爬虫简介

什么是爬虫?

是一种按照一定的规则，自动地抓取互联网信息的程序或者脚本。

所谓网页抓取，就是把 URL 地址中指定的网络资源从网络流中读取出来，保存到本地。 在 Python 中有很多库可以用来抓取网页

分类

通用爬虫（General Purpose Web Crawler)、聚焦爬虫（Focused Web Crawler)、增量式爬虫（Incremental Web Crawler)、深层爬虫（Deep Web Crawler）

通用网络爬虫

搜索引擎抓取系统（Baidu、Google、Yahoo 等）的重要组成部分。主要目的是将互联网上的网页下载到本地，形成一个互联网内容的镜像备份。

聚焦爬虫

是"面向特定主题需求"的一种网络爬虫程序，它与通用搜索引擎爬虫的区别在于： ***聚焦爬虫在实施网页抓取时会对内容进行处理筛选，尽量保证只抓取与需求相关的网页信息。***

增量式抓取

是指在具有一定量规模的网络页面集合的基础上，采用更新数据的方式选取已有集合中的过

时网页进行抓取，以保证所抓取到的数据与真实网络数据足够接近。进行增量式抓取的前提是，系统已经抓取了足够数量的网络页面，并具有这些页面被抓取的时间信息。

深度爬虫

针对起始 url 地址进行数据采集，在响应数据中进行数据筛选得到需要进行数据采集的下一波 url 地址，并将 url 地址添加到数据采集队列中进行二次爬取..以此类推，一致到所有页面的数据全部采集完成即可完成深度数据采集，这里的深度指的就是 url 地址的检索深度。

爬虫步骤

网页抓取，数据提取，数据存储

HTTP 协议

HTTP，HyperText Transfer Protocol，是互联网上应用最为广泛的一种网络协议。

是一个基于 TCP/IP 通信协议来传递数据，一个属于应用层的协议

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。

HTTPS（Hypertext Transfer Protocol over Secure Socket Layer）HTTP 的安全版，在 HTTP 下加入 SSL 层。

SSL（Secure Sockets Layer 安全套接层）主要用于 Web 的安全传输协议，在传输层对网络连接进行加密，保障在 Internet 上数据传输的安全。

- `HTTP` 的端口号为 `80`，
- `HTTPS` 的端口号为 `443`

urlib 的使用

在 Python3.x 中，我们可以使用 urlib 这个组件抓取网页，urllib 是一个 URL 处理包，这个包中集合了一些处理 URL 的模块

1.**urllib.request** 模块是用来打开和读取 URLs 的；

2.**urllib.error** 模块包含一些有 urllib.request 产生的错误，可以使用 try 进行捕捉处理；

3.**urllib.parse** 模块包含了一些解析 URLs 的方法；

4.**urllib.robotparser** 模块用来解析 robots.txt 文本文件.它提供了一个单独的 RobotFileParser 类，通过该类提供的 can_fetch()方法测试爬虫是否可以下载一个页面。

urllib.request.urlopen()

这个接口函数就可以很轻松的打开一个网站，读取并打印信息。

```
from urllib import request


if __name__ == "__main__":
    response = request.urlopen("http://fanyi.baidu.com")
    html = response.read()
    print(html)
```

说明

request.urlopen()打开和读取 URLs 信息，返回对象 response

可以调用 read()，进行读取。

print()，将读到的信息打印出来。

其实这就是浏览器接收到的信息，只不过我们在使用浏览器的时候，浏览器已经将这些信息转化成了界面信息供我们浏览。

网页编码

虽然我们已经成功获取了信息，但是显然他们都是二进制的乱码

可以通过简单的 decode()命令将网页的信息进行解码

```
from urllib import request


if __name__ == "__main__":
    response = request.urlopen("http://fanyi.baidu.com/")
    html = response.read()
    html = html.decode("utf-8")
    print(html)
```

自动获取网页编码

chardet

第三方库，用来判断编码的模块

使用 chardet.detect()方法，判断网页的编码方式

安装方法：

pip install chardet

代码：

```
from urllib import request

import chardet


if __name__ == "__main__":
    response = request.urlopen("http://fanyi.baidu.com/")
    html = response.read()
    charset = chardet.detect(html)
    print(charset)
```

Request 对象

反爬虫机制：

1、封杀爬虫程序

2、封杀指定 IP

3、封杀非人操作的程序

如果需要执行更复杂的操作，比如增加 HTTP 报头，必须创建一个 Request 实例来作为 urlopen()的参数；而需要访问的 url 地址则作为 Request 实例的参数。

```
from urllib import request

if __name__ == "__main__":
    req = request.Request("http://fanyi.baidu.com/")
    response = request.urlopen(req)
```

```
    html = response.read()
    html = html.decode("utf-8")
    print(html)
```

User Agent

浏览器就是互联网世界上公认被允许的身份，如果我们希望我们的爬虫程序更像一个真实用户，那我们第一步，就是需要伪装成一个被公认的浏览器。用不同的浏览器在发送请求的时候，会有不同的 User-Agent 头。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。

urllib 中默认的 User Agent，会有 Python 的字样，如果服务器检查 User Agent，可以拒绝 Python 程序访问网站。

常见浏览器 User-Agent:

https://blog.csdn.net/Kwoky/article/details/80381246

设置 User Agent

方法 1：在创建 Request 对象的时候，填入 headers 参数(包含 User Agent 信息)，这个 Headers 参数要求为字典；

方法 2：在创建 Request 对象的时候不添加 headers 参数，在创建完成之后，使用 add_header()的方法，添加 headers。

代码：

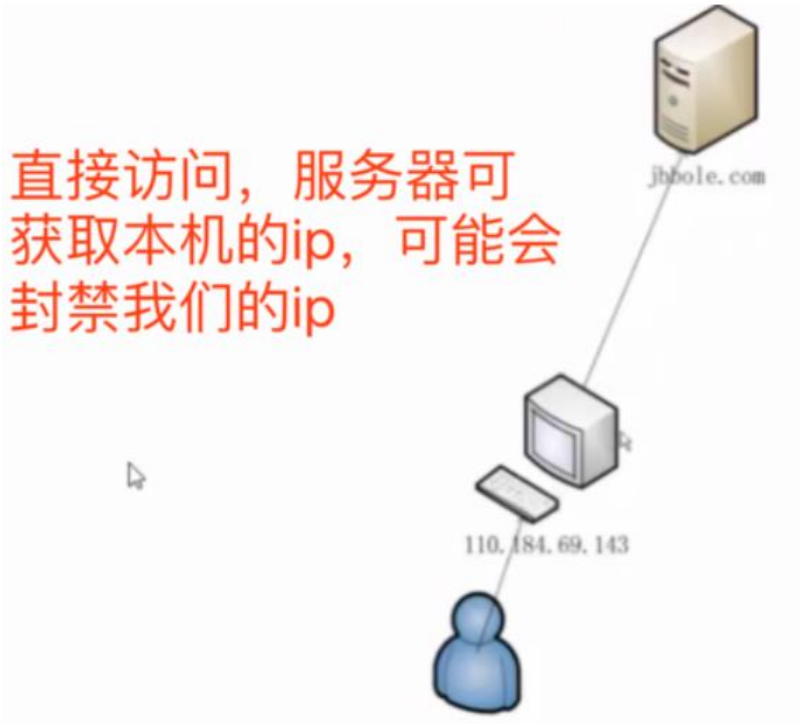
```
from urllib import request

if __name__ == "__main__":
    #以 CSDN 为例，CSDN 不更改 User Agent 是无法访问的
```

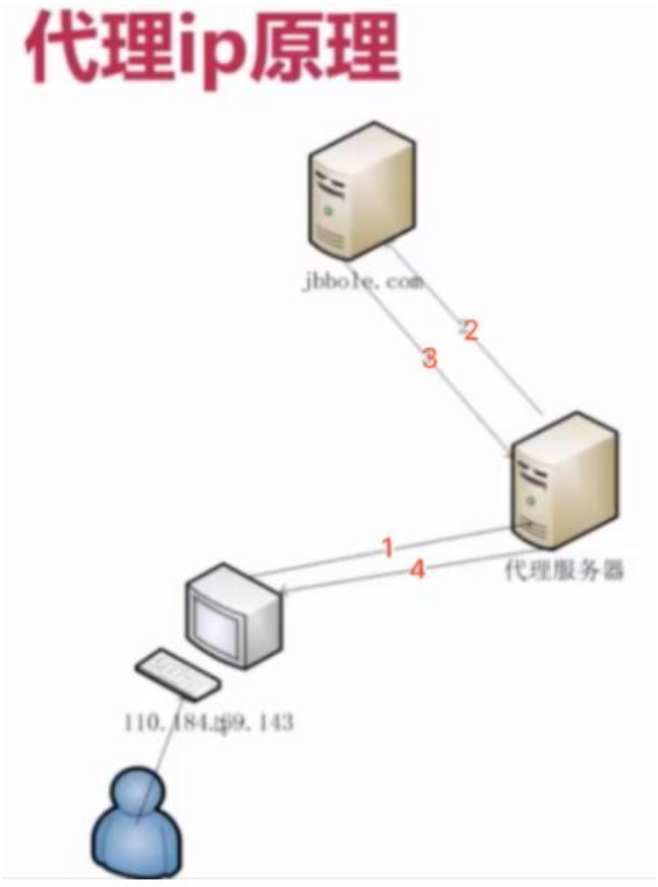
```
    url = 'http://www.csdn.net/'
    head = {}
    #写入 User Agent 信息
    head['User-Agent'] = 'Mozilla/5.0 (Linux; Android 4.1.1; Nexus 7 Build/JRO03D) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166  Safari/535.19'
    #创建 Request 对象
    req = request.Request(url, headers=head)
    # 也可以通过调用 Request.add_header() 添加/修改一个特定的 header
    #req.add_header("User-Agent", "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")
    req.add_header("Connection", "keep-alive")
    # 也可以通过调用 Request.get_header()来查看 header 信息
    print(req.get_header(header_name="Connection"))
    print(req.get_header(header_name="User-Agent"))
    #传入创建好的 Request 对象
    response = request.urlopen(req)
    #读取响应信息并解码
    html = response.read().decode('utf-8')
    #打印信息
    print(html)
```

IP 代理

直接访问网站



使用 ip 代理之后访问网站，可避免本机的 ip 暴露：



代理分类

代理可以分为三种，即高度匿名代理、普通匿名代理和透明代理。

高度匿名代理 隐藏客户的真实 IP，但是不改变客户机的请求，就像有个真正的客户浏览器在访问服务器。

普通匿名代理 能隐藏客户机的真实 IP，会改变客户的请求信息，服务器端不知道你的 ip 地址但有可能认为我们使用了代理。

透明代理 不但改变了我们的请求信息，还会传送真实的 IP 地址。

爬虫程序运行速度是很快,在网站爬取数据时，一个固定 IP 的访问频率就会很高，这不符合人为操作的标准。所以一些网站会设置一个 IP 访问频率的阈值，如果一个 IP 访问频率超过这个阈值，说明这个不是人在访问，而是一个爬虫程序。

解决办法一:设置延时

解决办法二:使用 IP 代理。可以设置一些代理服务器，每隔一段时间换一个代理，就算 IP 被禁止，依然可以换个 IP 继续爬取。

免费短期代理网站举例：

西刺免费代理 IP　http://www.xicidaili.com/

快代理免费代理　http://www.kuaidaili.com/free/inha/

聚合数据 https://www.juhe.cn/docs/api/id/62

代理 IP 选取

西刺网站为例，选出信号好的 IP

国家	IP地址	端口	服务器地址	是否匿名	类型	速度	连接时间	存活时间	验证时间
	114.222.164.173	808	江苏南京	高匿	HTTPS			22小时	18-05-20 13:20
	27.40.132.98	61234	广东湛江	高匿	HTTPS			3小时	18-05-20 13:20
	27.197.111.186	8118	山东临沂	高匿	HTTPS			13小时	18-05-20 13:20
	118.122.92.252	37901	四川成都	高匿	HTTPS			37天	18-05-20 13:18
	171.12.182.133	34281	河南	高匿	HTTP			1分钟	18-05-20 13:15
	1.25.106.84	61234	内蒙古	高匿	HTTPS			12小时	18-05-20 13:15
	219.145.254.198	23545	陕西商洛	高匿	HTTP			1分钟	18-05-20 13:15
	27.156.234.243	22929	福建福州	高匿	HTTP			1分钟	18-05-20 13:15
	182.34.54.48	29855	山东烟台	高匿	HTTPS			132天	18-05-20 13:15
	1.199.192.108	30822	河南新乡	高匿	HTTP			1分钟	18-05-20 13:15
	1.197.178.96	46112	河南南阳	高匿	HTTP			1分钟	18-05-20 13:15
	183.151.42.65	8070	浙江丽水	高匿	HTTP			1分钟	18-05-20 13:15
	60.182.186.218	27885	浙江金华	高匿	HTTP			1分钟	18-05-20 13:15
	183.157.169.1	8118	浙江杭州	高匿	HTTPS			4小时	18-05-20 13:13
	180.125.136.119	8000	江苏苏州	高匿	HTTPS			9小时	18-05-20 13:11
	122.114.31.177	808	河南郑州	高匿	HTTP			174天	18-05-20 13:11

使用代理的步骤：

(1)调用 urlib.request.ProxyHandler()，构建处理器对象，proxies 参数为一个字典。

class urllib.request. **ProxyHandler**(*proxies=None*)

(2)创建 Opener 对象

urllib.request. **build_opener**([*handler*, ...])

(3)安装 Opener

urllib.request. **install_opener**(*opener*)

使用 install_opener 方法之后，会将程序默认的 urlopen 方法替换掉。也就是说，如果使用 install_opener 之后，在该文件中，再次调用 urlopen 会使用自己创建好的 opener。

代码：

```
if __name__ == "__main__":
    #访问网址
    #url = 'http://www.baidu.com/'
    url='http://ip.27399.com/'
    #这是代理 IP
    proxy = {'http':'113.92.156.185:808'}
    #创建 ProxyHandler
    proxy_support = request.ProxyHandler(proxy)
    #创建 Opener
    opener = request.build_opener(proxy_support)
    #添加 User Angent
    opener.addheaders = [('User-Agent','Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36')]
```

```
#安装 OPener
request.install_opener(opener)
#使用自己安装好的 Opener
response = request.urlopen(url)
#读取相应信息并解码
html = response.read().decode("utf-8")
#打印信息
print(html)
```

注：http://ip.27399.com/ ip 地址测试网站

url 编码解码

urllib.parse.urlencode()函数帮我们将 key:value 这样的键值对转换成"key=value"这样的

的字符串，解码工作可以使用 urllib 的 unquote()函数。

```
from urllib import parse
word = {"title" : "天猫商城"}
url1 = parse.urlencode(word)
print(url1)
url2 =parse.unquote(url1)
print(url2)
```

运行结果：

```
title=%E5%A4%A9%E7%8C%AB%E5%95%86%E5%9F%8E
title=天猫商城

进程已结束,退出代码0
```

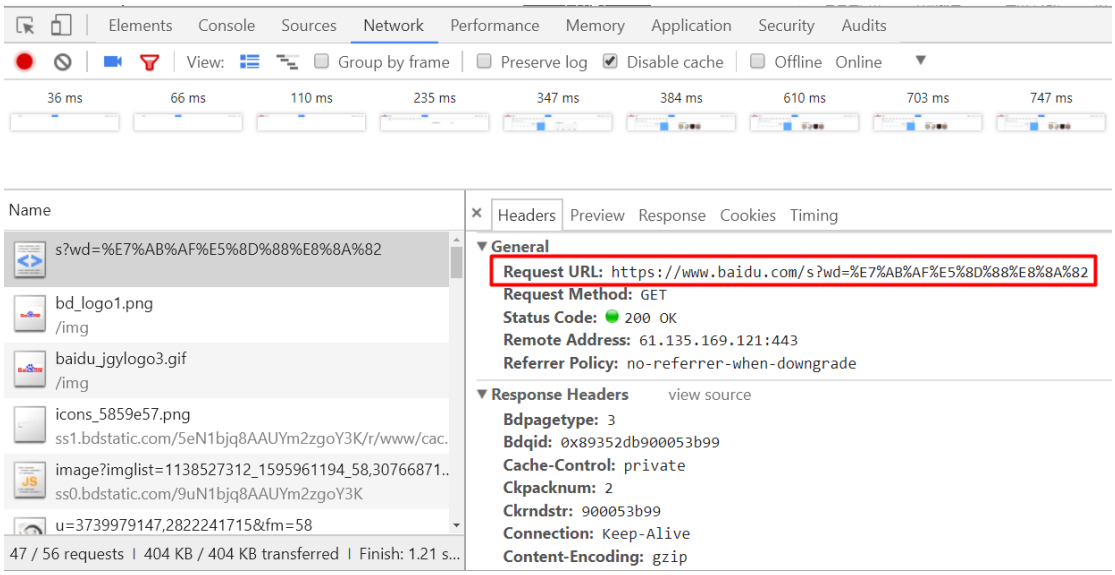
GET 请求

GET 请求一般用于向服务器获取数据

案例：百度搜索端午节

python 之——用 urllib 库爬取数据

审查元素



在其中可以看到在请求部分里，http://www.baidu.com/s? 之后出现一个长长的字符串，

其中就包含我们要查询的关键词，于是我们可以尝试用默认的 Get 方式来发送请求。

代码

```
from urllib import request
from urllib import parse
import chardet

url = "http://www.baidu.com/s"
word = {"wd":"端午节"}
word = parse.urlencode(word) #转换成 url 编码格式（字符串）
newurl = url + "?" + word        # url 首个分隔符就是 ?
headers={ "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36"}
req = request.Request(newurl, headers=headers)
response = request.urlopen(req)
html = response.read()
charset = chardet.detect(html)['encoding']
print(charset)
print(html.decode(charset))
```

POST 请求

urlopen 的 data 参数，是 POST 方法进行 Request 请求时，要传送给服务器的数据，字典

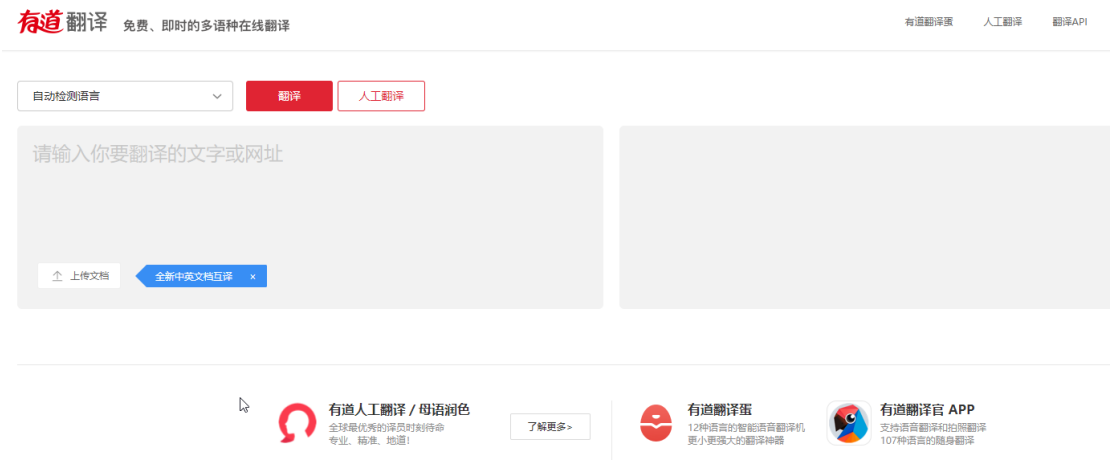
类型，里面要匹配键值对。

如果没有设置 urlopen()函数的 data 参数，HTTP 请求采用 GET 方式

使用 urllib.parse.urlencode()函数将发送的表单数据编码

案例：向有道翻译发送 data，得到翻译结果

1、打开有道翻译界面 http://fanyi.youdao.com/



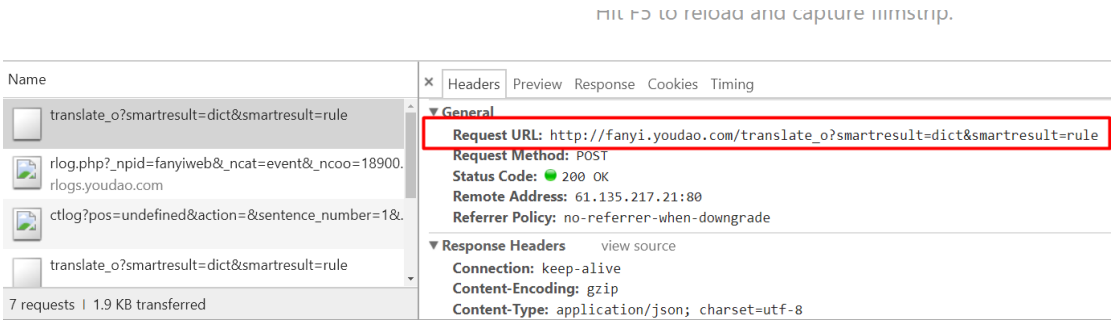
2、鼠标右键检查，也就是审查元素，如下图所示：

3、选择 Network

4、在左侧输入翻译内容 Tom

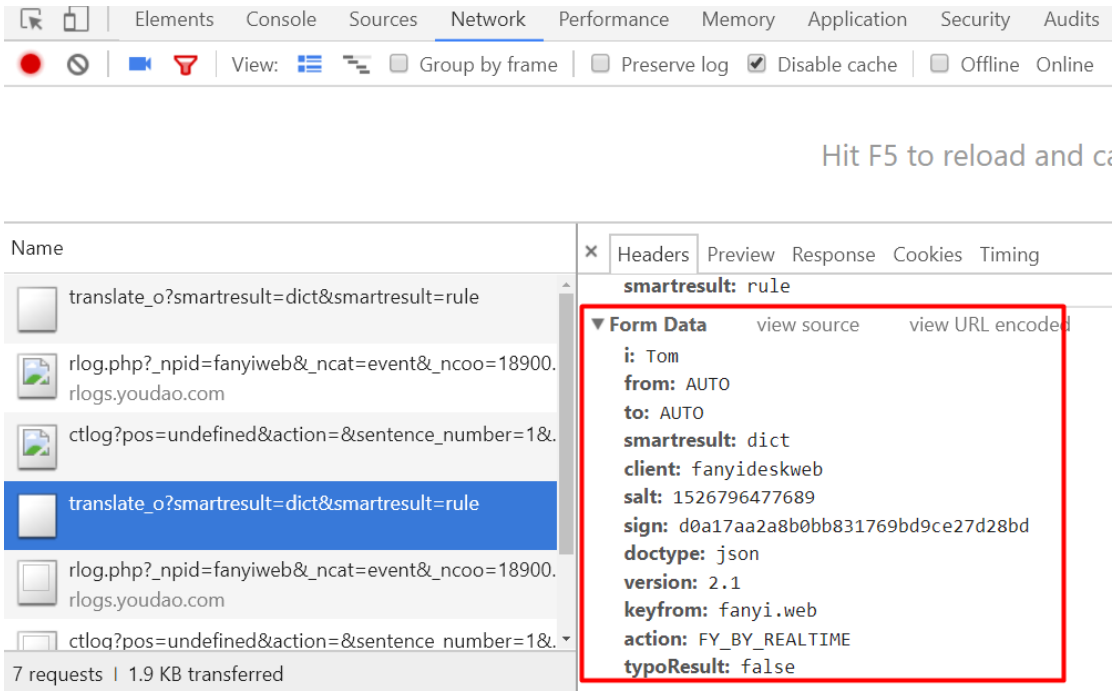
5、记下 Request URL:

```
http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule
```



6、记下 form data：

i: Tom
from: AUTO
to: AUTO
smartresult: dict
client: fanyideskweb
salt: 1526796477689
sign: d0a17aa2a8b0bb831769bd9ce27d28bd
doctype: json
version: 2.1
keyfrom: fanyi.web
action: FY_BY_REALTIME
typoResult: false



新建文件 translate_test.py，编写如下代码：

```
from urllib import request
from urllib import parse
import json

if __name__ == "__main__":
    #对应上图的 Request URL  为避免{"errorCode":50}的错误，去除 url 中的_o
    #Request_URL = 'http://fanyi.youdao.com/translate_o?smartresult=dict&smartresult=rule'
    Request_URL = 'http://fanyi.youdao.com/translate?smartresult=dict&smartresult=rule'
    #创建 Form_Data 字典，存储上图的 Form Data
    Form_Data = {}
    Form_Data['i'] = 'Tom'
    Form_Data['from'] = 'AUTO'
    Form_Data['to'] = 'AUTO'
    Form_Data['smartresult'] = 'dict'
    Form_Data['client'] = 'fanyideskweb'
    Form_Data['salt'] = '1526796477689'
    Form_Data['sign'] = 'd0a17aa2a8b0bb831769bd9ce27d28bd'
    Form_Data['doctype'] = 'json'
    Form_Data['version'] = '2.1'
    Form_Data['keyfrom'] = 'fanyi.web'
    Form_Data['action'] = 'FY_BY_REALTIME'
    Form_Data['typoResult'] = 'false'
    #使用 urlencode 方法转换标准格式
    data = parse.urlencode(Form_Data).encode('utf-8')
    head = {}
```

```
    #写入 User Agent 信息
    head['User-Agent'] = 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36'
    # 创建 Request 对象
    req = request.Request(Request_URL, headers=head)
    #传递 Request 对象和转换完格式的数据
    response = request.urlopen(req,data=data)
    #读取信息并解码
    html = response.read().decode('utf-8')
    #使用 JSON
    translate_results = json.loads(html)
    #找到翻译结果
    translate_results = translate_results['translateResult'][0][0]['tgt']
    #打印翻译信息
    print("翻译的结果是：%s" % translate_results)
```

运行结果：

翻译的结果是：汤姆

Cookie

HTTP是无状态的面向连接的协议，为了保持连接状态，引入了Cookie机制，在浏览器中存储用户信息

应用：Cookie 模拟登陆

```
# 获取一个有登录信息的 Cookie 模拟登陆
from urllib import request
import chardet

# 1. 构建一个已经登录过的用户的 headers 信息
headers = {
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36",
    "Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    # 便于终端阅读，表示不支持压缩文件
    # Accept-Encoding: gzip, deflate, sdch,
```

```
    "Accept-Language":"zh-CN,zh;q=0.9",
    "Cache-Control": "max-age=0",
    "Connection":"keep-alive",
    # 重点：这个 Cookie 是保存了密码无需重复登录的用户的 Cookie，这个 Cookie 里记录了用户名，密码(通常经过 RAS 加密)
    "Cookie": "__cfduid=d813c9add816556c64a8c087554cfb7af1508468882; BDUSS=kM2UnphaUpJRVphdjFrQ3R0TX5KM1NhY25mLVFmdTUtbTJ2ZndiMkw1TWRYRkJiQUFBQUFBJCQAAAAAAAAAAAEAAABUN0cMVG9temt5AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB3PKFsdzyhba; BAIDUID=675B04E847A8A586ECC76203C511EA12:FG=1; PSTM=1529894757; BD_UPN=12314753; BIDUPSID=269D2DF8A150308A1A953FED45517BA8; BDORZ=B490B5EBF6F3CD402E515D22BCDA1598; BD_CK_SAM=1; PSINO=1; H_PS_PSSID=1428_21106_20927; BD_HOME=1",
    "Host":"www.baidu.com",
    "Upgrade-Insecure-Requests":"1",
}

# 2. 通过 headers 里的报头信息（主要是 Cookie 信息），构建 Request 对象
req = request.Request("https://www.baidu.com/", headers = headers)

# 3. 直接访问 renren 主页，服务器会根据 headers 报头信息（主要是 Cookie 信息），判断这是一个已经登录的用户，并返回相应的页面
response = request.urlopen(req)
# 4. 打印响应内容
html = response.read()
charset = chardet.detect(html)['encoding']
print(charset)
print(html.decode(charset))
```

cookielib 库 和 HTTPCookieProcessor 处理器

Python 处理 Cookie，一般是通过 cookielib 模块和 urllib 的 request 模块的 HTTPCookieProcessor 处理器类一起使用。

cookielib 模块：主要作用是提供用于存储 cookie 的对象

HTTPCookieProcessor 处理器：主要处理 cookie 对象，并构建 handler 对象。

cookielib 库

主要作用是提供用于存储 cookie 的对象

该模块主要的对象有 CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

CookieJar：管理 HTTP cookie 值、存储 HTTP 请求生成的 cookie、向传出的 HTTP 请求添加 cookie 的对象。整个 cookie 都存储在内存中。

FileCookieJar (filename,delayload=None,policy=None)：CookieJar 派生而来，将 cookie 存储到文件中。filename 是存储 cookie 的文件名。delayload 为 True 时支持延迟访问文件，即只有在需要时才读取文件或在文件中存储数据。

MozillaCookieJar (filename,delayload=None,policy=None)：从 FileCookieJar 派生而来，MozillaCookieJar 实例与 Mozilla 浏览器 cookies.txt 兼容。

LWPCookieJar (filename,delayload=None,policy=None)：从 FileCookieJar 派生而来，实例与 libwww-perl 标准的 Set-Cookie3 文件格式兼容。

HTTPCookieProcessor 处理器：处理 cookie 对象，并构建 handler 处理器对象。

案例：获取 Cookie，生成 CookieJar()对象中

```
from urllib import request
import http.cookiejar

# 构建一个 CookieJar 对象实例来保存 cookie
cookiejar = http.cookiejar.CookieJar()

# 使用 HTTPCookieProcessor()来创建 cookie 处理器对象，参数为 CookieJar()对象
handler=request.HTTPCookieProcessor(cookiejar)

# 通过 build_opener() 来构建 opener
opener = request.build_opener(handler)

# 4. 以 get 方法访问页面，访问之后会自动保存 cookie 到 cookiejar 中
opener.open("https://www.baidu.com")

## 可以按标准格式将保存的 Cookie 打印出来
cookieStr = ""
for item in cookiejar:
    cookieStr = cookieStr + item.name + "=" + item.value + ";"

## 舍去最后一位的分号
print(cookieStr[:-1])
```

案例：获得网站 cookie，保存在 cookie 文件中

```
import http.cookiejar
from urllib import request

#保存 cookie 的本地磁盘文件名
filename = 'cookie.txt'
#1. 声明一个 MozillaCookieJar(有 save 实现)对象实例来保存 cookie，之后写入文件
cookiejar = http.cookiejar.MozillaCookieJar(filename)
#2. 使用 HTTPCookieProcessor()来创建 cookie 处理器对象，参数为 CookieJar()对象
handler = request.HTTPCookieProcessor(cookiejar)
#3. 通过 build_opener() 来构建 opener
opener = request.build_opener(handler)
#4. 创建一个请求，原理同 urllib2 的 urlopen
response = opener.open("http://www.baidu.com")
#5. 保存 cookie 到本地文件
cookiejar.save()
```

案例：从文件中获取 cookies，做为请求的一部分

```
from urllib import request
import http.cookiejar
import chardet

# 创建 MozillaCookieJar(有 load 实现)实例对象
cookiejar = http.cookiejar.MozillaCookieJar()
# 从文件中读取 cookie 内容到变量
cookiejar.load('cookie.txt')
# 使用 HTTPCookieProcessor()来创建 cookie 处理器对象，参数为 CookieJar()对象
handler = request.HTTPCookieProcessor(cookiejar)
#通过 build_opener() 来构建 opener
opener = request.build_opener(handler)
response = opener.open("https://www.baidu.com")
# 4. 打印响应内容
html = response.read()
charset = chardet.detect(html)['encoding']
print(charset)
print(html.decode(charset))
```

案例：利用 cookielib 和 post 登录人人网

```
from urllib import request
from urllib import parse
import http.cookiejar

# 1. 构建一个 CookieJar 对象实例来保存 cookie
cookie = http.cookiejar.CookieJar()

# 2. 使用 HTTPCookieProcessor()来创建 cookie 处理器对象，参数为 CookieJar()对象
cookie_handler = request.HTTPCookieProcessor(cookie)

# 3. 通过 build_opener() 来构建 opener
opener = request.build_opener(cookie_handler)

# 4. addheaders 接受一个列表，里面每个元素都是一个 headers 信息的元祖, opener 将附带 headers 信息
opener.addheaders = [("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36")]

# 5. 需要登录的账户和密码
data = {"email":"mr_mao_hacker@163.com", "password":"alaxxxxxime"}

# 6. 通过 urlencode()转码
postdata = parse.urlencode(data).encode(encoding='UTF8')
print(postdata)
# 7. 构建 Request 请求对象，包含需要发送的用户名和密码
req = request.Request("http://www.renren.com/PLogin.do", data = postdata)

# 8. 通过 opener 发送这个请求，并获取登录后的 Cookie 值，
opener.open(req)

# 9. opener 包含用户登录后的 Cookie 值，可以直接访问那些登录后才可以访问的页面
response = opener.open("http://www.renren.com/966745694/profile")

# 10. 打印响应内容
print(response.read().decode())
```

获取 AJAX 加载的内容

AJAX 一般返回的是 JSON,直接对 AJAX 地址进行 post 或 get，就返回 JSON 数据了。

```
from urllib import request
from urllib import parse

url = "https://movie.douban.com/j/chart/top_list?type=11&interval_id=100%3A90&action"
headers={"User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36"}
# 变动的是这两个参数，从 start 开始往后显示 limit 个
formdata = {
        'start':'0',
        'limit':'10'
}
data = parse.urlencode(formdata).encode(encoding='UTF8')
req = request.Request(url, data = data, headers = headers)
response = request.urlopen(req)
print(response.read().decode('utf-8'))
```

HTTPS 请求 SSL 证书验证

现在随处可见 https 开头的网站，urllib2 可以为 HTTPS 请求验证 SSL 证书，就像 web 浏览器一样，如果网站的 SSL 证书是经过 CA 认证的，则能够正常访问，如：

https://www.baidu.com/等...

如果 SSL 证书验证不通过，或者操作系统不信任服务器的安全证书，比如浏览器在访问 12306 网站如：https://www.12306.cn/mormhweb/的时候，会警告用户证书不受信任。

（据说 12306 网站证书是自己做的，没有通过 CA 认证）

urllib 在访问:

```
from urllib import request

url = "https://www.12306.cn/mormhweb/"
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}
req = request.Request(url, headers = headers)
response = request.urlopen(req)
print(response.read())
```

会报出 SSLError:

```
urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:720)>

进程已结束,退出代码1
```

如果以后遇到这种网站，我们需要单独处理 SSL 证书，让程序忽略 SSL 证书验证错误，即可正常访问。

```
from urllib import request
# 1. 导入 Python SSL 处理模块
import ssl

# 2. 表示忽略未经核实的 SSL 证书认证
context = ssl._create_unverified_context()

url = "https://www.12306.cn/mormhweb/"
```

```
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}
req = request.Request(url, headers = headers)
response = request.urlopen(req,context=context)
print(response.read().decode('utf8'))
```

urllib.error

urllib.error 可以接收有 urllib.request 产生的异常。

exception urllib.error. **URLError**

The handlers raise this exception (or derived exceptions) when they run into a problem. It is a subclass of OSError.

reason

The reason for this error. It can be a message string or another exception instance.

Changed in version 3.3: URLError has been made a subclass of OSError instead of IOError.

exception urllib.error. **HTTPError**

Though being an exception (a subclass of URLError), an HTTPError can also function as a non-exceptional file-like return value (the same thing that urlopen() returns). This is useful when handling exotic HTTP errors, such as requests for authentication.

URLError

产生的原因主要有：

- 没有网络连接
- 服务器连接失败
- 找不到指定的服务器

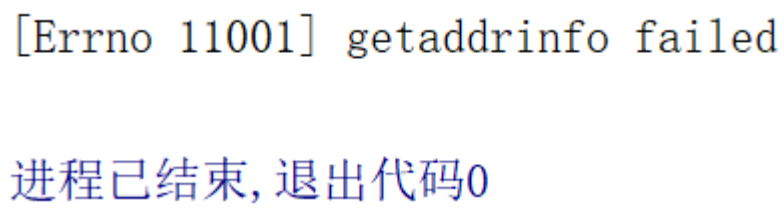
案例：访问了一个不存在的域名，用 try except 语句来捕获相应的异常

```
from urllib import request
from urllib import error

if __name__ == "__main__":
    #一个不存在的连接
    url = "http://www.ajkfhafwjqh.com/"
    req = request.Request(url)
```

```
    try:
        response = request.urlopen(req)
        html = response.read().decode('utf-8')
        print(html)
    except error.URLError as e:
        print(e.reason)
```

运行结果：



HTTPError

HTTPError 是 URLError 的子类，我们发出一个请求时，服务器上都会对应一个 response 应答对象，其中它包含一个数字**"响应状态码"**。

如果 urlopen 不能处理的，会产生一个 HTTPError，对应相应的状态码，HTTP 状态码表示 HTTP 协议所返回的响应的状态。

```
from urllib import request
from urllib import error

if __name__ == "__main__":
    #一个不存在的连接
    url = "http://www.zhihu.com/AAA.html"
    #url = "http://blog.csdn.net/cqcre"
    req = request.Request(url)
    try:
        responese = request.urlopen(req)
        html = responese.read()
        print(html)
    except error.HTTPError as e:
        print(e.code)
```

```
D:\test\vritualenv\env1\Scripts\p
404

进程已结束,退出代码0
```

混合使用

如果想用 HTTPError 和 URLError 一起捕获异常，那么需要将 HTTPError 放在 URLError 的前面，因为 HTTPError 是 URLError 的一个子类。如果 URLError 放在前面，出现 HTTP 异常会先响应 URLError，这样 HTTPError 就捕获不到错误信息了。

```
from urllib import request
from urllib import error

req = request.Request("http://www.douyu.com/tom_kwok.html")
try:
    request.urlopen(req)
except error.HTTPError as err:
    print(err.code)
except error.URLError as err:
    print(err)
else:
    print("Good Job")
```

也可以使用 hasattr 函数判断一场对象含有的属性，如果含有 reason 属性表明是 URLError，如果含有 code 属性表明是 HTTPError。

```
if __name__ == "__main__":
    #一个不存在的连接
    url = "http://www.douyu.com/tom_kwok.html"
    req = request.Request(url)
    try:
        responese = request.urlopen(req)
    except error.URLError as e:
        if hasattr(e, 'code'):
            print("HTTPError")
            print(e.code)
        elif hasattr(e, 'reason'):
```

```
        print("URLError")
        print(e.reason)
```

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，且有五种可能取值：

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
常见状态代码、状态描述、说明：
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

范例：批量爬取贴吧数据

需求：爬取百度贴吧

输入贴吧名称，起始页码，结束页码，爬取贴吧数据，以‘第 x 页.html’命名，保存为 html 文件

url 规律：

第一页：

https://tieba.baidu.com/f?kw=王者荣耀&ie=utf-8&pn=0

第二页：

https://tieba.baidu.com/f?kw=王者荣耀&ie=utf-8&pn=50

第三页：

https://tieba.baidu.com/f?kw=王者荣耀&ie=utf-8&pn=100

每个页面不同之处，就是 url 最后的 pn 的值

范例：模拟登录微博

需求：
先用帐号登录微博，获取一个有登录信息的 Cookie，模拟登陆

问题思考

1、什么是 http 协议?

2、 http 和 https 的区别?

3、什么是爬虫，有哪些分类?

4、数据爬取的步骤有哪些?

5、如何审查页面元素?

6、什么是 cookie?

7、 get 方法和 pos 方法的区别?

8、常见的状态码有哪些?