使用urllib获取www资源

目标：学习使用urllib编写简单爬虫程序，获取webpage资源

1. 掌握单页面普通网页的获取方法

urllib.request.urlopen()方法应用

urllib.request中最常用的方法是urlopen(),它也是我们使用urllib获取普通网页的基本方法。
在应用之前，我们先看一下urllib的源代码，这是从事IT软件类技术工作要养成的职业习惯。
由于urllib是python3内置库，所以无需安装。源代码的路径可以在import urllib或import.request后，使用"**file**"属性查看。
从头部注释中可以了解urlopen方法需要传入一个字符串参数：页面的URL，然后它会打开这个URL，返回类文件对象的响应对象。

> def urlopen(url, data=None, timeout=socket._GLOBAL_DEFAULT_TIMEOUT,*, cafile=None, capath=None, cadefault=False, context=None)

查看上面urlopen方法原型，了解它的功能和调用方法。可以看到，url是必须给定的参数，其他参数可以默认。下面我们尝试使用urlopen打开百度网页。 我们使用了with...as...语句调用，这样会更有利于在不使用时正常关闭连接。返回的结果是HTTPResponse对象。调用这个对象的read()方法，可以访问具体的文件内容。

In [ ]:

```
"""使用urlopen()实现最简单的url访问
"""
import urllib.request

#可以使用语句查看摘要信息：print(urllib.request.__all__)
#可以使用语句查看urllib的本地位置：print(urllib.request.__file__)


url = 'http://www.baidu.com'

with urllib.request.urlopen(url) as response:
    print(response)
    #print(response.read())
    #print(response.read().decode('utf-8'))
    print(response.read()[100:2000].decode('utf-8'))
```

urllib.request.urlretrieve()方法

urllib.request.urlretrieve()方法能够以另一种形式获取页面内容，它会将页面内容存为临时文件并获取response头。
可以查看urlretrieve方法的原型： `def urlretrieve(url, filename=None, reporthook=None, data=None)`

In [ ]:

```
"""使用urlretrieve()将页面内容存为临时文件，并获取response头
"""
import urllib.request

url = 'http://www.baidu.com'
localfile, headers = urllib.request.urlretrieve(url)
print(headers)
print('--'*10)
print(localfile)
```

理解HTTPResponse对象

HTTPResponse对象是一种类文件对象，除了可以文件的read()方法读取它的内容外，还有别的属性和方法。例如：r.code与r.status属性存放本次请求的响应码;r.headers属性存放响应头；r.url属性存放了发出响应的服务器URL；还可以尝试info()和geturl()方法。使用response的geturl()和info方法来验证请求与响应是否如我们希望的一样。有时会出现请求发往的服务器与应答服务器不是同一台主机的情况。

In [ ]:

```
"""理解HTTPResponse对象
"""
import urllib.request

url = 'http://www.baidu.com'
with urllib.request.urlopen(url) as r:
    print(r)
    print(r.code)
    print(r.status)
    print(r.headers)
    print(r.url)
    print(response.info())
    print(response.geturl())
```

2. 掌握使爬虫更像浏览器的方法

默认情况下，urllib发出的请求头大致如下所示：

```
GET / HTTP/1.1
Accept-Encoding: identity Host: 10.10.10.135
User-Agent: Python-urllib/3.6
Connection: close
```

大多数网站的服务器端会进行内容审查，检查客户端类型，一方面是为了满足多样化的需求；另一方面也可以限制一些网络爬虫程序的访问。
上面内容中的User-Agent就是一个内容审查重点，一般的浏览器发出的请求头如下所示：

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,/;q=0.8
Accept-Encoding: gzip, deflate, br Accept-Language: zh-CN,zh;q=0.9 Cache-Control: max-
age=0 Connection: keep-alive Cookie:
BAIDUID=AFA97F911C6ADE2D2B06C704D9581140:FG=1;
BIDUPSID=AFA97F911C6ADE2D2B06C704D9581140; PSTM=1537772835;
BD_UPN=12314753; BDORZ=B490B5EBF6F3CD402E515D22BCDA1598; ispeed_lsm=2;
MCITY=-131%3A; delPer=0; BD_CK_SAM=1; PSINO=2;
H_PS_PSSID=26523_1434_21094_20929;
H_PS_645EC=c9eazWKQBOzC0E25N41dsXkBqUjMnVbupNHxd9OHwW7BqVtt5h0N0eT2dFs;
BD_HOME=0 DNT: 1 Host: www.baidu.com Referer: https://www.baidu.com/s?ie=utf-
8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=python%20urllib%20request%20%E5%86%85%E5%AE%
(https://www.baidu.com/s?ie=utf-
8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=python%20urllib%20request%20%E5%86%85%E5%AE%
Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
```

服务器发现不是正常浏览器可以拒绝提供服务，例如访问www.z.cn 时，使用下面代码会报出 HTTP Error 503: Service Unavailable： ()

```
with urllib.request.urlopen(url) as response:
    print(response.status)
()
```

这时我们可以定制请求对象HTTPRequest，是指更像是浏览器发出的。

In [ ]:

```
"""定制request对象，使爬虫更像浏览器
"""
import urllib.request

url = 'https://www.amazon.cn/gp/goldbox/ref=cngwIter_DEAL_PER_DAY_title?pf_rd_p=1d2f8bbc-3f28-47e6-acb8-ce35d618c4e5&pf_rd_s=desktop-1&pf_rd_t=36701&pf_rd_i=desktop&pf_rd_m=A1AJ19PSB66TGU&pf_rd_r=8XXT48R1QRS14P1OW9M5&pf_rd_r=8XXT48R1QRS14P1OW9M5&pf_rd_p=1d2f8bbc-3f28-47e6-acb8-ce35d618c4e5'
'''使用下面代码会报出 HTTP Error 503: Service Unavailable
with urllib.request.urlopen(url) as response:
    print(response.status)

with urllib.request.urlopen(url) as response:
    print(response.status)
'''
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36'}
request = urllib.request.Request(url,headers=headers)
with urllib.request.urlopen(request) as response:
    print(response.status)
```

3. 掌握向服务器传递参数的方法

许多HTTP方法都可以用来向服务器提供数据，最常见的GET和POST方法都可以，但方式不同

使用GET方法向服务器提供数据

In [ ]:

```
"""使用GET方法，向百度服务器发送查询请求
"""
import urllib.request
import urllib.parse

querystr = {'wd':'北航'}
querystr_encode = urllib.parse.urlencode(querystr)
print(querystr_encode)
#https://www.baidu.com/s?wd=%E5%8C%97%E8%88%AA
url = 'http://www.baidu.com/s?' + querystr_encode
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
request = urllib.request.Request(url,headers=headers)
with urllib.request.urlopen(request) as response:
    print(response.status)
    print(response.headers)
    print(response.read().decode('utf-8'))
```

使用POST方法向服务器提供数据

In [ ]:

```
"""利用POST方法，向http://httpbin.org 提交
事前应在该网站进行设置，启动试用链接。
"""
import urllib.request
import urllib.parse

url = "http://httpbin.org/post"
payload = {'key1': 'value1', 'key2': 'value2'}
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
req = urllib.request.Request(url,data=urllib.parse.urlencode(payload).encode('utf-8'),headers=headers)
with urllib.request.urlopen(req) as r:
    print(r.read().decode('utf-8'))
```

In [ ]:

```
"""利用POST方法，向http://10.10.10.135/WebGoat/ 提交用户名和密码
"""
import urllib.request
import urllib.parse

url = 'http://10.10.10.135/dvwa/login.php'
cookie = 'PHPSESSID=898c1rsum58475qh3nros002n7; path=/'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
         'Cookie':cookie,
          }
authstr = {'username':'admin',
           'password':'admin',
           'Login':'Login',}
data = urllib.parse.urlencode(authstr).encode('utf-8')

request = urllib.request.Request(url,data=data,headers=headers)
with urllib.request.urlopen(request) as response:
    print(response.status)
    print(response.headers)
    cookie1 = response.headers['Set-Cookie']

url = 'http://10.10.10.135/dvwa/index.php'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          'Cookie':cookie+';'+cookie1,
          }
print(headers)
request = urllib.request.Request(url,headers=headers)
with urllib.request.urlopen(request) as response:
    print(response.read())
```

4. 掌握设置超时访问限制和处理异常的方法

urllib.error处理异常,两个常用异常类：urllib.error.URLError和HTTPError

In [ ]:

```
""" 设置time-out"""
import socket
import urllib.request
# timeout in seconds
timeout = 3
socket.setdefaulttimeout(timeout)
# this call to urllib.request.urlopen now uses the default timeout
# we have set in the socket module
req = urllib.request.Request('http://www.python.org/')
a = urllib.request.urlopen(req).read()
print(a)
```

In [ ]:

```
"""使用urllib.error处理异常
URLError继承自OSError，是urllib的异常的基础类
HTTPError是验证HTTP response实例的一个异常类。

HTTP protocol errors是有效的response，有状态码、headers、body。

一个成熟的程序需要管理所有输出，不仅有希望见到的输出，还要有意料之外的异常。
logging的使用可以参考https://docs.python.org/3.5/howto/logging.html
"""

import urllib.request
import urllib.error
import urllib.parse
import logging

logging.basicConfig(format='%(asctime)s:%(levelname)s:%(message)s',
                    datefmt='%Y-%m-%d %H:%M:%S',
                    filename='C:\\Users\\leo\Documents\\crawlerslesson1_crawler.log',
                    level=logging.DEBUG)
try:
    #url = 'http://www.baidu11.com'
    url = 'http://10.10.10.135/WebGoat/attack'
    headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
    request = urllib.request.Request(url,headers=headers)
    with urllib.request.urlopen(request) as response:
        print(response.status)
        print(response.read().decode('utf-8'))

except urllib.error.HTTPError as e:
    import http.server
    #print(http.server.BaseHTTPRequestHandler.responses[e.code])
    logging.error('HTTPError code: %s and Messages: %s'% (str(e.code),http.server.BaseHTTPRequestHandler.responses[e.code]))
    logging.info('HTTPError headers: ' + str(e.headers))
    logging.info(e.read().decode('utf-8'))
    print('不好意思，服务器卡壳儿了，请稍后重试。')
except urllib.error.URLError as e:
    logging.error(e.reason)
    print('不好意思，服务器卡壳儿了，请稍后重试。')
```

5. 实例：从百度贴吧下载多页话题内容

先了解以下百度贴吧http://tieba.baidu.com/f? (http://tieba.baidu.com/f?) 我们定义几个函数：

- loadPage(url) 用于获取网页
- writePage(html,filename) 用于将已获得的网页存储为本地文件
- tiebaCrawler(url,beginpPage,endPage,keyword)用于调度，提供需要抓取的页面URLs
- main：程序主控模块，完成基本命令行交互接口

In [ ]:

```
"""A case of crawler is used to fetch the content of baidu's tieba url, in according to user's i
nput keywords.

"""
import urllib.request
import urllib.parse

def loadPage(url):
    """
        Function: Fetching url and accessing the webpage content.
        url: the wanted webpage url.
    """
    headers = {'Accept': 'text/html','User-Agent':'Mozilla/5.0',}
    print('To send http request to %s' % url)
    request = urllib.request.Request(url,headers=headers)

    return  urllib.request.urlopen(request).read().decode('utf-8')

def writePage(html,filename):
    """
        Fuction: To write the content of html into a local file.
        html: The response content.
        filename: the local filename to be used stored the response.
    """
    print('To write html into a local file %s ...' % filename)
    with open(filename,'w') as f:
        f.write(str(html))
    print('Work done.')
    print('-'*10)

def tiebaCrawler(url,beginpPage,endPage,keyword):
    """
        Function: The scheduler of tieba crawler, is used to access every wanted url in turns.
        url: the url of baidu's tieba webpage
        beginPage: initial page
        endPage: end page
        keyword: the wanted keyword
    """
    filename = keyword + '_tieba.html'
    for page in range(beginpPage,endPage+1):
        pn = (page - 1) * 50
        queryurl = url + '&pn=' + str(pn)
        writePage(loadPage(queryurl),filename)

if __name__ == '__main__':
    kw = input('Pl input the wanted tieba\'s name:' )
    beginPage = int(input('The beginning page number:'))
    endPage = int(input('The ending page number:'))
    # 百度贴吧查询url例子：http://tieba.baidu.com/f?ie=utf-8&kw=%E5%8C%97%E8%88%AA&fr=search&red
_tag=i2305631770
    url = 'http://tieba.baidu.com/f?'
    key = urllib.parse.urlencode({'kw':kw})
    queryurl = url+ key
    tiebaCrawler(queryurl,beginPage,endPage,kw)
```

6. 掌握自定义opener的方法

urllib.request.urlopen()调用了HTTPHandler来处理无错误的HTTP请求，它的功能是有限的。对于urllib.request.urlopen()不支持的功能，我们可以通过自己定义opener，调用其他Handler实现。urllib中的opener都是urllib.request.OpenerDirector类的实例。OpenerDirector类管理着Handler对象集合，这些Handlers完成实际的http请求工作，每个Handler实现了一种特定协议或选项内容。OpenerDirector作为一个组合型对象，调用各种Handlers打开请求的URL。 例如：HTTPHandler执行HTTP GET和POST请求，处理无错误返回；HTTPRedirectHandler自动处理HTTP 301、302、303和307重定型错误；HTTPDigestAuthHandler处理digest认证。

urllib.request中含有多个使用的Handler类，例如：

- HTTPHandler
- HTTPSHandler
- FileHandler
- DataHandler
- CacheFTPHandler
- HTTPRedirectHandler
- HTTPCookieProcessor
- ProxyHandler
- HTTPBasicAuthHandler
- HTTPDigestAuthHandler
- ProxyBasicAuthHandler
- ProxyDigestAuthHandler
- UnknownHandler

自定义的Opener对象都由OpenerDirector加载不同的Handler来生成。

自定义opener需要先初始化一个OpenerDirector，使用build_opener方法实现， 这是一个使用调用单一函数调用多个handlers生成opener实例的方法。
install_opener可以用于生成一个opener对象，形成全局默认的opener，这使你再次使用urlopen时，不再使用系统原opener，而使用你定义的opener。
不准备替换全局默认的opener时，可以使用opener实例中的open方法，访问url。

In [ ]:

```
"""自定义opener"""
import urllib.request

# demo 1
httphandler = urllib.request.HTTPHandler()
opener = urllib.request.build_opener(httphandler)
request = urllib.request.Request('http://www.baidu.com/')
response = opener.open(request)
print(response.read().decode('utf-8'))
```

In [ ]:

```
"""自定义opener"""
# demo 2
# 在开发中，如果需要了解HTTPHandler的调试信息，可以使用下列语句,无需输出语句。
httphandler = urllib.request.HTTPHandler(debuglevel=1)
opener = urllib.request.build_opener(httphandler)
request = urllib.request.Request('http://www.baidu.com/')
response = opener.open(request)
```

In [ ]:

```
"""自定义opener"""
# demo3  使用install_opener方法使自定义的opener成为全局默认opener

request = urllib.request.Request('http://www.baidu.com/')
httphandler = urllib.request.HTTPHandler()
opener = urllib.request.build_opener(httphandler)
urllib.request.install_opener(opener)

a = urllib.request.urlopen(request).read().decode("utf8")
print(a)
```

7. 掌握HTTP basic authentication方法

登陆网页前遇到的要求输入用户名和密码的程序，通常称为身份认证程序。HTTP认证可以保护一个作用域（称为一个realm）内的资源不受非法访问。当一个请求要求取得受保护的资源时，网页服务器回应一个 401 Unauthorized error 错误码。这个回应包含一个指定了验证方法和领域的 WWW-Authenticate 头信息。把这个领域想像成一个存储着用户名和密码的数据库，它将被用来标识受保护资源的有效的用户。比如网站使用http basic auth时，尝试访问该网站上标识为“Private Files”的资源，服务器响应可能是：WWW-Authenticate:Basic realm="Private Files" 。

HTTP规范中定义了两种认证模式：Basic Auth和digest auth。若server采用Basic Auth保护资源，那么你访问这些被保护资源时就会首先看到一个用户认证表单，要求输入用户名和密码。用户输入后用户名和密码都会以Base64编码形式发送给服务器。认证的基本过程是：

1. 客户请求访问网页；
2. 服务器端返回401错误，要求认证（401消息的头里面带了挑战信息，例如：认证头： WWW-Authenticate: Basic realm="zhouhh@mydomain.com" ）；
3. 客户端重新提交请求并附以认证信息，这部分信息将被编码；
4. 服务器检查信息，通过则给以正常服务页面；否则返回401错误。

第一次服务器返回401错误时，会返回headers字典信息，其中会包含信息：WWW-Authenticate: Basic realm="cPanel"。我们假定已知用户名和密码，之后利用一定的编码格式将realm名、用户名、密码等信息；编码后就可以传递给服务器，认证就可通过。

In [ ]:

```
"""HTTP basic auth case
"""
import urllib.request

url = 'http://'
```

In [2]:

```
"""实例1 httpbin.org提供了Auth demo
使用前先登录该网站，进行设置后可获得相应示例的request url，复制该url到本程序作为初始访问的url。
"""
import urllib.request
import urllib.error
import urllib.parse

url = 'http://httpbin.org/basic-auth/user/123456'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }

username = 'user'
password = '123456'

request = urllib.request.Request(url,headers=headers)

passmgr = urllib.request.HTTPPasswordMgrWithDefaultRealm()
# this creates a password manager
passmgr.add_password(None, url, username, password)
# because we have put None at the start it will always
# use this username/password combination for  urls
# for which `theurl` is a super-url

authhandler = urllib.request.HTTPBasicAuthHandler(passmgr)
# create the AuthHandler

opener = urllib.request.build_opener(authhandler)

try:
    with opener.open(request) as response:
        print(response.status)
        print(response.read().decode('utf-8'))
except urllib.error.URLError as e:
    print(e)
except urlllib.error.HTTPError as e:
    print(e)
    print(e.headers)
except:
    print('unkown error.')
```

```
200
{
  "authenticated": true,
  "user": "user"
}
```

In [ ]:

```
"""实例2 不使用HTTPPasswordMgrWithDefaultRealm类的验证An case of passing basic authentication wi
th urllib

"""

import urllib.request
import urllib.error
import urllib.parse
import logging

url = 'http://10.10.10.135/WebGoat/attack'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apn
g,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/69.0.3497.100 Safari/537.36',
          }
request = urllib.request.Request(url,headers=headers)

def passBasicAuth(realm):

    import base64
    username = 'webgoat'
    password = 'webgoat'
    bstr = username+':'+password
    schemastr , realmname = realm.split('=')
    if schemastr.lower().find('basic') >= 0:
        schema ='Basic'
    else:
        print('The authentication schema isn\'t basic, programe exit.')
        exit(-1)

    base64str = base64.b64encode(bstr.encode('utf-8'))
    authheader = 'Basic %s' % base64str.decode('utf-8')

    request.add_header('Authorization',authheader)
    print(request.headers)
    with urllib.request.urlopen(request) as response:
        print(response.status)
        print(response.read().decode('utf-8'))



try:
    with urllib.request.urlopen(request) as response:
        print(response.status)
        print(response.info())
        #print(response.read().decode('utf-8'))
except urllib.error.URLError as e:
    if  hasattr(e,'code'):

        print(e.code)
        print(e.info())
        if e.code == 401:
            passBasicAuth(e.headers['WWW-Authenticate'])
    elif hasattr(e,'reason'):
        print(e.reason)
    else:
        print('unkown error.')
```

8.掌握自动提交身份认证的方法

In [ ]:

```
"""实例2的改进 HTTPPasswordMgrWithDefaultRealm 的验证An case of passing basic authentication wit
h urllib

Worked!

"""
import urllib.request
import urllib.parse
import urllib.error

url = 'http://10.10.10.135/WebGoat/attack'
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apn
g,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/69.0.3497.100 Safari/537.36',
          }
request = urllib.request.Request(url,headers=headers)
username = 'webgoat'
password = 'webgoat'

passman = urllib.request.HTTPPasswordMgrWithDefaultRealm()
# this creates a password manager
passman.add_password(None, url, username, password)
# because we have put None at the start it will always
# use this username/password combination for  urls
# for which `theurl` is a super-url

authhandler = urllib.request.HTTPBasicAuthHandler(passman)
# create the AuthHandler

opener = urllib.request.build_opener(authhandler)

urllib.request.install_opener(opener)
# All calls to urllib2.urlopen will now use our handler
# Make sure not to include the protocol in with the URL, or
# HTTPPasswordMgrWithDefaultRealm will be very confused.
# You must (of course) use it when fetching the page though.

with urllib.request.urlopen(request) as response:
    print(response.status)
    print(response.read().decode('utf-8'))
# authentication is now handled automatically for us
```

HTTP Digest Auth

Basic Auth是一种很不安全的认证方式，如果有人截留了合法用户成功认证的过程，那么就可以解码Base64编码的username和password，实现重放攻击。这种认证方式较basic auth安全一些，因为它不会将密码明文单独发送给服务器。Digest Auth使用MD5散列算法处理密钥，防止了密钥被窃取后解密（当然这安全也并不稳妥）。具体情况可以参考rfc2617文档。
使用Digest Auth认证的基本过程：

1. 客户发送请求后；
2. 收到一个401（未授权）消息，包含一个Challenge和一个唯一的字符串：nonce，其值每次请求都不一样；
3. 客户将用户名密码和401消息返回的挑战一起MD5加密后传给服务器(这样即使有窃听，也无法进行重放攻击)；
4. 服务器检查是否合法。

通常在第2步会收到下列响应头信息（部分）：

> Www-Authenticate: Digest realm="me@kennethreitz.com",
> nonce="8978f018e5e52314ae7be58725230601", qop="auth",
> opaque="6c48b639d829d34653e7a61707451786", algorithm=MD5, stale=FALSE

参数解释：

- WWW-Authenticate：http中用于提供认证信息一个标头；
- realm：表示保护域，其值可以是一个简单的字符串，而rfc2617上要求是一个email类型的字符串；
- qop：是认证的(校验)方式，这个信息较为重要，对后面md5的加密过程有影响；
- nonce：表示一次性会话密钥，其值是一个字符串，每次登录服务器都会产生一个新的随机字符串作为nonce值；如果不严格，可以随机生成一个GUID（即唯一、不重复的）；如果严格，则需要包含时间信息、客户端IP信息和其它信息，因为认证过程的时间很短，所以如果服务器收到认证信息后发现这个时间和服务器的时间相去甚远，那说明不正常，直接拒绝，以防止攻击。附有客户端IP信息有利服务器了解哪些IP持续试探，可以将其置入blacklist。这些严格的做法主要是为了防止攻击。在rfc2617上有较为详细的描述。
- opaque：由服务器指定的一个字符串，通常是base64或hexadecimal编码数据，这个字串由服务器发给客户端后，客户端会原样传回，它只是透传而已，即客户端还会原样返回过来。实际上，上面的那些域，客户端都还是会原样返回的，但返回时除了以上的那些域之外，还会增加新的内容进来。
- algorithm：表示加密算法，通常为MD5散列算法，并不是严格的加密。
- stale：一个标志位，用来指示前一个请求是否被拒绝了。

In [5]:

```
"""尝试通过http digest auth
首先在httpbin.org网站上设置auth-digest-auth，并获得测试用的url"""
import urllib.request
import urllib.error

def auth():
    url = 'http://httpbin.org/digest-auth/auth/leoleo1/123456789'
    username = 'leo1'
    password = '123456'
    realm = "me@kennethreitz.com"

    headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
    request = urllib.request.Request(url,headers=headers)
    try:

        passmgr = urllib.request.HTTPPasswordMgr()
        passmgr.add_password(realm=realm,uri=url,user=username,passwd=password)
        authHandler = urllib.request.HTTPDigestAuthHandler(passmgr)

        opener = urllib.request.build_opener(authHandler)

        with opener.open(request) as resp:
            print(resp.status)
            print(resp.read().decode('utf-8'))
    except urllib.error.HTTPError as e:
        print(e)
        print(e.headers)
        '''
        authdict = {}
        authdict["realm"] = e.headers['Www-Authenticate'].split(',')[0].split('=')[-1]
        authdict["nonce"] = e.headers['Www-Authenticate'].split(',')[1].split('=')[-1]
        authdict["qop"] =  e.headers['Www-Authenticate'].split(',')[2].split('=')[-1]
        authdict["opaque"] =  e.headers['Www-Authenticate'].split(',')[3].split('=')[-1]
        authfoo(authdict)
        '''
    except urllib.error.URLError as e:
        print(e)

if __name__ == '__main__':
    auth()
```

```
HTTP Error 401: UNAUTHORIZED
Connection: close
Server: gunicorn/19.9.0
Date: Mon, 15 Oct 2018 12:35:19 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Www-Authenticate: Digest realm="me@kennethreitz.com", nonce="5d14fdd4a654a4b2a537c
4c20fc5ae71", qop="auth", opaque="a04d6ca62bea4d640435b3b2ceb29a83", algorithm=MD
5, stale=FALSE
Set-Cookie: stale_after=never; Path=/
Set-Cookie: fake=fake_value; Path=/
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Via: 1.1 vegur
```

8. 使用代理

使用代理是对抗反爬虫机制的常用做法。很多网站会检测某一段时间某个外来IP地址对服务器的访问次数等信息。
如果访问次数或方式不符合安全策略，就会禁止该外来IP对服务器的访问。所以，爬虫设计者可以用一些代理服务器，使自己真实IP地址被隐藏，免于被禁止。
urllib中使用ProxyHandler来设置代理服务器的使用。

网络上通常有两类代理：免费代理、收费代理。免费代理可以通过百度/google搜索，或从以下网站查找：西刺免费代理IP、快代理免费代理、Proxy360代理、全网代理IP...
免费开放代理一般会有很多人都在使用，而且代理有寿命短，速度慢，匿名度不高，HTTP/HTTPS支持不稳定等缺点（免费没好货）。
专业爬虫工程师或爬虫公司会使用高品质的私密代理，这些代理通常需要找专门的代理供应商购买，再通过用户名/密码授权使用（舍不得孩子套不到狼）。
可以组织一个代理列表，在一定时间策略下，随机使用，免于被server禁止访问。

In [ ]:

```
"""使用代理"""

#demo1 使用ProxyHandler通过指定的一个免费代理访问目标网站
import urllib.request

request = urllib.request.Request('http://www.baidu.com/')
proxy_support = urllib.request.ProxyHandler({'http': '210.1.58.212:8080'})

opener = urllib.request.build_opener(proxy_support)
response = opener.open(request)
print(response.read().decode('utf-8'))
```

In [ ]:

```
"""使用代理"""
#demo2 使用有身份认证的代理
import urllib.request
username = 'leo'
password = 'leo'
proxydict = {'http':'106.185.26.199:25'}
proxydict['http'] = username+':'+password+'@'+proxydict['http']
httpWithProxyHandler = urllib.request.ProxyHandler(proxydict)
opener = urllib.request.build_opener(httpWithProxyHandler)
request = urllib.request.Request('http://www.google.com/')
response = opener.open(request)
print(response.read().decode('utf-8'))
```

In [ ]:

```
#demo3 使用urllib推荐做法改进上述过程
username = 'leo'
password = 'leo'
proxyserver = {'106.185.26.199:25'}

#    1.构建一个密码管理对象，用来保存需要处理的用户名和密码；
passwordMgr = urllib.request.HTTPPasswordMgrWithDefaultRealm()
#    2.添加用户信息，第一个参数realm是与远程服务器相关的域的信息，默认为None，可通过response头查看
#      后面3个参数分别为代理服务器、用户名、密码；
passwordMgr.add_password(None,proxyserver,username,password)
#    3.构建一个代理基础用户名/密码验证的Handler对象，参数为密码管理对象；
proxyauth_handler = urllib.request.ProxyBasicAuthHandler(passwordMgr)
#    4.通过build_opener()定义opener对象
opener = urllib.request.build_opener(proxyauth_handler)
#    5.构造请求request
request = urllib.request.Request('http://www.google.com')
#    6.使用自定义opener发送请求
response = opener.open(request)
#    7.打印响应内容
print(response.read().decode('utf-8'))
```

In [ ]:

```
#demo5 使用从http://www.goubanjia.com/, wwwkuaidaili.com/dps 获取的代理列表
#可以使用快代理在线测试代理可行性
import random

proxylist = [{"https":"180.210.205.199:8888"},
                {"http":"185.22.174.65:10010"},
                {"http":"54.153.171.50:3128"},
                {"http":"54.79.47.128:8080"},
                {"https":"195.235.204.60:3128"},
                {"http":"203.174.90.201:8080"},
                {"http":"67.63.33.7:80"},
                {"http":"150.138.220.247:80"},
                {"http":"94.16.117.29:3128"},
                {"http":"210.1.58.212:8080"},
                {"http":"125.39.9.35:9000"},
                {"http":"94.242.55.108:10010"},
                {"http":"74.82.50.155:3128"},
                {"https":"119.28.195.93:8888"},
                {"http":"151.106.12.251:1080"},
                {"https":"51.254.50.239:3128"},
                {"http":"140.227.60.114:3128"},
                {"http":"165.227.45.213:8080"},
                {"http":"212.77.138.161:41258"},
                {"http":"37.61.224.107:8195"},
            ]

def randomTryProxy(retry):
    """# 策略1 随机选
    """
    try:

        proxy = random.choice(proxylist)


        print('Try %s : %s' % (retry,proxy))
        #    使用代理构建处理对象
        httpProxyHandler = urllib.request.ProxyHandler(proxy)
        opener = urllib.request.build_opener(httpProxyHandler)
        request = urllib.request.Request('http://www.google.com')
        response = opener.open(request,timeout=6)
        print(response.read().decode('utf-8'))
    except:
        print('Connect error.Please retry')
        if retry > 0:
            randomTryProxy(retry-1)

def inorderTryProxy(proxy):
    """# 策略2 依次选择尝试
    """
    try:
        print('Try %s :' % proxy)
        #    使用代理构建处理对象
        httpProxyHandler = urllib.request.ProxyHandler(proxy)
        opener = urllib.request.build_opener(httpProxyHandler)
        request = urllib.request.Request('http://www.google.com')
        response = opener.open(request,timeout=6)
        print(response.read().decode('utf-8'))
    except:
        print('Connect error.Please retry')
```

```
if __name__ == '__main__':
    #randomTryProxy(5)
    for p in proxylist:
        inorderTryProxy(p)
```

10.掌握获取ajax异步加载网页内容的方法

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。 AJAX 最大的优点是在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。 AJAX 不需要任何浏览器插件，但需要用户允许JavaScript在浏览器上执行。

这里以 https://movie.douban.com/tag/#/ (https://movie.douban.com/tag/#/) 为例 先使用抓包工具查看一下这个页面，通过测试可以发现每次点击“更多”会增加一个响应 https://movie.douban.com/j/new_search_subjects?sort=U&range=0,10&tags=&start=40 (https://movie.douban.com/j/new_search_subjects?sort=U&range=0,10&tags=&start=40) 将其直接在浏览器中打开，可以看到它以json格式记录了新加载的电影信息。 找到这个文件后，就可开始尝试了。

In [ ]:

```
"""抓取ajax页面"""
#demo1
import urllib.request
import urllib.parse
import urllib.error

# https://movie.douban.com/j/chart/top_list?type=24&interval_id=100%3A90&action=&start=60&limit=20
url = 'https://movie.douban.com/j/new_search_subjects?'
movietype = '动作'
params = {'sort':'U','range':'0,10','tags':'','start':'1','tags':''}
params['tags'] = movietype
params_encode = urllib.parse.urlencode(params).encode('utf-8')
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
          }
try:
    request = urllib.request.Request(url,data=params_encode,headers=headers)
    with urllib.request.urlopen(request) as response:
        print(response.read().decode('utf-8'))
except urllib.error.HTTPError as e:
    print(e)
except urllib.error.URLError as e:
    print(e)
```

In [ ]:

```
"""抓取ajax页面"""
# demo2
import urllib.request
import urllib.parse
import json

url = 'https://www.toutiao.com/api/pc/feed/?category=news_hot&utm_source=toutiao&widen=1&max_beh
ot_time=1539571709&max_behot_time_tmp=1539571709&tadrequire=true&as=A1B5FBFC9483892&cp=5BC473A8E
9B2DE1&_signature=wz2C-AAAmQJpEsJ4owX5fMM9gu'
params = {'category':'news_hot','utm_source':'toutiao','tadrequire':'true'}
params_encode = urllib.parse.urlencode(params).encode('utf-8')
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apn
g,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/69.0.3497.100 Safari/537.36',
          }
try:
    request = urllib.request.Request(url,headers=headers)
    with urllib.request.urlopen(request) as response:

        #原来的方法得到编码后数据，因为json将其当作字符串用引号包围起来了。
        #print(response.read().decode('utf-8'))
        #可以查看响应头，有Content-Type: application/json
        #print(response.headers)
        print(json.loads(response.read()))
except urllib.error.HTTPError as e:
    print(e)
except urllib.error.URLError as e:
    print(e)
```

11.掌握使用cookie获取需认证网页的方法

Cookie，指某些网站为了辨别用户身份、进行session跟踪而储存在用户本地终端上的数据（通常经过加密)。比如说有些网站需要登录后才能访问某个页面，在登录之前，你想抓取某个页面内容，登陆前与登陆后是不同的，或者不允许的。 使用Cookie和使用代理IP一样，也需要创建一个自己的opener。在HTTP包中，提供了cookiejar模块，用于提供对Cookie的支持。

方法1 urllib通过已登录的cookie值，以登录用户身份访问网页。

首先用浏览器登录，获取登陆后的cookie，通常这个cookie会非常长。 我们以访问 http://www.renren.com/968196747/profile (http://www.renren.com/968196747/profile) 这个登录后链接为例 如果成功会得到相关内容. 将cookkie值作为字符串加载在headers里。

In [ ]:

```
"""urllib通过已登录的cookie值，以登录用户身份访问网页"""
import urllib.request
import urllib.parse

##demo1 通过已登录的cookie值，以登录用户身份访问网页
url = 'http://www.renren.com/968196747/profile'

headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',

          }
'''
headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',
           'Cookie':'anonymid=jmn5bx3v-51e0e; depovince=GW; _r01_=1; ick_login=c9a5ab7b-7432-4e79-8d8d-d9e3f89ab72c; ick=a0121b31-6fe8-4eef-bf25-4278b6f19743; XNESSESSIONID=d51be5b82a20; WebOnLineNotice_968196747=1; JSESSIONID=abcISInMXWcrBd2AlxKyw; jebe_key=ed233682-75b8-496c-a199-557f631f200e%7Cc377fecba1c1e1def24233f88b207b06%7C1538208356017%7C1%7C1538208354004; wp_fold=0; jebecookies=dea9e704-78ec-4756-90d7-1a93dae3be67|||||; _de=D6104CF9DBA07C121FF9E00605E6865D; p=a9d3694434120963a494d947ab2906477; first_login_flag=1; ln_uact=13141055789; ln_hurl=http://head.xiaonei.com/photos/0/0/men_main.gif; t=3395ed44534775ea30e64655670389627; societyguester=3395ed44534775ea30e64655670389627; id=968196747; xnsid=dba2a4e4; loginfrom=syshome',
          }
'''
request = urllib.request.Request(url)#,headers=headers)
with urllib.request.urlopen(request) as response:
    print(response.read().decode('utf-8'))
```

方法2 使用cookie处理库自动抓取登录cookie

上面的方法略显笨拙，可以使用更加机智的方法。 我们可以使用urllib.request.cookielib库和urllib.request.HTTPCookieProcessor处理器，自动收集cookie，不用再手工抓包。 http.cookiejar可以用于web客户端的http cookie处理，管理cookie值，存储http请求产生额cookie，向传出的请求添加cookie对象

In [ ]:

```
"""使用更加机智的方法"""
##    使用urllib.request.cookielib库和urllib.request.HTTPCookieProcessor处理器，自动收集cookie，不再手工抓包；
#    以中国博士网为例
import urllib.request
import http.cookiejar
import urllib.parse
cookie = http.cookiejar.CookieJar()

cookieHandler = urllib.request.HTTPCookieProcessor(cookie)
opener = urllib.request.build_opener(cookieHandler)

url = 'http://www.chinaphd.com/cgi-bin/loginout.cgi?forum='
formdata = {'action':'login',
            'forum':'',
            'inmembername': 'hhhparty',
            'inpassword':'tdzj1234',
            'hidden':'0',
            'CookieDate':'0',
            'onlineview':'1',
            'viewMode':'',
            'selectstyle': '',
            'tanchumsg':'',
            'freshtime':'',
           }
data = urllib.parse.urlencode(formdata).encode('utf-8')

opener.addheaders = [('User-Agent','Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36')]

request = urllib.request.Request(url,data=data)#,headers=headers)
response =  opener.open(request)
print(response.read().decode('gbk'))

#登录后，还可以继续访问其他页面
response_message = opener.open('http://www.chinaphd.com/cgi-bin/messanger.cgi?action=inbox')
print(response_message.read().decode('gbk'))
```

12.访问https站点

需要CA证书才能访问,证书是用于加密连接和身份认证的数字凭据，通常由公信机构发放。 尝试访问 http://www.baidu.com (http://www.baidu.com) 与https://www.baidu.com， (https://www.baidu.com，) 观察它们的不同 百度访问https时会有跳转。 再尝试访问12306网站 'http://www.12306.cn/mormhweb/ (http://www.12306.cn/mormhweb/)' 与 https://www.12306.cn/mormhweb/ (https://www.12306.cn/mormhweb/) 的不同 可以看到访问 https://www.12306.cn/mormhweb/ (https://www.12306.cn/mormhweb/) 时会报出错误: CertificateError: hostname 'www.12306.cn' doesn't match either of 'webssl.chinanetcenter.com' ssl库ssl.

In [ ]:

```
"""访问https"""

import urllib.request
import ssl
#尝试访问http://www.baidu.com 与https://www.baidu.com，观察它们的不同
#再尝试访问12306网站 'http://www.12306.cn/mormhweb/' 与 https://www.12306.cn/mormhweb/的不同

url = 'https://www.12306.cn/mormhweb/'

#导入证书需要使用ssl库,可以查看urllib源码如何使用这个函数
context = ssl._create_unverified_context()


#request = urllib.request.Request(url,unverifiable=True)
request = urllib.request.Request(url)
#分别尝试下列语句块，查看不同的结果。
with urllib.request.urlopen(request,context=context) as response:
    print(response.read().decode('utf-8'))
'''
with urllib.request.urlopen(request) as response:
    print(response.read().decode('utf-8'))
'''
'''
with urllib.request.urlopen(url,cafile=cafile) as response:
    print(response.read().decode('utf-8'))
'''
```