一、爬虫基础

"大数据时代"，获取数据方式有哪些?

- 企业产生的数据：百度搜索指数、腾讯公司业绩数据、阿里巴巴集团财务及运营数据、新浪微博微指数　等...

> 大型互联网公司拥有海量用户，有天然的数据积累优势，还有一些有数据意识的中小型企业，也开始积累自己的数据。

- 数据平台购买数据：数据堂、国云数据市场、贵阳大数据交易所　等...

> 在各个数据交易平台上购买各行各业各种类型的数据，根据数据信息、获取难易程度的不同，价格也会有所不同。

- 政府/机构公开的数据：中华人民共和国国家统计局数据、中国人民银行调查统计、世界银行公开数据、联合国数据、纳斯达克、新浪财经美股实时行情 等...

> 通常都是各地征服统计上报，或者是行业内专业的网站、机构等提供。

- 数据管理咨询公司：麦肯锡、埃森哲、尼尔森、中国互联网络信息中心、艾瑞咨询　等...

> 通常这样的公司有很庞大的数据团队，一般通过市场调研、问卷调查、固定的样本检测、与各行各业的其他公司合作、专家对话来获取数据，并根据客户需求制定商业解决方案。

- 爬取网络数据：

> 如果数据市场上没有需要的数据，或者价格太高不愿意购买，那么可以利用爬虫技术，抓取网站上的数据。

关于Python网络爬虫，我们需要学习的有：

1. Python基础语法学习（基础知识）
2. 对HTML页面的内容抓取（Crawl）
3. 对HTML页面的数据解析（Parse）
4. 动态HTML的处理/验证码的处理 (针对反爬处理)
5. Scrapy框架以及scrapy-redis分布式策略（第三方框架）
6. 爬虫(Spider)、反爬虫(Anti-Spider)、反反爬虫(Anti-Anti-Spider)之间的斗争....

1.基本原理

我们可以把互联网比作一张大网，而爬虫即（网络爬虫）便是在网上爬行了蜘蛛。把网的节点比作个个网页，爬虫爬到这就相当于访问了该页面，获取了其信息。可以把节点间的连线比作网页与网页之间的链接关系，这样蜘蛛通过一个节点后，可以顺着节点连线继续爬行到达下一个节点，即通过一个网页继续获取后续的网页，这样整个网的节点便可以被蜘蛛全部爬行到，网站的数据就以被抓取下来了

2.基本概述

简单来说，爬虫就是获取网页并提取和保存信息的自动化程序

1）获取网页

爬虫首先要做的工作就是获取网页，这里就是获取网页的源代码。源代码里包含了网页的部分有用信息，所以只要把源代码获取下来，就可以从中提取想要的信息了。

之前讲过请求和响应的概念，向网站的服务器发送一个请求，返回的响应体便是网页源代码。所以，最关键的部分就是构造一个请求并发送给服务器，然后接收到响应并将其解析出来，那么这个流程怎样实现呢？

Python提供了许多库来帮助我们实现这个操作，如urlib.requests 等我们可以用这些库来帮助我们实现HTTP请求操作，请求和响应都可以用类库提供的数据结构来表示，得到响应之后只需要解析数据结构中的Body部分即可，即得到网页的源代码，这样我们可以用程序来实现获取网页的过程了。

2）提取信息

获取网页源代码后，接下来就是分析网页源代码，从中提取我们想要的数据。首先，最通用的方法便是采用正则表达式提取，这是一个万能的方法，但是在构造正则表达式时比较复杂且容易出错。

另外，由于网页的结构有一定的规则，所以还有一些根据网页节点属性、CSS选择器或XPath来提取网页信息的库，如Beauiful Soup、pyquery、lxml 等。使用这些库，我们可以高效快速地从中提取网页信息，如节点的属性、文本值等。

提取信息是爬虫非常重要的部分，它可以使杂乱的数据变得条理清晰，以便我们后续处理和分析数据

3）保存数据

提取信息后，我们一般会将提取到的数据保存到某处以便后续使用。这里保存形式有多种多样，可以简单保存为TXT 文本或JSON文本，也可以保存到数据库，如MySQL、MongoDB和redis等

4）自动化程序

说自动化程序，意思是说爬虫可以代替人来完成这些操作。首先，我们手工当然可以提取这些信息，但是当量特别大或者想快速获取大量数据的话，肯定还是要借助程序。爬虫就是代替我们来完成这份爬取工作的自动化程序，它可以在抓取过程中进行各种异常处理、错误重试等操作，确保爬取持续高效地运行。

5）爬虫分类

根据使用场景，网络爬虫又可分为 通用爬虫 和 聚焦爬虫 两种.

- 通用网络爬虫，是搜索引擎抓取系统（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。

二、抓包工具：Fiddler

1.工作原理

Fiddler是强大的抓包工具，它的原理是以web代理服务器的形式进行工作的，使用的代理地址是：127.0.0.1，端口默认为8888，我们也可以通过设置进行修改。

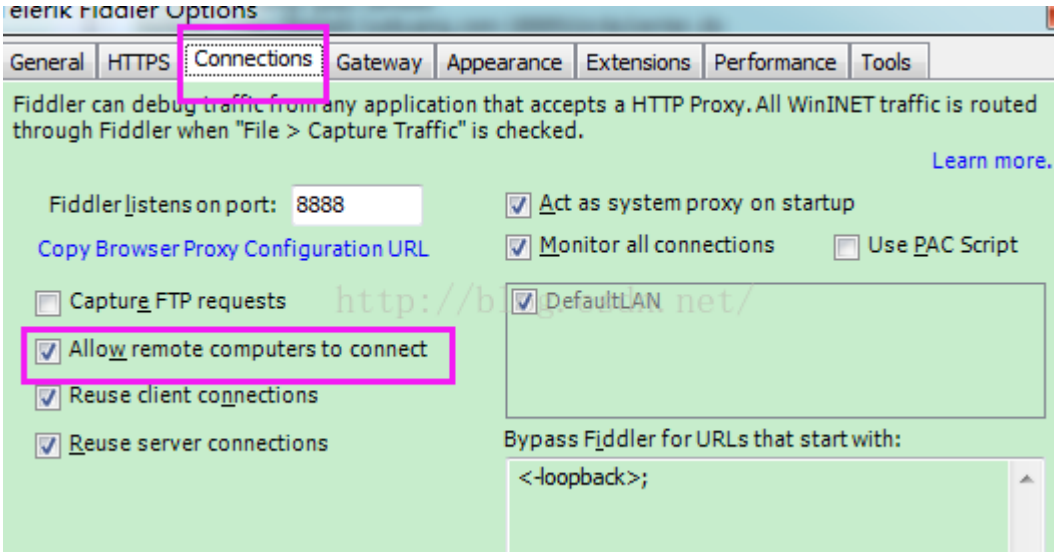
代理就是在客户端和服务器之间设置一道关卡，客户端先将请求数据发送出去后，代理服务器会将数据包进行拦截，代理服务器再冒充客户端发送数据到服务器；同理，服务器将响应数据返回，代理服务器也会将数据拦截，再返回给客户端。

Fiddler可以抓取支持http代理的任意程序的数据包，如果要抓取https会话，要先安装证书。

2.Fiddler抓取HTTPS设置

1. 启动Fiddler，打开菜单栏中的 Tools -> Telerik Fiddler Options -> HTTPS，打开“Fiddler Options”对话框。
2. 对Fiddler进行设置：
    - 选中Capture HTTPS CONNECTs (捕捉HTTPS连接)，并点击点击右侧Actions从下拉菜单中选择：Trust Root Certificate（受信任的根证书）
    - 选中Decrypt HTTPS traffic（解密HTTPS通信）
    - 另外我们要用Fiddler获取本机所有进程的HTTPS请求，所以中间的下拉菜单中选中...from all processes（从所有进程）
    - 选中下方Ignore server certificate errors（忽略服务器证书错误）

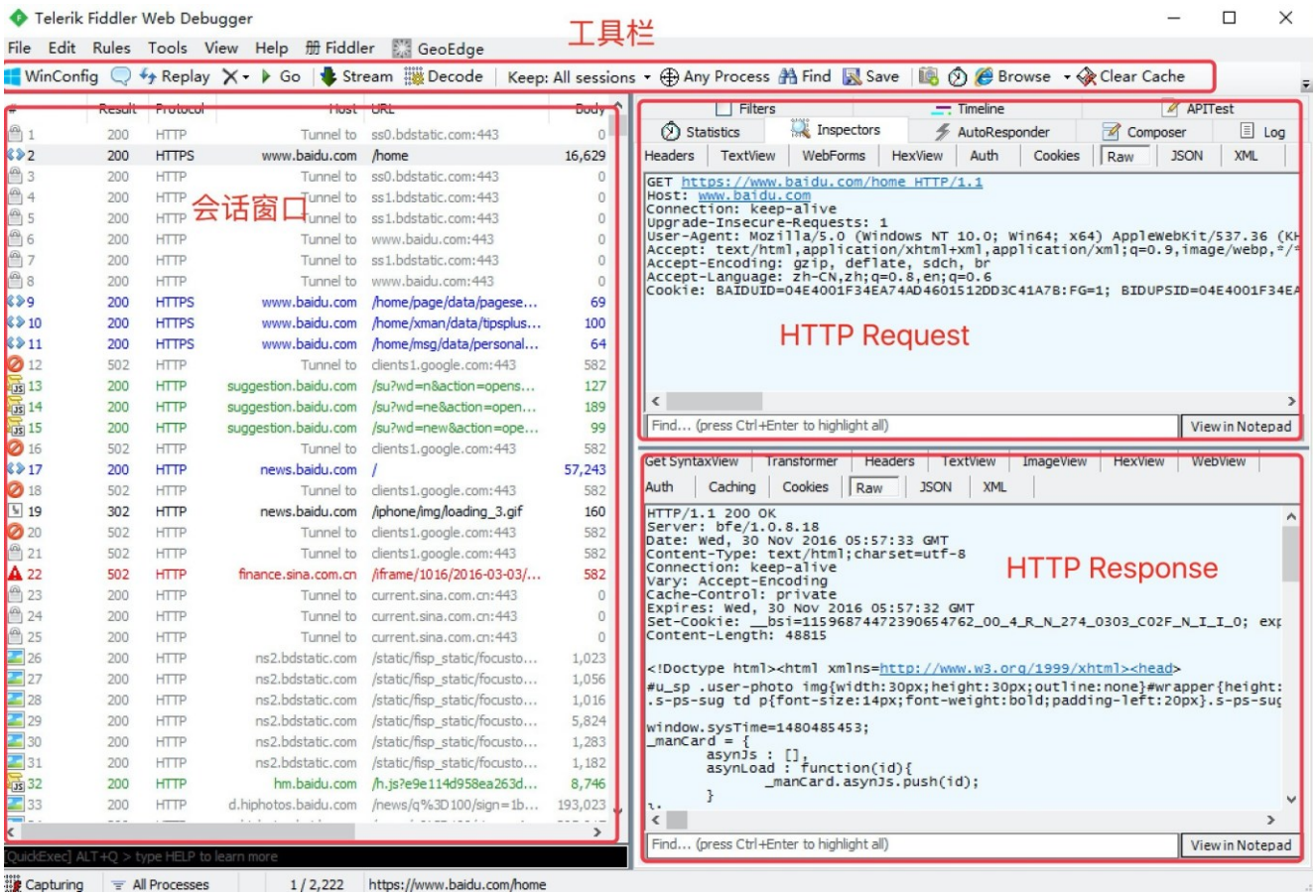
3. Fiddler 主菜单点击Tools -> Telerik Fiddler Options -> Connections，
    - 勾选allow remote computers to connect（允许远程连接）
    - 勾选Act as system proxy on startup（作为系统启动代理）
    - 默认监听端口为8888（下图Fiddler listens on port就是端口号），若端口被占用可以设置成其他的，配置好后要重新启动fiddler



4. 配置好之后重启Fiddler，使配置生效（必须）。

3.Fiddler界面

设置好后，本机HTTP通信都会经过127.0.0.1:8888代理，也就会被Fiddler拦截到。



请求 (Request) 部分详解

- Headers： 显示客户端发送到服务器的 HTTP 请求的 header，显示为一个分级视图，包含了 Web 客户端信息、Cookie、传输状态等。
- Textview： 显示 POST 请求的 body 部分为文本。
- WebForms： 显示请求的 GET 参数和 POST body 内容。
- HexView ： 用十六进制数据显示请求。
- Auth： 显示响应 header 中的 Proxy-Authorization(代理身份验证)和Authorization(授权) 信息.
- Raw ： 将整个请求显示为纯文本。
- JSON ： 显示JSON格式文件。
- XML： 如果请求的 body 是 XML 格式，就是用分级的 XML 树来显示它。

响应 (Response) 部分详解

- Transformer： 显示响应的编码信息。
- Headers： 用分级视图显示响应的 header。
- TextView： 使用文本显示相应的 body。
- ImageVies ： 如果请求是图片资源，显示响应的图片。
- HexView ： 用十六进制数据显示响应。
- WebView： 响应在 Web 浏览器中的预览效果。
- Auth： 显示响应 header 中的 Proxy-Authorization(代理身份验证) 和 Authorization(授权) 信息。
- Caching： 显示此请求的缓存信息。
- Privacy： 显示此请求的私密 (P3P) 信息。
- Raw ： 将整个响应显示为纯文本。
- JSON ： 显示JSON格式文件。
- XML： 如果响应的 body 是 XML 格式，就是用分级的 XML 树来显示它 。

三、urllib

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抓取出来。

在Python中有很多库可以用来抓取网页，在python2中有urllib和urllib2两个库来实现请求的发送，但是在python3中已经不存在urllib2了，统一为urllib，官方文档链接为：https://docs.python.org/3/library/urllib.html

Urllib是python内置的HTTP请求库，也就是不需要额外去安装了可以直接使用，他包含了以下4个模块：

- urllib.request： 请求模块
- urllib.error： 异常处理模块
- urllib.parse ： url解析模块
- urllib.robotparser： robots.txt解析模块

1. urllib的基础使用

1) urlopen

关于urllib.request.urlopen参数的介绍：

```
urllib.request.urlopen(url, data=None, [timeout, ]*,cafile=None, capath=None, cadefault=False, context=None)
```

url

先写一个简单的例子：

```
import urllib.request

response = urllib.request.urlopen('http://www.baidu.com')

print(response.read().decode('utf-8'))
```

```
<html>
<head>

    <meta http-equiv="content-type" content="text/html;charset=utf-8">
    <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}
```

实际上，如果我们在浏览器上打开百度主页， 右键选择“查看源代码”，你会发现，跟我们刚才打印出来的是一模一样。也就是说，上面的3行代码就已经帮我们把百度的首页的全部代码爬了下来。

response.read()可以获取到网页的内容，如果没有read()，将返回如下内容

```
import urllib.request

response = urllib.request.urlopen('http://www.baidu.com')
print(response)
```

```
<http.client.HTTPResponse object at 0x7ff84809f550>
```

可以发现它是一个HTTPResponse类型的对象，

主要包含read(),readinto(),getheader(name).getheader(),fileno()等方法

data

上面的例子是通过请求百度的get请求获得百度，下面使用urllib的post请求，

这里通过http://httpbin.org/post 网站演示（该网站可以作为练习使用urllib的一个站点使用，可以模拟各种请求操作）。

```
import urllib.parse
import urllib.request

data = bytes(urllib.parse.urlencode({'word': 'python'}), encoding='utf8')
print(data)

response = urllib.request.urlopen('http://httpbin.org/post', data=data)
print(response.read())
```

这里就用到urllib.parse，通过bytes(urllib.parse.urlencode())可以将post数据进行转换放到urllib.request.urlopen的data参数中。这样就完成了一次post请求。 所以如果我们添加data参数的时候就是以post请求方式请求，如果没有data参数就是get请求方式

GET和POST的区别?

- GET方式是直接以链接形式访问，链接中包含了所有的参数，服务器端用Request.QueryString获取变量的值。如果包含了密码的话是一种不安全的选择，不过你可以直观地看到自己提交了什么内容。
- POST则不会在网址上显示所有的参数，服务器端用Request.Form获取提交的数据，在Form提交的时候。但是HTML代码里如果不指定 method 属性，则默认为GET请求，Form中提交的数据将会附加在url之后，以?分开与url分开。
- 表单数据可以作为 URL 字段（method="get"）或者 HTTP POST （method="post"）的方式来发送。比如在下面的HTML代码中，表单数据将因为 （method="get"） 而附加到 URL 上：

timeout

在某些网络情况不好或者服务器端异常的情况会出现请求慢的情况，或者请求异常，所以这个时候我们需要给请求设置一个超时时间，而不是让程序一直在等待结果。

```
import urllib.request

response = urllib.request.urlopen('http://httpbin.org/get', timeout=1)

print(response.read())
```

request

在上一个例子里，urlopen()的参数就是一个url地址；

但是如果需要执行更复杂的操作，比如有很多网站为了防止程序爬虫爬网站造成网站瘫痪，会需要我们携带一些headers头部信息才能访问，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。

```
import urllib.request

## url 作为Request()方法的参数，构造并返回一个Request对象
request = urllib.request.Request('https://www.baidu.com')

## Request对象作为urlopen()方法的参数，发送给服务器并接收响应
response =urllib.request.urlopen(request)

print(response.read().decode('utf-8'))
```

上面我们说了，有很多网站不喜欢被程序（非人为访问）访问，为了防止程序爬虫爬网站造成网站瘫痪，网站会限制需要我们携带一些headers头部信息才能访问，打个很形象的例子，我和你两个人，你能直接进你家，但是我直接进你家的话就会被你爸拦下来，因为你是你爸的儿子而我不是，所以要进你家的话我需要一个身份，你和你爸说我是你朋友来你家玩，这样你爸肯定让我进了，这个身份就是所谓的User-Agent头，你带上了这个就会被允许访问。

```
import urllib.request

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

User_Agent = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36'}

## url 和 User_Agent 作为Request()方法的参数，构造并返回一个Request对象
request = urllib.request.Request(url=url,header=User_Agent)

## Request对象作为urlopen()方法的参数，发送给服务器并接收响应
response =urllib.request.urlopen(request)

print(response.read().decode('utf-8'))
```

添加请求头的第二种方式,添加更多的Header信息,这种添加方式有个好处是自己可以定义一个请求头字典，然后循环进行添加

```
import urllib.request

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

User_Agent = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36'}

request = urllib.request.Request(url=url,header=User_Agent)

request.add_header("Connection",  "keep-alive")

#   也可以通过调用Request.get_header()来查看header信息
# request.get_header(header_name="Connection")

response =urllib.request.urlopen(request)
```

```
print response.code      #可以查看响应状态码

print(response.read().decode('utf-8'))
```

2) URL解析

urlencode

这个方法可以将字典转换为url参数，

```
import urllib.request, urllib.parse

word = {"wd": "不掉发的羊驼"}

# 通过urlencode将字典键值对按URL编码转换，从而能被web服务器接受。
word_encode = urllib.parse.urlencode(word)
print(word_encode)

# 通过unquote把 URL编码字符串，转换回原先字符串。
word_str = urllib.parse.unquote(word_encode)
print(word_str)
```

一般HTTP请求提交数据，需要编码成　URL编码格式，然后做为url的一部分，或者作为参数传到Request对象中。

urlparse

```
urllib.parse.urlparse(urlstring, scheme='', allow_fragments=True)
```

该方法可以实现URl的识别和分段，看下面实例：

```
from urllib.parse import urlparse

result = urlparse("https://www.baidu.com/s?wd=python")
print(result)
```

```
ParseResult(scheme='https', netloc='www.baidu.com', path='/s', params='', query='wd=python',
fragment='')
```

将url分为6个部分，返回一个包含6个字符串项目的元组：协议、位置、路径、参数、查询、片段。

其中scheme 是协议, netloc 是域名服务器,path 相对路径 ,params是参数，query是查询的条件

urlunpars

其实功能和urlparse的功能相反，它是用于拼接，它接受的参数是一个可迭代对象，但是它的长度必须是6，否则会抛出参数数量不足或者过多的问题。

```
from urllib.parse import urlunparse

data = ['http','www.baidu.com','index.html','user','a=123','commit']
print(urlunparse(data))
```

```
http://www.baidu.com/index.html;user?a=123#commit
```

3) 异常处理

在很多时候我们用urlopen或opener.open方法发出一个请求访问页面时，如果urlopen或opener.open不能处理这个response，页面就会产生错误,比如404，500等

这里主要说的是URLError和HTTPError，以及对它们的错误处理。

URLError

URLError 产生的原因主要有：

1. 没有网络连接
2. 服务器连接失败
3. 找不到指定的服务器

我们可以用try except语句来捕获相应的异常。下面的例子里我们访问了一个不存在的域名：

```
from urllib import request,error

try:
    response = request.urlopen("http://pythonsite.com/1111.html")
except error.URLError as e:
    print(e.reason)
```

结果：

```
Not Found
```

URLError里只有一个属性：reason,即抓异常的时候只能打印错误信息，类似刚才的例子

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或opener.open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。

注意，urllib2可以为我们处理重定向的页面（也就是3开头的响应码），100-299范围的号码表示成功，所以我们只能看到400-599的错误号码。

```
from urllib import request, error

try:
    response = request.urlopen("http://pythonsite.com/1111.html")
except error.HTTPError as e:
    print(e.reason)
    print(e.code)
    print(e.headers)
except error.URLError as e:
    print(e.reason)

else:
    print("reqeust successfully")
```

```
Not Found
404
Date: Thu, 10 May 2018 02:05:36 GMT
Server: Apache
Vary: Accept-Encoding
Content-Length: 207
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

同时，e.reason其实也可以在做深入的判断，

```
import socket
from urllib import error,request

try:
    response = request.urlopen("http://www.pythonsite.com/",timeout=0.001)
except error.URLError as e:
    print(type(e.reason))
    if isinstance(e.reason,socket.timeout):
        print("time out")
```

```
<class 'socket.timeout'>
time out
```

HTTP响应状态码：

```
1xx:信息

100 Continue
服务器仅接收到部分请求，但是一旦服务器并没有拒绝该请求，客户端应该继续发送其余的请求。
101 Switching Protocols
服务器转换协议：服务器将遵从客户的请求转换到另外一种协议。
```

2xx:成功

200 OK
请求成功（其后是对GET和POST请求的应答文档）
201 Created
请求被创建完成，同时新的资源被创建。
202 Accepted
供处理的请求已被接受，但是处理未完成。
203 Non-authoritative Information
文档已经正常地返回，但一些应答头可能不正确，因为使用的是文档的拷贝。
204 No Content
没有新文档。浏览器应该继续显示原来的文档。如果用户定期地刷新页面，而Servlet可以确定用户文档足够新，这个状态代码是很有用的。
205 Reset Content
没有新文档。但浏览器应该重置它所显示的内容。用来强制浏览器清除表单输入内容。
206 Partial Content
客户发送了一个带有Range头的GET请求，服务器完成了它。

3xx:重定向

300 Multiple Choices
多重选择。链接列表。用户可以选择某链接到达目的地。最多允许五个地址。
301 Moved Permanently
所请求的页面已经转移至新的url。
302 Moved Temporarily
所请求的页面已经临时转移至新的url。
303 See Other
所请求的页面可在别的url下被找到。
304 Not Modified
未按预期修改文档。客户端有缓冲的文档并发出了一个条件性的请求（一般是提供If-Modified-Since头表示客户只想比指定日期更新的文档）。服务器告诉客户，原来缓冲的文档还可以继续使用。
305 Use Proxy
客户请求的文档应该通过Location头所指明的代理服务器提取。
306 Unused
此代码被用于前一版本。目前已不再使用，但是代码依然被保留。
307 Temporary Redirect
被请求的页面已经临时移至新的url。

4xx:客户端错误

400 Bad Request
服务器未能理解请求。
401 Unauthorized
被请求的页面需要用户名和密码。
401.1
登录失败。
401.2
服务器配置导致登录失败。

401.3
由于 ACL 对资源的限制而未获得授权。
401.4
筛选器授权失败。
401.5
ISAPI/CGI 应用程序授权失败。
401.7
访问被 Web 服务器上的 URL 授权策略拒绝。这个错误代码为 IIS 6.0 所专用。
402 Payment Required
此代码尚无法使用。
403 Forbidden
对被请求页面的访问被禁止。
403.1
执行访问被禁止。
403.2
读访问被禁止。
403.3
写访问被禁止。
403.4
要求 SSL。
403.5
要求 SSL 128。
403.6
IP 地址被拒绝。
403.7
要求客户端证书。
403.8
站点访问被拒绝。
403.9
用户数过多。
403.10
配置无效。
403.11
密码更改。
403.12
拒绝访问映射表。
403.13
客户端证书被吊销。
403.14
拒绝目录列表。
403.15
超出客户端访问许可。
403.16
客户端证书不受信任或无效。
403.17
客户端证书已过期或尚未生效。
403.18
在当前的应用程序池中不能执行所请求的 URL。这个错误代码为 IIS 6.0 所专用。
403.19
不能为这个应用程序池中的客户端执行 CGI。这个错误代码为 IIS 6.0 所专用。
403.20
Passport 登录失败。这个错误代码为 IIS 6.0 所专用。
404 Not Found

服务器无法找到被请求的页面。
404.0
没有找到文件或目录。
404.1
无法在所请求的端口上访问 Web 站点。
404.2
Web 服务扩展锁定策略阻止本请求。
404.3
MIME 映射策略阻止本请求。
405 Method Not Allowed
请求中指定的方法不被允许。
406 Not Acceptable
服务器生成的响应无法被客户端所接受。
407 Proxy Authentication Required
用户必须首先使用代理服务器进行验证，这样请求才会被处理。
408 Request Timeout
请求超出了服务器的等待时间。
409 Conflict
由于冲突，请求无法被完成。
410 Gone
被请求的页面不可用。
411 Length Required
"Content-Length" 未被定义。如果无此内容，服务器不会接受请求。
412 Precondition Failed
请求中的前提条件被服务器评估为失败。
413 Request Entity Too Large
由于所请求的实体的太大，服务器不会接受请求。
414 Request-url Too Long
由于url太长，服务器不会接受请求。当post请求被转换为带有很长的查询信息的get请求时，就会发生这种情况。
415 Unsupported Media Type
由于媒介类型不被支持，服务器不会接受请求。
416 Requested Range Not Satisfiable
服务器不能满足客户在请求中指定的Range头。
417 Expectation Failed
执行失败。
423
锁定的错误。

5xx:服务器错误

500 Internal Server Error
请求未完成。服务器遇到不可预知的情况。
500.12
应用程序正忙于在 Web 服务器上重新启动。
500.13
Web 服务器太忙。
500.15
不允许直接请求 Global.asa。
500.16
UNC 授权凭据不正确。这个错误代码为 IIS 6.0 所专用。
500.18

URL 授权存储不能打开。这个错误代码为 IIS 6.0 所专用。
500.100
内部 ASP 错误。
501 Not Implemented
请求未完成。服务器不支持所请求的功能。
502 Bad Gateway
请求未完成。服务器从上游服务器收到一个无效的响应。
502.1
CGI 应用程序超

时。 · 502.2

CGI 应用程序出错。
503 Service Unavailable
请求未完成。服务器临时过载或当机。
504 Gateway Timeout
网关超时。
505 HTTP Version Not Supported
服务器不支持请求中指明的HTTP协议版本

2. urllib的高级使用

1) Opener

opener是 urllib.request.OpenerDirector 的实例，我们之前一直都在使用的urlopen，它是一个特殊的opener（也就是模块帮我们构建好的）。

但是基本的urlopen()方法不支持代理、Cookie等其他的 HTTP/HTTPS高级功能。所以要支持这些功

能：使用相关的 Handler处理器 来创建特定功能的处理器对象；

然后通过 urllib.request.build_opener()方法使用这些处理器对象，创建自定义opener对象；

使用自定义的opener对象，调用open()方法发送请求。

注意：如果程序里所有的请求都使用自定义的opener，可以使用urllib.request.install_opener() 将自定义的opener 对象 定义为 全局opener，表示如果之后凡是调用urlopen，都将使用这个opener（根据自己的需求来选择）。

简单的自定义opener()

```
import urllib.request

# 构建一个HTTPHandler处理器对象，支持处理HTTP请求
http_handler =urllib.request.HTTPHandler()


#   调用urllib.request.build_opener()方法，创建支持处理HTTP请求的opener对象
opener = urllib.request.build_opener(http_handler)

# 构建 Request请求
request = urllib.request.Request("http://www.baidu.com/")
```

```
#   调用自定义opener对象的open()方法，发送request请求
#    （注意区别：不再通过urllib.request.urlopen()发送请求）
response = opener.open(request)

# 获取服务器响应内容
print(response.read())
```

这种方式发送请求得到的结果，和使用urllib.request.urlopen()发送HTTP/HTTPS请求得到的结果是一样的。

如果在 HTTPHandler()增加 debuglevel=1参数，还会将 Debug Log 打开，这样程序在执行的时候，会把收包和发包的报头在屏幕上自动打印出来，方便调试，有时可以省去抓包的工作。

```
# 仅需要修改的代码部分：

# 构建一个HTTPHandler 处理器对象，支持处理HTTP请求，同时开启Debug Log，debuglevel 值默认 0
http_handler = urllib.request.HTTPHandler(debuglevel=1)

# 构建一个HTTPHSandler 处理器对象，支持处理HTTPS请求，同时开启Debug Log，debuglevel 值默认 0
https_handler = urllib.request.HTTPSHandler(debuglevel=1)
```

2) ProxyHandler,代理

使用代理IP，这是爬虫/反爬虫的第二大招，通常也是最好用的。

网站它会检测某一段时间某个IP 的访问次数，如果访问次数过多，它会禁止你的访问,所以这个时候需要通过设置代理来爬取数据,这个时候我们可以通过rullIib.request.ProxyHandler()设置代理

```
import urllib.request

#构建代理Handler
proxy_handler =
    urllib.request.ProxyHandler({ 'http':
    'http://127.0.0.1:9743'
})

opener = urllib.request.build_opener(proxy_handler)

#    只有使用opener.open()方法发送请求才使用自定义的代理，而urlopen()则不使用自定义代理。
response = opener.open('http://httpbin.org/get')

#  将opener应用到全局，之后所有的，不管是opener.open()还是urlopen()    发送请求，都将使用自定义代理。
#urllib.request.install_opener(opener)
# response = urlopen(request)

print(response.read())
```

免费的开放代理获取基本没有成本，我们可以在一些代理网站上收集这些免费代理，测试后如果可以用，就把它收集起来用在爬虫上面。

如果代理IP足够多，就可以像随机获取User-Agent一样，随机选择一个代理去访问网站。

```
import urllib.request
import random

proxy_list = [
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"},
    {"http" : "124.88.67.81:80"}
]

# 随机选择一个代理
proxy = random.choice(proxy_list)

# 使用选择的代理构建代理处理器对象
httpproxy_handler = urllib.request.ProxyHandler(proxy)

opener = urllib.request.build_opener(httpproxy_handler)

request = urllib.request.Request("http://www.baidu.com/")
response = opener.open(request)

print(response.read())
```

> 但是，这些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）。
>
> 匿名度：通常情况下，使用免费代理是可以看到真实IP的，所以也叫透明代理。透明代理的请求报头有X-Forwarded-For部分，值是原始客户端的 IP。()
>
> 所以，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）。

3) cookie

HTTP是无状态的面向连接的协议，服务器和客户端的交互仅限于请求/响应过程，结束之后便断开，在下一次请求时，服务器会认为新的客户端。为了维护他们之间的链接，让服务器知道这是之前某个用户发送的请求，则必须在一个地方保存客户端的信息。

> Cookie：通过在 客户端 记录的信息确定用户的身份。 Session：通过在 服务器端 记录的信息确定用户的身份。

Cookie 是指某些网站服务器为了辨别用户身份和进行Session跟踪，而储存在用户浏览器上的文本文件，Cookie可以保持登录信息到用户下次与服务器的会话。

Cookie属性

Cookie是http请求报头中的一种属性

```
Cookie名字（Name）
Cookie的值（Value）
Cookie的过期时间（Expires/Max-Age）
Cookie作用路径（Path）
Cookie所在域名（Domain），
使用Cookie进行安全连接（Secure）。
```

> 前两个参数是Cookie应用的必要条件，另外，还包括Cookie大小（不同浏览器对Cookie个数及大小限制是有差异的）。

Cookie由变量名和值组成，根据 Netscape公司的规定，Cookie格式如下： `Set - Cookie: NAME=VALUE; Expires=DATE; Path=PATH; Domain=DOMAIN_NAME; SECURE`

Cookie应用

Cookies在爬虫方面最典型的应用是判定注册用户是否已经登录网站，cookie中保存中我们常见的登录信息，有时候爬取网站需要携带cookie信息访问或者在下一次进入此网站时保留用户信息简化登录或其他验证过程。

```
import urllib.request

# 1.构建一个已经登录过的用户的headers信息
headers = {
    "Host":"www.baidu.com",
    "Connection":"keep-alive",
    "Cache-Control": "max-age=0"
    "Upgrade-Insecure-Requests": "1"
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/66.0.3359.139 Safari/537.36"
```

```
    "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"

    #  重点：这个Cookie是一个保存了用户登录状态的Cookie
    "Cookie": "BAIDUID=D1749A6566F1233FA8124AE8AD791F0D:FG=1;
BIDUPSID=D1749A6566F1233FA8124AE8AD791F0D; PSTM=1523859359; MCITY=-257%3A; BD_UPN=12314753;
BDUSS=9yQmhVbm5PalZCZmFnUFVoc35VY2VqSWs5NFQ4UEV1SEZjLXZLdWQ4RUMzUWhiQVFBQUFBJCQAAAAAAAAAAAEAAAC1
UD4D0P7M7NTC07AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAJQ4VoC
UOFaem; ispeed_lsm=2; BD_CK_SAM=1; PSINO=7; BD_HOME=1; H_PS_PSSID=1435_21118_22160; sug=3;
sugstore=0; ORIGIN=0; bdime=21110"
}

# 2. 通过headers里的报头信息（主要是Cookie信息），构建Request对象
req = urllib.request.Request("http://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 = urllib.request.urlopen(req)

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

4) HTTPCookiProcessor

我们一般用的是http.cookijar，用于获取cookie以及存储cookie

```
import http.cookiejar, urllib.request

# 声明一个cookijar对象
cookie = http.cookiejar.CookieJar()

#   利用HTTPCookieProcessor构建一个handler创建cookie处理器对象
handler = urllib.request.HTTPCookieProcessor(cookie)

# 构建Opener
opener = urllib.request.build_opener(handler)

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

for item in cookie:
    print(item.name+"="+item.value)
```

在上面的例子中我们将Cookie保存到cookiejar对象中，然后打印出了cookie中的值，也就是访问百度首页的Cookie值.

同时cookie可以写入到文件中保存，有两种方式http.cookiejar.MozillaCookieJar和http.cookiejar.LWPCookieJar()，当然你自己用哪种方式都可以

```
import http.cookiejar, urllib.request

# 保存cookie的本地磁盘文件名
filename = "cookie.txt"

#   声明一个LWPCookieJar(有save实现)对象实例来保存cookie，之后写入文件
# cookie = http.cookiejar.LWPCookieJar(filename)

#   声明一个MozillaCookieJar(有save实现)对象实例来保存cookie，之后写入文件
cookie = http.cookiejar.MozillaCookieJar(filename)

#   使用HTTPCookieProcessor()来创建cookie处理器对象
handler = urllib.request.HTTPCookieProcessor(cookie)

opener = urllib.request.build_opener(handler)

response = opener.open('http://www.baidu.com')

cookie.save(ignore_discard=True,   ignore_expires=True)
```

同样的如果想要通过获取文件中的cookie做为请求的一部分去访问，可以通过load方式获取

```
import http.cookiejar, urllib.request

#创建一个LWPCookieJar(有load实现)对象实例
cookie =http.cookiejar.LWPCookieJar()

# 从文件中读取cookie内容到变量
cookie.load('cookie.txt', ignore_discard=True, ignore_expires=True)

#   使用HTTPCookieProcessor()来创建cookie处理器对象
handler = urllib.request.HTTPCookieProcessor(cookie)

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

response =opener.open('http://www.baidu.com')

print(response.read().decode('utf-8'))
```