网络爬虫与数据采集课程

第3章 Web页面爬取

课前引导

本节课我们将介绍使用python开发网络爬虫的基本方法，尝试获取web页面信息。

上节回顾

上节课介绍了：

- 网络爬取的入口
- 网页下载的原理
- 内容解析的对象
- 存储数据的仓库

本节课程主要内容

内容列表：

- 本节目标
- 爬取Web页面的基本过程；
- 使用Urllib实现基本Web页面爬取；
- 使用Requests优化页面爬取过程；
- 本节总结
- 课后练习

1 本节目标

本节课的主要目标：

- 使学生理解web页面爬取的基本过程
- 使学生能够运用python urllib库设计自己的简单web页面爬取程序；
- 使学生能够应用requests库，设计可用性良好的web页面爬取程序。

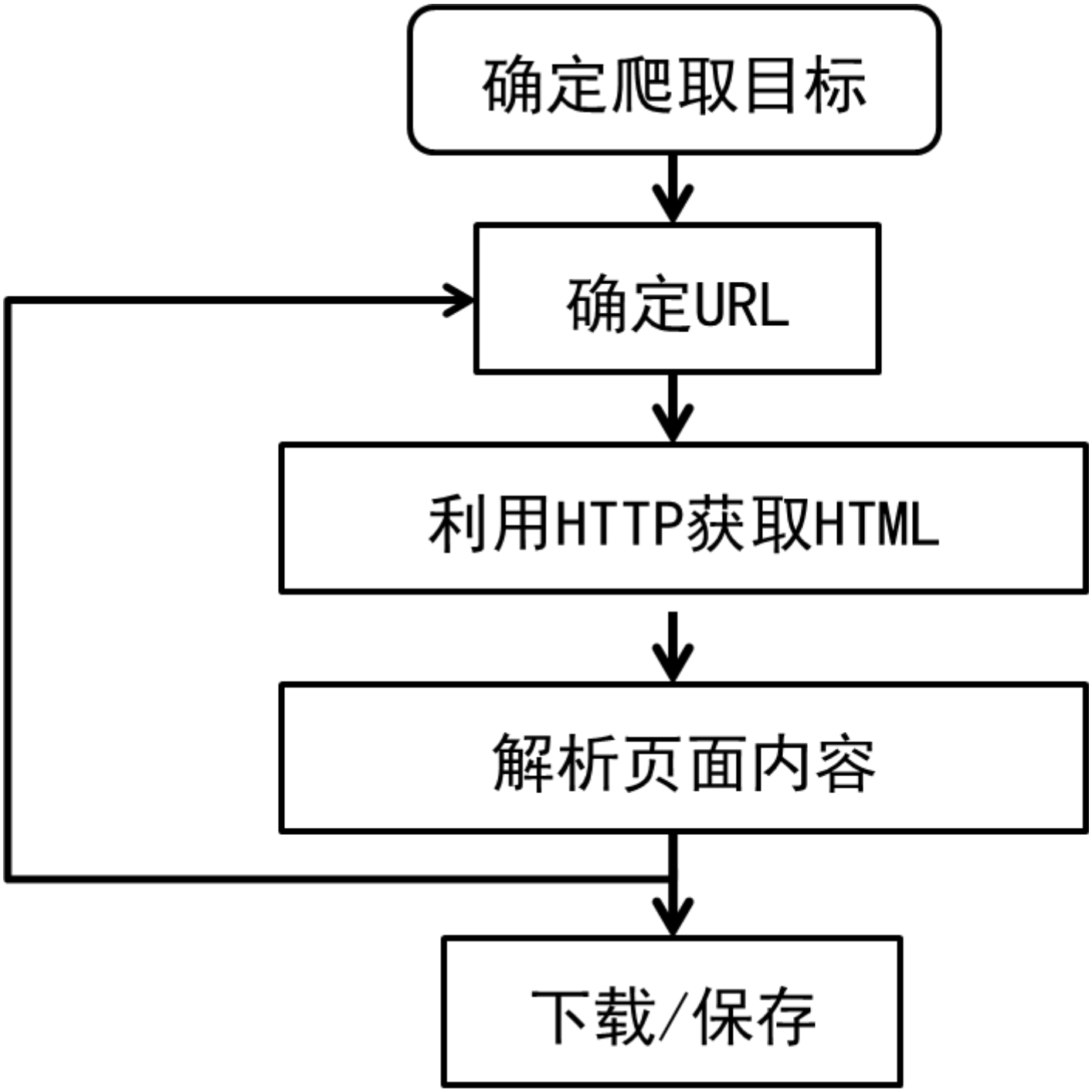
重点：

- 理解web页面爬取的基本过程
- 掌握urllib的基本用法，构建简单爬虫程序
- 掌握requests的基本用法，对简单爬虫程序进行优化

2 爬取Web页面的基本过程

我们先介绍使用网络爬虫爬取web页面的基本过程。

知识讲解



- 第一步，需要人工确定待爬取的目标网页的URL，并将这个URL放入待爬取队列。
- 第二步，网络爬虫程序要能够像浏览器一样，根据输入的URL向远程服务器发出HTTP请求，尝试获得响应内容，通常是目标数据所在的HTML文档。
- 第三步，网络爬虫程序要能够解析已下载HTML文档内容。这里的解析指的是检索HTML文档，提取出目标数据的过程。除了目标数据，可能还需要获得新的URL，为后续爬取提供种子。
- 第四步，网络爬虫程序需要将提取出的数据，按照用户所需的格式存储下来，通常会使用某种格式的文件或数据库来存储数据，将需要的图片或视频资源下载到特定的文件夹中，而爬得的新URL将放入待爬取队列中；
- 第五步，网络爬虫程序将按上述步骤，逐一处理待爬取队列中的URL，直到队列为空。

以上步骤，就是使用网络爬虫程序爬取Web页面的基本过程。在思路上，它并不复杂，那么我们能不能将之付诸实践呢？

案例与应用

暂无

模块练习与答案

见习题集

内容小结

本小节主要介绍了Web页面爬取的5个步骤。这几点需要学生牢记。

使用Urllib实现基本Web页面爬取

知识讲解

工欲利其事，必先利其器。换句话说，我们需要先搭建开发网络爬虫程序的编程环境。

我们建议大家选用Anaconda集成开发工具。

Anaconda是一个开源的Python发行版本，其包含了conda、Python等180多个科学包及其依赖项。对于学习数据科学的同学，这是非常好的一个集成工具。对于开发网络爬虫而言，我们将借助它的交互式编程环境，以及多种支持库，来构建我们的程序。

Anaconda3的安装过程较为简单，下面就以Windows系统中的安装过程为例，介绍一下安装Anaconda的过程和启动Jupyter notebook编辑器的方法。

- 1.请从Anaconda官方网站或清华大学开源软件镜像站下载支持Python3 的Anaconda安装包并启动安装过程。Windows下适用的Anaconda安装包是一个exe可执行程序，双击它启动安装过程；
- 2.在阅读产品的License文档并点击“I Agree”后，选择为“Just Me”或是“All users”安装该程序，然后选择存放Anaconda的本地路径，我们建议初学者使用默认路径。如果要设置自定义路径，请不要使用有空格和非unicode字符的路径，否则会出现错误。
- 3.选择将Anaconda增加到本地PATH路径，并选择将Anaconda作为您的默认python编译器。
- 4.点击“Finish”完成安装，之后您可以在Windows“启动”的程序列表中查看到Anaconda的启动快捷方式，为了验证安装是否成功，您可以点击运行“Anaconda navigator”，查看是否有弹出一个Anaconda 的导航界面。

经过上述4步，Anaconda就安装好了，但如果你在安装中出现了异常错误，可以在互联网上搜索解答信息。

安装好Anaconda后，我们需要启动交互式的Python开发环境。我们选择目前十分流行的Jupyter notebook作为我们的交互式开发环境。

启动方法很简单，在“Anaconda navigator”界面中就有启动Jupyter notebook的按钮，点击之后就可以启动了。此外，在Windows的程序列表中还有单独的“Jupyter notebook”启动项，点击后也可启动它。启动后的Jupyter notebook，通常会使用默认的浏览器打开一个页面，这个页面的URL是http://localhost:8888/tree,

为了讨论方便，同学们可以在Windows当前用户目录中的“我的文档”中新建一个名为“MyWebCrawlers”的新文件夹，用来存放我们将要编写的爬虫程序。通过Jupyter notebook打开页面的“new”按钮也可以实现这一点。

上述准备工作完成之后，下面我们就开始来编写自己的第一个网络爬虫程序吧？

首先让我们在Jupyter notebook打开的浏览器页面上，进入刚才建立的“MyWebCrawlers”文件夹，然后，点击右上方的“new”按钮，生成一个“Python3”文档。这个文档将是我们编写程序的主要场所。

我们编写Python网络爬虫时，需要借助一些支持库来简化我们的编写过程，例如使用Python3内置的Urllib库就是一个不错的选择。Urllib库中包含了4个处理URL的模块，分别是：

- Urllib.request，它用于打开和读取URL。
  - 在Urllib.request模块中定义了一些打开url的方法和类，除了可以帮助我们获取web页面，还可以帮助我们处理简单或摘要类型的页面认证，页面重定向、以及cookies等访问web页面时的常见问题。
- urllib.error ，它包含了request模块可能引发的异常；
- urllib.parse ，它用于解析URL
- urllib.robotparser 用于解析 robots.txt 文件。

这些模块的使用细节，我们将在接下来的课程中依次讲解。

下面，我们将使用urllib完成以下几个任务：

1. 访问指定的URL，获取响应结果；
2. 使设置HTTP请求头参数，使网络爬虫更像浏览器行为；
3. 使用GET或POST方法，向服务器传递参数；
4. 学会处理常见异常

要想访问指定的URL，通常会使用urllib.request中的urlopen函数。

使用urllib首先要引入urllib库。

```
import urllib

help(urllib)
```

```
    Help on package urllib:

    NAME
        urllib

    PACKAGE CONTENTS
        error
        parse
        request
        response
        robotparser

    FILE
        d:\pythonspace\anaconda3\lib\urllib\__init__.py
```

在我们了解了urllib中的包选项（即子模块）后，就使用更为精确的引入方式。

```
import urllib.error
import urllib.parse
import urllib.request
import urllib.robotparser
import urllib.response
```

这其中我们将频繁使用的request包，使用help方法，可以查看其基本信息。

可以从中看出，urllib.request的主要功能是使用各种网络协议（例如http）打开url。它也是使用urlopen获取某个网页最简单的方式。

在帮助中，还提供了一个例子。但这个例子优点复杂，我们稍后再进行介绍。

```
help(urllib.request)

# 以下为结果
    Help on module urllib.request in urllib:

    NAME
        urllib.request - An extensible library for opening URLs using a variety of protocols

    DESCRIPTION
        The simplest way to use this module is to call the urlopen function,
        which accepts a string containing a URL or a Request object (described
        below).  It opens the URL and returns the results as file-like
        object; the returned object has some extra methods described below.

        ...

        urlopen(url, data=None) -- Basic usage is the same as original
        urllib.  pass the url and optionally data to post to an HTTP URL, and
        get a file-like object back.  One difference is that you can also pass
        a Request instance instead of URL.  Raises a URLError (subclass of
        OSError); for HTTP errors, raises an HTTPError, which can also be
        treated as a valid response.
        ...

        Request -- An object that encapsulates the state of a request.  The
        state can be as simple as the URL.  It can also include extra HTTP
        headers, e.g. a User-Agent.

        ...

        Example usage:

        import urllib.request

        # set up authentication info
        authinfo = urllib.request.HTTPBasicAuthHandler()
        authinfo.add_password(realm='PDQ Application',
                              uri='https://mahler:8092/site-updates.py',
                              user='klem',
                              passwd='geheim$parole')

        proxy_support = urllib.request.ProxyHandler({"http" : "http://ahad-haam:3128"})

        # build a new opener that adds authentication and caching FTP handlers
        opener = urllib.request.build_opener(proxy_support, authinfo,
                                             urllib.request.CacheFTPHandler)

        # install it
        urllib.request.install_opener(opener)

        f = urllib.request.urlopen('http://www.python.org/')
```

```
...


VERSION
    3.7

FILE
    d:\pythonspace\anaconda3\lib\urllib\request.py
```

1 使用urllib.request.urlopen 方法访问指定的URL

为了使初学者不至于被过多的细节所困扰，我们下面先介绍使用urllib.request中最常用的urlopen方法。

urlopen它也是我们使用urllib获取普通网页的基本方法。

我们可以使用 help方法，获取这个函数的原型

```
help(urllib.request.urlopen)

# 以下为结果
    Help on function urlopen in module urllib.request:

    urlopen(url, data=None, timeout=<object object at 0x000002C994D8E6A0>, *, cafile=None, capat
        Open the URL url, which can be either a string or a Request object.

        *data* must be an object specifying additional data to be sent to
        the server, or None if no such data is needed.  See Request for
        details.

        urllib.request module uses HTTP/1.1 and includes a "Connection:close"
        header in its HTTP requests.

        The optional *timeout* parameter specifies a timeout in seconds for
        blocking operations like the connection attempt (if not specified, the
        global default timeout setting will be used). This only works for HTTP,
        HTTPS and FTP connections.

        If *context* is specified, it must be a ssl.SSLContext instance describing
        the various SSL options. See HTTPSConnection for more details.

        The optional *cafile* and *capath* parameters specify a set of trusted CA
        certificates for HTTPS requests. cafile should point to a single file
        containing a bundle of CA certificates, whereas capath should point to a
        directory of hashed certificate files. More information can be found in
        ssl.SSLContext.load_verify_locations().

        The *cadefault* parameter is ignored.

        This function always returns an object which can work as a context
        manager and has methods such as

        * geturl() - return the URL of the resource retrieved, commonly used to
          determine if a redirect was followed

        * info() - return the meta-information of the page, such as headers, in the
          form of an email.message_from_string() instance (see Quick Reference to
          HTTP Headers)

        * getcode() - return the HTTP status code of the response.  Raises URLError
          on errors.

        For HTTP and HTTPS URLs, this function returns a http.client.HTTPResponse
        object slightly modified. In addition to the three new methods above, the
        msg attribute contains the same information as the reason attribute ---
        the reason phrase returned by the server --- instead of the response
        headers as it is specified in the documentation for HTTPResponse.

        For FTP, file, and data URLs and requests explicitly handled by legacy
```

```
        URLopener and FancyURLopener classes, this function returns a
        urllib.response.addinfourl object.

        Note that None may be returned if no handler handles the request (though
        the default installed global OpenerDirector uses UnknownHandler to ensure
        this never happens).

        In addition, if proxy settings are detected (for example, when a *_proxy
        environment variable like http_proxy is set), ProxyHandler is default
        installed and makes sure the requests are handled through the proxy.
```

从上面的帮助中，我们可以看出，这个函数的一些细节：

- 功能：函数 urlopen 用于打开参数url指定的页面，这个url可以是字符串或是一个请求对象。
- 参数：函数 urlopen 有7个参数。
  - url：用于指定将要访问的网页url，无默认值，必须由用户给出。
  - data：用于加载用户传递给服务器的数据；默认值为None；
  - timeout：用于设置超时时间，是一个时间类型的=<object object at 0x000001BAE831E6A0>,
  - cafile=None, capath=None, cadefault=False，这三个参数与用户证书有关
  - context=None，这个参数用于传递各类型的SSL参数。
- 返回值：
  - 对于HTTP、HTTPS类型的URL，函数返回是http.client.HTTPResponse对象；
  - 对于FTP、文件、数据URL和请求对象型的URL，函数返回一个urllib.response.addinfourl对象。

下面我们尝试使用urlopen，来实现**目标1.访问指定的url，获取响应结果**。

```
import urllib.request

url = "http://www.baidu.com"
r = urllib.request.urlopen(url)
print(r)
```

```
<http.client.HTTPResponse object at 0x000002C9975FA588>
```

由于urlopen的返回值是http.client.HTTPResponse类型，所以为了得到人们可读的信息，我们下面简单探索一下这个类型。

还是使用help方法,我们主要关注它的方法

```
help(r)

# 以下为结果

    Help on HTTPResponse in module http.client object:

    class HTTPResponse(io.BufferedIOBase)
     |  HTTPResponse(sock, debuglevel=0, method=None, url=None)
     |
     |  Base class for buffered IO objects.
     |
     |  The main difference with RawIOBase is that the read() method
     |  supports omitting the size argument, and does not have a default
     |  implementation that defers to readinto().
     |
     |  In addition, read(), readinto() and write() may raise
     |  BlockingIOError if the underlying raw stream is in non-blocking
     |  mode and not ready; unlike their raw counterparts, they will never
     |  return None.
     |
     |  A typical implementation should not inherit from a RawIOBase
     |  implementation, but wrap one.
```

http.client.HTTPResponse对象可调用的方法中，有几个很重要，使用它们可以获得我们所需的信息.

它们分别是：

- read()方法,用于以字节形式，读取并返回响应内容；
- geturl()方法，用于返回页面真实的url；
- info()方法，用于返回与URL相关的元信息；
- getheaders()方法，用于返回响应头部信息；
- getcode()方法，用于返回http响应代码；

为了得到人类可读的信息，我们需要使用的是read方法。我们把上面的代码做简单修改,就可以看到响应内容了。

```
import urllib.request

url = "http://www.baidu.com"
r = urllib.request.urlopen(url)
print(r.read())

# 以下为部分执行结果

    b'<!DOCTYPE html>\n<!--STATUS OK-->\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\
    ......
    'result_kw\':"#kw"\n});\n</script>\n\n<script>\nif(navigator.cookieEnabled){\n\tdocument.coc
```

事实上，上面的r，即http.client.HTTPResponse对象，会以文件形式保存在缓存中，我们使用read方法将它读出。

由于读取方法属于io操作，为了确保在读取IO资源后正确关闭资源，所以我们推荐使用下列方法。即使用with语句，with语句是python中处理IO资源的经典语句，它可以确保打开的资源在执行完with语句之后正确的被关闭，而不会因程序员遗忘而形成隐患。

可以看到，响应内容由很多的字符组成，这些字符事实上就是我们使用浏览器访问www.baidu.com 这个url时得到的网页源代码。

为了验证这一事实，我们可以使用google浏览器chrome的开发者工具来查看。

由于read方法返回的是字节型数据，我们可以将其返回值视为字符串，字符串在python中是一种特殊的列表，所以如果只需要获得响应结果中的部分内容，可以使用列表切片来实现。例如：

```
import urllib.request

url = "http://www.baidu.com"
with urllib.request.urlopen(url) as r:
    print(r.read()[500:1000])

# 以下为部分执行结果

    b'<meta http-equiv="X-UA-Compatible" content="IE=Edge">\n\t<meta content="always" name="refe
```

可以看到，上面的响应结果中存在一些不可读编码，特别是一些有意义的标签内容，例如title的内容。

如何将这些编码转换为可读编码呢？

我们可以使用decode()方法，例如：

```
import urllib.request

url = "http://www.baidu.com"
with urllib.request.urlopen(url) as r:
    print(r.read()[500:1000].decode('utf-8'))
# 以下为部分执行结果

    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
        <meta content="always" name="referrer">
        <meta name="theme-color" content="#2932e1">
        <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
        <link rel="search" type="application/opensearchdescription+xml" href="/content-search.xm
        <link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu_85beaf5496f291521eb75b


        <link rel="dns-prefetch" href="//s1.bdstatic.com"/>
```

如果你想把缓存的文件保存下来，可以使用urllib.request.urlretrieve()方法。

```
help(urllib.request.urlretrieve)

# 以下为部分执行结果

    Help on function urlretrieve in module urllib.request:

    urlretrieve(url, filename=None, reporthook=None, data=None)
        Retrieve a URL into a temporary location on disk.

        Requires a URL argument. If a filename is passed, it is used as
        the temporary file location. The reporthook argument should be
        a callable that accepts a block number, a read size, and the
        total file size of the URL target. The data argument should be
        valid URL encoded data.

        If a filename is passed and the URL points to a local resource,
        the result is a copy from local file to new file.

        Returns a tuple containing the path to the newly created
        data file as well as the resulting HTTPMessage object.
```

urllib.request.urlretrieve()方法用于访问指定url，并将响应内容存储到本地的临时文件中。

使用时需要特别指定的参数有：

- url：待访问的页面地址。
- filename：存储响应结果的文件路径。

urllib.request.urlretrieve()方法的返回值是一个元组，包括两方面：

- 新生成的文件名
- 响应产生的HTTPMessage对象

```
import urllib.request

url = "http://www.baidu.com"
localfile, headers = urllib.request.urlretrieve(url)
print("响应头信息：")
print(headers)
print("本地文件名：")
print(localfile)

# 以下为部分执行结果
```

```
响应头信息：
Bdpagetype: 1
Bdqid: 0xb7726796003073b2
Cache-Control: private
Content-Type: text/html
Cxy_all: baidu+d6a8029e425f552015c6586a032a3206
Date: Thu, 11 Jul 2019 07:15:07 GMT
Expires: Thu, 11 Jul 2019 07:14:49 GMT
P3p: CP=" OTI DSP COR IVA OUR IND COM "
Server: BWS/1.1
Set-Cookie: BAIDUID=006F323E4A1132EC633BD24290A34F73:FG=1; expires=Thu, 31-Dec-37 23:55:55 G
Set-Cookie: BIDUPSID=006F323E4A1132EC633BD24290A34F73; expires=Thu, 31-Dec-37 23:55:55 GMT;
Set-Cookie: PSTM=1562829307; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/
Set-Cookie: delPer=0; path=/; domain=.baidu.com
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=1423_21093_29523_29521_29238_28518_29099_28836_29220; path=/; domain=
Vary: Accept-Encoding
X-Ua-Compatible: IE=Edge,chrome=1
Connection: close
Transfer-Encoding: chunked


本地文件名：
C:\Users\leo\AppData\Local\Temp\tmp__eiqy59
```

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。

- 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；

- r.info()
- r.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。

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

url = 'http://www.baidu.com'
with urllib.request.urlopen(url) as r:
    print('响应码：')
    print(r.code)
    #print(r.status)
    print('响应头信息：')
    #print(r.headers)
    print(r.info())
    print('获得页面的真实url：')
    #print(r.url)
    print(r.geturl())


# 以下为部分执行结果

    响应码：
    200
    响应头信息：
    Bdpagetype: 1
    Bdqid: 0xdc6d87de002f9bc1
    Cache-Control: private
    Content-Type: text/html
    Cxy_all: baidu+74098a06a74a3752310420b2c161e5ce
    Date: Thu, 11 Jul 2019 07:15:48 GMT
    Expires: Thu, 11 Jul 2019 07:15:35 GMT
    P3p: CP=" OTI DSP COR IVA OUR IND COM "
    Server: BWS/1.1
    Set-Cookie: BAIDUID=386F6B1508A7F49276508F9C2C09BA9E:FG=1; expires=Thu, 31-Dec-37 23:55:55 G
    Set-Cookie: BIDUPSID=386F6B1508A7F49276508F9C2C09BA9E; expires=Thu, 31-Dec-37 23:55:55 GMT;
    Set-Cookie: PSTM=1562829348; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/
    Set-Cookie: delPer=0; path=/; domain=.baidu.com
    Set-Cookie: BDSVRTM=0; path=/
    Set-Cookie: BD_HOME=0; path=/
    Set-Cookie: H_PS_PSSID=1433_21117_29522_29519_29237_28519_29098_28836_29220_26350; path=/; d
    Vary: Accept-Encoding
    X-Ua-Compatible: IE=Edge,chrome=1
    Connection: close
    Transfer-Encoding: chunked


    获得页面的真实url：
    http://www.baidu.com
```

2 使设置HTTP请求头参数，使网络爬虫更像浏览器行为

下面，我们尝试使用urllib完成本讲的第2个任务。

Web服务器在接收客户端请求时，有时会根据客户端类型给出响应结果，例如手机浏览器访问百度与pc浏览器访问百度的结果就不相同。

我们做两个个实验来验证：

- 实验1 利用httpbin网站的user-agent响应功能，查看请求头中的user-agent

使用这个功能时，需要先登录httpbin网站，设置user-agent功能。

```
import urllib.request

url = "http://www.httpbin.org/user-agent"
with urllib.request.urlopen(url) as r:
    print(r.read())
# 以下为部分执行结果

    b'{\n  "user-agent": "Python-urllib/3.7"\n}\n'
```

```
import urllib.request

url = "http://www.httpbin.org/user-agent"
headers = {'User-Agent':'Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) Apple
request = urllib.request.Request(url,headers=headers)

with urllib.request.urlopen(request) as r:
    print(r.read())

# 以下为部分执行结果

    b'{\n  "user-agent": "Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) Appl
```

- 实验2 使用不同的user-agent访问百度

```
import urllib.request

url = "http://www.baidu.com"
with urllib.request.urlopen(url) as r:
    print(r.read()[500:2000].decode('utf-8'))

# 以下为部分执行结果

    <meta http-equiv="X-UA-Compatible" content="IE=Edge">


        <title>百度一下，你就知道</title>
```

```
import urllib.request

url = "http://www.baidu.com"
headers = {'User-Agent':'Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) Apple
request = urllib.request.Request(url,headers=headers)

with urllib.request.urlopen(request) as r:
    print(r.read()[500:2000].decode('utf-8'))


# 以下为部分执行结果

<title>百度一下</title><script>window._performanceTimings=[['firstLine',+new Date
```

通过上面的实验，我们可以确定的是，客户端使用不同的user-agent值，服务器会有不同的响应。

有些网站服务器还会根据这一点，限制网络爬虫的访问，所以我们需要设置爬虫的user-agent，使其更像一般的PC或手机。

下面举例说明：

将爬虫程序伪装为PC版chrome浏览器

```
import urllib.request

url = "http://www.baidu.com"
headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge
request = urllib.request.Request(url,headers=headers)

with urllib.request.urlopen(request) as r:
    print(r.read()[500:2000].decode('utf-8'))
```

```
# 以下为部分执行结果
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
        <meta content="always" name="referrer">
        <meta name="theme-color" content="#2932e1">
        <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
        <link rel="search" type="application/opensearchdescription+xml" href="/content-search.xml" title="百度搜索
        <link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu_85beaf5496f291521eb75ba38eacbd87.svg">


        <link rel="dns-prefetch" href="//s1.bdstatic.com"/>
        <link rel="dns-prefetch" href="//t1.baidu.com"/>
        <link rel="dns-prefetch" href="//t2.baidu.com"/>
        <link rel="dns-prefetch" href="//t3.baidu.com"/>
        <link rel="dns-prefetch" href="//t10.baidu.com"/>
        <link rel="dns-prefetch" href="//t11.baidu.com"/>
        <link rel="dns-prefetch" href="//t12.baidu.com"/>
        <link rel="dns-prefetch" href="//b1.bdstatic.com"/>

        <title>百度一下，你就知道</title>


    <style id="css_index" index="index" type="text/css">html,body{height:100%}
    html{overflow-y:auto}
    body{font:12px arial;text-align:;background:#fff}
    body,p,form,ul,li{margin:0;padding:0;list-style:none}
    body,form,#fm{position:relative}
    td{text-align:left}
    img{border:0}
    a{color:#00c}
    a:active{color:#f60}
    input{border:0;padding:0}
    #wrapper{position:relative;_position:;min-height:100%}
    #head{padding-bottom:100px;text-align:center;*z-index:1}
    #ftCon{height:50px;position:absolute;bottom:47px;text-align:left;width:100%;margin:0 auto;z-index:0;overflow:
    .ftCon-Wrapper{overflow:hid
```

将爬虫程序伪装为移动版 **chrome** 浏览器

```python
import urllib.request

url = "http://www.baidu.com"
headers = {'User-Agent':'Mozilla/5.0(Linux;U;Android2.2.1;zh-cn;HTC_Wildfire_A3333Build/FRG83D)A
request = urllib.request.Request(url,headers=headers)

with urllib.request.urlopen(request) as r:
    print(r.read()[500:2000].decode('utf-8'))
```

将爬虫程序伪装为移动版 **QQ** 浏览器

```
import urllib.request

url = "http://www.baidu.com"
headers = {'User-Agent':'MQQBrowser/26Mozilla/5.0(Linux;U;Android2.3.7;zh-cn;MB200Build/GRJ22;Cy
request = urllib.request.Request(url,headers=headers)

with urllib.request.urlopen(request) as r:
    print(r.read()[500:2000].decode('utf-8'))
```

3 使用GET或POST方法，向服务器传递参数

下面我们尝试使用urllib，完成本讲的第3个任务。

浏览器或网络爬虫等客户端，在查询网站信息或实现用户登录时，需要使用GET方法或POST方法实现数据的传递。

3.1 向服务器传递GET方法参数

首先分析一下浏览器上使用百度搜索关键字的过程。

以搜索“北京”关键字为例。

接下来，我们尝试使用爬虫程序完成百度搜索，即利用Get方法向baidu提交查询请求，然后获取百度的查询结果。

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

url = 'http://www.baidu.com/s?'
querystr = {'wd':'北京'}
querystr_encode = urllib.parse.urlencode(querystr)
print(querystr_encode)
#https://www.baidu.com/s?wd=%E5%8C%97%E8%88%AA
theurl = url + querystr_encode
print(theurl)
headers = {'Accept':'text/html',
           'User-Agent':'Mozilla/5.0',
          }
request = urllib.request.Request(theurl,headers=headers)
with urllib.request.urlopen(request) as response:
    if response.status == 200:
        print(response.read()[500:].decode('utf-8'))

# 以下为部分执行结果

    wd=%E5%8C%97%E4%BA%AC
    http://www.baidu.com/s?wd=%E5%8C%97%E4%BA%AC
          <link rel="icon" sizes="any" mask href="//www.baidu.com/img/baidu_85beaf5496f291521eb7
            <link rel="search" type="application/opensearchdescription+xml" href="/content-searc


    <title>北京_百度搜索</title>
    ......

    <div class="opr-recommends-merge-content">

        <div class="cr-title c-clearfix">
                <a class="cr-title-sub opr-recommends-merge-more-btn" href="javascript:;" onclic
            <span title="北京著名景点">北京著名景点</span>
                </div>
            ......
        <div class="c-span4  opr-recommends-merge-item " data-click="{'rsv_re_ename':'故宫','rsv
                                                    <div class="opr-recommends-merge-p">
                <a target="_blank" href="/s?wd=%E6%95%85%E5%AE%AB&usm=4&ie=utf-8&rsv_cq=%E5%8C%9
                <a class="opr-recommends-merge-mask" target="_blank" href="/s?wd=%E6%95%85%E5%AE
            <div class="c-gap-top-small"><a target="_blank" title="故宫" href="/s?wd=%E6%95%85%E
                            <div class="opr-recommends-merge-d">
                                <p class="opr-recommends-merge-width-text">明清两代的皇宫</p>
                        </div>
                                </div>


    </html>
```

3.2 向服务器传递 POST 方法 参数

首先分析一下浏览器上使用百度翻译的情况。

我们以访问 http://xwqy.gsxt.gov.cn/etps/productInfoList.do 网站为例。在查询金融服务信息时会使用post方法传递信息。我们可以使用浏览器的开发者模式进行观察这一过程。

接下来，我们尝试使用爬虫程序获取查询结果，即利用POST方法向 http://xwqy.gsxt.gov.cn/etps/productInfoList.do 提交查询请求，然后获取查询的结果。

```
import urllib.request
import urllib.parse

url ='http://xwqy.gsxt.gov.cn/etps/productInfoList.do'
#url = 'https://fanyi.baidu.com/v2transapi'
payload = {'loanQuota': '100万元及以下'}

payload_encode = urllib.parse.urlencode(payload).encode('utf-8')
print(payload_encode)

headers = {'Accept':'text/html',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge
request = urllib.request.Request(url,headers=headers,data = payload_encode,method='POST')
r= urllib.request.urlopen(request)
with urllib.request.urlopen(request) as response:
    if response.status == 200:
        rf = response.read()
        print(rf.decode('utf-8'))
```

```
# 以下为部分执行结果
    <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1" />
        <title>小微企业名录</title>

    <base href="http://xwqy.gsxt.gov.cn/" />
    ......
                                <div class="clist"> <span class="xdtag">抵押</span>
                                <h3><a href="javascript:" onclick="funcApp('000000006a1eee1c016a49b50d9201aa')" t
                                <div class="cs">
                                        <strong>1-500万元</strong>
                                        <i>贷款利率<b>4.35-8.00%</b></i>
                                        <span>贷款期限：1年以下（含1年）</span>
                                        <span>贷款银行：中国工商银行</span>
                                </div>
                                <div class="sqbox">
                                  <input type="submit" onclick="appPage('000000006a1eee1c016a49b50d9201aa');retur
                                </div>
                            </div>

                                <div class="clist"> <span class="xdtag">信用</span>
                                <h3><a href="javascript:" onclick="funcApp('0000000065f68eaa0166242584b50139')" t
                                <div class="cs">
                                        <strong>1-200万元</strong>
                                        <i>贷款利率<b>5.44-5.44%</b></i>
                                        <span>贷款期限：1年以下（含1年）</span>
                                        <span>贷款银行：中国建设银行</span>
                                </div>
                                <div class="sqbox">
                                  <input type="submit" onclick="appPage('0000000065f68eaa0166242584b50139');retur
                                </div>
                            </div>

    </body>
    </html>
    </div>
    <div class="footer">
         <p>版权所有：国家市场监督管理总局</p>
         <p>地址：北京市西城区三里河东路八号  邮政编码：100820  技术支持电话：010-88650856</p>
    </div>
    </body>
    </html>
```

4 常见异常处理

下面我们来尝试使用urllib完成本讲的第4个任务，即处理异常。

网络爬虫利用网络获取数据，而网络通信中难免会有异常，所以要想使爬虫程序变得健壮就需要引入异常处理机制。

首先看一个没有异常处理的例子：

```
import urllib.request
import urllib.parse

url ='http://www.google.com/'
headers = {'Accept':'text/html',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge


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

with urllib.request.urlopen(request) as response:
    if response.status == 200:
        rf = response.read()
        print(rf.decode('utf-8'))


# 以下为部分执行结果

    ---------------------------------------------------------------------------

    TimeoutError                              Traceback (most recent call last)

    D:\pythonspace\anaconda3\lib\urllib\request.py in do_open(self, http_class, req, **http_conn_args)
       1316                 h.request(req.get_method(), req.selector, req.data, headers,
    -> 1317                           encode_chunked=req.has_header('Transfer-encoding'))
       1318             except OSError as err: # timeout error


    D:\pythonspace\anaconda3\lib\http\client.py in request(self, method, url, body, headers, encode_chunked)
       1228         """Send a complete request to the server."""
    -> 1229         self._send_request(method, url, body, headers, encode_chunked)
       1230

    ......

    D:\pythonspace\anaconda3\lib\urllib\request.py in do_open(self, http_class, req, **http_conn_args)
       1317                           encode_chunked=req.has_header('Transfer-encoding'))
       1318             except OSError as err: # timeout error
    -> 1319                 raise URLError(err)
       1320             r = h.getresponse()
       1321         except:


    URLError: <urlopen error [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。>
```

接下来，我们尝试引入异常处理机制。

```
import urllib.request
import urllib.parse

url ='http://www.google.com/'
headers = {'Accept':'text/html',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge

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

    with urllib.request.urlopen(request) as response:
        if response.status == 200:
            rf = response.read()
            print(rf.decode('utf-8'))
except urllib.error.URLError as e:
    print(e)



# 以下为部分执行结果
    <urlopen error [WinError 10060] 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。>
```

我们还可以使自己的程序更专业一些，将异常信息不仅显示在控制台上，还将其存入日志文件。

注意，此时要使jupyter notebook有权限写文件。

```
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='.\log\mycrawler.log',
                    level=logging.DEBUG,decode='utf-8')


try:

    import socket
    # timeout in seconds
    timeout = 3
    socket.setdefaulttimeout(timeout)

    url ='http://www.google.com/'
    headers = {'Accept':'text/html',
           'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Ge

    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.BaseHTTPReques
    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。详细信息可以查看日志文件。')
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。详细信息可以查看日志文件。')
```

案例

见第三章案例1

习题

见习题集

内容小结

本小节主要介绍了：

- Anaconda的安装使用
- Urllib库基本情况
- urllib.request.urlopen()的基本用法
- 结合4个任务介绍了实现Web页面爬取的基本过程和细节。

3 使用Python Requests优化页面爬取过程

使用urllib库可以实现包括访问URL、传递参数、设置代理、进行身份认证等功能，但它的一些操作不太简洁。为了使我们的爬虫程序更加优雅，我们下面介绍另一个Python库 ：requests库，使用它使页面爬取过程更加优化。

知识讲解

Requests库基于urllib库，也是一个用于处理URL的工具库。使用它比urllib更加简洁。它的主要特点包括：

- 能够自动保持与网站的TCP连接，
- 内含带持久cookie的会话机制，
- 能够提供像浏览器类似的SSL认证，
- 能够自动完成内容解码，
- 能够支持文件分块上传和流式下载，
- 能够自动对下载图片、文件的解压，
- 支持Unicode响应结果等等。

下面我们将使用requests，完成以下任务：

1. 构建requests爬虫程序框架，完成对指定URL的访问，获取并分析响应结果；
2. 定制http请求头，完成复杂POST请求和响应；
3. 使用request处理cookie，使用requests会话对象；
4. 设置代理服务器
5. 设置SSL证书验证

安装requests库

首先介绍requests库的安装过程。

1. 大家启动anaconda prompt；
2. 确保自己的电脑能访问互联网；
3. 在打开的命令行下键入如下命令：

```
pip install requests
```

4. 安装过程一般不会有异常，安装结束后启动jupyter notebook，运行下列代码，测试requests库是否可以正常导入；
5. 如果没有问题，则安装成功；否则，可以考虑重新执行上述步骤。

```
import requests

help(requests)
```

```
# 以下为部分执行结果
    Help on package requests:

    NAME
        requests

    DESCRIPTION
        Requests HTTP Library
        ~~~~~~~~~~~~~~~~~~~~~

        Requests is an HTTP library, written in Python, for human beings. Basic GET
        usage:

           >>> import requests
           >>> r = requests.get('https://www.python.org')
           >>> r.status_code
           200
           >>> 'Python is a programming language' in r.content
           True

        ... or POST:

           >>> payload = dict(key1='value1', key2='value2')
           >>> r = requests.post('https://httpbin.org/post', data=payload)
           >>> print(r.text)
           {
             ...
             "form": {
               "key2": "value2",
               "key1": "value1"
             },
             ...
           }

        The other HTTP methods are supported - see `requests.api`. Full documentation
        is at <http://python-requests.org>.

        :copyright: (c) 2017 by Kenneth Reitz.
        :license: Apache 2.0, see LICENSE for more details.

    PACKAGE CONTENTS
        __version__
        _internal_utils
        adapters
        api
        auth
        certs
        compat
        cookies
        exceptions
        help
```

```
    hooks
    models
    packages
    sessions
    status_codes
    structures
    utils

FUNCTIONS
    check_compatibility(urllib3_version, chardet_version)

DATA
    __author_email__ = 'me@kennethreitz.org'
    __build__ = 139520
    __cake__ = '✨ 🍰 ✨'
    __copyright__ = 'Copyright 2018 Kenneth Reitz'
    __description__ = 'Python HTTP for Humans.'
    __license__ = 'Apache 2.0'
    __title__ = 'requests'
    __url__ = 'http://python-requests.org'
    codes = <lookup 'status_codes'>
    cryptography_version = '2.4.2'

VERSION
    2.21.0

AUTHOR
    Kenneth Reitz

FILE
    d:\pythonspace\anaconda3\lib\site-packages\requests\__init__.py
```

初识**requests**库

requests库是一个python语言编写的http工具库，它基于urllib，但比urllib更加易用。

查看帮助信息，可以看到requests库的是使用方法十分简单。

下面，我们结合本讲的任务，举例说明requests的使用方法。

3.1 构建**requests**爬虫程序框架，完成对指定**URL**的访问，获取并分析响应结果。

```
import requests

r = requests.get('https://www.baidu.com')
print("响应码：")
print(r.status_code)
print("响应内容：")
print(r.text)
```

可以看到上面的响应内容中存在一些不可读的编码，我们需要做一些编码转换。而且上面的代码过于简单，不能防止异常的发生，我们还需要引入一定的异常处理机制。

下面，我们构建一个利用request访问url的函数。我们可以在今后的网络爬虫程序中能够复用它。

```
def fetchUrl(url):
    try:
        r = requests.get(url)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except:
        return "Some exceptions were raised."

url = "https://www.baidu.com"
fetchUrl(url)
```

理解响应对象

上面的处理，使用了响应对象的若干方法和属性。下面再介绍一些有关rquests响应对象的知识。

```
"""理解响应"""
import requests

r = requests.get('http://api.github.com/events')

print("响应状态码：")
print(r.status_code)
print('响应头信息：')
print(r.headers)
print('响应内容编码格式：')
print(r.encoding)
print('cookie：')
print(r.cookies)
print("重定向与历史:")
print(r.history)
print("访问的url:")
print(r.url)
```

```
响应状态码：
200
响应头信息：
{'Date': 'Thu, 11 Jul 2019 07:31:08 GMT', 'Content-Type': 'application/json; charset=utf-8', 'Tr
响应内容编码格式：
utf-8
cookie：
<RequestsCookieJar[]>
重定向与历史：
[<Response [301]>]
访问的url：
https://api.github.com/events
```

3.2 定制http请求头，完成复杂POST请求和响应

使用requests库访问url时，我们的爬虫程序作为客户端，其user-agent值为python-requests/2.21.0，这一点我们可以通过访问httpbin的user-agent页面进行验证。

```
import requests

url = "http://www.httpbin.org/user-agent"
with requests.get(url) as r:
    print(r.text)
```

```
{
  "user-agent": "python-requests/2.21.0"
}
```

为了使爬虫程序借用浏览器的user-agent，使其更像浏览器，可以使用下列方法定制http请求头。

```
import requests

def fetchUrl(url):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }
        r = requests.get(url,headers=headers)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except:
        return "Some exceptions were raised."

url = "http://www.httpbin.org/user-agent"
fetchUrl(url)
```

```
'{\n  "user-agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
```

如何使用requests库向网站服务器提交信息呢？

3.3 使用GET和POST方法提交信息

使用GET方法提交信息

```
import requests

def fetchUrl(url,querystr):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }
        queryload = {'wd':querystr}
        r = requests.get(url,params = queryload,headers = headers)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except:
        return "Some exceptions were raised."

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

fetchUrl(url,'北京')
```

使用POST方法提交信息

```
import requests

def fetchUrl(url,queryload):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }

        r = requests.post(url,data = queryload,headers = headers)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except requests.exceptions.HTTPError as e:
        print(e)
        return "Some exceptions were raised."


url = ' http://xwqy.gsxt.gov.cn/etps/productInfoList.do '
payload = {'loanQuota': '100万元及以下'}
fetchUrl(url,payload)
```

3.4 使用requests会话对象保存和传递cookies

cookie是网站为了辨别用户身份、进行会话跟踪而储存在用户本地端和服务器上的数据。

下面的代码显示了访问www.baidu.com 时，百度服务器生成的cookie值。

```
import requests

def fetchUrl(url):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }

        r = requests.get(url,headers = headers)
        r.raise_for_status()
        r.encoding = r.apparent_encoding

        return print(r.cookies)

    except:
        return "Some exceptions were raised."


url = 'https://tieba.baidu.com/index.html'

fetchUrl(url)
```

很多网站服务器提供给已登录用户和未登录用户的信息是不同的。而记录用户状态的是cookies。

为了使用爬虫程序访问已登录用户才能流量的网页，我们可以采取以下策略：

1. 手动登录某个网站；
2. 通过浏览器开发者模式，获取当前cookies
3. 在爬虫程序中建立Session对象；
4. 使用cookies数据更新Session对象的cookies，之后再使用爬虫访问受保护的网页。

我们以访问百度为例说明这个过程。

```
import requests

def fetchUrl(url,cookies):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }
        print("提交请求前的cookies值：")
        session = requests.Session()
        session.cookies.update(cookies)
        print(session.cookies)
        r = session.get(url,headers = headers)

        r.raise_for_status()
        r.encoding = r.apparent_encoding

        print("服务器响应中的cookies值：")
        print(r.cookies)

        return r.text

    except requests.exceptions.HTTPError as e:
        print(e)
        return "Some exceptions were raised."

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

cookies = dict(BAIDUID='3E1B4DE81836633B09BA505869736F3C:FG=1',
               PSTM='1561023988',
               BIDUPSID='E53026ACD99DA573C9474C276849AFFA',
               BDORZ='B490B5EBF6F3CD402E515D22BCDA1598',
               MCITY='-131%3A',
               yjs_js_security_passport='74440fdbd35296135d63011f8d0abe55d5e700f5_1562132807_js'
               BDUSS='GJFdmdPcjVPMUUteVpZcXoybkFsS1ZsMmVTfnFFbGk3YX5SYmhUcm15Slk3RU5kSVFBQUFBJCQ
               H_PS_PSSID='1437_21111_29135_29238_28518_29098_28833_29220_29439_20718',
               delPer='0',
               PSINO='1',
               PHPSESSID='rt2292okka0ubt079se46cpp47',
               Hm_lvt_4010fd5075fcfe46a16ec4cb65e02f04='1562139865,1562141343',
               Hm_lpvt_4010fd5075fcfe46a16ec4cb65e02f04='1562141343',)
cookies['BDRCVFR\[feWj1Vr5u3D\]']='I67x6TjHwwYf0'
fetchUrl(url,cookies)
```

3.5 设置代理服务器

如果需要使用代理，你可以通过为任意请求方法提供 proxies 参数来配置单个请求:

```
import requests

def fetchUrl(url,proxies = None):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }
        if proxies:
            r = requests.get(url,headers = headers,proxies = proxies)
        else:
            r = requests.get(url,headers = headers)

        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except requests.exceptions.HTTPError as e:
        print(e)
        return "Some exceptions were raised."


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

proxies = {
  "http": "59.108.125.241:8080",
  "http": "47.104.172.108:8118",
}
fetchUrl(url,proxies = proxies)
```

3.6 设置SSL证书验证

Requests 可以为 HTTPS 请求验证 SSL 证书，就像 web 浏览器一样。

SSL 验证默认是开启的，如果证书验证失败，Requests 会抛出 SSLError.

下面举例说明：

```
import requests
import os
def fetchUrl(url,cafile):
    try:
        headers = {'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
                   'User-Agent':'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
                  }
        r = requests.get(url,headers = headers,verify =cafile)

        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text

    except requests.exceptions.HTTPError as e:
        print(e)
        return "Some exceptions were raised."


url = 'https://www.citibank.com.cn/sim/index.htm'

cafile = 'sampleca.cer'
fetchUrl(url,cafile)
```

案例

见第三章案例2

习题

见习题集

内容小结

本小节主要介绍了Requests库爬取web页面的方法。

Requests库较Urllib库更加人性化，有利于快速构建Web页面获取程序。

Requests库的应用关键在于理解requests.get()、requests.post、requests.Session等方法或类的应用。

本节总结

本节主要介绍了如何通过python自带库urllib和第三方 python http库 requests来获取web页面内容。

urllib库作为python自带的http方法库，可以构建Web页面爬取程序、处理异常、解析url编码，但操作时需要注意较多的技术细节，新手使用时应多查看帮助文件，从根本上弄通道理。

requests库是基于urllib的http方法库，它的可用性较高，开发速度更快，建议多加练习使用。

课后练习

见习题集。