国       韩语     7.2    35      3814
```

3.6.2 作业：待定

3.7 爬取动态数据

什么时候会遇到动态数据？

Ajax 请求、分页请求、服务器端 API 请求，实现前后端分离的架构。一句话，服务器端返回的不是网页本身，而是数据本身，通常会以 json 格式字符串发送到客户端。

例如：App 数据（“食派士”APP）

例如：今日头条、豆瓣电影的“选电影”页面 -- “瀑布流”

例如：公共数据平台 API（比如天气预报数据、聚合数据等）

3.7.1 案例：爬取天气预报数据

天气预报 API 接口：http://wthrcdn.etouch.cn/weather_mini?city=北京

```
"""
爬取天气预报数据：请求服务器端的 API
"""
```

```
import requests
import re

# url
# url = "http://wthrcdn.etouch.cn/weather_mini?citykey=101010100"
url = "http://wthrcdn.etouch.cn/weather_mini?city=上海"

# 假装自己是一个浏览器
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3"
}

# 请求
response = requests.get(url, headers=headers)

# 输出
# print(response.text)
print(response.json())
print(response.json()["data"]['yesterday'])
print(response.json()["data"]['city'])
print(response.json()["data"]['forecast'])

for day in response.json()["data"]['forecast']:
    print(day['date'], end=", ")
    print(day['high'], end=", ")
    # fengli = re.match(r"<!\[CDATA\[(.*)]]>", day['fengli']).group(1)
    fengli = re.findall(r"<!\[CDATA\[(.*)]]>", day['fengli'])[0]
    print(fengli, end=", ")
    print(day['low'], end=", ")
    print(day['fengxiang'], end=", ")
    print(day['type'])
```


3.7.2 案例：爬取“食派士”APP 数据

网站 URL：http://newsite.sherpa.com.cn

Python 代码实现：

```
import requests

# url = "http://newsite.sherpa.com.cn"
url = "http://newsite.sherpa.com.cn/sherpa-web-api/restaurant?poiId=B000A7YZT5&language=zh_CN&offset=0"

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3',
    # 'Cookie  ': 'UM_distinctid=167e36861e91c-0…rent_visit_time=1545726081810',
    # 'devType': 30,
    # 'Host': 'newsite.sherpa.com.cn',
    'Referer': 'http://newsite.sherpa.com.cn/'
}

r = requests.get(url, headers)
r.encoding = "utf8"

jsonTxt = r.text
# print(jsonTxt)

#  把 json 格式的字符串转换为 dict
content = r.json()

#  或者也可使用 json 模块
# import json
# content = json.loads(jsonTxt)

# print(content)
```

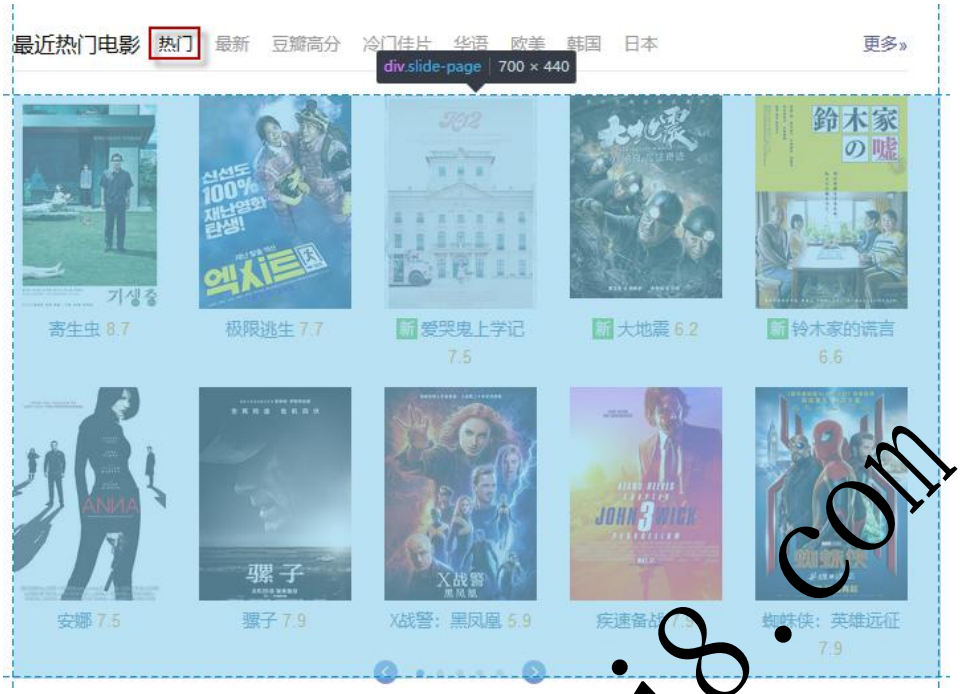
```
# 提取餐馆部分的信息
data = content["data"]
# print(data)

# 获得菜肴信息
# cuisines = data["cuisines"]
# for cuisine in cuisines:
#         print(cuisine)


# 获取餐馆信息
def get_rests(data):
    rests = data["rests"]
    for rest in rests:
        print("餐馆 ID：{}，餐馆名称：{}，餐馆地址：{}，菜系：{}".format(rest['restId'], rest['rest'], rest['branch'],
rest["cuisine"]))

# 执行函数，提取餐馆信息
get_rests(data)
```

执行以上代码，输出结果如下：

```
餐馆 ID：11386001，餐馆名称：拉兹印度音乐餐厅(鼓楼)，餐馆地址：拉兹印度音乐餐厅(鼓楼)，菜系：印度菜
餐馆 ID：11663002，餐馆名称：星期五餐厅(金融街)，餐馆地址：星期五餐厅(金融街)，菜系：美国菜
餐馆 ID：11679404，餐馆名称：西贡在巴黎餐厅(鼓楼)，餐馆地址：西贡在巴黎餐厅(鼓楼)，菜系：越南菜
餐馆 ID：11680198145，餐馆名称：牛油果子(鼓楼)，餐馆地址：牛油果子(鼓楼)，菜系：融合菜
餐馆 ID：11624002，餐馆名称：老石水饺(宝钞胡同)，餐馆地址：老石水饺(宝钞胡同)，菜系：中国菜
餐馆 ID：11679872，餐馆名称：起司家地安店(安门外大街)，餐馆地址：起司家地安店(安门外大街)，菜系：甜点
餐馆 ID：11679466，餐馆名称：西贡在巴黎餐厅(西单)，餐馆地址：西贡在巴黎餐厅(西单)，菜系：越南菜
餐馆 ID：11680198093，餐馆名称：本気家(西单老佛爷)，餐馆地址：本気家(西单老佛爷)，菜系：日本菜
餐馆 ID：11679646，餐馆名称：啧啧杯子蛋糕(五道营胡同)，餐馆地址：啧啧杯子蛋糕(五道营胡同)，菜系：甜点
餐馆 ID：11679634，餐馆名称：街旁(东大街)，餐馆地址：街旁(东大街)，菜系：融合菜
餐馆 ID：11680091，餐馆名称：北平机器啤酒屋(方家胡同)，餐馆地址：北平机器啤酒屋(方家胡同)，菜系：特色菜
餐馆 ID：11679494，餐馆名称：闲(五道营)，餐馆地址：闲(五道营)，菜系：混合类
餐馆 ID：11679595，餐馆名称：摩卡站(西单)，餐馆地址：摩卡站(西单)，菜系：特色菜
餐馆 ID：11679493，餐馆名称：The Veggie Table(五道营胡同)，餐馆地址：The Veggie Table(五道营胡同)，菜系：
特色菜
餐馆 ID：11680198186，餐馆名称：沃歌斯(西单君太)，餐馆地址：沃歌斯(西单君太)，菜系：特色菜
餐馆 ID:11383001，餐馆名称:卵石庭院美式墨西哥餐厅(五道营)，餐馆地址:卵石庭院美式墨西哥餐厅(五道营)，菜
系：墨西哥菜
餐馆 ID：11679593，餐馆名称：堡爵仕餐厅(西单) (暂时不可点)，餐馆地址：堡爵仕餐厅(西单) (暂时不可点)，菜系：
美国菜
餐馆 ID：11666001，餐馆名称：箴言之口(东直门内大街)，餐馆地址：箴言之口(东直门内大街)，菜系：意大利菜
餐馆 ID：11679021，餐馆名称：小云南餐厅(后永康)，餐馆地址：小云南餐厅(后永康)，菜系：中国菜
餐馆 ID：11679866，餐馆名称：法兰之吻蛋糕(三里屯)，餐馆地址：法兰之吻蛋糕(三里屯)，菜系：甜点
```

3.7.3 案例：爬取豆瓣电影网站首页的“热门电影”

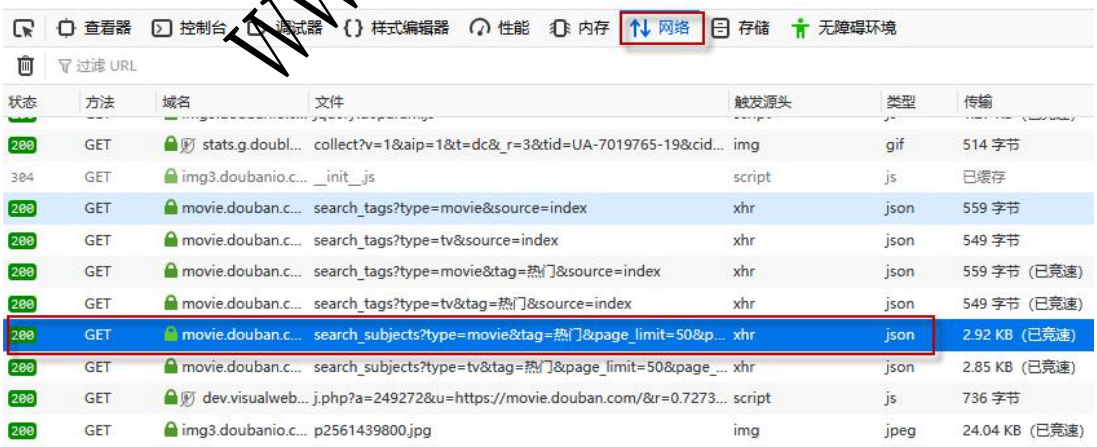
要爬取的内容：



这些最近热门电影是以 JSON 格式从服务器端返回的数据，然后在客户端动态填充到页面中。要查看请求 JSON 数据的 URL，通过以下方式：

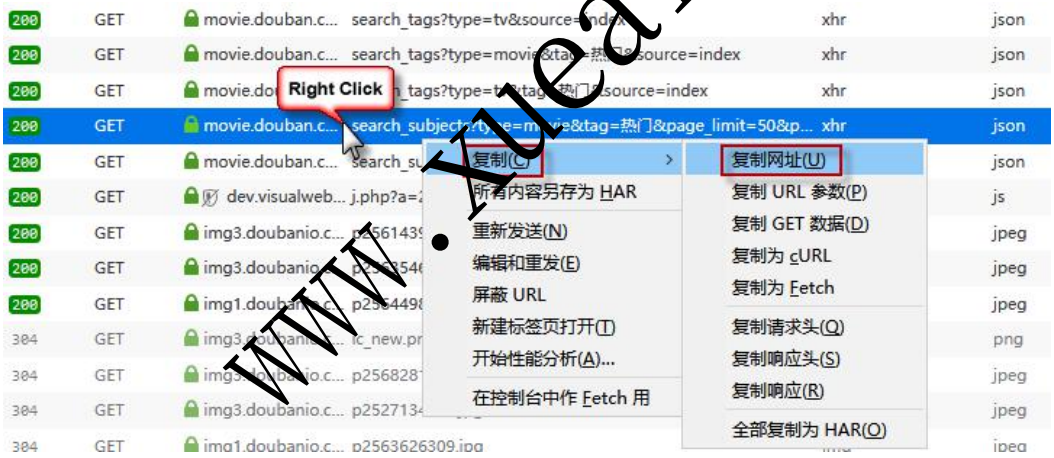
- Fn + F12，打开浏览器的开发者模式
- 找到“网络”选项卡
- 刷新页面
- 点击请求的 url，在右侧查看响应数据

1）热门电影数据为动态数据，是使用 JavaScript 的 Ajax 技术动态加载，然后填充到前端页面中的。



2）该 Ajax 请求返回的是 JSON 格式的数据：

3）获取热门电影数据请求的 URL：



得到的请求 URL 如下：

https://movie.douban.com/j/search_subjects?type=movie&tag=%E7%83%AD%E9%97%A8&page_limit=50&page_start=0

总页数：243 页

4）Python 代码实现：

```
"""
爬取豆瓣电影首页的“热门电影”。
经过分析，这是一个使用 Ajax 动态加载的 JSON 格式数据。所以需要找到其 Ajax 请求的 URL，然后请求回 JSON 数据，并解析。
"""
```

```
import requests
from bs4 import BeautifulSoup

# 豆瓣电影首页的 url
url =
"https://movie.douban.com/j/search_subjects?type=movie&tag=%E7%83%AD%E9%97%A8&page_limit=50&page_start=0"

my_headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
    }

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url, headers=my_headers)
movies_json_array = response.json()

print(movies_json_array.get("subjects")[0])
```

重构一：get 请求的 url 参数，单独编码传递

```
"""
爬取豆瓣电影首页的“热门电影”。
经过分析，这是一个使用 Ajax 动态加载的 JSON 格式数据。所以需要找到其 Ajax 请求的 URL，然后请求回 JSON 数据，并解析。
"""
import requests
from bs4 import BeautifulSoup

# 豆瓣电影首页的 url
url = "https://movie.douban.com/j/search_subjects"

# 要传递的参数
params = {
    "type": "movie",
    "tag": "热门",
    "page_limit": 50,
    "page_start": 0
}

# 假装自己是浏览器
my_headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
}

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url, headers=my_headers, params=params)

# 返回的是 json 数组，转换
movies_json_array = response.json()
```

```
# 遍历返回的电影数据
for movie in movies_json_array.get("subjects"):
    print(movie)
```

重构二：封装到函数中

```
"""
爬取豆瓣电影首页的“热门电影”。
经过分析，这是一个使用 Ajax 动态加载的 JSON 格式数据。所以需要找到其 Ajax 请求的 URL，然后请求回 JSON 数据，并解析。
"""
import requests
from bs4 import BeautifulSoup

# 参数 page 为要爬取的页码
def get_hot_movies_one_page(page):
    # 豆瓣电影首页的 url
    url = "https://movie.douban.com/j/search_subjects"

    # 要传递的参数
    params = {
        "type": "movie",
        "tag": "热门",
        "page_limit": 50,
        "page_start": page
    }

    # 假装自己是浏览器
    my_headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
    }

    # 向服务器发起请求，并保存返回的响应对象
    response = requests.get(url, headers=my_headers, params=params)
    movies_json_array = response.json()

    # 遍历输出每部电影的数据
    for movie in movies_json_array.get("subjects"):
        print(movie)


# 执行函数，爬取第一页的数据
get_hot_movies_one_page(0)

print("--------------------------------------------------------------------------------")
# 执行函数，爬取第二页的数据
get_hot_movies_one_page(1)
```

重构三：将爬取到的数据打包到 DataFrame 中并返回

```
"""
爬取豆瓣电影首页的“热门电影”。
经过分析，这是一个使用 Ajax 动态加载的 JSON 格式数据。所以需要找到其 Ajax 请求的 URL，然后请求回 JSON 数据，并解析。
"""
import requests
from bs4 import BeautifulSoup
import pandas as pd

pd.set_option('display.max_columns', None)   # 显示所有列
pd.set_option('display.max_rows', None)      # 显示所有行
pd.set_option('display.width', 200)          # 行宽，默认 80
pd.set_option('max_colwidth', 80)            # 设置 value 的显示长度为 80，默认为 50


def get_hot_movies_one_page(page):
    # 豆瓣电影首页的 url
    url = "https://movie.douban.com/j/search_subjects"

    # 要传递的参数
    params = {
        "type": "movie",
        "tag": "热门",
        "page_limit": 50,
        "page_start": page
    }

    # 假装自己是浏览器
    my_headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
    }

    # 向服务器发起请求，并保存返回的响应对象
    response = requests.get(url, headers=my_headers, params=params)
    movies_json_array = response.json()

    # 解析每部电影的数据：id title url rate cover_x cover_y cover
    ids = []
    titles = []
    urls = []
    rates = []
    cover_xes = []
    cover_yes = []
    covers = []
    for movie in movies_json_array.get("subjects"):
        ids.append(movie['id'])
        titles.append(movie['title'])
        urls.append(movie['url'])
```

```
            rates.append(float(movie['rate']))        # 注意：json 中传过来的是字符串类型
            cover_xes.append(movie['cover_x'])
            cover_yes.append(movie['cover_y'])
            covers.append(movie['cover'])

        data = {"id": ids,
                "title": titles,
                "url": urls,
                "rate": rates,
                "cover_x": cover_xes,
                "cover_y": cover_yes,
                "cover": covers
                }
        df = pd.DataFrame(data=data)
        return df


# 执行函数，爬取第一页的数据
one_page_df = get_hot_movies_one_page(0)

# 查看返回的数据类型
print(type(one_page_df))

# 查看第一条数据
print(one_page_df.iloc[0, :])

# 查看全部数据
print(one_page_df)
```

重构四：爬取所有的热门电影数据（共 244 页）

```
"""
爬取豆瓣电影首页的“热门电影”。
经过分析，这是一个使用 Ajax 动态加载的 JSON 格式数据。所以需要找到其 Ajax 请求的 URL，然后请求回 JSON
数据，并解析。
"""
import requests
from bs4 import BeautifulSoup
import pandas as pd

pd.set_option('display.max_columns', None)   # 显示所有列
pd.set_option('display.max_rows', None)      # 显示所有行
pd.set_option('display.width', 200)          # 行宽，默认 80
pd.set_option('max_colwidth', 80)            # 设置 value 的显示长度为 80，默认为 50

# 爬取一页数据
def get_hot_movies_one_page(page):
    # 豆瓣电影首页的 url
    url = "https://movie.douban.com/j/search_subjects"

    # 要传递的参数
```

```
    params = {
        "type": "movie",
        "tag": "热门",
        "page_limit": 50,
        "page_start": page
    }

    # 假装自己是浏览器
    my_headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/61.0.3163.100 Safari/537.36'
    }

    # 向服务器发起请求，并保存返回的响应对象
    response = requests.get(url, headers=my_headers, params=params)
    movies_json_array = response.json()

    # 解析每部电影的数据：id title url rate cover_x cover_y cover
    ids = []
    titles = []
    urls = []
    rates = []
    cover_xes = []
    cover_yes = []
    covers = []
    for movie in movies_json_array.get("subjects"):
        ids.append(movie['id'])
        titles.append(movie['title'])
        urls.append(movie['url'])
        rates.append(float(movie['rate']))  # 注意：json 中传过来的是字符串类型
        cover_xes.append(movie['cover_x'])
        cover_yes.append(movie['cover_y'])
        covers.append(movie['cover'])

    data = {"id": ids,
            "title": titles,
            "url": urls,
            "rate": rates,
            "cover_x": cover_xes,
            "cover_y": cover_yes,
            "cover": covers
            }
    df = pd.DataFrame(data=data)
    return df

# 爬取多页数据
def get_hot_movies(page_total):
    # 构造一个空的 DataFrame，用来存放所有页的数据
    all_pages_df = pd.DataFrame()
```

```
    # 依次爬取每一页电影数据，并所加到 all_pages_df 中
    for page in range(page_total):
        one_page = get_hot_movies_one_page(page)
        print("page: " + str(page))
        print(one_page.shape)
        all_pages_df = all_pages_df.append(one_page)

    return all_pages_df


# 爬取 244 页热门电影数据
# all_hot_movies = get_hot_movies(244)
all_hot_movies = get_hot_movies(2)

print(all_hot_movies.shape)
print(all_hot_movies.columns)

# 查看前 5 条数据
print(all_hot_movies.iloc[:5, :])
```


3.7.4 作业：爬取豆瓣电影“选电影”页面的热门电影数据

提示

1）找 URL：在首页，点"最近热门电影"|"更多"，跳转到"选电影"页面。

2）选电影是动态加载的瀑布流结构：

每次点击“加载更多”链接时，会使用 Ajax 技术从服务器端请求新的电影数据，并动态填充到页面中。

3.8 如何向服务器传递参数

get 方法请求时的参数传递
带有查询字符串：URL 中可选的“?…”被称为“查询字符串”。

```
import requests

url = 'http://www.webscrapingfordatascience.com/paramhttp/?query=test'
r = requests.get(url)
print(r.text)          # 会显示: I don't have any information on "test"
```

查询字符串如果带有非字典字符（例如空格、特殊字符、中文等），会被编码。- URL 重写

```
import requests

url = 'http://www.webscrapingfordatascience.com/paramhttp/?query=a query with spaces'
r = requests.get(url)

# 参数会被编码为'a%20query%20with%20spaces'
# 执行下面的语句可以看得到:
print(r.request.url)      # 将显示 [...]/paramhttp/?query=a%20query%20with%20spaces

print(r.text)    # 将显示: I don't have any information on "a query with spaces"
```

在 requests 中有简单的方法：

```
import requests

url = 'http://www.webscrapingfordatascience.com/paramhttp/'
parameters = {
        'query': 'a query with /, spaces and ?&'
}

r = requests.get(url, params=parameters)
print(r.url)
print(r.text)
```

传参数

```
# http://localhost:8084/jxshoping/detail?type=clothing&id=tb1

url = "http://localhost:8084/jxshoping/detail"
query_str = {'type': 'clothing', 'id': 'tb1', 'key3': None}

# 说明：任何参数值为 None 的字典 key 都不会添加到 URL 的查询字符串中

response = requests.get(url, params=query_str)
response.encoding = "utf8"
content = response.text

print("################传参数####################")
print(response.request.url)
```

```
print(response.url)
print(content)
```

参数也可以传递列表：query_str = {'key1': 'value1', 'key2': ['value2', 'value3']}

post 方法请求时的参数传递

POST 请求：模拟表单提交

```
url = 'http://localhost:8084/jxshoping/LoginServlet'

print("################模拟提交表单######################")
# 只需简单地传递一个字典给 data 参数。数据字典在发出请求时会自动编码为表单形式:
form_data = {'username': 'xlw', 'userpwd': '123456'}  # 设置表单数据
response = requests.post(url, data=form_data)
print(response.text)
```

3.98 保存爬取的数据到文件中

3.9.1 案例：爬取电影数据并存储到本地 csv 文件中

爬取豆瓣电影网站首页的“最近热门电影”数据，并保存到本地.csv 文件中。

```
...
# 保存到本地.csv 文件
all_hot_movies.to_csv("hot_movies.csv",encoding="utf8")
```

结果如下所示（部分）：

实现图片下载：

```
url = "http://localhost:8084/WebDemo2/images/TB1_400x400.jpg"
response = requests.get(url)

with open("tb.jpg", "wb") as file:
    file.write(response.content)
print("download over.")
```

当流下载时，用 Response.iter_content 或许更方便些。requests.get(url)默认是下载在内存中的，下载

完成才存到硬盘上，可以用 Response.iter_content 来边下载边存硬盘。

```
filename = "aa.txt"
with open(filename, 'wb') as file:
    for chunk in r.iter_content(chunk_size=1024):
        file.write(chunk)
```

如果要查看下载保存进度，可以使用进度条。

首先安装 tqdm：

```
$ pip install tqdm
```

然后在程序中导入使用即可：

```
from tqdm import tqdm

filename = "aa.txt"
with open(filename, 'wb') as file:
    for chunk in tqdm(r.iter_content(chunk_size=1024)):
        file.write(chunk)
```

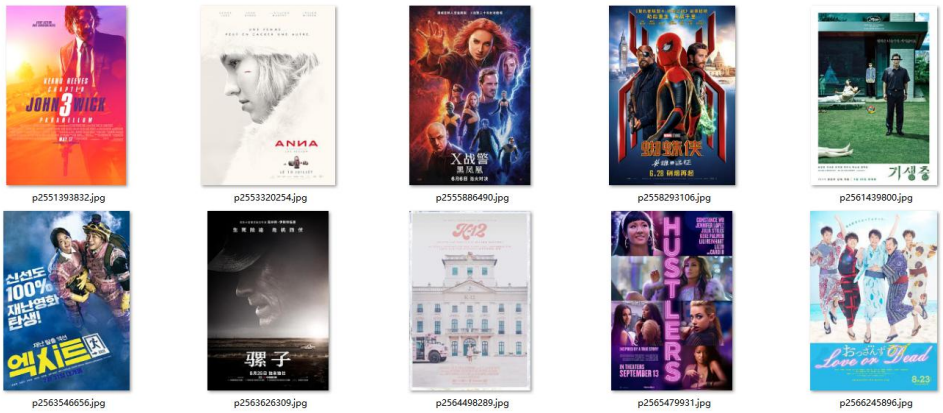
3.9.2 案例：爬取电影海报并存储到本地

爬取豆瓣电影网站首页的“最近热门电影”海报图片，并保存到本地。

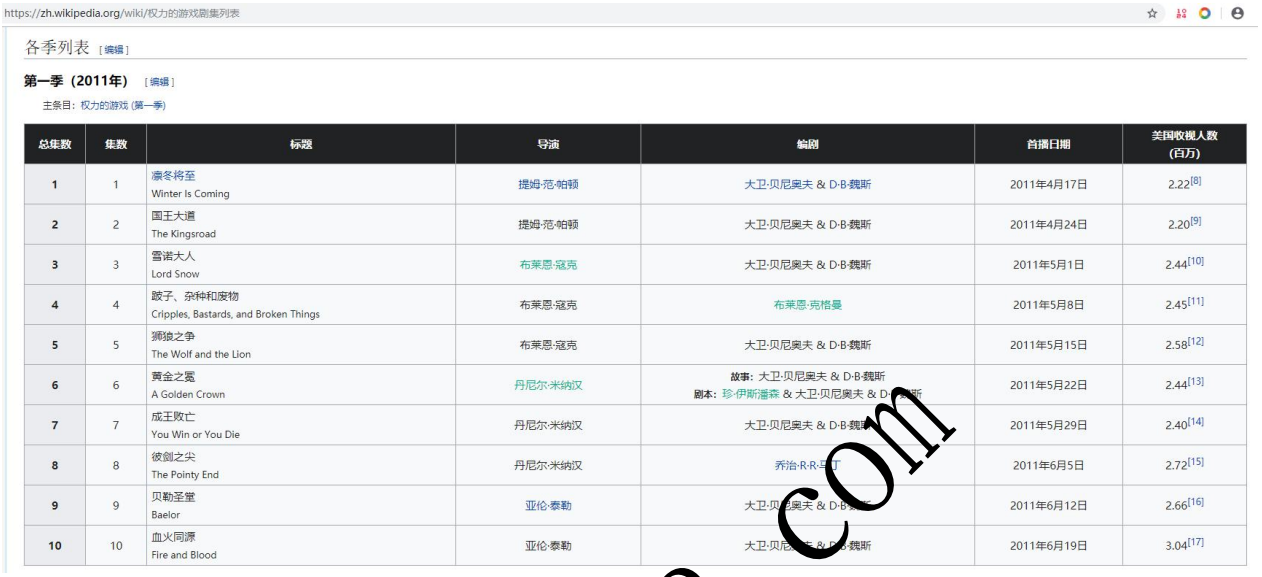
```
"""
下载豆瓣电影网站首页"热门电影"电影海报，并保存到本地。

技巧：
    网站提供的是.webp 格式的图像文件。
    这种格式谷歌（google）开发的一种旨在加快图片加载速度的图片格式。图片压缩体积大约只有 JPEG 的 2/3，
并能节省大量的服务器宽带资源和数据空间。
    但是这种格式文件下载到本地无法打开。
    所以一个小技巧：将后缀改为 jpg，然后再下载
"""
import pandas as pd
import requests
import time
import random
import os
from tqdm import tqdm      # pip install tqdm     安装进度条模块

pd.set_option('display.max_columns', None)   # 显示所有列
pd.set_option('display.max_rows', None)      # 显示所有行
pd.set_option('display.width', 180)          # 行宽，默认 80
pd.set_option('max_colwidth', 100)           # 设置 value 的显示长度为 80，默认为 50

df = pd.read_csv("hot_movies.csv")
print(df.shape)

# 如果文件夹不存在，就创建
```

```
imgs_dir = "posters"
if not os.path.exists(imgs_dir):
    os.mkdir(imgs_dir)

# 获取所有电影海报的 url
# movie_urls = df.iloc[:, -1]
movie_urls = df.iloc[:10, -1]          # 暂时只下载前 10 张

# 假装我是一个浏览器
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/sig
ned-exchange;v=b3"
}

# 遍历每个 url，请求图片文件，并保存到本地
for url in movie_urls:
    # 构造文件名
    filename = url.split("/")[-1].split(".")[0]
    filepath = imgs_dir + "/" + filename + ".jpg"
    print(filepath)
    # 构造 url
    jpg_url = "https://img3.doubanio.com/view/photo/s_ratio_poster/public/" + filename + ".jpg"

    # 下载并保存
    r = requests.get(jpg_url, headers=headers, stream=True)
    with open(filepath, 'wb') as file:
        for chunk in tqdm(r.iter_content(chunk_size=4096)):  # 边下载边存盘
            file.write(chunk)
    # 随机暂停几秒钟，防止被封
    time.sleep(random.randint(1, 5))

print("download over.")
```

爬取结果如下：



3.9.3 作业：爬取维基百科“权利的游戏”电视剧集列表并存入文件



https://zh.wikipedia.org/wiki/权力的游戏剧集列表

各季列表 [编辑]

第一季（2011年） [编辑]

主条目：权力的游戏 (第一季)

总集数	集数	标题	导演	编剧	首播日期	美国收视人数（百万）
1	1	凛冬将至 Winter Is Coming	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月17日	2.22[8]
2	2	国王大道 The Kingsroad	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月24日	2.20[9]
3	3	雪诺大人 Lord Snow	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月1日	2.44[10]
4	4	跛子、杂种和废物 Cripples, Bastards, and Broken Things	布莱恩·寇克	布莱恩·克格曼	2011年5月8日	2.45[11]
5	5	狮狼之争 The Wolf and the Lion	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月15日	2.58[12]
6	6	黄金之冕 A Golden Crown	丹尼尔·米纳汉	故事：大卫·贝尼奥夫 & D·B·魏斯 剧本：珍·伊斯温森 & 大卫·贝尼奥夫 & D·B·魏斯	2011年5月22日	2.44[13]
7	7	成王败亡 You Win or You Die	丹尼尔·米纳汉	大卫·贝尼奥夫 & D·B·魏斯	2011年5月29日	2.40[14]
8	8	彼剑之尖 The Pointy End	丹尼尔·米纳汉	乔治·R·R·马丁	2011年6月5日	2.72[15]
9	9	贝勒圣堂 Baelor	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月12日	2.66[16]
10	10	血火同源 Fire and Blood	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月19日	3.04[17]

URL 地址：

https://zh.wikipedia.org/wiki/权利的游戏剧集列表

要求：

编写 Python 爬虫程序，将《权利的游戏》所有剧集爬取下来，并存入.csv 文件中。

3.10 保存爬取的数据到数据库中

一般数据采集流程：

请求服务器端数据 -> 解析返回的数据 -> 存入数据库

常用的关系型数据库：MySQL

Python 的 MySQL 数据库驱动：PyMySQL。如果没有安装 PyMySQL 驱动，使用下面的命令安装：

```
$ pip install PyMySQL
```

3.10.1 案例：爬取电影数据并存入 MySQL 数据库中

要保存的数据有：id, title, url, rate, cover_x, cover_y, cover

1）在 MySQL 中创建数据库

```
create table douban_hot_movies(
  id        varchar(20),
```

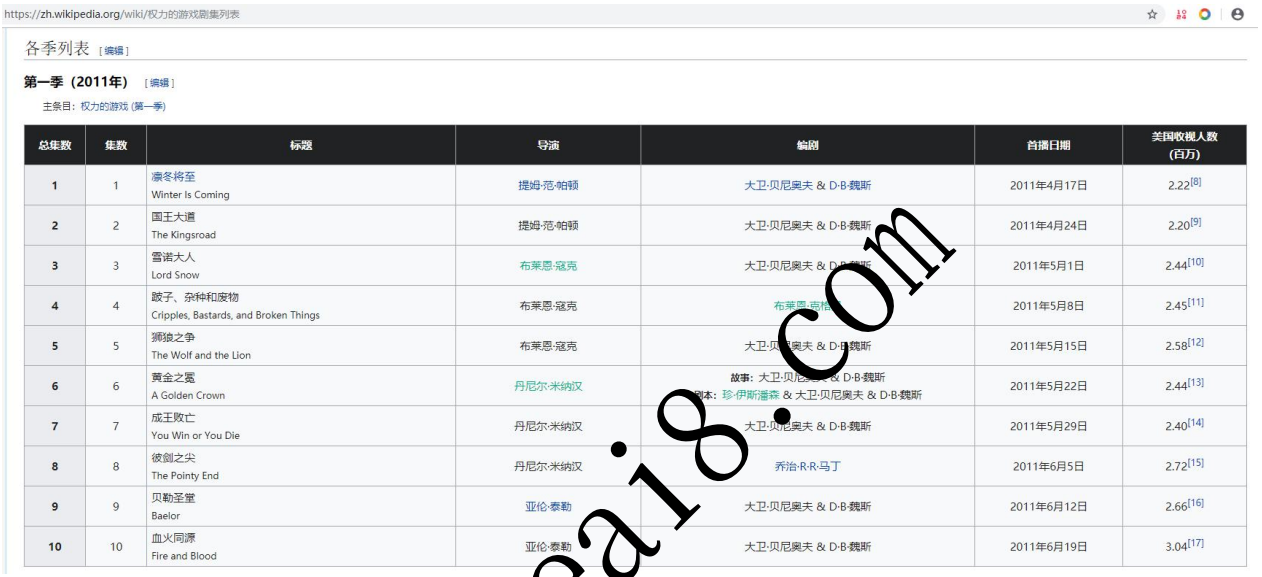
```
  title      varchar(50),
  url        varchar(255),
  rate       int,
  cover_x int,
  cover_y int,
  cover      varchar(300)
);
```

2）数据库程序：

```
"""
爬取豆瓣电影首页的“最近热门电影”数据，并存入到 MySQL 数据库中。

如果没有安装 PyMySQL 驱动，使用下面的命令安装：pip install PyMySQL
"""
import pandas as pd
import pymysql

pd.set_option('display.max_columns', None)   # 显示所有列
pd.set_option('display.max_rows', None)      # 显示所有行
pd.set_option('display.width', 180)          # 行宽，默认 80
pd.set_option('max_colwidth', 100)           # 设置 value 的显示长度为 80，默认为 50

# 注：这里为了简单，直接加载的.csv 文件
#       可替换为前面示例中爬虫代码
df = pd.read_csv("hot_movies.csv")
print(df.shape)

# df = df.iloc[:10, :]
# print(df)

# 将 DataFrame 数据写入到数据库的函数
def save_from_df(df):
    # 打开数据库连接
    con = pymysql.connect("localhost", "root", "admin", "cda", use_unicode=True, charset="utf8")

    # cursor 对象
    cursor = con.cursor()

    # SQL  插入语句  id,title,url,rate,cover_x,cover_y,cover
    for row in range(df.shape[0]):
        movie = df.iloc[row, :]
        sql = "insert into douban_hot_movies values ('%s', '%s', '%s', '%s', '%s', '%s', '%s') " % \
            (movie[0], movie[1], movie[2], movie[3], movie[4], movie[5], movie[6])
        try:
            cursor.execute(sql)              # 执行 sql 语句
            con.commit()                     # 注意：这里需要提交
        except:
            con.rollback()                   # 发生错误时回滚

    # 关闭数据库连接
```

```
    con.close()


save_from_df(df)
```

3.10.2 作业：爬取维基百科“权利的游戏”电视剧集列表并存入数据库



https://zh.wikipedia.org/wiki/权力的游戏剧集列表

各季列表 [编辑]

第一季（2011年） [编辑]

主条目：权力的游戏 (第一季)

总集数	集数	标题	导演	编剧	首播日期	美国收视人数（百万）
1	1	凛冬将至 Winter Is Coming	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月17日	2.22[8]
2	2	国王大道 The Kingsroad	提姆·范·帕顿	大卫·贝尼奥夫 & D·B·魏斯	2011年4月24日	2.20[9]
3	3	雪诺大人 Lord Snow	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月1日	2.44[10]
4	4	跛子、杂种和废物 Cripples, Bastards, and Broken Things	布莱恩·寇克	布莱恩·寇克	2011年5月8日	2.45[11]
5	5	狮狼之争 The Wolf and the Lion	布莱恩·寇克	大卫·贝尼奥夫 & D·B·魏斯	2011年5月15日	2.58[12]
6	6	黄金之冕 A Golden Crown	丹尼尔·米纳汉	故事：大卫·贝尼奥夫 & D·B·魏斯 剧本：珍·伊斯塔森 & 大卫·贝尼奥夫 & D·B·魏斯	2011年5月22日	2.44[13]
7	7	成王败亡 You Win or You Die	丹尼尔·米纳汉	大卫·贝尼奥夫 & D·B·魏斯	2011年5月29日	2.40[14]
8	8	彼剑之尖 The Pointy End	丹尼尔·米纳汉	乔治·R·R·马丁	2011年6月5日	2.72[15]
9	9	贝勒圣堂 Baelor	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月12日	2.66[16]
10	10	血火同源 Fire and Blood	亚伦·泰勒	大卫·贝尼奥夫 & D·B·魏斯	2011年6月19日	3.04[17]

URL 地址：

https://zh.wikipedia.org/wiki/权力的游戏剧集列表

要求：

编写 Python 爬虫程序，将《权利的游戏》所有剧集爬取下来，并存入 MySQL 数据库中。

3.11 如何伪装不是爬虫（对付反爬虫）

怎么知道网站让不让我们爬取数据？查看其 rotots.txt 文档声明。

https://www.douban.com/robots.txt

怎么破解？

使用 User Agent 隐藏身份。

为何要设置 User Agent？

有一些网站不喜欢被爬虫程序访问，所以会检测连接对象，如果是爬虫程序，也就是非人点击访问，它就会不让你继续访问。所以为了要让程序可以正常运行，需要隐藏自己的爬虫程序的身份。此时，我们就可以通过设置 User Agent 的来达到隐藏身份的目的，User Agent 的中文名为用户代理，简称 UA。

User Agent 存放于 Headers 中，服务器就是通过查看 Headers 中的 User Agent 来判断是谁在访问。在 Python 中，如果不设置 User Agent，程序将使用默认的参数，那么这个 User Agent 就会有 Python 的字样，如果服务器检查 User Agent，那么没有设置 User Agent 的 Python 程序将无法正常访问网站。

Python 允许我们修改这个 User Agent 来模拟浏览器访问。标识浏览器身份的是 User-Agent。所以要在爬虫程序中伪装自己是一个浏览器，就要设置请求头部（request headers）中的 User-Agent 参数。

常见的 User Agent

- Android
  - Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Safari/535.19
  - Mozilla/5.0 (Linux; U; Android 4.0.4; en-gb; GT-I9300 Build/IMM76D) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
  - Mozilla/5.0 (Linux; U; Android 2.2; en-gb; GT-P1000 Build/FROYO) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
- Firefox
  - Mozilla/5.0 (Windows NT 6.2; WOW64; rv:21.0) Gecko/20100101 Firefox/21.0
  - Mozilla/5.0 (Android; Mobile; rv:14.0) Gecko/14.0 Firefox/14.0
- Google Chrome
  - Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.94 Safari/537.36
  - Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.133 Mobile Safari/535.19
- iOS
  - Mozilla/5.0 (iPad; CPU OS 5_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A334 Safari/7534.48.3
  - Mozilla/5.0 (iPod; U; CPU like Mac OS X; en) AppleWebKit/420.1 (KHTML, like Gecko) Version/3.0 Mobile/3A101a Safari/419.3

上面列举了 Andriod、Firefox、Google Chrome、iOS 的一些 User Agent，直接拷贝就能用。这是最简单的方法。

也可以通过以下手段查看有哪些 User-Agent 可以使用。

方法一：在浏览器工具里，查看请求的头部信息；

方法二：运行下面的程序，抓取常见浏览器的 User-Agent 信息，然后任意选择其中一个：

```
import requests
import json

# 爬取浏览器代理（UserAgent）值，写入到指定文件中
def write_browser_info_to_file():
    url = "https://fake-useragent.herokuapp.com/browsers/0.1.11"
    my_user_agent = requests.get(url)

    # print(my_user_agent.json().get("browsers").get("firefox"))
    # print(my_user_agent.json().get("browsers").get("chrome"))
```

```
    # print(my_user_agent.json().get("browsers").get("opera"))
    # print(my_user_agent.json().get("browsers").get("internetexplorer"))
    # print(my_user_agent.json().get("browsers").get("safari"))

    # 查看不同浏览器的 UserAgent 值
    agents = my_user_agent.json().get("browsers").get("safari")
    for agent in agents:
        print(agent)

    # 也可以选择将其保存到文件中
    # with open("browser_info_agent.json","w") as f:
    #     json.dump(my_user_agent.text, f)


write_browser_info_to_file()
```

小贴士：

- ✓ 所有浏览器都假装自己是 Mozilla/5.0
- ✓ Chrome 假装自己是所有内核（应该是 WebKit，却有 KHTML 和 Gecko）除了 IE(Trident)
- ✓ IE 假装自己是 Gecko,同时也认为是自己
- ✓ EDGE 假装自己是 Webkit/Gecko，同时认为也是自己
- ✓ Opera 假装自己是 Webkit/Gecko,同时认为也是自己
- ✓ Firefox 假装自己是自己。好吧，我以前是老大，假装现在也是。

设置 User-Agent：

```
url = "http://localhost:8084/jxshoping/CommonServlet"

# 自定义头部信息
my_headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 ' +
                  ' (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36'
}

print("################没 headers#####################")
# 在 requests 中，通过参数 headers 来发送自定义的 header
response = requests.get(url, headers=my_headers)  # 加入自定义的头部信息
print(response.text)

# 显示请求头部分信息
print("请求头部信息为：")
print(response.request.headers)
```

查看浏览器设置：https://www.whatismybrowser.com/。这个网站可以让服务器测试浏览器的属性。

```
import requests
from bs4 import BeautifulSoup

headers = {
```

```
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3"
}

url = "https://www.whatismybrowser.com/developers/what-http-headers-is-my-browser-sending"

req = requests.get(url, headers=headers)
bsObj = BeautifulSoup(req.text, 'lxml')
# print(bsObj.find("table", {"class": "table-striped"}).get_text)
print(req.request.headers)
```

3.12 如何控制爬取的速度（对付反爬虫）

降低访问频率

为了提高爬虫的效率，很多人在爬虫中使用多线程、异步等手段，使得爬虫在一个小时内甚至可以访问数十万次页面，给服务器带来了不小的压力，因此有一些防护措施较完备的网站就可能会阻止你快速地提交表单，或者快速地与网站进行交互。即使没有这些安全措施，用一个比普通人快很多的速度从一个网站下载大量信息也可能让自己被网站封杀。

遇到这种网站，如果你对爬虫效率要求不高，可以尝试降低爬虫访问速度，比如不再使用多线程技术，每次访问后设置等待时间，限制每小时访问量等。比如在 python 中：

```
import time,random                  # 导入包
time.sleep(random.randint(1,9))     # 设置时间间隔为随机几秒钟
```

但如果你对爬虫效率要求比较高，每次访问都等待无法满足要求，那么首先需要确定网站是根据什么条件进行限制的，再想办法去突破限制。比如很多网站都是通过 IP 来限制访问量的，那么就可以使用代理 IP，每请求几次就换一个 IP，这样就可以避免被禁了。

3.13 如何突破访问量限制（对付反爬虫）

使用代理 IP 隐藏身份。

为何要设置代理 IP？

写爬虫，大家都知道，抓的网站和数据多了，如果爬虫抓取速度过快，免不了触发网站的防爬机制，几乎用的同一招就是封 IP。解决方案有 2 个：

1、同一 IP，放慢速度(爬取速度慢)

2、使用代理 IP 访问(推荐)

第一种方案牺牲的就是时间和速度。第二种方案需要设计代理 IP。

所谓代理就是介于用户与网站之间的第三者：用户先将请求发到代理，然后代理再发到服务器，这

样看起来就像是代理在访问那个网站了。这时，服务器会将这次访问算到代理头上。同时用多个代理的话，单个 IP 的访问量就降下去了。

Python 中的 requests 使用代理非常简单。如果需要使用代理，可以用 proxy 参数为任何 request 方法配置单独的请求：

```
import requests

proxies = {
   'http': 'http://10.10.1.10:3128',
   'https': 'http://10.10.1.10:1080',
}

r = requests.get("http://www.baidu.com", proxies=proxies)
print(r)
print(r.text)
print(r.status_code)
```

下面代码实现了从代理 IP 池中随机挑选一个有效高匿代理 IP 的功能。高匿代理 IP 来自以下地址：https://www.xicidaili.com/nn/。

```
import requests
import time, random

# 高匿代理池
proxies_pool = [
        {"http": "114.239.147.75:808"},
        {"https": "171.12.113.35:9999"},
        {"http": "120.83.106.21:9999"},
        {"http": "58.253.157.72:9999"},
        {"https": "58.253.159.117:9999"},
        {"http": "120.83.109.129:9999"},
        {"https": "58.253.159.251:9999"},
        {"https": "163.204.244.11:9999"},
        {"https": "60.13.42.108:9999"},
        {"https": "27.43.186.70:9999"},
        {"https": "58.253.159.157:9999"},
        {"https": "115.219.108.3:8010"},
        {"https": "113.120.60.252:24967"},
        {"http": "114.239.148.184:808"},
        {"https": "61.157.136.105:808"},
        {"https": "221.206.100.133:34073"},
        {"https": "123.163.97.75:9999"},
        {"https": "58.253.157.182:9999"},
        {"https": "1.198.72.234:9999"},
        {"https": "115.53.35.54:9999"},
        {"https": "113.120.61.174:30413"},
        {"https": "113.124.95.251:46895"},
        {"http": "1.197.204.179:9999"},
        {"http": "114.239.147.75:808"},
        {"http": "112.85.168.67:9999"},
        {"http": "115.53.33.147:9999"},
```

```
        {"http": "1.198.73.64:9999"},
        {"http": "123.163.96.94:9999"}
    ]

# 随机获取一个有效的代理 IP
def get_one_proxies(proxies_pool):
    index = random.randint(0, len(proxies_pool)-1)
    valided_proxies = proxies_pool[index]
    print("正在尝试:", end=" ")
    print(valided_proxies)
    while not valid_proxies(valided_proxies):
        index = random.randint(0, len(proxies_pool)-1)
        valided_proxies = proxies_pool[index]
        print("正在尝试:", end=" ")
        print(valided_proxies)
        time.sleep(random.randint(1,9))
    print("有效的代理 IP：", end=" ")
    print(valided_proxies)
    return valided_proxies

# 检验代理是否有效
def valid_proxies(proxies):
    my_proxies = proxies
    # 假装自己是浏览器
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/74.0.3729.169 Safari/537.36",
        "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exch
ange;v=b3"
    }
    try:
        # 使用代理 IP 请求百度首页。如果返回状态码是 200，说明这是一个有效的代理 IP
        r = requests.get("http://www.baidu.com", proxies=my_proxies, headers=headers)
        return r.status_code == 200
    except Exception as e:
        return False


proxey_ip = get_one_proxies(proxies_pool)
print(proxey_ip)
```

那么从哪里能找到这么多代理 IP 呢？可以从网上搜“高匿代理 IP”，有很多。这里推荐下面这个国内高匿代理 IP：https://www.xicidaili.com/nn/。

根据代理的匿名程度，代理可以分为如下类别。

- **高度匿名代理**：会将数据包原封不动地转发，在服务端看来就好像真的是一个普通客户端在访问，而记录的IP是代理服务器的IP。
- **普通匿名代理**：会在数据包上做一些改动，服务端上有可能发现这是个代理服务器，也有一定几率追查到客户端的真实IP。代理服务器通常会加入的HTTP头有 `HTTP_VIA` 和 `HTTP_X_FORWARDED_FOR` 。
- **透明代理**：不但改动了数据包，还会告诉服务器客户端的真实IP。这种代理除了能用缓存技术提高浏览速度，能用内容过滤提高安全性之外，并无其他显著作用，最常见的例子是内网中的硬件防火墙。
- **间谍代理**：指组织或个人创建的用于记录用户传输的数据，然后进行研究、监控等目的的代理服务器。

使用 requests 和 Beautifulsoup 爬取代理 IP 数据，并保存到文本文件中。代码如下：

```
import requests
import os
from bs4 import BeautifulSoup

os.chdir(r'C:\Users\Administrator\Desktop\scrapy\proxy')

headers = {'User-Agent':'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/56.0.2924.87 Mobile Safari/537.36'}
url = 'http://www.xicidaili.com/nn/1'
s = requests.get(url,headers = headers)

soup = BeautifulSoup(s.text,'lxml')
ips = soup.select('#ip_list tr')
fp = open('host.txt','w')
for i in ips:
    try:
        ipp = i.select('td')
        ip = ipp[1].text
        host = ipp[2].text
        fp.write(ip)
        fp.write('\t')
        fp.write(host)
        fp.write('\n')
    except Exception as e :
        print ('no ip !')
fp.close()
```


如果有多个代理 IP，那么可以建立一个 IP 池，每次随机从 IP 池中选择一个进行访问，如果访问失败了，则将其从池中去除即可。

代理检验

有很多 ip 是不可用的，我们可以检验下，代码如下：

```
import requests
import os
from bs4 import BeautifulSoup

os.chdir(r'C:\Users\Administrator\Desktop\scrapy\proxy')
```

```
# headers = {'User-Agent':'Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/56.0.2924.87 Mobile Safari/537.36'}

fp = open('host.txt','r')
ips = fp.readlines()
proxys = list()

for p in ips:
    ip =p.strip('\n').split('\t')
    proxy = 'http:\\' +    ip[0] + ':' + ip[1]
    proxies = {'proxy':proxy}
    proxys.append(proxies)

url = 'https://www.baidu.com'
for pro in proxys:
    try :
        s = requests.get(url,proxies = pro)
        print (s)
    except Exception as e:
        print (e)
```

查看返回的响应状态码。如果是 200，就说明该代理 IP 是可用的。

使用代理 IP：

```
s = requests.get(url, headers = headers, proxies=random.choice(proxys))
```

3.14 如何绕过域名限制（对付反盗链）

什么是 Referer?

Referer 是请求头的一部分，假设 A 站上有 B 站的链接，在 A 站上点击 B 站的链接，请求头会带有 Referer 而 Referer 的值为 A 站的链接。例如，某电商网站在各个搜索引擎和合作友好站点上投放广告进行引流，那么如何评估各个搜索引擎和友好网站的引流效果呢？方法就是查看用户访问的 URL 中的 Referer 值。

有一部分网站会对 Referer 进行检测（一些资源网站的防盗链就是检测 Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加 Headers，将 Referer 值修改为目标网站域名。

“欺骗”

有的网站，如果是来自站外的请求，不允许下载图片。这时可以简单地“欺骗”：设置“Referer” header 头部参数。

```
url = "http://localhost:8084/jxshoping/RefererServlet"

# 有的网站，如果是来自站外的请求，不允许下载图片
# 这时可以简单地“欺骗”：设置“Referer” header
```

```
my_headers = {
    'Referer': 'http://localhost:8084/jxshoping/'
}

print("###############“欺骗”####################")
# response = requests.get(url)    # 无法下载
response = requests.get(url, headers=my_headers)
print(response.text)
```

3.15 如何实现登录状态爬取-Cookie

Cookies 是存储在客户端计算机上的文本文件，并保留了各种跟踪信息。在采集一些网站时 cookie 是不可或缺的。要在一个网站上持续保持登录状态，需要在多个页面中保存一个 cookie。有些网站不要求在每次登录时都获得一个新 cookie，只要保存一个旧的“已登录”的 cookie 就可以访问。

例如，在电商网站中，客户登录以后在不同页面执行“加入购物车”活动。服务器端怎么知道这是同一个用户的购物行为？

识别返回用户包括三个步骤：

1) 服务器脚本向浏览器发送一组 Cookies。例如：姓名、年龄或识别号码等。
2) 浏览器将这些信息存储在本地计算机上，以备将来使用。
3) 当下一次浏览器向 Web 服务器发送任何请求时，浏览器会把这些 Cookies 信息发送到服务器，服务器将使用这些信息来识别用户。

传递 cookies

1）首选手动登录，在浏览器的“开发者工具”里查看 cookie，并复制；

2）在代码中请求 url 时，携带上此 cookie，即可。

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# 豆瓣电影首页的 url
url = "https://movie.douban.com/"

# 设置一个代理
proxies = {"http": "114.239.147.75:808"}

headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36",

"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3",
    "referer":"https://accounts.douban.com/passport/login?redir=https%3A%2F%2Fmovie.douban.com%2F"
}
```

```
# 设置 cookie
cookies = {'cookie':'bid=B5BekwtSopc; douban-fav-remind=1;
__gads=ID=0a3f14ad78739e56:T=1562902870:S=ALNI_MYry55U4n5sr-P8xrg0T163Tm76mQ; ll="118237";
__yadk_uid=lgNw5TQuc6kJ8OH1fXcrRFOgvl0Uqx4t;
_vwo_uuid_v2=DBDDE948A4D76F162F7F2D14F78440F9E|1416f10d9202359658952632bb5edaed;
dbcl2="155364214:gy0CGHBl+ec"; ck=ESOB;
_pk_ref.100001.4cf6=%5B%22%22%2C%22%22%2C1568725180%2C%22https%3A%2F%2Faccounts.douban.
com%2Fpassport%2Flogin%3Fredir%3Dhttps%253A%252F%252Fmovie.douban.com%252F%22%5D;
_pk_id.100001.4cf6=2cfe467fa1fcd5ee.1568725180.1.1568725180.1568725180.; _pk_ses.100001.4cf6=*;
__utma=30149280.1816932353.1568725180.1568725180.1568725180.1; __utmb=30149280.0.10.1568725180;
__utmc=30149280;
__utmz=30149280.1568725180.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/p
assport/login; __utma=223695111.1599022813.1568725180.1568725180.1568725180.1;
__utmb=223695111.0.10.1568725180; __utmc=223695111;
__utmz=223695111.1568725180.1.1.utmcsr=accounts.douban.com|utmccn=(referral)|utmcmd=referral|utmcct=/
passport/login; push_noty_num=0; push_doumail_num=0'}

# 向服务器发起请求，并保存返回的响应对象
response = requests.get(url, cookies=cookies, headers=headers, proxies=proxies)

# 响应的网页内容
html = response.text
```

Cookies 也可以在 Cookie Jar 中传递。它们提供了一个更完整的接口，可以在多个 path 路径上使用这些 cookie。

```
import requests

jar = requests.cookies.RequestsCookieJar()
jar.set('first_cookie', 'first', domain='httpbin.org', path='/cookies')
jar.set('second_cookie', 'second', domain='httpbin.org', path='/extra')
jar.set('third_cookie', 'third', domain='httpbin.org', path='/cookies')

url = 'http://httpbin.org/cookies'
req = requests.get(url, cookies=jar)
req.text
```

Session 对象

有时，跨多个请求保存某些参数是有用的。Session 对象就是这样做的。例如，它将在使用相同会话发出的所有请求之间持久化 cookie 数据。

Session 对象使用 urllib3 的连接池。这意味着底层 TCP 连接将被重用，用于向同一主机发出的所有请求。

这可以显著提高性能。还可以将请求对象的方法与 Session 对象一起使用。

当希望跨所有请求发送相同的数据时，Session 也很有用。例如，如果决定向给定域发送一个 cookie 或一个包含所有请求的 user-agent 头，可以使用 Session 对象。

```
import requests

ssn = requests.Session()
```

```
ssn.cookies.update({'visit-month': 'February'})

reqOne = ssn.get('http://httpbin.org/cookies')
print(reqOne.text)
# 输出"visit-month" cookie 的信息

reqTwo = ssn.get('http://httpbin.org/cookies', cookies={'visit-year': '2017'})
print(reqTwo.text)
# 输出"visit-month" 和"visit-year" cookie 的信息

reqThree = ssn.get('http://httpbin.org/cookies')
print(reqThree.text)
# 输出"visit-month" cookie 的信息
```

3.16 如果网站速度慢怎么办？

设置超时

如果为超时指定一个值，如下所示:

```
r = requests.get('https://github.com', timeout=5)
```

超时值将应用于连接和读取超时。如果想单独设置值，请指定一个元组:

```
r = requests.get('https://github.com', timeout=(3.05, 27))
```

如果远程服务器非常慢，可以告诉 Requests 永远等待响应，方法是传递 None 作为超时值，然后去喝上一杯咖啡。

```
r = requests.get('https://github.com', timeout=None)
```

3.17 如果爬取的内容很大怎么办？

流式请求：

```
r = requests.get('http://httpbin.org/stream/20', stream=True)
for line in r.iter_lines():
    pass
```

或分块请求：

```
r = requests.get('http://httpbin.org/stream/20', stream=True)
for line in iter_content(chunk_size = 1, decode_unicode = False):
    pass
```

因此，这个方法每次以 chunk_size 字节数为单位，遍历响应数据。当对请求设置了 stream=True 时，该方法将避免一次性将整个文件读入内存，只获得较大的响应。

请注意，chunk_size 参数可以是整数，也可以是 None。但是，当设置为整数值时，chunk_size 确定应该立即读入内存的字节数。

当 chunk_size 被设置为 None, stream 被设置为 True 时，数据在到达时将被读取，无论接收到的块的大小和时间。但是，当 chunk_size 被设置为 None, stream 被设置为 False 时，所有数据将只作为一个数据块返回。

3.18 使用 Selenium 爬取 Web 数据

Selenium 是什么？Selenium 是一个 web 测试库，是自动化测试工具，它用于自动化浏览器活动。它支持各种浏览器，包括 Chrome，Safari，Firefox 等主流界面式浏览器，如果你在这些浏览器里面安装一个 Selenium 的插件，那么便可以方便地实现 Web 界面的测试。换句话说叫 Selenium 支持这些浏览器驱动。

官网：https://www.seleniumhq.org/

文档：https://pypi.org/project/selenium/

API：https://seleniumhq.github.io/selenium/docs/api/py/api.html

安装 Selenium：

```
$ pip install selenium
```

Selenium 需要一个驱动程序来与所选的浏览器进行交互。例如，Chrome 浏览器需要 chromedriver，在运行代码之前需要安装（下载）它。确保该驱动程序位于 PATH 路径上。

不同的浏览器都有自己的驱动程序可用。下面是一些最流行的浏览器驱动程序的下载链接。

- Chrome: https://sites.google.com/a/chromium.org/chromedriver/downloads
- Edge: https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
- Firefox: https://github.com/mozilla/geckodriver/releases
- Safari: https://webkit.org/blog/6900/webdriver-support-in-safari-10/

注：windows 系统下载驱动后复制到路径D:\anaconda\Library\bin，这样就不用设置环境变量了 也不用在 driver = webdriver.Chrome() 的 "()" 中输入路径。

WebDriver 是一个用于跨多个浏览器自动测试 web 应用程序的开源工具。它提供了导航到 web 页面、用户输入、JavaScript 执行等功能。ChromeDriver 是一个独立的服务器，它实现了 W3C WebDriver 标准。ChromeDriver 适用于 Android 上的 Chrome 和台式机上的 Chrome (Mac、Linux、Windows 和 ChromeOS)。

ChromeDriver 是一个单独的可执行文件，Selenium WebDriver 使用它来控制 Chrome。

示例 1：自动启动浏览器，打开百度，获取百度首页的 HTML 内容。

```
import time
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

# 执行下面代码前，确保已经将驱动添加到了 PATH 路径上
browser = webdriver.Chrome("../driver/chromedriver.exe")
browser.get('http://www.baidu.com/')
assert '百度' in browser.title

# 获取当前页面标题
title = browser.title
print(title)
```

```
# 获取当前的 url
current_url = browser.current_url
print(current_url)

# 获取当前页面源码
html = browser.page_source
print(html)

time.sleep(5)

browser.quit()
```

示例 2：自动启动浏览器，打开百度，搜索“CDA 数据分析师”，返回搜索结果。

```
import time
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

# 执行下面代码前，确保已经将驱动添加到了 PATH 路径上
browser = webdriver.Chrome("../driver/chromedriver.exe")
browser.get('http://www.baidu.com/')
assert '百度' in browser.title

time.sleep(5)

search_box = browser.find_element_by_name('wd')      # by_name：按 name 属性搜索 input 输入框
search_box.send_keys('CDA 数据分析师' + Keys.RETURN)
search_box.submit()

time.sleep(5)

# 获取当前页面源码（即返回的搜索结果）
html = browser.page_source
print(html)

time.sleep(50)

browser.quit()
```


示例 3：不过每次都要看着浏览器执行这些操作，有时候有点不方便. 我们可以让 selenium 不弹出浏览器窗口，让它”安静”地执行操作。在创建 driver 之前定义几个参数就能摆脱浏览器的身体了。

```
from selenium.webdriver.chrome.options import Options

chrome_options = Options()
chrome_options.add_argument("--headless")          # define headless

driver = webdriver.Chrome(chrome_options=chrome_options)
...
```

使用选择器从返回的 HTML 源码中搜索元素：

- browser.find_element_by_id()
- browser.find_element_by_class_name()
- browser.find_element_by_css_selector()
- browser.find_element_by_name()
- browser.find_element_by_tag_name()
- browser.find_element_by_link_text()
- browser.find_element_by_partial_link_text()
- browser.find_element_by_xpath()
- browser.find_elements_by_id()
- browser.find_elements_by_class_name()
- browser.find_elements_by_css_selector()
- browser.find_elements_by_name()
- browser.find_elements_by_tag_name()
- browser.find_elements_by_link_text()
- browser.find_elements_by_partial_link_text()
- browser.find_elements_by_xpath()

什么时候会要用到 Selenium 呢?

- 发现用普通方法爬不到想要的内容
- 网站跟你玩”捉迷藏”，太多 JavaScript 内容
- 需要像人一样浏览的爬虫

3.18.1 案例：爬取“东方财富网”股票吧评论数据

```
"""
东方财富网，股吧“热门帖子”分页爬取
http://guba.eastmoney.com/default,99_1.html
http://guba.eastmoney.com/default,99_2.html
http://guba.eastmoney.com/default,99_3.html
......
http://guba.eastmoney.com/default,99_12.html

"""
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd
import time
import random


# 定义一个爬取页面的函数
def get_page(page):
    # 定义要请求的 url
    url = "http://guba.eastmoney.com/default,99_%d.html" % page
```

```
    # 配置 webdriver 使用 Chrome 浏览器，必须设置 chrommedriver 路径
    driver = webdriver.Chrome("../driver/chromedriver.exe")

    # 请求
    driver.get(url)

    # 浏览
    # print(driver.page_source)

    # 构造 BeautifulSoup 对象实例
    soup = BeautifulSoup(driver.page_source, "html.parser")

    # 先找到 div.balist
    # div_balist = soup.find("div", attrs={"class":"balist"})
    div_balist = soup.find("div", class_="balist")     # 与上一句等价

    # <ul class="newlist" tracker-eventcode="gb_xgbsy_ lbqy_rmlbdj">
    ul_newlist = div_balist.find_all("li")

    data = []           # 存储所有的帖子
    for li in ul_newlist:
        row = []   # 存储每个帖子
        yuedu = li.find_all("cite")[0].text.strip()             # 阅读
        row.append(yuedu)
        pinglun = li.find_all("cite")[1].text.strip()           # 评论
        row.append(pinglun)
        tieba_a = li.find("a", class_="balink")                 # 贴吧
        if tieba_a:
            tieba = tieba_a.text
        else:
            tieba = None
        row.append(tieba)
        note = li.find("a", class_="note").text             # 帖子标题
        row.append(note)
        author = li.find("font").text                       # 帖子作者
        row.append(author)
        date = li.find("cite", class_="last").text          # 帖子发布时间
        row.append(date)

        data.append(row)
        # print(row)

    columns = ["阅读", "评论", "贴吧", "标题", "作者", "更新时间"]
    df = pd.DataFrame(data=data, columns=columns)
    return df

df_all = pd.DataFrame()
for i in range(1, 3):
    print("----------第%d 页----------" % i)
```

```
    df = get_page(i)
    print(df)
    df_all = df_all.append(df)
    time.sleep(random.randint(1, 9))

print(df_all.shape)


# 将爬取的内容保存到.csv 文件中
# df_all.to_csv("贴吧评论.csv")
```

3.18.2 案例：爬取“东方财富网”股票吧评论数据

```
"""
东方财富网，股吧“热门帖子”分页爬取，模拟用户单击分布导航链接的过程
http://guba.eastmoney.com/default,99_1.html
http://guba.eastmoney.com/default,99_2.html
http://guba.eastmoney.com/default,99_3.html
......
http://guba.eastmoney.com/default,99_12.html

"""
from bs4 import BeautifulSoup
import pandas as pd
import time
import random
from selenium import webdriver
from selenium.webdriver.chrome.options import Options

# 定义一个爬取页面的函数
def parse_page(html):
    # 构造 BeautifulSoup 对象实例
    soup = BeautifulSoup(html, "html.parser")

    # 先找到 div.balist
    # div_balist = soup.find("div", attrs={"class":"balist"})
    div_balist = soup.find("div", class_="balist")  # 与上一句等价

    # <ul class="newlist" tracker-eventcode="gb_xgbsy_ lbqy_rmlbdj">
    ul_newlist = div_balist.find_all("li")

    data = []  # 存储所有的帖子
    for li in ul_newlist:
        row = []  # 存储每个帖子
        yuedu = li.find_all("cite")[0].text.strip()  # 阅读
        row.append(yuedu)
        pinglun = li.find_all("cite")[1].text.strip()  # 评论
        row.append(pinglun)
        tieba_a = li.find("a", class_="balink")  # 贴吧
```

```
            if tieba_a:
                tieba = tieba_a.text
            else:
                tieba = None
            row.append(tieba)
            note = li.find("a", class_="note").text   # 帖子标题
            row.append(note)
            author = li.find("font").text   # 帖子作者
            row.append(author)
            date = li.find("cite", class_="last").text   # 帖子发布时间
            row.append(date)

            data.append(row)
            # print(row)

        columns = ["阅读", "评论", "贴吧", "标题", "作者", "更新时间"]
        df = pd.DataFrame(data=data, columns=columns)
        return df

# 配置 webdriver 使用 Chrome 浏览器，必须设置 chrommedriver 路径
chrome_options = Options()
chrome_options.add_argument("--headless")          # define headless
option.add_experimental_option('excludeSwitches', ['enable-automation'])      # 防止被检测
driver = webdriver.Chrome("../driver/chromedriver.exe", chrome_options=chrome_options)

# 打开首页
url = "http://www.eastmoney.com/"
driver.get(url)

# 模拟点击首页上的“股吧”链接
driver.find_element_by_css_selector("ul[class='mu104']").find_element_by_css_selector('a[class="first-label"]').cl
ick()
time.sleep(2)

# 窗口跳转（切换通过窗口句柄）
allhandles=driver.window_handles                # 获得所有窗口
driver.switch_to.window(allhandles[1])          # 切换到新窗口

df_all = pd.DataFrame()
for page in range(1, 13):
    # 单击“下一页”
    selector = "a[data-page='{}']".format(page)
    driver.find_element_by_id("pageArea").find_element_by_css_selector(selector).click()
    time.sleep(2)

    # 窗口跳转（切换通过窗口句柄）
    allhandles = driver.window_handles          # 获得所有窗口
    driver.switch_to.window(allhandles[1])      # 切换到新窗口

    # 解析新一页中的帖子
```

```
        df = parse_page(driver.page_source)
        print("----------第%d 页----------" % page)
        print(df)
        df_all = df_all.append(df)

print(df_all.shape)

# 将爬取的内容保存到.csv 文件中
df_all.to_csv("贴吧评论.csv")
```