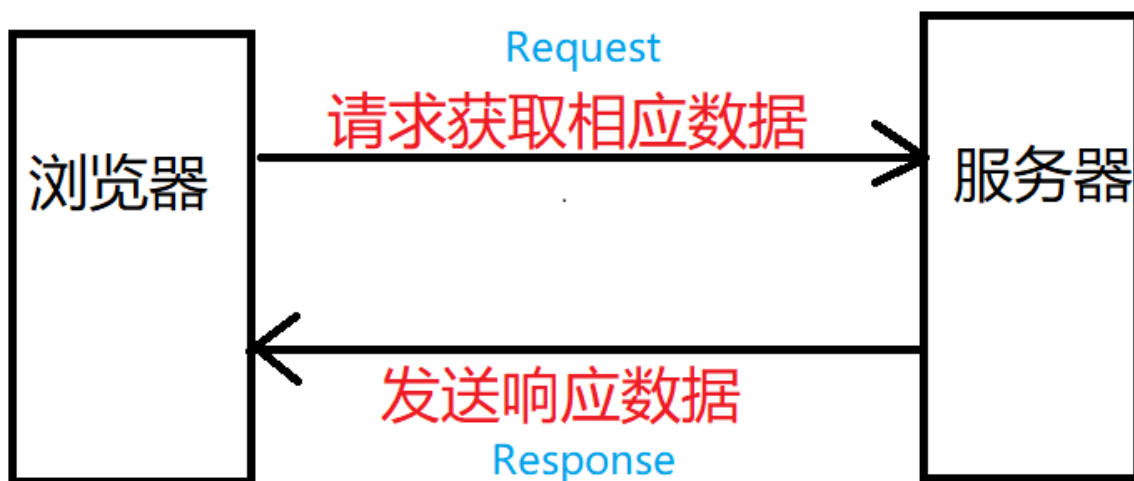


01 Requests模块

网络爬虫——一种批量化地采集网站数据的手段

Requests模块可以模拟浏览器对服务器进行数据的请求：



Requests模块完成的工作就是上述图形所说的

一般的网页会有一个robots协议，用于约束访问的客户端以及禁止访问的内容，比如淘宝的协议：<http://www.taobao.com/robots.txt>，内容如下：

```
User-agent: Baiduspider
Disallow: /
```

```
User-agent: baiduspider
Disallow: /
```

User-Agent：客户端、Disallow：禁止访问

上图这两段结合起来意思就是禁止Baidu访问当前网页的所有内容（/表示根目录）

1.1 Requests基本用法

Requests模块是一个第三方模块，需要先安装模块，在cmd或者pycharm的Terminal中运行如下语句：

```
pip install requests
```

尝试获取百度的html：

```
import requests

# 目标：获取百度搜索首页的html，并写入到本地中
# 获取html，直接使用get请求即可
url = 'https://baidu.com'

# get：发送请求给url对应的服务器
# 返回值：封装过的Response对象，其中包含了服务器响应的数据
```

```
# 这里要获取html，所以服务器返回的响应数据应该是Html
response = requests.get(url)
# 这里通过打断点的方式可以发现response.encoding='ISO-8859-1'
# 所以乱码的问题可以尝试修改编码方式改为utf-8
response.encoding = 'utf-8'

# 将html保存在本地
with open('baidu.html', 'w', encoding='utf-8') as f:
    f.write(response.text)
    # f.write(response.content.decode()) # decode()默认就是解码为utf-8
```

但是目前还存在一个问题：html源代码不完整



```
1 <!DOCTYPE html>
2
3 <!--STATUS OK--><html> <head><meta http-equiv=content
4
5 |
1555 $.setCookie('virus-2020-clicked', '1');
1556 e.removeClass('dot');
1557 });
1558 var hasClicked = $.getCookie && $.getCookie('virus-2020-clicked');
1559 if (!hasClicked) {
1560     e.addClass('dot');
1561 }
1562 });
1563 </script><script src="https://dss0.bdstatic.com/5aV1bjqh_Q23odCf/static/sup
1564     if(navigator.cookieEnabled){
1565         document.cookie="NOJS=;expires=Sat, 01 Jan 2000 00:00:00 GMT";
1566     }
1567 </script><script src="https://dss0.bdstatic.com/5aV1bjqh_Q23odCf/st
```

从百度首页，右键点击“查看网页源代码”，可以发现原始代码是有1500行以上的，而前面脚本获取的只有5行

为什么会出现这个现象？

1.2 Requests请求头Headers

通过 httpbin.org 查看requests发送的请求包含的参数是否完整：

```
import requests
from pprint import pprint

url = 'https://httpbin.org/get' # 该网站可以查看我们通过requests发送的一切参数
# 每次发送请求，必须携带Headers，但这里没有给吗？
response = requests.get(url)
# 获取并打印通过requests发送的内容
pprint(response.json()) # json()可以获取json数据并转为Python的对象

"""
{'args': {},
```

```
'headers': {'Accept': '*/*',
            'Accept-Encoding': 'gzip, deflate, br',
            'Host': 'httpbin.org',
            'User-Agent': 'python-requests/2.27.1',
            'X-Amzn-Trace-Id': 'Root=1-638d49bc-4f116c1828fd28472bd8c627'},
'origin': '116.21.12.158',
'url': 'https://httpbin.org/get'}
''''
```

将运行结果与浏览器进行对比：

```
{
  "args": {},
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Host": "httpbin.org",
    "Sec-Ch-Ua": "\"Google Chrome\";v=\"107\", \"Chromium\";v=\"107\", \"Not=A?Brand\";v=\"24\"",
    "Sec-Ch-Ua-Mobile": "?0",
    "Sec-Ch-Ua-Platform": "\"Windows\"",
    "Sec-Fetch-Dest": "document",
    "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "none",
    "Sec-Fetch-User": "?1",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-638d49b6-7cde36f972df6fa2567ddda7"
  },
  "origin": "116.21.12.158",
  "url": "https://httpbin.org/get"
}
```

最主要的差别
就是这两个
User-Agent

```
{'args': {},
'headers': {'Accept': '*/*',
            'Accept-Encoding': 'gzip, deflate, br',
            'Host': 'httpbin.org',
            'User-Agent': 'python-requests/2.27.1',
            'X-Amzn-Trace-Id': 'Root=1-638d49bc-4f116c1828fd28472bd8c627'},
'origin': '116.21.12.158',
'url': 'https://httpbin.org/get'}
```

根据打印的结果，可以获知如下信息：

- 即使我们没写Headers，requests也会自动构造并携带Headers字典
- User-Agent这个键值对在浏览器和requests发送的区别最大，如果不修改，就相当于告诉对方我们是通过脚本来访问服务器的

使用 Network 查看User-Agent：

Request URL: https://www.baidu.com/
Request Method: GET
Status Code: 200 OK
Remote Address: 14.215.177.39:443
Referrer Policy: strict-origin-when-cross-origin

Response Headers (15)
View source

Request Headers

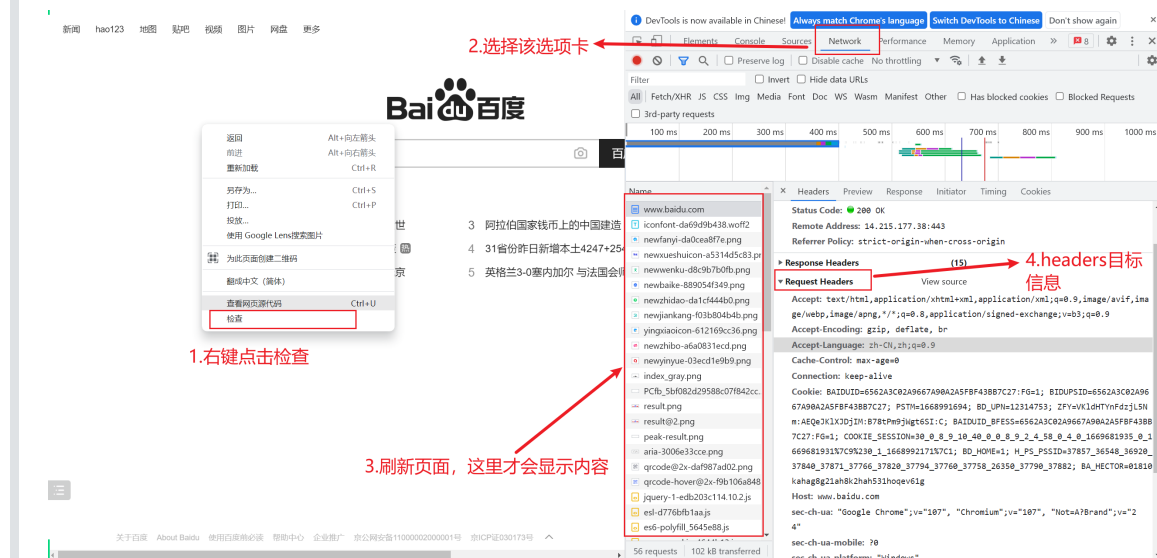
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Cookie: BAIDUID=6562A3C02A9667A90A2A5F8F43887C27;FG=1; BIDUPSID=6562A3C02A9667A90A2A5F8F43887C27; PSTM=1668991694; BD_UPN=12314753; ZFYvK1dTYnfDzjL5Nm:AEQeX1XJ0J1H:878TPn5jngt651:C; BAIDUID_BFESS=6562A3C02A9667A90A2A5F8F43887C27;FG=1; COOKIE_SESSION=30_0_8_9_10_40_0_8_9_2_4_58_0_4_0_1669681935_0_16696819317C9%230_1_1668992171%7C1; BD_HOME=1; H_PS_PSSID=37857_36548_36920_37840_37871_37766_37820_37794_37760_37758_26350_37790_37882; BA_HECTOR=01810kahag8g21ah8K2ah531hoqev6ig
Host: www.baidu.com
sec-ch-ua: "Google Chrome";v="107", "Chromium";v="107", "Not=A?Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36

注意：Headers会被许多网址作为反爬的基本手段，其中如下几个参数是经常成为反爬参数的内容：

- **User-Agent**：它表示浏览器和操作系统的各种信息
- **Host**：主机的域名
- **Refferr**：防盗链
- **Cookie**：用户信息保存的键值对

以上的这些信息，可以通过网页上面点击右键 --> 检查 --> Network选项卡 --> 刷新页面



构造携带Headers的请求：

```
# 只要将Headers构造成字典即可
import requests

# 目标：获取百度搜索首页的html，并写入到本地中
# 获取html，直接使用get请求即可
url = 'https://baidu.com'
# 构造headers
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36'
}
# get请求中可以携带headers参数
response = requests.get(url, headers=headers)
response.encoding = 'utf-8'

# 将html保存在本地
with open('baidu.html', 'w', encoding='utf-8') as f:
    f.write(response.text)
```

现在才能获取到正常的百度页面：

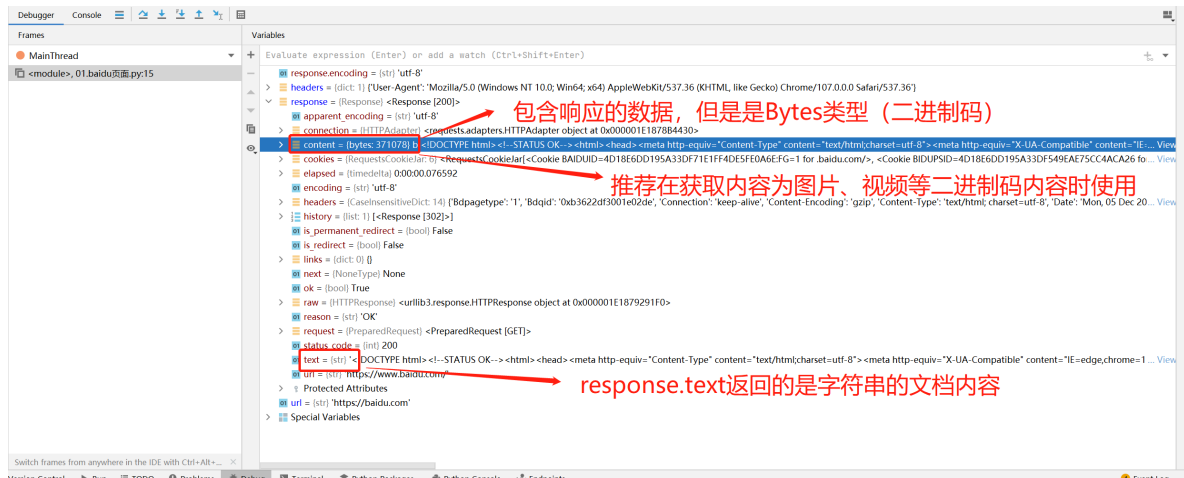


1.3 Response对象

Response实际上就是将从服务器获取的响应数据封装成一个对象。

Response对象常用的属性和方法：

- **response.text**: 获取网页文档内容，获取的内容是字符串
- **response.content**: 获取相应数据内容，类型是Bytes类型
- **response.encoding**: 默认编码方式
- **response.status_code**: 响应状态码，200表示请求成功，3开头表示重定向，4表示客户端的错误/要找的内容在服务器上并不存在，5开头表示服务器出错
- **response.json()**: 将获取到的json数据转为Python对象



对于text和content的梳理：

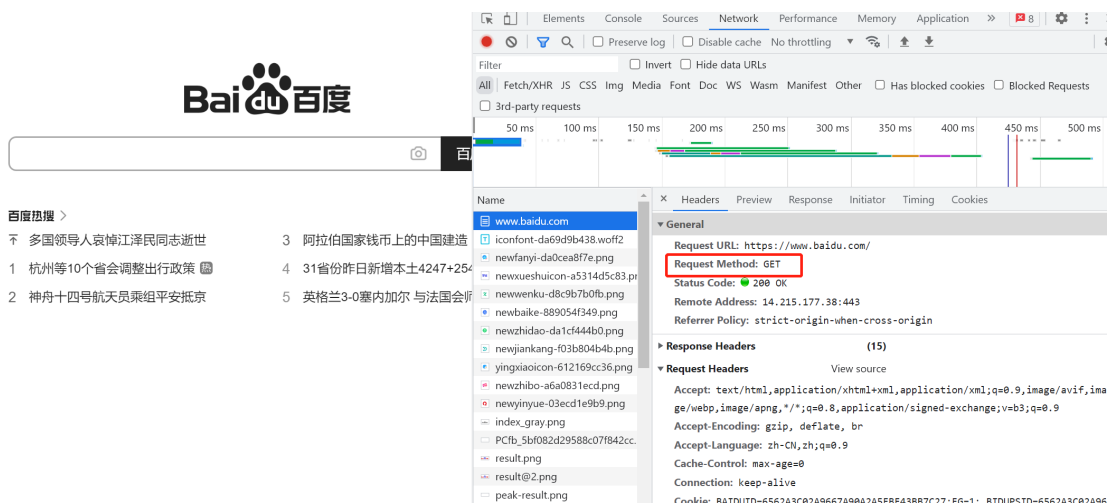
- **content**: 获取的是bytes（二进制字节码的数据），通常可以用于直接获取图片、视频等二进制文件的内容，也可以获取html，但是需要使用decode()解码为字符串
- **text**: 获取的是html的字符串

1.4 请求方法

常用的请求方法：

- **get**: 一般数据获取的接口会使用get
- **post**: 用于发送表单数据，相比get会更安全些
- 其他方法: put、delete、option...

找到某个接口的请求方法：



get和post两者最大的不同，在于请求参数的携带上

1.4.1 get方法的请求参数

```
import requests
url = 'https://httpbin.org/get' # 该网站可以查看我们通过requests发送的一切参数
```

```
# 通过get方法携带请求参数的几种方式
# 方式一：直接写在url的字符串中即可
# 在url结尾开头用 ? 在每个键值对之间用 & 连接
res1 = requests.get(url + '?a=python&b=java&c=golang')
# 获取并打印通过requests发送的内容
print(res1.json()['args']) # {'a': 'python', 'b': 'java', 'c': 'golang'}

# 方式二：使用get方法的params参数，通过字典的形式传入参数
params = {
    'a': 'java',
    'b': 'C++',
    'c': 'golang'
}
res2 = requests.get(url, params=params)
print(res2.json()['args']) # {'a': 'java', 'b': 'C++', 'c': 'golang'}

# 方式一和方式二是可以合并使用的，这样做相当于取参数的并集
res3 = requests.get(url + '?a=python&b=java&c=golang', params=params)
print(res3.json()['args']) # {'a': ['python', 'java'], 'b': ['java', 'C++'], 'c': ['golang', 'golang']}
```

1.4.2 post方法的请求参数

```
import requests
url = 'https://httpbin.org/post'

data = {
    'a': 'java',
    'b': 'C++',
    'c': 'golang'
}
# post方法可以使用data与json来携带表单数据
res1 = requests.post(url, data=data)
print(res1.json()) # 'form': {'a': 'java', 'b': 'C++', 'c': 'golang'} 表单数据
# json参数会将字典转为json字符串再进行传输
res2 = requests.post(url, json=data)
print(res2.json()) # 'data': '{"a": "java", "b": "C++", "c": "golang"}'
# 虽然上面两者看似没什么区别，但实际上最推荐使用json参数

# 另外，post的表单参数和get的请求参数是可以一起用的
res3 = requests.post(url, params=data, json=data)
print(res3.json())
# 'args': {'a': 'java', 'b': 'C++', 'c': 'golang'}
# 'data': '{"a": "java", "b": "C++", "c": "golang"}'
```

02 Html数据解析

爬取数据的页面大致上分为两种：

- HTML渲染的页面
- Ajax+JavaScript渲染的页面

两者的区别：

- HTML是服务器渲染好的结果，也就是服务器提前将Html中所需要的数据全部加载好再发送给客户端（浏览器）

- Ajax是先获取HTML，但是Html不包含任何数据，而是在加载好Html的基本骨架之后额外发送请求去获取该页面的数据，再通过客户端的浏览器渲染页面

注意：最简单的识别方法是，在页面上右键点击“查看源代码”，然后检查是否包含主页的数据，如果包含数据则为Html渲染，如果不包含数据则为Ajax渲染

这里使用的从HTML页面中提取数据的手段是：Xpath

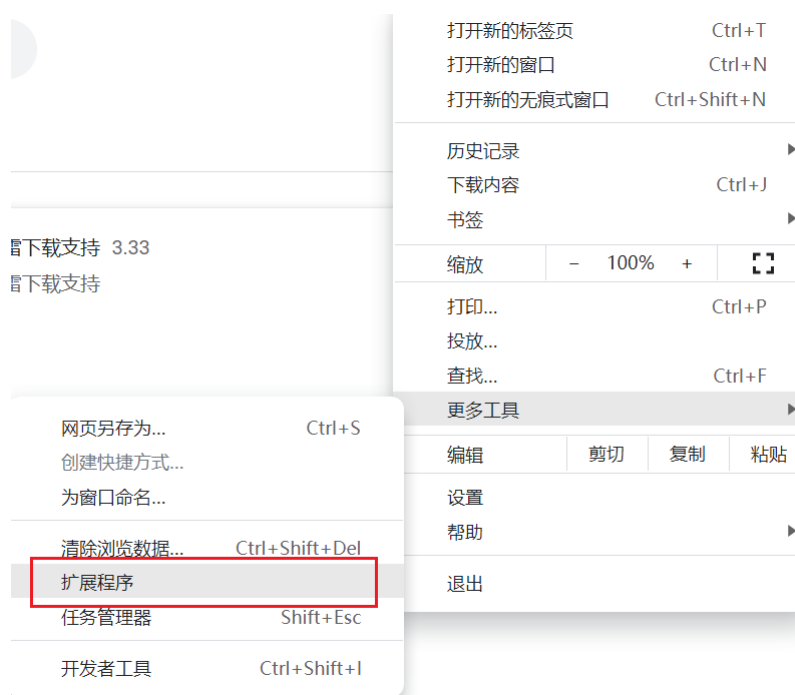
BS4、PyQuery、正则表达式

2.1 Xpath插件安装

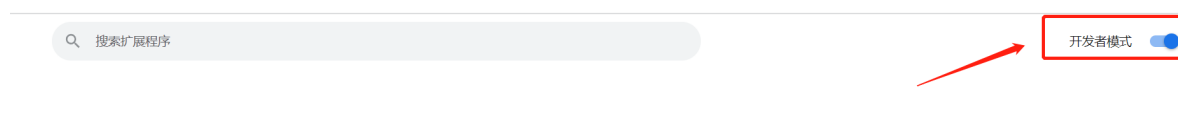
找到群里的 xpath_helper_2_0_2.rar，下载之后进行解压：

名称	状态	修改日期	类型
📁 xpath_helper_2_0_2	✅	2022/11/21 19:22	文件夹

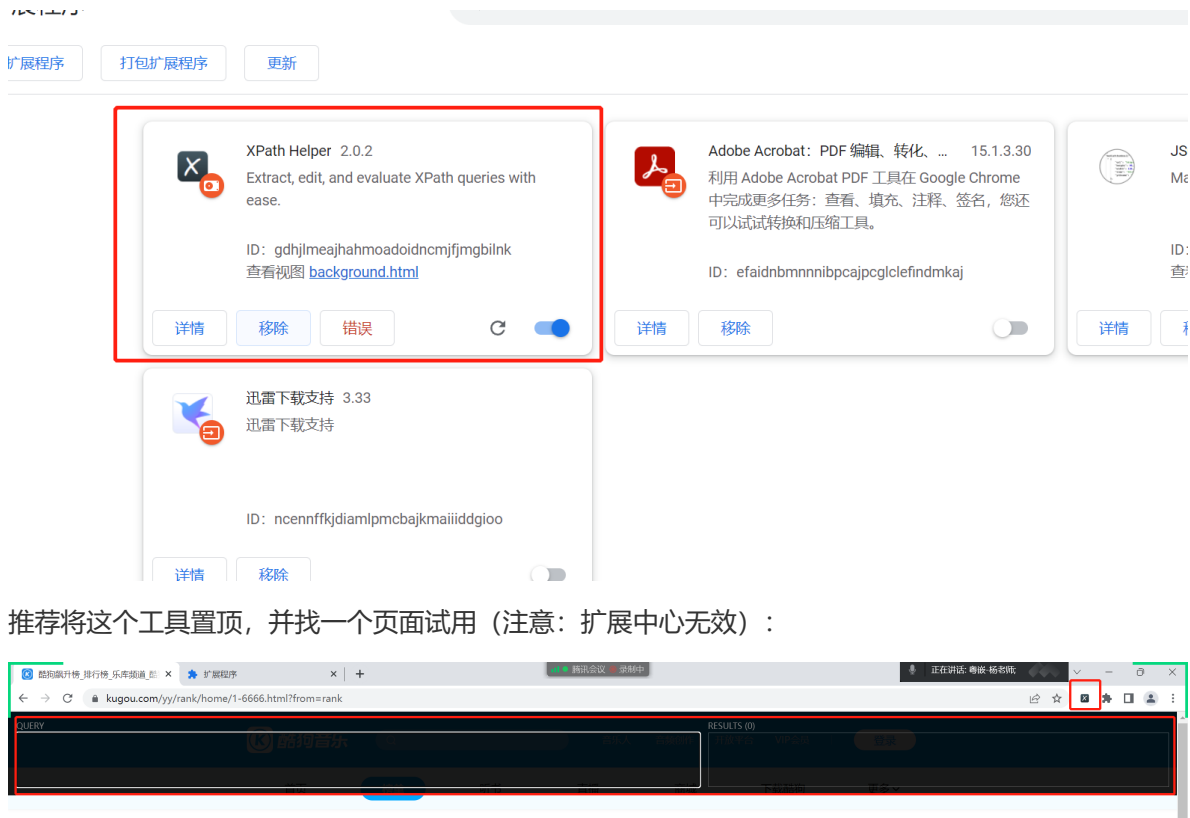
接着打开浏览器的扩展中心：



打开开发者模式：



然后将刚刚解压的文件夹整个拖进来即可：



推荐将这个工具置顶，并找一个页面试用（注意：扩展中心无效）：

2.2 Xpath语法

表达式	描述
nodename	选取此节点的所有子节点
/	从根节点选取，根节点就是标签
//	从匹配选择的当前节点选择文档中的节点，忽略任意的层级去选取符合语法的标签
.	选取当前节点
..	选取当前节点的父节点
@	获取属性值
text()	获取标签中的文本节点
//ul/li[1]	获取ul下的第一个li标签

. 和 .. 可以用相对路径的内容去理解

在浏览器中使用xpath_helper插件完成如下操作：

这里以test.html页面举例，说明一下，在使用xpath_helper时，**被选中的标签会含有 class="xh-highlight" 这个属性**，如果页面内容可见，那么标签内容标黄色，图片内容是变淡

选取html节点: `/html`
选中head标签: `/html/head`
选中body标签: `/html/body` 或者 `//body`

获取两张图片的src: `/html/body/img/@src` 或者 `//img/@src`
获取img的父节点 (body): `//img/..`

获取p标签: `//p`
获取p标签的文本: `//p/text()`

获取第一个ul下的所有歌手名: `//body/ul/li/text()`
获取第一个ul下的所有歌曲: `//body/ul/li/span/text()`
获取第一个ul下的所有歌手和歌曲: `//body/ul/li//text()`

属性选取语法:

表达式	描述
<code>//title[@lang]</code>	选取拥有lang属性的title标签
<code>//title[@lang="eng"]</code>	选取lang=eng的title标签
<code>//title[contains(@lang, "eng")]</code>	选取lang属性包含eng值的title标签
<code>*</code>	表示任意节点, 比如 <code>//*</code> 表示选中页面所有标签
<code>@*</code>	表示选取所有属性

举例:

选中class为list的ul: `//ul[@class="list"]`
使用contains方法: `//ul[contains(@class, "list")]`

选取id为div_text的标签: `//div[@id="div_text"]`
这样写也可以: `//*[@id="div_text"]`

选择class为text的标签: `//*[@class="text"]`
选择class为text的div标签: `//div[@class="text"]`

选择class为block的标签: `//*[@class="block"]`
选择class含有block的标签: `//*[contains(@class, "block")]`

获取含有多个属性的div的所有属性: `//*[@id="div_text"]/@*`

2.3 Python使用xpath语法

安装lxml模块: `pip install lxml`

导出对象: `from lxml import etree`

- 借助 `etree.HTML(html)` 或者 `etree.parse(path, parser)` 两个方法返回的对象即可使用 `xpath` 方法。
 - 上面两个方法获取都是Element对象, 该对象即可使用`xpath()`方法

注意: `xpath()`返回的内容就是一个列表, 列表返回值的情况:

- 返回空列表: `xpath`表达式写错或者html中不存在对应元素

- 返回字符串列表：获取的内容是文本、属性值
- 返回Element元素的列表：代表获取的是标签本身

使用案例：

```
from lxml import etree

# 获取html
html = open('test.html', 'r', encoding='utf-8').read()
# print(html)
# 解析方法有两个：etree.HTML和etree.parse
# 针对字符串的解析使用etree.HTML
html_ele = etree.HTML(html)
print(html_ele) # <Element html at 0x2304f3ae0c0>
# 读取html文件使用etree.parse，参数1为路径，参数2是解析器
# html_ele = etree.parse('./test.html', etree.HTMLParser())
# print(html_ele) # <lxml.etree._ElementTree object at 0x000002420A3D9140>

# 获取所有图片的src
srcs = html_ele.xpath('//img/@src')
print(srcs) # 因为获取的是src的值，所以获取的是字符串的列表
# 获取img标签本身
imgs = html_ele.xpath('//img')
# html中的标签在这里也是Element对象
print(imgs) # [<Element img at 0x276ecfac0c0>, <Element img at 0x276ecfac100>]

# 获取歌曲信息的ul的所有li标签
lis = html_ele.xpath('//body/ul/li')
print(lis) # [<Element li at 0x207ee62c400>, <Element li at 0x207ee62c440>, <Element li at 0x207ee62c480>]
for li in lis: # Element对象本身就可以继续使用xpath
    # name = li.xpath('text()')[0] # 获取歌手名
    name = li.xpath('./text()')[0] # 获取歌手名
    print("歌手: " + name)
    music = li.xpath('span/text()')[0] # 获取歌名
    print("歌曲: " + music)

# 将两个一起获取
# name_music = li.xpath('//text()') # 这里相当于面对整个html进行匹配
# print(li.xpath('/html')) # [<Element html at 0x1eaebd78500>]
name_music = li.xpath('./text()') # 如果要使用//或者 / ，一定要使用 . 表示从当前位置出发
print(name_music)
```

03 爬取案例：酷狗音乐

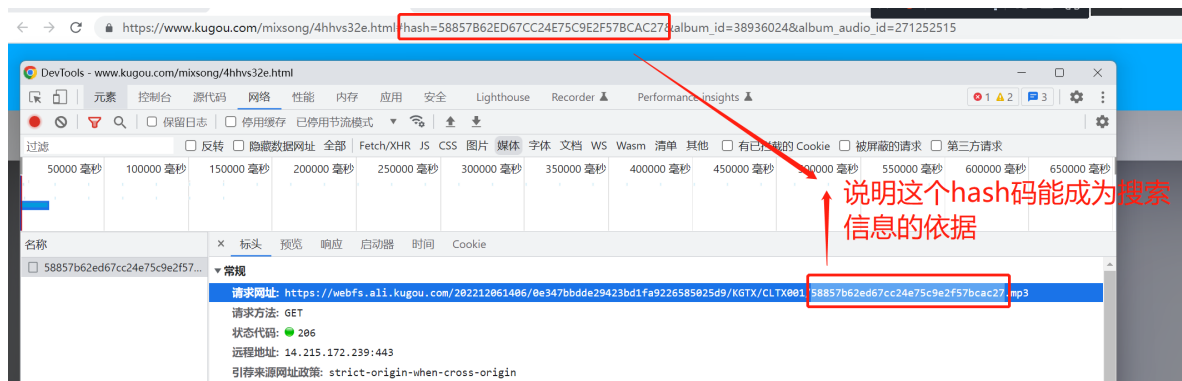
目标页面：<https://www.kugou.com/yy/rank/home/1-6666.html?from=rank>

大致步骤：

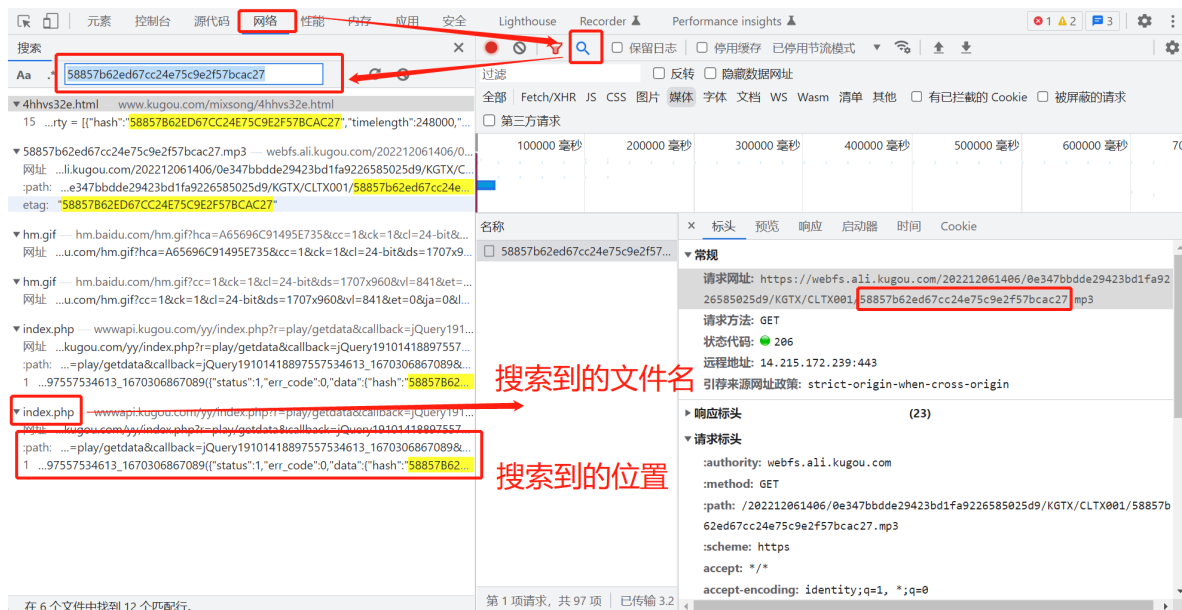
- 获取一首歌
- 获取其中一个榜单的音乐
- 获取整个榜单的音乐

3.1 获取一首歌

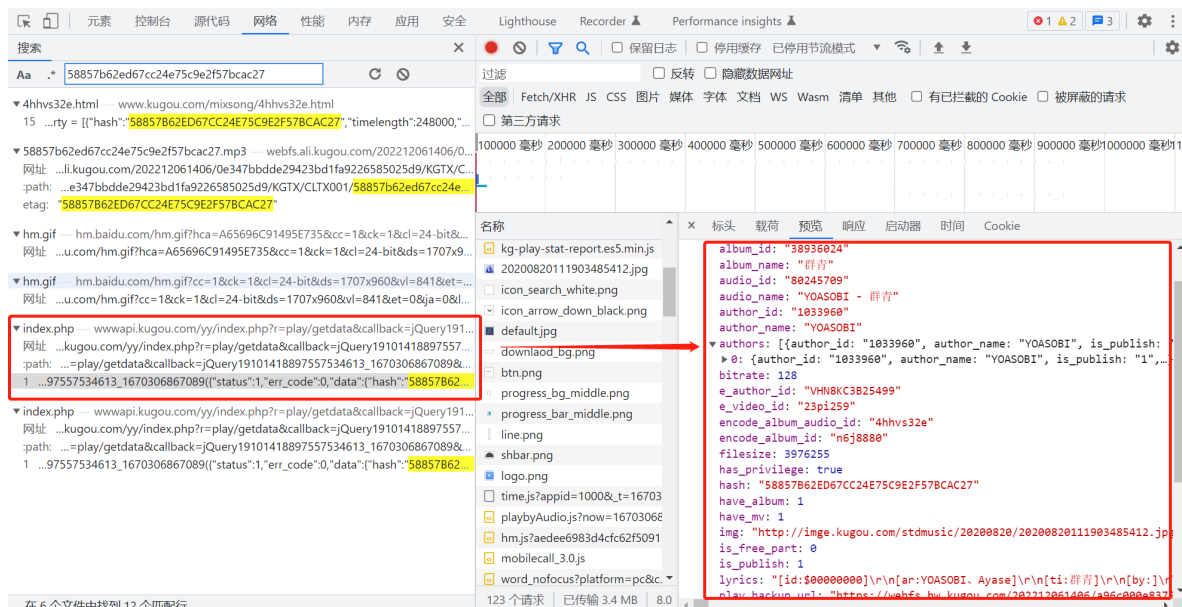
在榜单是上面进入到一首歌的播放页，在播放页点击右键检测（或者F12），然后刷新一下播放页。



将歌曲的hash值放入Network的搜索中：



然后手动筛选搜索结果：

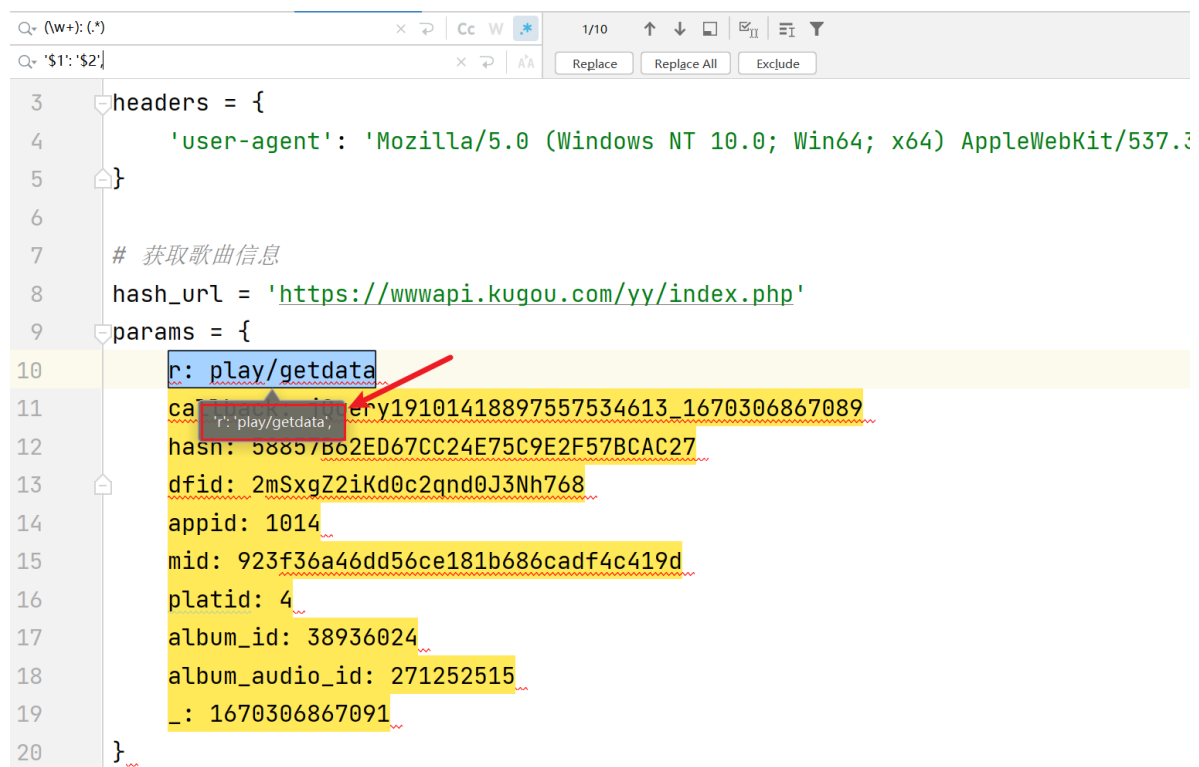


构造该请求所需要的请求参数：

r: play/getdata
callback: jQuery19101418897557534613_1670306867089
hash: 58857B62ED67CC24E75C9E2F57BCAC27
dfid: 2mSxgZ2iKd0c2qnd0J3Nh768
appid: 1014
mid: 923f36a46dd56ce181b686cadf4c419d
platid: 4
album_id: 38936024
album_audio_id: 271252515

_: 1670306867091

复制到Pycharm中，再手动为它们添加引号和逗号：



```
3 headers = {  
4     'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36'  
5 }  
6  
7 # 获取歌曲信息  
8 hash_url = 'https://wwwapi.kugou.com/yy/index.php'  
9 params = {  
10     'r: play/getdata'  
11     'callback': 'jQuery19101418897557534613_1670306867089'  
12     'hash': '58857B62ED67CC24E75C9E2F57BCAC27'  
13     'dfid': '2mSxgZ2iKd0c2qnd0J3Nh768'  
14     'appid': '1014'  
15     'mid': '923f36a46dd56ce181b686cadf4c419d'  
16     'platid': '4'  
17     'album_id': '38936024'  
18     'album_audio_id': '271252515'  
19     '_': '1670306867091'  
20 }
```

按Ctrl + R 调出替换选项，上面用正则表达式选中内容，下面用分组选择语法为它修改内容

上面的正则表达式：(w+):(.*)

下面的表达式：'\$1': '\$2'，（\$1和\$2分别表示正则表达式中的两个括号选中的内容，也就是分组内容）

确定没错之后点击“Replace All”，记得启用正则表达式

```
import requests  
  
headers = {  
    'user-agent': 'Mozilla/5.0 (windows NT 10.0; win64; x64) Applewebkit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36'  
}  
  
# 获取歌曲信息  
hash_url = 'https://wwwapi.kugou.com/yy/index.php'
```

```

params = {
    'r': 'play/getdata',
    # callback选项一般用于转化对象给JavaScript使用，所以这里注释掉
    # 'callback': 'jQuery19101418897557534613_1670306867089',
    'hash': '58857B62ED67CC24E75C9E2F57BCAC27',
    'dfid': '2mSxgZ2iKd0c2qnd0J3Nh768',
    'appid': '1014',
    'mid': '923f36a46dd56ce181b686cadf4c419d',
    'platid': '4',
    'album_id': '38936024',
    'album_audio_id': '271252515',
    '_': '1670306867091',
}
hash_resp = requests.get(hash_url, headers=headers, params=params)
# print(hash_resp.json())
hash_data = hash_resp.json()['data']
audio_name = hash_data['audio_name']
play_url = hash_data['play_url']

# 下载歌曲
music_url = play_url
resp = requests.get(music_url, headers=headers)
with open(audio_name + ".mp3", 'wb') as f:
    f.write(resp.content)
print(f'{audio_name} 已经下载成功! ')

```

3.2 获取一个榜单的音乐