回顾

- 并行和并发
  - 表示程序/计算机具有处理多个任务的能力
    - 并行表示可以同时处理多个任务（几个多核CPU）
    - 并发无法同时处理多个任务，但是可以基于时间片轮转法在多任务间快速切换的执行任务。
- 同步和异步
  - 在基于并行或者并发处理任务的时候，任务中如果出现阻塞操作，就可以选择使用同步或者异步的方式进行处理阻塞操作。
    - 同步处理：让CPU等待阻塞操作结束后，在继续执行处理阻塞后面的其他操作。
    - 异步处理：遇到阻塞操作当前任务会交出CPU的使用权，让CPU可以继续处理其他任务。
      - 因此，在异步应用的过程中，如果一个任务出现了阻塞操作，可以将阻塞操作单独封装成一个单独的任务。
- 进程和线程
  - 是用来实现异步的两种技术手段
  - 在异步应用的过程中，如果一个任务出现了阻塞操作，

可以将阻塞操作单独封装成一个单独的任务，这个任务就是进程或者线程。

协程（重要！）

- 我们知道，无论是多进程还是多线程，在遇到IO阻塞时都会被操作系统强行剥夺走CPU的执行权限(使得cup执行其他操作，其他操作可能是我们程序的其他部分，也可能是其他的应用程序)，我们自己程序的执行效率因此就降低了下来。
- 解决这一问题的关键在于：
  - 我们自己从自己的应用程序级别检测到IO阻塞，然后使得cpu切换到我们自己程序的其他部分/任务执行（这里的任务指的是当前我们自己程序表示的进程或线程中的某一组操作/子程序），这样可以把我们程序的IO阻塞降到最低，我们的程序处于就绪态就会增多，以此来迷惑操作系统，操作系统便以为我们的程序是IO阻塞比较少的程序，从而会尽可能多的分配CPU给我们，这样也就达到了提升程序执行效率的目的。
    - 通俗理解：
      - 一个线程/进程可以表示一组指定行为的操作，这个操作可以由多个执行步骤组成，这些执行步骤有的是阻塞操作有的非阻塞操作，那么，当cpu执行当前进程/线程的时候遇到了阻塞的执行步骤的时

候，如果不对其处理，则包含当前执行步骤的进程/线程就会被挂起进入到阻塞状态，且交出cpu的使用权（cpu就被别人抢走了）。那么如果遇到阻塞的执行步骤，我们的程序可以检测出它是阻塞的，且可以将cpu切换到我们自己程序其他非阻塞的执行步骤时，则包含这些执行步骤的进程/线程就不会进入到阻塞状态，从而减少进程/线程的阻塞状态，增加就绪状态（牢牢抢占cup）极大幅度提升程序执行的效率。

        - 因此，有了协程后，在单进程或者单线程的模式下，就可以大幅度提升程序的运行效率了！

- 在python3.5之后新增了asyncio模块，可以帮我们检测IO（只能是网络IO【HTTP连接就是网络IO操作】），实现应用程序级别的切换（**异步IO**）。

- 接下来让我们来了解下协程的实现，从 Python 3.4 开始，Python 中加入了协程的概念，但这个版本的协程还是以生成器对象为基础的，在 Python 3.5 则增加了 asyncio，使得协程的实现更加方便。首先我们需要了解下面几个概念：

    - 特殊函数：

        - 在函数定义前添加一个async关键字，则该函数就变为了一个特殊的函数！

        - 特殊函数的特殊之处是什么？

            - 1.特殊函数被调用后，函数内部的程序语句（函数

体）没有被立即执行
        - 2.特殊函数被调用后，会返回一个协程对象
    - 协程：
        - 协程对象，特殊函数调用后就可以返回/创建了一个协程对象。
        - 协程对象 == 特殊的函数 == 一组指定形式的操作
            - 协程对象 == 一组指定形式的操作
    - 任务：
        - 任务对象就是一个高级的协程对象。高级之处，后面讲，不着急！
        - 任务对象 == 协程对象 == 一组指定形式的操作
            - 任务对象 == 一组指定形式的操作
    - 事件循环：
        - 事件循环对象（Event Loop）,可以将其当做是一个容器，该容器是用来装载任务对象的。所以说，让创建好了一个或多个任务对象后，下一步就需要将任务对象全部装载在事件循环对象中。
        - 思考：为什么需要将任务对象装载在事件循环对象?
            - 当将任务对象装载在事件循环中后，启动事件循环对象，则其内部装载的任务对象对应的相关操作就会被立即执行。

```
import asyncio
import time
#特殊的函数
async def get_request(url):
    print('正在请求的网址是:',url)
    time.sleep(2)
    print('请求网址结束! ')
    return 123
#创建了一个协程对象
c = get_request('www.1.com')
#创建任务对象
task = asyncio.ensure_future(c)
#创建事件循环对象
loop = asyncio.get_event_loop()
#将任务对象装载在loop对象中且启动事件循环对象
loop.run_until_complete(task)
```

- 任务对象对比协程对象的高级之处重点在于:
  - 可以给任务对象绑定一个回调函数!
  - 回调函数有什么作用?
    - 回调函数就是回头调用的函数，因此要这么理解，当任务对象被执行结束后，会立即调用给任务对象绑定的这个回调函数!

```
import asyncio
import time
```

```
#特殊的函数
async def get_request(url):
    print('正在请求的网址是:',url)
    time.sleep(2)
    print('请求网址结束! ')
    return 123
#回调函数的封装:必须有一个参数
def t_callback(t):
    #参数t就是任务对象
    # print('回调函数的参数t是: ',t)
    # print('我是任务对象的回调函数! ')
    data = t.result()#result()函数就可以返回特殊函数内部的返回值
    print('我是任务对象的回调函数! ,获取到特殊函数的返回值为:',data)
#创建协程对象
c = get_request('www.1.com')
#创建任务对象
task = asyncio.ensure_future(c)
#给任务对象绑定回调函数:add_done_callback的参数就是回调函数的名字
task.add_done_callback(t_callback)

#创建事件循环对象
loop = asyncio.get_event_loop()
loop.run_until_complete(task)
```

- 多任务的协程

```
import asyncio
import time
start = time.time()
urls = [

 'www.1.com','www.2.com','www.3.com'
]
async def get_request(url):
    print('正在请求: ',url)
    time.sleep(2)
    print('请求结束:',url)
#有了三个任务对象和一个事件循环对象
if __name__ == '__main__':
    tasks = []
    for url in urls:
        c = get_request(url)
        task =
asyncio.ensure_future(c)
        tasks.append(task)
    #将三个任务对象，添加到一个事件循环对
象中
    loop = asyncio.get_event_loop()

 loop.run_until_complete(asyncio.wai
t(tasks))
```

```
    print('总耗时:',time.time()-
start)
```

- 出现两个问题：
  - 1.没有实现异步效果
  - 2.wait()是什么意思?

- wait()函数：
  - 给任务列表中的每一个任务对象赋予一个可被挂起的权限！当cpu执行的任务对象遇到阻塞操作的时候，当前任务对象就会被挂起，则cup就可以执行其他任务对象，提高整体程序运行的效率！
  - 挂起任务对象：让当前正在被执行的任务对象交出cpu的使用权，cup就可以被其他任务组抢占和使用，从而可以执行其他任务组。
  - **注意：特殊函数内部，不可以出现不支持异步模块的代码，否则会中断整个异步效果！**
- await关键字：挂起发生阻塞操作的任务对象。在任务对象表示的操作中，凡是阻塞操作的前面都必须加上await关键字进行修饰！
- 完整的实现了，多任务的异步协程操作

```
import asyncio
```

```
import time
start = time.time()
urls = [

 'www.1.com','www.2.com','www.3.com'
]
async def get_request(url):
    print('正在请求：',url)
    # time.sleep(2) #time模块不支持异步
    await asyncio.sleep(2)
    print('请求结束:',url)
#有了三个任务对象和一个事件循环对象
if __name__ == '__main__':
    tasks = []
    for url in urls:
        c = get_request(url)
        task = asyncio.ensure_future(c)
        tasks.append(task)
    #将三个任务对象，添加到一个事件循环对象中
    loop = asyncio.get_event_loop()

 loop.run_until_complete(asyncio.wait(tasks))
```

```
    print('总耗时:',time.time()-
start)
```

- 真正的将多任务的异步协程作用在爬虫中
  - 需求：爬取自己服务器中的页面数据，并将其进行数据解析操作
  - aiohttp:是一个基于网络请求的模块，功能和requests相似，但是，requests是不支持异步的，而aiohttp是支持异步的模块。
    - 环境安装：pip install aiohttp
    - 具体用法：
      - 1.先写大致架构

```
    with
aiohttp.ClientSession() as
sess:
        #基于请求对象发起请求
        #此处的get是发起get请求,
常用参数:
url,headers,params,proxy
        #post方法发起post请求，常
用参数：url,headers,data,proxy
        #发现处理代理的参数和
requests不一样（注意），此处处理代
理使用proxy='http://ip:port'
         with
sess.get(url=url) as
response:
            page_text =
response.text()
            #text():获取字符串形
式的响应数据
            #read()：获取二进制形
式的响应数据
            return page_text
```

- 2.在第一步的基础上补充细节
  - 在每一个with前加上async关键字
  - 在阻塞操作前加上await关键字

- 完整代码：

```
async def
get_request(url):
    #requests是不支持异步的
模块
    # response = await
requests.get(url=url)
    # page_text =
response.text
    #创建请求对象（sess）
    async with
aiohttp.ClientSession()
as sess:
        #基于请求对象发起请
求
        #此处的get是发起get
请求，常用参数：
url,headers,params,proxy
        #post方法发起post
请求，常用参数：
url,headers,data,proxy
        #发现处理代理的参数
和requests不一样（注意），此
处处理代理使用
proxy='http://ip:port'
```

```
            async with await
sess.get(url=url) as
response:
                page_text =
await response.text()
                #text():获取字
符串形式的响应数据
                #read(): 获取
二进制形式的响应数据
                return
page_text
```

- 多任务异步爬虫的完整代码实现：

```
import requests
import asyncio
import time
from lxml import etree
import aiohttp
start = time.time()
urls = [
    'http://127.0.0.1:5000/bobo',
    'http://127.0.0.1:5000/jay',
    'http://127.0.0.1:5000/tom'
]
#该任务是用来对指定url发起请求，获取响应数据
async def get_request(url):
```

```
    #requests是不支持异步的模块
    # response = await
requests.get(url=url)
    # page_text = response.text
    #创建请求对象（sess）
    async with aiohttp.ClientSession()
as sess:
        #基于请求对象发起请求
        #此处的get是发起get请求，常用参数：
url,headers,params,proxy
        #post方法发起post请求，常用参数：
url,headers,data,proxy
        #发现处理代理的参数和requests不一样
（注意），此处处理代理使用
proxy='http://ip:port'
        async with await
sess.get(url=url) as response:
            page_text = await
response.text()
            #text():获取字符串形式的响应数
据
            #read()：获取二进制形式的响应
数据
            return page_text
def parse(t):#回调函数专门用于数据解析
    #获取任务对象请求到的页面源码数据
    page_text = t.result()
```

```
    tree = etree.HTML(page_text)
    a =
tree.xpath('//a[@id="feng"]/@href')[0]
    print(a)

tasks = []
for url in urls:
    c = get_request(url)
    task = asyncio.ensure_future(c)
    task.add_done_callback(parse)
    tasks.append(task)
loop = asyncio.get_event_loop()
loop.run_until_complete(asyncio.wait(t
asks))

print('总耗时:',time.time()-start)
```

异步爬虫具体应用

注意：通常，异步操作主要作用在耗时环节。

同步操作

url：https://www.pkdoutu.com/photo/list/?page=1

```
```

```
import requests
from lxml import etree
import os
dirName = 'imgLibs'
if not os.path.exists(dirName):
    os.mkdir(dirName)
headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/113.0.0.0
Safari/537.36"
}
for page in range(1,2):
    url = 'https://www.pkdoutu.com/photo/list/?
page=%d'%page
    page_text =
requests.get(url,headers=headers).text
    tree = etree.HTML(page_text)
    a_alist = tree.xpath('//*[@id="pic-
detail"]/div/div[2]/div[2]/ul/li/div/div/a')
    for a in a_alist:
        #图片是滑动滚轮后单独加载出来的（图片懒加载）
        img_src = a.xpath('./img/@data-
original')[0]
        img_title = img_src.split('/')[-1]
        img_path = dirName+'/'+img_title
```

```
        img_data =
requests.get(img_src,headers=headers).content
        with open(img_path,'wb') as fp:
            fp.write(img_data)
        print(img_title,': 爬取保存成功! ')
```

多线程爬取斗图

url：https://www.pkdoutu.com/photo/list/?page=1

```
import requests
from lxml import etree
import os
from threading import Thread

dirName = 'imgLibs'
if not os.path.exists(dirName):
    os.mkdir(dirName)

headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/113.0.0.0
Safari/537.36"
}
```

```
# 爬取多页的图片链接，将其存储起来，然后准备后面的异步图片爬取
def get_img_msg():
    img_msg = [] #存储图片相关信息
    for page in range(1,2):
        url = 'https://www.pkdoutu.com/photo/list/?page=%d'%page
        page_text = requests.get(url,headers=headers).text
        tree = etree.HTML(page_text)
        a_alist = tree.xpath('//*[@id="pic-detail"]/div/div[2]/div[2]/ul/li/div/div/a')
        for a in a_alist:
            #图片是滑动滚轮后单独加载出来的（图片懒加载）
            img_src = a.xpath('./img/@data-original')[0]
            img_title = img_src.split('/')[-1]
            dic = {}
            dic['img_url'] = img_src
            dic['img_title'] = img_title
            img_msg.append(dic)
    return img_msg

def get_request(dic):
    img_title = dic['img_title']
```

```
    img_src = dic['img_url']
    img_path = dirName + '/' + img_title
    img_data = requests.get(img_src,
headers=headers).content
    with open(img_path, 'wb') as fp:
        fp.write(img_data)
    print(img_title, ': 爬取保存成功! ')


if __name__ == '__main__':
    img_msg = get_img_msg()
    for dic in img_msg:
        thread = Thread(target=get_request,
args=(dic,))
        thread.start()
```

协程爬取斗图

url：https://www.pkdoutu.com/photo/list/?page=1

```
import requests
from lxml import etree
import os
import aiohttp
import asyncio
```

```
dirName = 'imgLibs'
if not os.path.exists(dirName):
    os.mkdir(dirName)

headers = {
    "User-Agent": "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/113.0.0.0
Safari/537.36"
}

# 爬取多页的图片链接，将其存储起来，然后准备后面的异步图
片爬取
def get_img_msg():
    img_msg = [] #存储图片相关信息
    for page in range(1,2):
        url =
'https://www.pkdoutu.com/photo/list/?
page=%d'%page
        page_text =
requests.get(url,headers=headers).text
        tree = etree.HTML(page_text)
        a_alist = tree.xpath('//*[@id="pic-
detail"]/div/div[2]/div[2]/ul/li/div/div/a')
        for a in a_alist:
```

```
            #图片是滑动滚轮后单独加载出来的（图片懒加载）
            img_src = a.xpath('./img/@data-original')[0]
            img_title = img_src.split('/')[-1]
            dic = {}
            dic['img_url'] = img_src
            dic['img_title'] = img_title
            img_msg.append(dic)
    return img_msg

async def get_request(dic):
    async with aiohttp.ClientSession() as sess:
        async with await sess.get(url=dic['img_url']) as response:
            img_data = await response.read()
            dic['img_data'] = img_data
            return dic

#任务对象回调函数：用来进行图片的持久化存储
def saveImg(task):
    dic = task.result()
    img_path = dirName+'/'+dic['img_title']
    with open(img_path, 'wb') as fp:
        fp.write(dic['img_data'])
    print(dic['img_title'], '：爬取保存成功！')
```

```
if __name__ == '__main__':
    img_msg = get_img_msg()
    tasks = []
    for dic in img_msg:
        c = get_request(dic)
        task = asyncio.ensure_future(c)
        task.add_done_callback(saveImg)
        tasks.append(task)
    loop = asyncio.get_event_loop()

 loop.run_until_complete(asyncio.wait(tasks))
```

M3U8流视频数据爬虫

HLS技术介绍

现在大部分视频客户端都采用HTTP Live Streaming（HLS，Apple为了提高流播效率开发的技术），而不是直接播放MP4等视频文件。HLS技术的特点是将流媒体切分为若干【TS片段】（比如几秒一段），然后通过一个【M3U8列表文件】将这些TS片段批量下载供客户端播放器实现实时流式播放。因此，在爬取HLS的流媒体文件的思路一般是先【下载M3U8文件】并分析其中内容，然后在批量下载文件中定义的【TS片段】，最后将其【组合】成mp4文件或者直接保存TS片段。

M3U8文件详解

如果想要爬取HLS技术下的资源数据，首先要对M3U8的数据结构和字段定义非常了解。M3U8是一个扩展文件格式，由M3U扩展而来。那么什么事M3U呢?

M3U文件

M3U这种文件格式，本质上说不是音频视频文件，它是音频视频文件的列表文件，是纯文本文件。

M3U这种文件被获取后，播放软件并不是播放它，而是根据它的记录找到媒体的网络地址进行在线播放。也就是说，M3U格式的文件只是存储多媒体播放列表，并提供了一个指向其他位置的音频视频文件的索引，播放的是那些被指向的文件。

为了能够更好的理解M3U的概念，我们先简单做一个M3U文件（myTest.m3u）。在电脑中随便找几个MP3，MP4文件依次输入这些文件的路径，myTest.m3u文件内容如下

```
E:\Users\m3u8\刘德华 - 无间道.mp4
E:\Users\m3u8\那英 - 默.mp3
E:\Users\m3u8\周杰伦 - 不能说的秘密.mp4
E:\Users\m3u8\花粥 - 二十岁的某一天.mp3
E:\Users\m3u8\周深 - 大鱼.mp4
```

M3U8文件

M3U8也是一种M3U的扩展格式（高级的M3U，所以也属于M3U）。下面我们将了解一下M3U8中定义的几个非常重要的关键字：

```
#EXTM3U:每个M3U文件第一行必须是这个tag标识。(简单了解)

#EXT-X-VERSION:版本，此属性可用可不用。(简单了解)

#EXT-X-TARGETDURATION:目标持续时间，是用来定义每个TS的【最大】duration（持续时间）。(简单了解)

#EXT-X-ALLOW-CACHE是否允许允许高速缓存。(简单了解)

#EXT-X-MEDIA-SEQUENCE定义当前M3U8文件中第一个文件的序列号，每个ts文件在M3U8文件中都有固定唯一的序列号。(简单了解)

#EXT-X-DISCONTINUITY:播放器重新初始化(简单了解)



#EXTINF:指定每个媒体段(ts文件)的持续时间，这个仅对其后面的TS链接有效，每两个媒体段(ts文件)间被这个tag分隔开。(简单了解)

#EXT-X-ENDLIST表明M3U8文件的结束。(简单了解)
```

M3U8示例：大家会看到在该文件中有大量的ts文件的链接地址，这个就是我们之前描述的真正的视频文件。其中任何一个ts文件都是一小段视频，可以单独播放。我们做视频爬虫的目标就是把这些ts文件都爬取下来。

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:6
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:3.127,
/20230512/RzGw5hDB/1500kb/hls/YZefAiEF.ts
#EXTINF:3.127,
/20230512/RzGw5hDB/1500kb/hls/FsliUCL6.ts
#EXTINF:3.127,
/20230512/RzGw5hDB/1500kb/hls/DD7c47bz.ts
#EXT-X-ENDLIST
```

实战

需求：

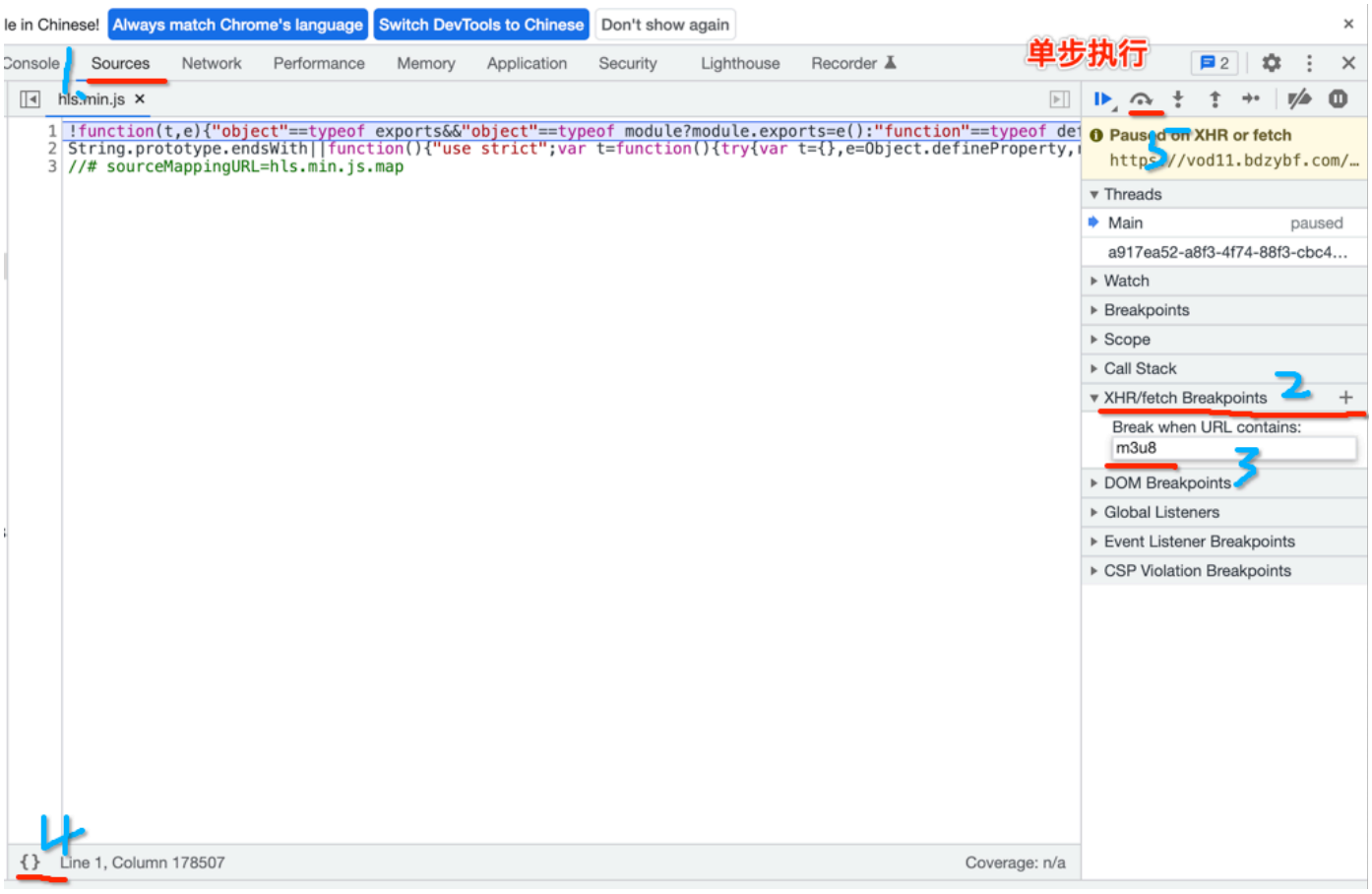
- https://www.99meijutt.com/

具体操作

打开开发者工具，定位到Sources选项卡。在“Sources”选项卡（顶端）中，点击“XHR/fetch Breakpoints”的加号，在编辑框中输入“m3u8”，本操作的意思是：当网页中有任何URL中包含m3u8字段的网络访问就暂停执行。

完成以上操作后重新刷新页面，此时Chrome开发者工具就会暂停在访问M3U8的代码位置。暂停后，我们可以点击左下方的{}图标提高暂停部分代码的可读性。

然后点击”单步“执行按钮，单步执行js代码，执行几部后，会看到一级m3u8的文件地址使用requests进行请求发送，获取一级m3u8文件内容。



- 同步操作代码

```
import requests
from urllib.parse import urljoin
import re
import os
# 需要安装. pip install pycryptodome
from Crypto.Cipher import AES
dirName = 'tsLib'
if not os.path.exists(dirName):
    os.mkdir(dirName)

headers  = {
    'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36'
}
#一级m3u8地址
m1_url = "https://c2.monidai.com/20230415/7Z6l3a9h/index.m3u8"
m1_page_text = requests.get(url=m1_url,headers=headers).text
#从一级m3u8文件中解析出二级m3u8地址
m1_page_text = m1_page_text.strip()#取出收尾的回车
#二级m3u8地址
m2_url = ''
```

```
for line in m1_page_text.split('\n'):
    if not line.startswith('#'):
        m2_url = line
        #补充m2_url:缺少域名
        m2_url =
'https://h0.rzisytn.cn/'+m2_url
        #至此就获取到了完整的二级文件地址
#请求二级文件地址内容
m2_page_text =
requests.get(url=m2_url,headers=headers).text
m2_page_text = m2_page_text.strip()

#解析出每一个ts切片的地址
ts_url_list = []
for line in m2_page_text.split('\n'):
    if not line.startswith('#'):
        ts_url = line
        #不同ts下载地址
        ts_url = 'https://h0.rzisytn.cn'+ts_url
        ts_url_list.append(ts_url)

#请求到每一个ts切片的数据
for url in ts_url_list:
    #获取ts片段的数据
    ts_data =
requests.get(url=url,headers=headers).content
    ts_name = url.split('/')[-1]
```

```
    ts_path = dirName+'/'+ts_name
    with open(ts_path,'wb') as fp:
        #需要将解密后的数据写入文件进行保存
        fp.write(ts_data)
    print(ts_name,'下载保存成功！')

# ts文件的合并，最好网上找专业的工具进行合并，自己手动合并会经常出问题
```

- 异步操作代码

```
```

练习

- 注意：
  - m3u8文件地址需要自己尝试不同的拼接方式探究真正的地址是什么
  - 如果不知道ts的下载地址，可以在浏览器抓包工具中查看ts数据包的url地址，找寻ts完整地址的规律。
- 网站：https://www.meijuw.com/