# 正则表达式

## 常用匹配规则

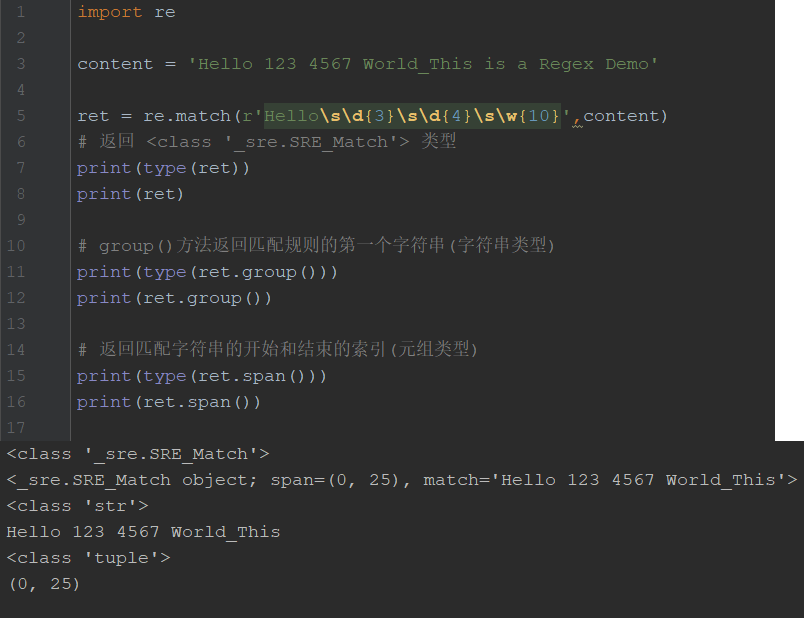
|  |  |
| --- | --- |
| 修饰符 | 描述 |
| \w | 匹配字母、数字及下划线 |
| \W | 匹配不是字母、数字及下划线的字符 |
| \s | 匹配任意空白字符，等价于[\t\n\r\f] |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于[0-9] |
| \D | 匹配任意非数字的字符 |
| \A | 匹配字符串开头 |
| \Z | 匹配字符串结尾，如果存在换行，只匹配换行前的结束字符串 |
| \z | 匹配字符串结尾，如果存在换行，同时还会匹配换行符 |
| \G | 匹配最后匹配完成的位置 |
| \n | 匹配一个换行符 |
| \t | 匹配一个制表符 |
| ^ | 匹配以一行字符串的开头 |
| $ | 匹配以一行字符串结尾 |
| **.** | 匹配任意字符，除了换行符,当re.DOTALL或re.S标记被指定时,则可以匹配包含换行符的任意字符 |
| […] | 用来表示一组字符，单独列出，比如[amk]匹配a、m、或k |
| [^…] | 不在[]中的字符，比如[^abc]匹配除了a、b、c之外的字符 |
| \* | 匹配0个或多个表达式 |
| + | 匹配1个或多个表达式 |
| ? | 匹配0个或多个前面正则表达式定义的片段,非贪婪模式 |
| {n} | 精确匹配n个前面的表达式 |
| {n,m} | 精确匹配n到m次由前面正则表达式定义的片段,贪婪模式 |
| a|b | 匹配a或b |
| ( ) | 匹配括号内的表达式,也表示一个组 |

## match()方法

match()方法会尝试从字符串的其实位置匹配正则表达式，如果匹配，就返回匹配成功的结果，如果不匹配，就返回None

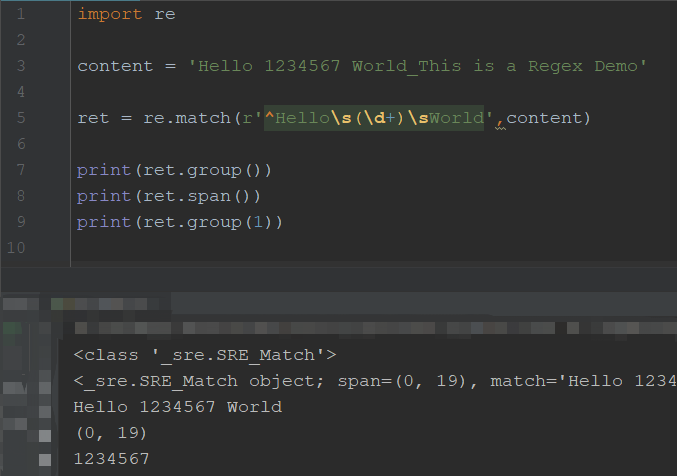
（即从字符串的开头开始匹配，一旦开头匹配不成功，那么整个匹配就失败了）

它更适合用来检测某个字符串是否符合某个正则表达式的规则



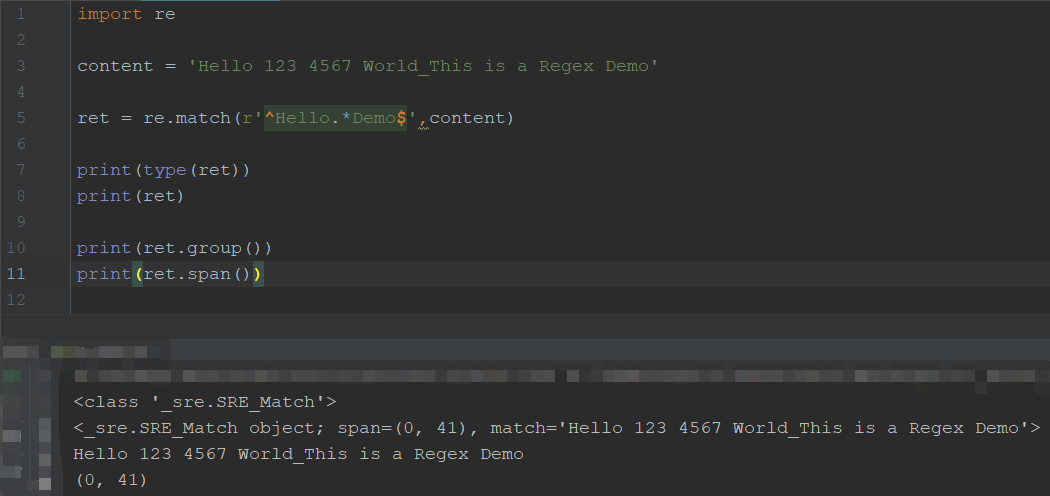
### 1.匹配目标

使用()来精准匹配目标



### 2.通用匹配

使用 **.\*** 来实现通用匹配

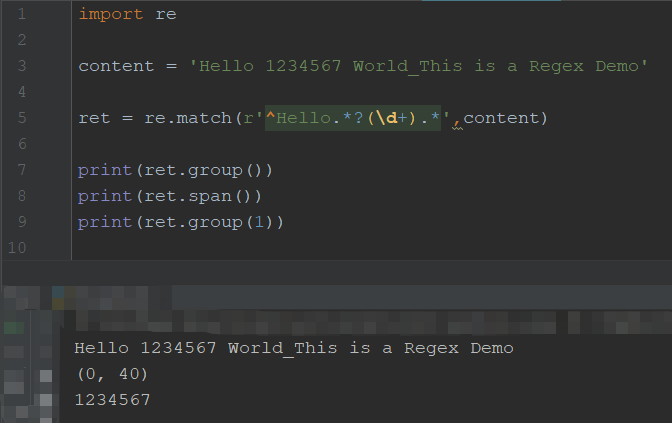


### 3.贪婪与非贪婪

贪婪匹配：尽可能多的匹配字符 **.\***



非贪婪匹配：尽可能匹配少的字符 **.\*?**

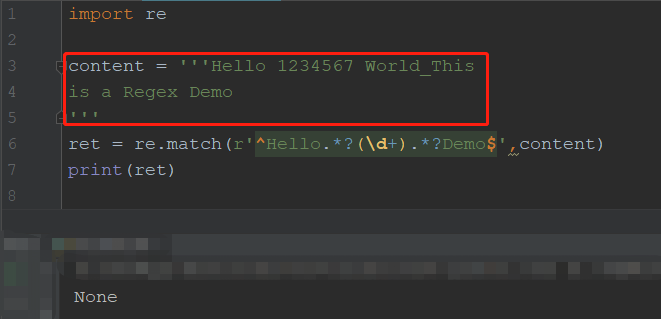


如果匹配的结果在字符串结尾, **.\*?** 可能匹配不到内容了，因为它会匹配尽可能少的内容

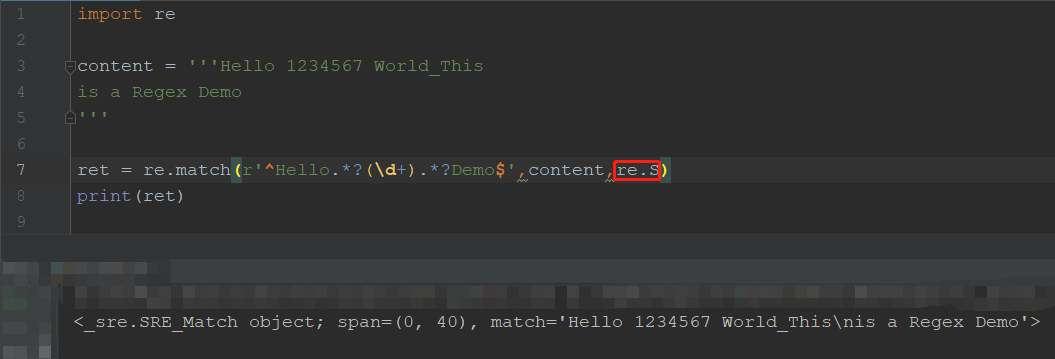


### 4.修饰符

字符串中加上换行符，正则表达式还是一样，结果返回None



需要加上修饰符，修改如下



|  |  |
| --- | --- |
| 修饰符 | 描述 |
| **re.I** | 使匹配对大小写不敏感（常用） |
| **re.L** | 做本地化识别匹配 |
| **re.M** | 多行匹配，影响^和$ |
| **re.S** | 使 **.** 匹配换行在内的所有字符（常用） |
| **re.U** | 根据Unicode字符集解析字符，这个标志影响\w、\W、\b、和 \B |
| **re.X** | 该标志通过给予你更灵活的格式以便你将正则表达式写的更易于理解 |

### 5.转义匹配

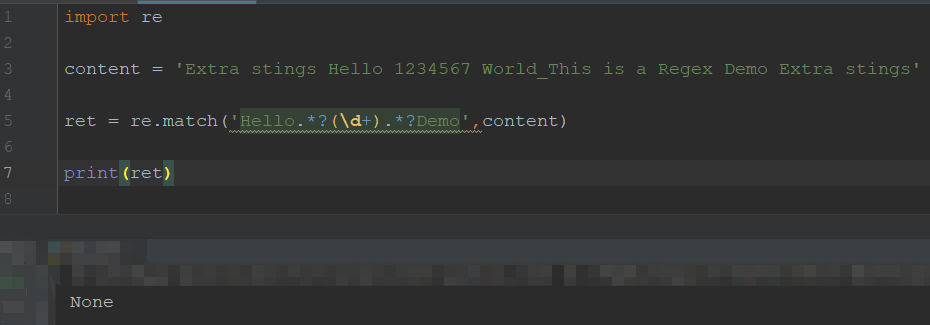
**.**可以匹配除换行符意外的任意字符，但如果目标字符串里面含有**.**那该怎么办



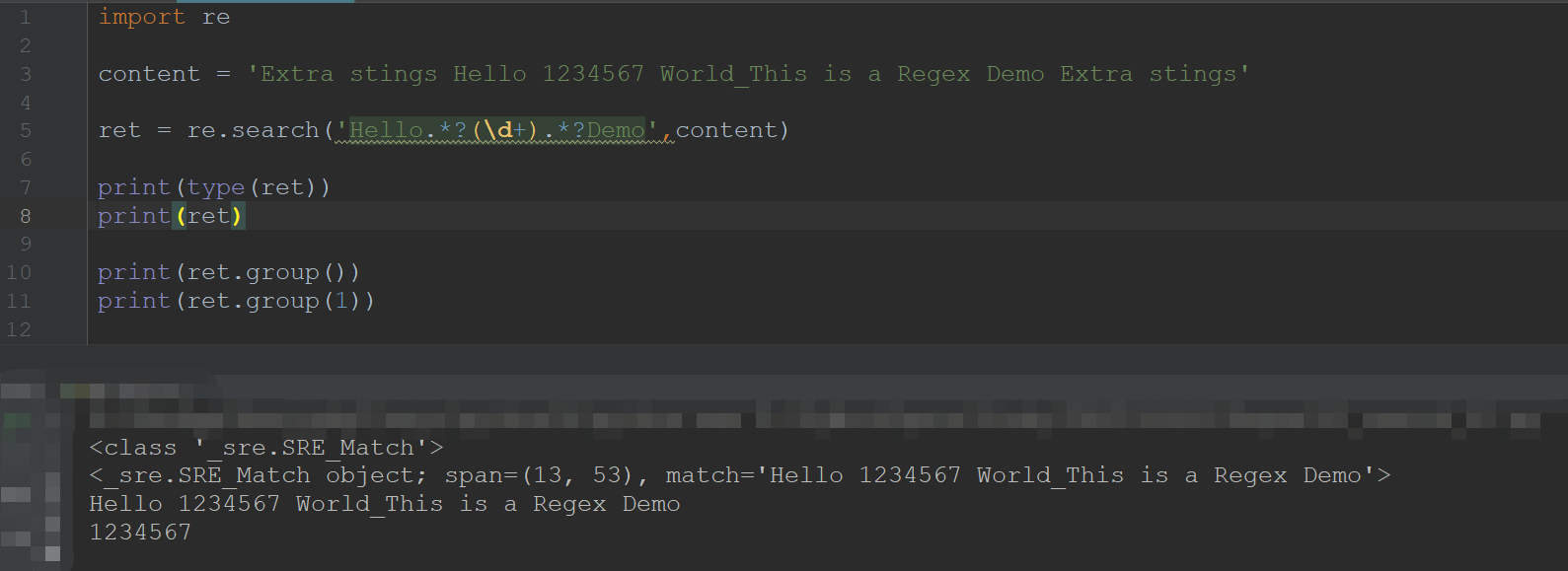
## search()方法

它在匹配时会扫描整个字符串，然后返回第一个成功匹配的结果，也就是说正则表达式可以是字符串的一部分，在匹配时，search()方法会依次扫描字符串，直到找到第一个符合规则的字符串,然后返回匹配内容，如果搜索完了还没找到，就返回None

match()方法匹配，如下



search()方法，如下



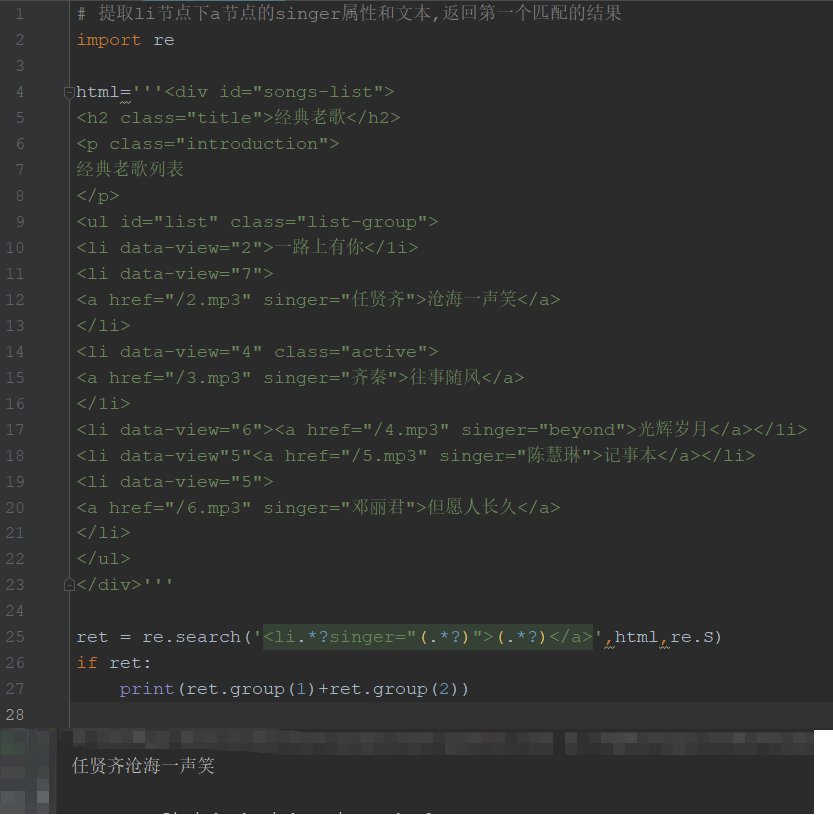
**以下有一段待匹配的HTML文本**

html='''<div id="songs-list">  
<h2 class="title">经典老歌</h2>  
<p class="introduction">  
经典老歌列表  
</p>  
<ul id="list" class="list-group">  
<li data-view="2">一路上有你</li>  
<li data-view="7">  
<a href="/2.mp3" singer="任贤齐">沧海一声笑</a>  
</li>  
<li data-view="4" class="active"><a href="/3.mp3" singer="齐秦">往事随风</a></li>  
<li data-view="6"><a href="/4.mp3" singer="beyond">光辉岁月</a></li>  
<li data-view="5">  
<a href="/5.mp3" singer="陈慧琳">记事本</a>  
</li>  
<li data-view="5">  
<a href="/6.mp3" singer="邓丽君">但愿人长久</a>  
</li>  
</ul>  
</div>'''

提取第三个li节点下a节点的singer属性和文本



提取li节点下a节点的singer属性和文本,返回第一个匹配的结果



## findall()方法

该方法会搜索整个字符串，然后返回匹配正则表达式的**所有内容，返回列表数据类型，**可能是元组列表或字符串列表

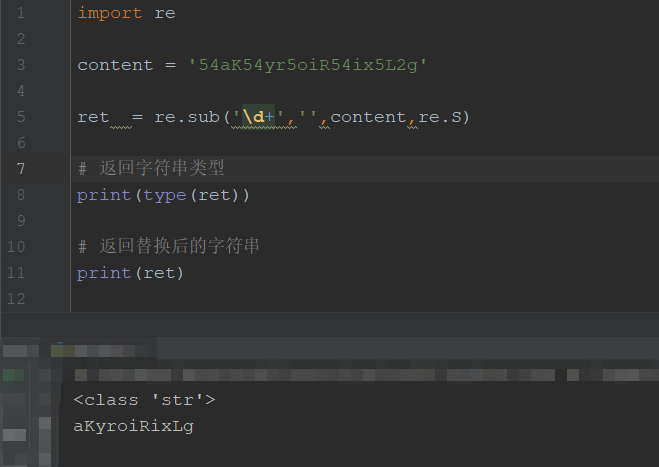


如果只需要获取一个内容可以用search方法，当需要提取多个内容时，可以用findall方法

## sub()方法

修改或者替换文本，比如想要把一串文本中的所有数字都去掉，如果只用字符串的replace()方法，那就太繁琐了，这时可以借助sub() 方法。sub方法不支持修饰符 re.S….

将一串文本中的数字替换成空

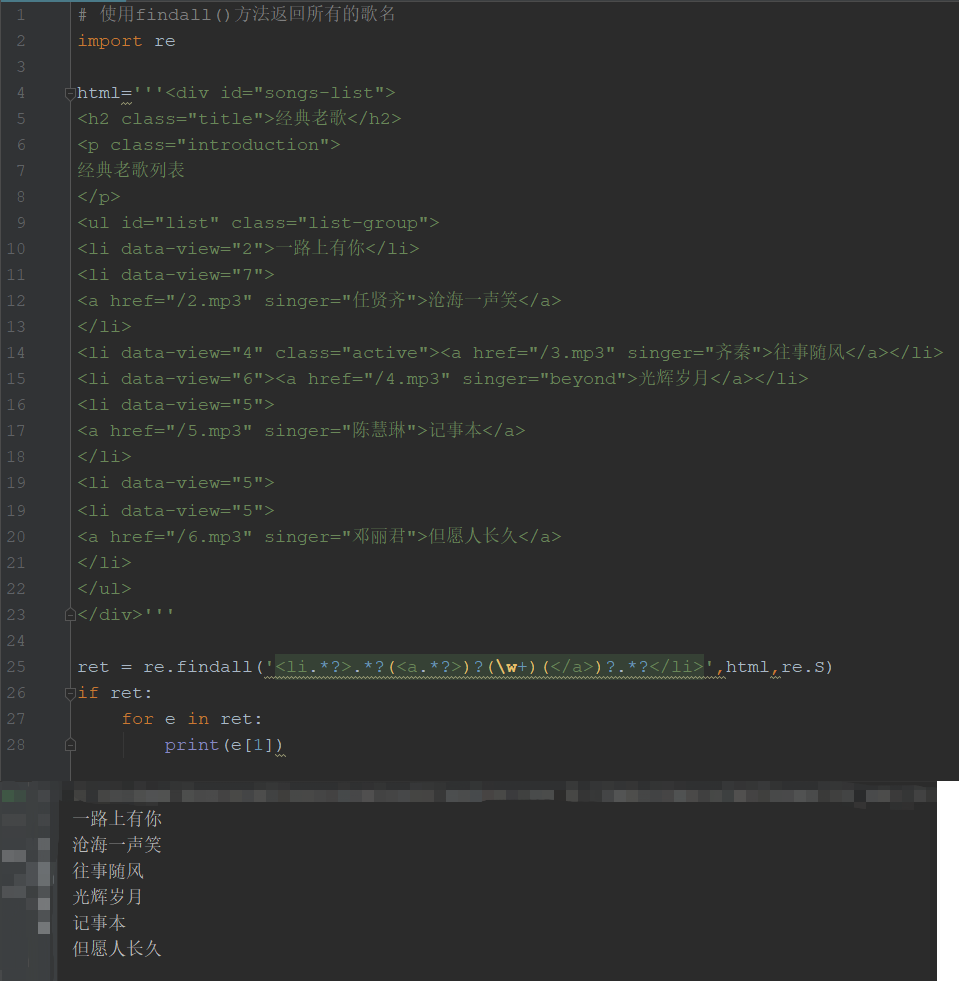


**注意：遇上换行符，sub可以使用修饰符，但必须指定参数名flags=re.S，否则将无效**

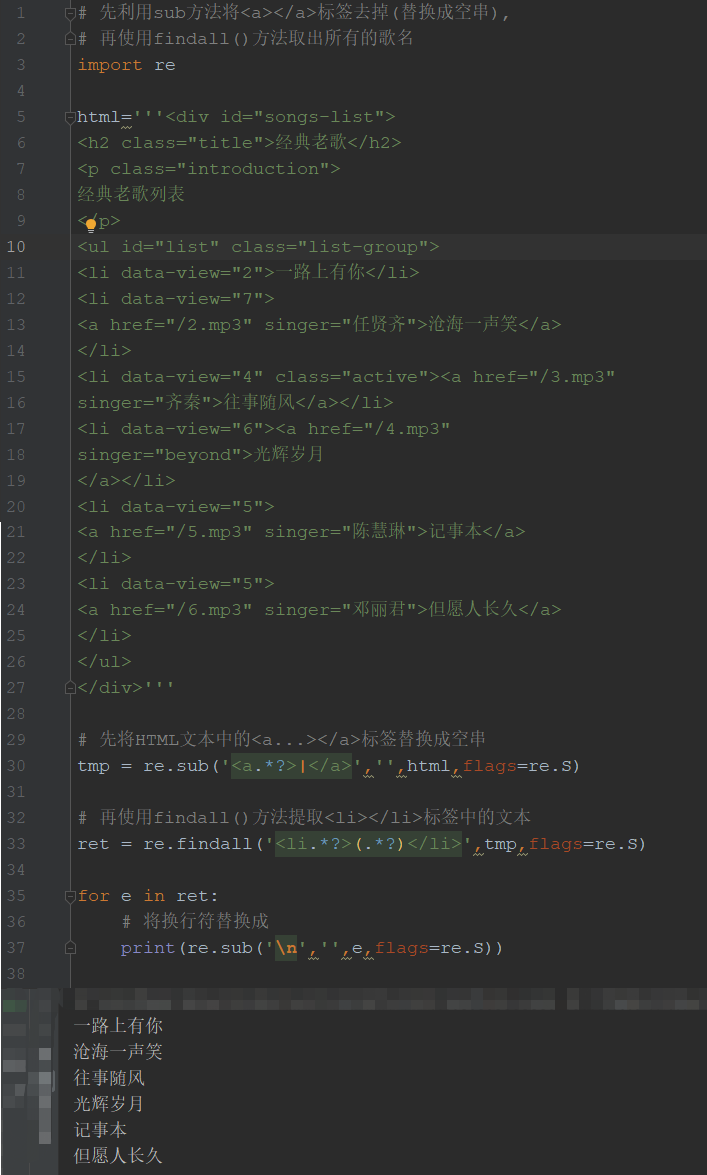
**如下：**

****

获取以上HTML文本中所有的歌名  
使用findall()方法

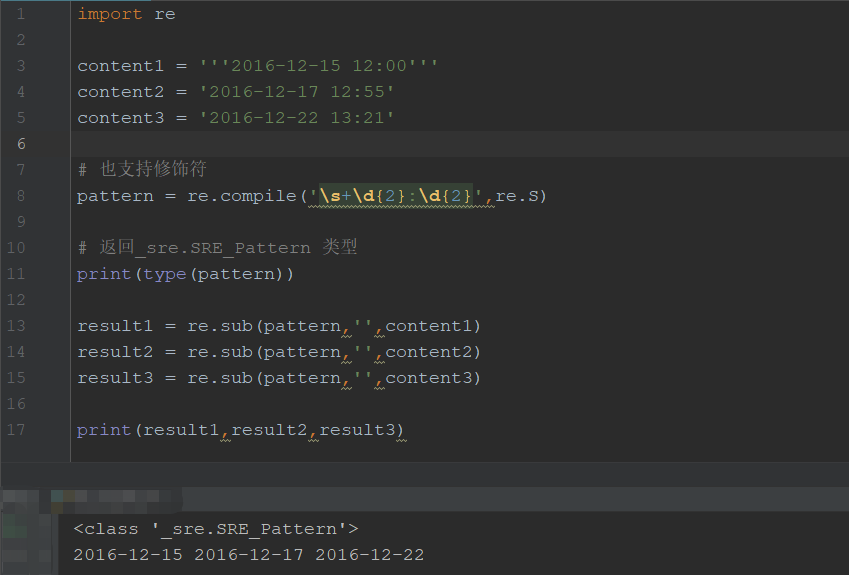


使用sub()方法



## compile()方法

这个方法可以将正则字符串编译成正则表达式对象，以便在后面的匹配中**复用**



# urllib库的使用

官方文档地址

<https://docs.python.org/3.5/library/>

它是Python内置的HTTP请求库，（即无需额外安装即可使用），包含4个模块：request、error、parse、robotparser模块

## request模块

它是基本的HTTP请求模块，用来模拟发送请求（常用）

<https://docs.python.org/3.5/library/http.client.html#http.client.HTTPResponse.readinto>

### urlopen方法()

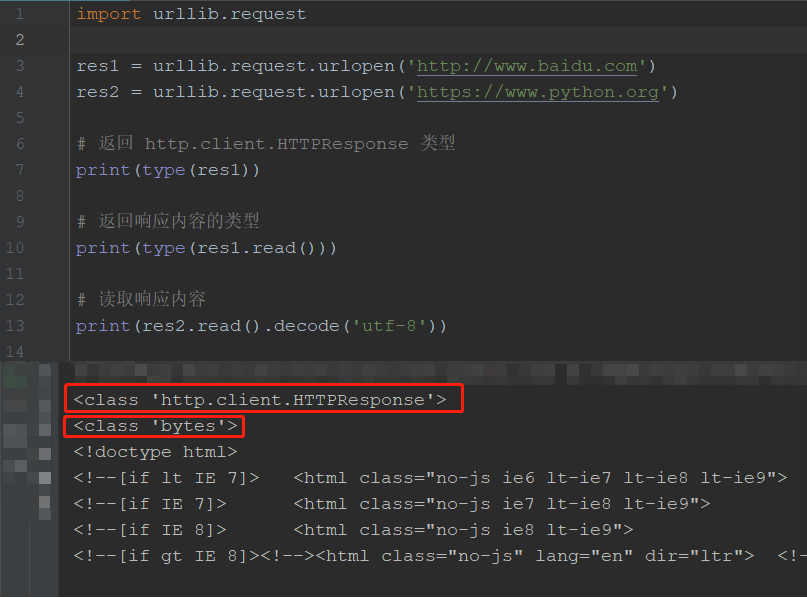
返回 **http.client.HTTPResponse** 类型的对象

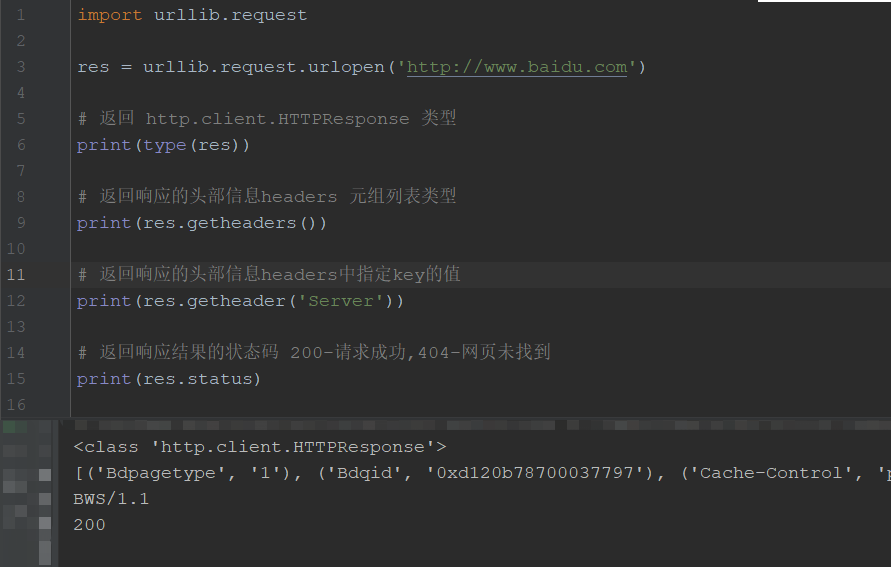
主要方法**read()、readinto()、getheader()、getheaders()、fileno()**方法、

以及**msg、status、reason、version、debuglevel、closed** 属性

使用read()方法后，后续就不可再调用read方法了（重复读取），否则将返回None

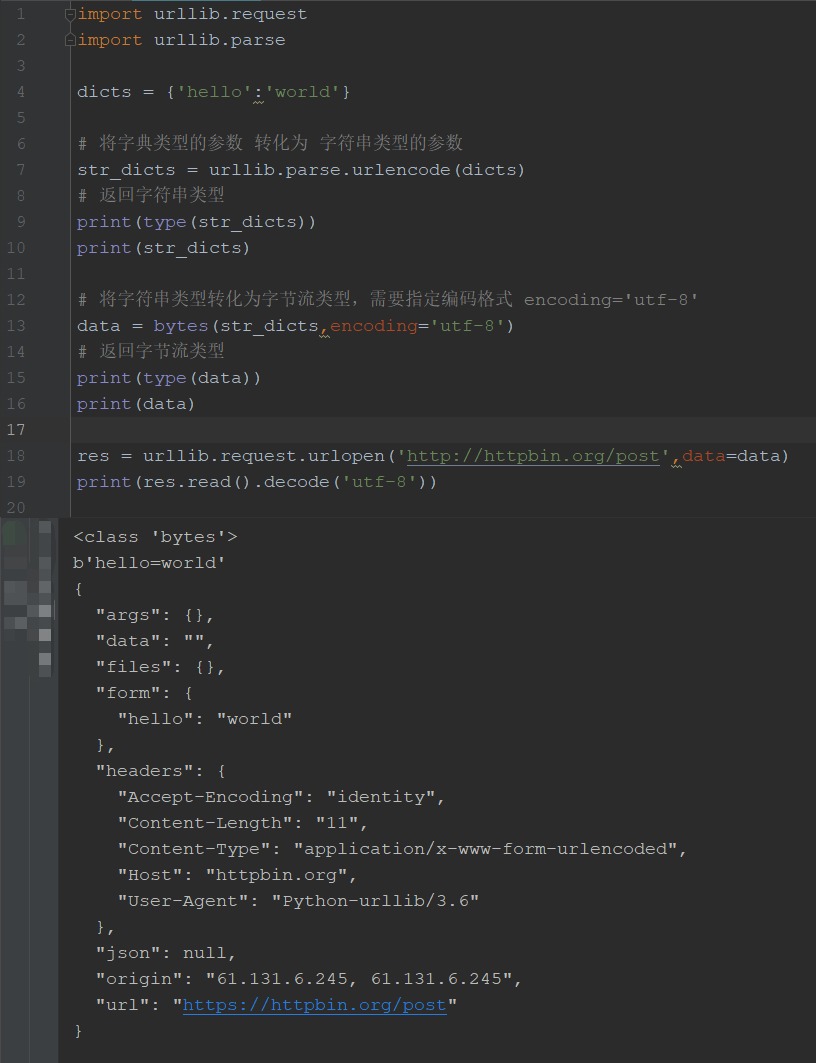
read()方法返回网页的内容，类型为Byte类型





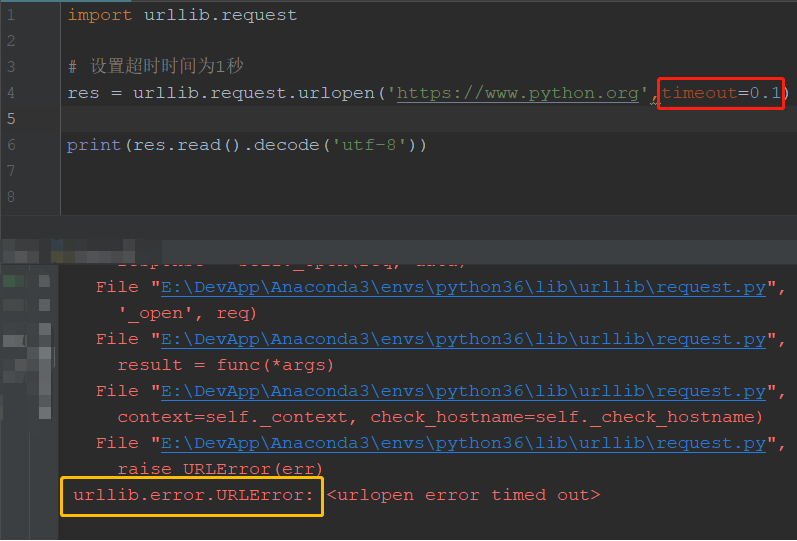
#### data参数

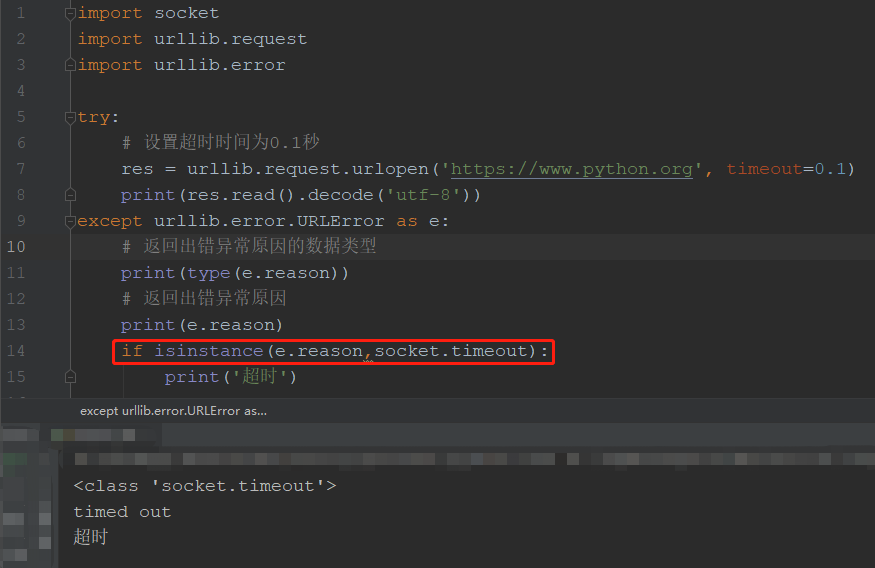
data参数是可选的，如果要添加该参数，需要使用bytes()方法将参数转化为字节流格式的内容，即bytes类型



#### timeout参数

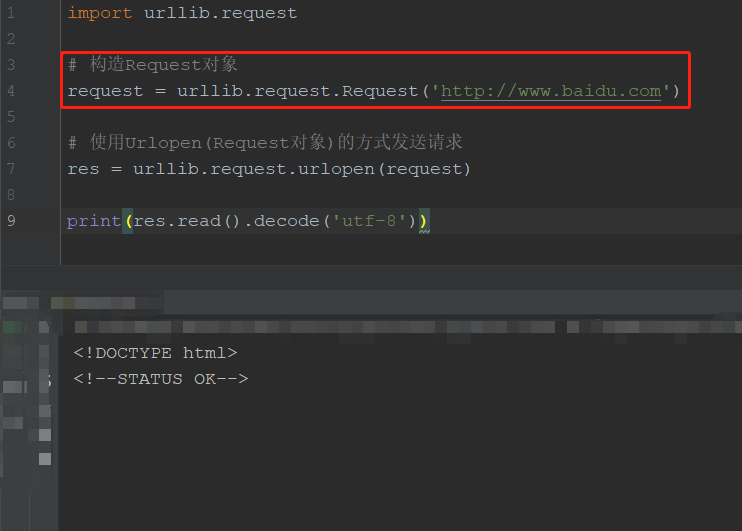
用于设置超时时间，单位为秒，如果请求超出设置的这个时间，还没有得到响应，就会抛出异常，如果不指定会使用全局默认时间，它支持Http、Https、FTP请求

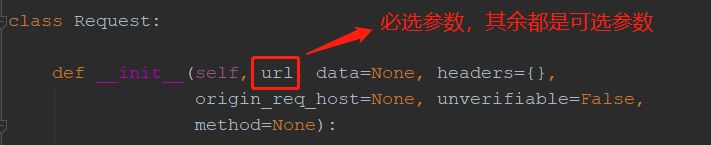




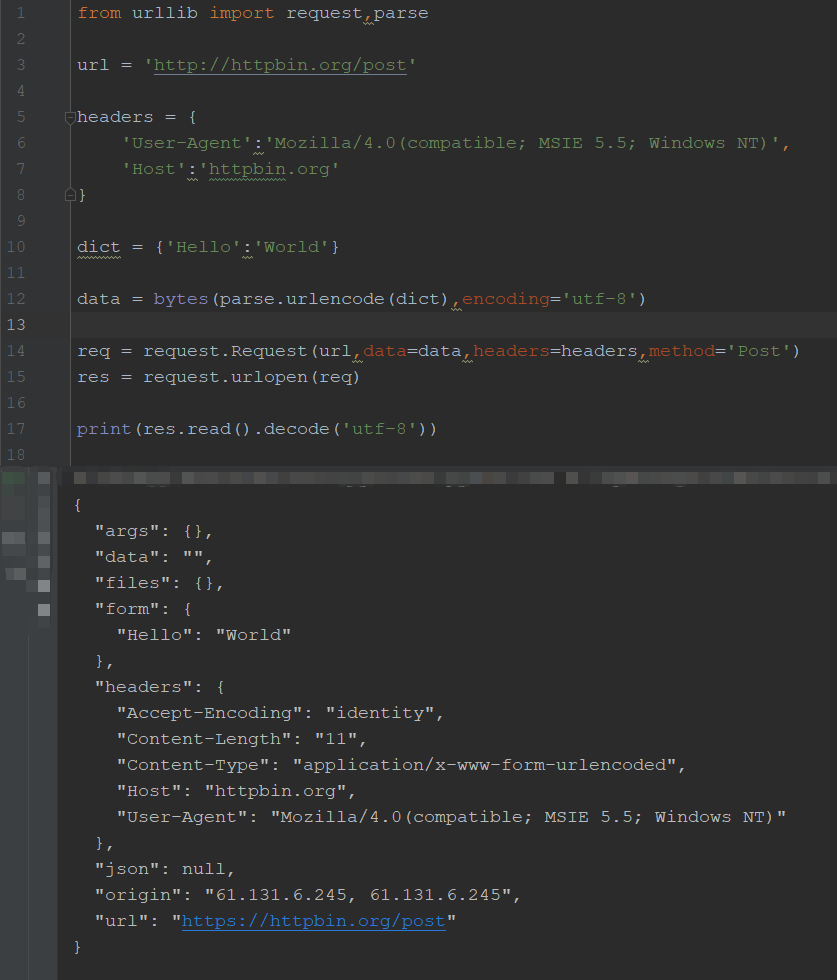
### Request类

以下先来感受下Request的用法

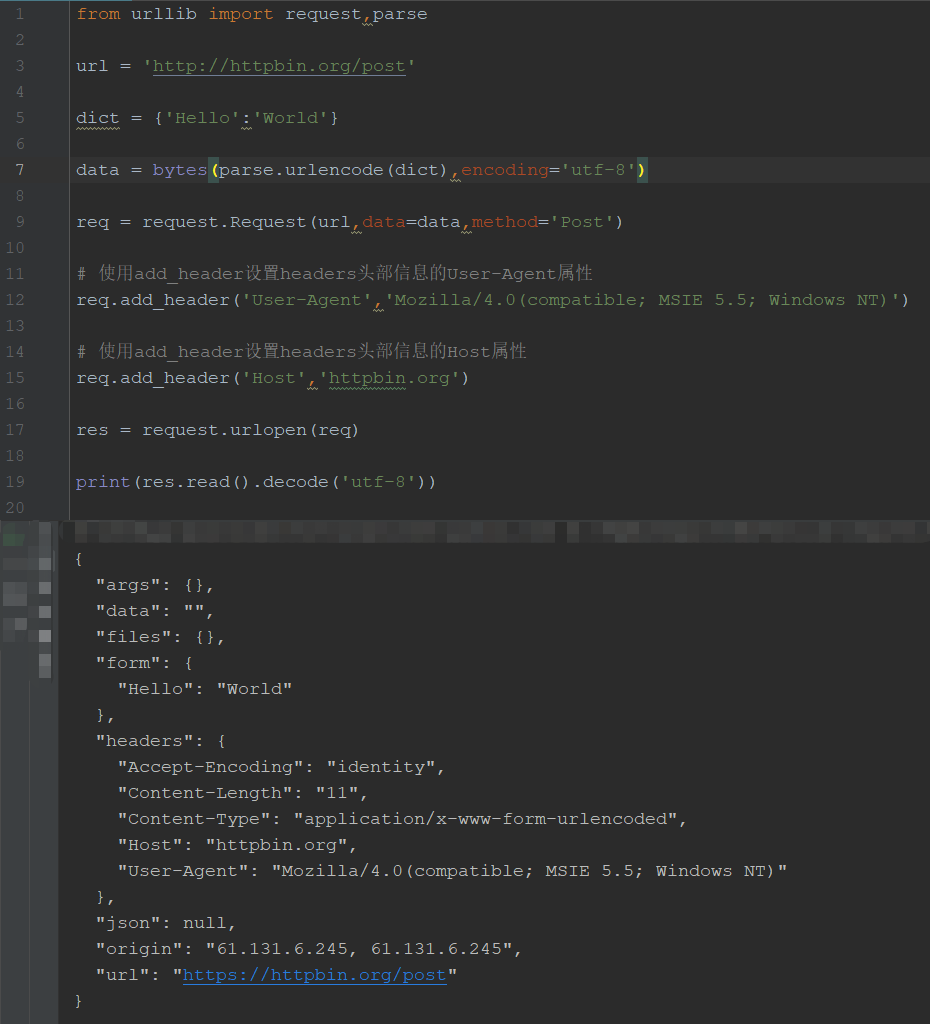




第1个参数url是必选参数，其余都是可选参数



headers头部信息参数也可以用add\_header()方法来添加



### 高级用法

在上面的过程中，我们虽然可以构造请求，但是对于一些高级的操作（比如Cookies处理、代理设置等），那该怎么办？

接下来，就需要强大的工具**Handler**登场了，我们可以把它理解为各种处理器，有专门处理登录验证的、有处理Cookies的，有处理代理设置的，利用他们我们几乎可以做到Http请求中的所有事情

#### BaseHandler类

它是其他Handler的父类，它提供了最基本的方法，例如default\_open()、default\_request()方法

<https://docs.python.org/3/library/urllib.request.html#urllib.request.BaseHandler>

有各种Handler子类继承BaseHandler类，如下：

**HTTPDefaultErrorHandler**：用于处理HTTP响应错误，错误会抛出HTTPError类型异常

**HTTPRedirectHandler**：用于处理重定向

**HTTPCookiesProcessor**：用于处理Cookies

**ProxyHandler**：用于设置代理，默认代理为空

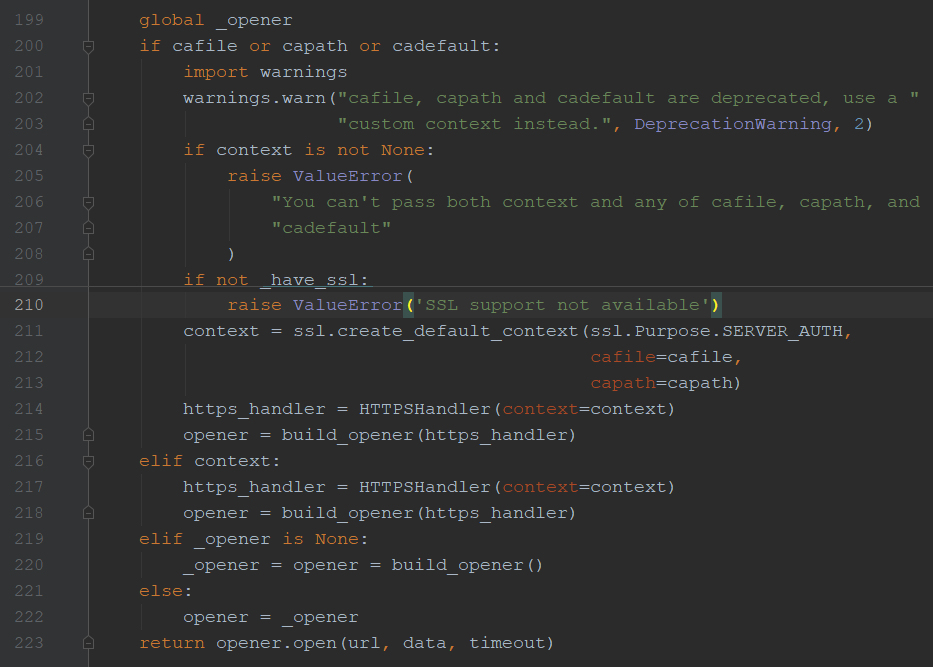
**HTTPPasswordMgr**：用于管理密码，它维护了用户名和密码的表

**HTTPBasicAuthHandler**：用于管理认证，如果一个链接打开时需要认证，那么可以用它来解决认证问题

#### OpenerDirector类

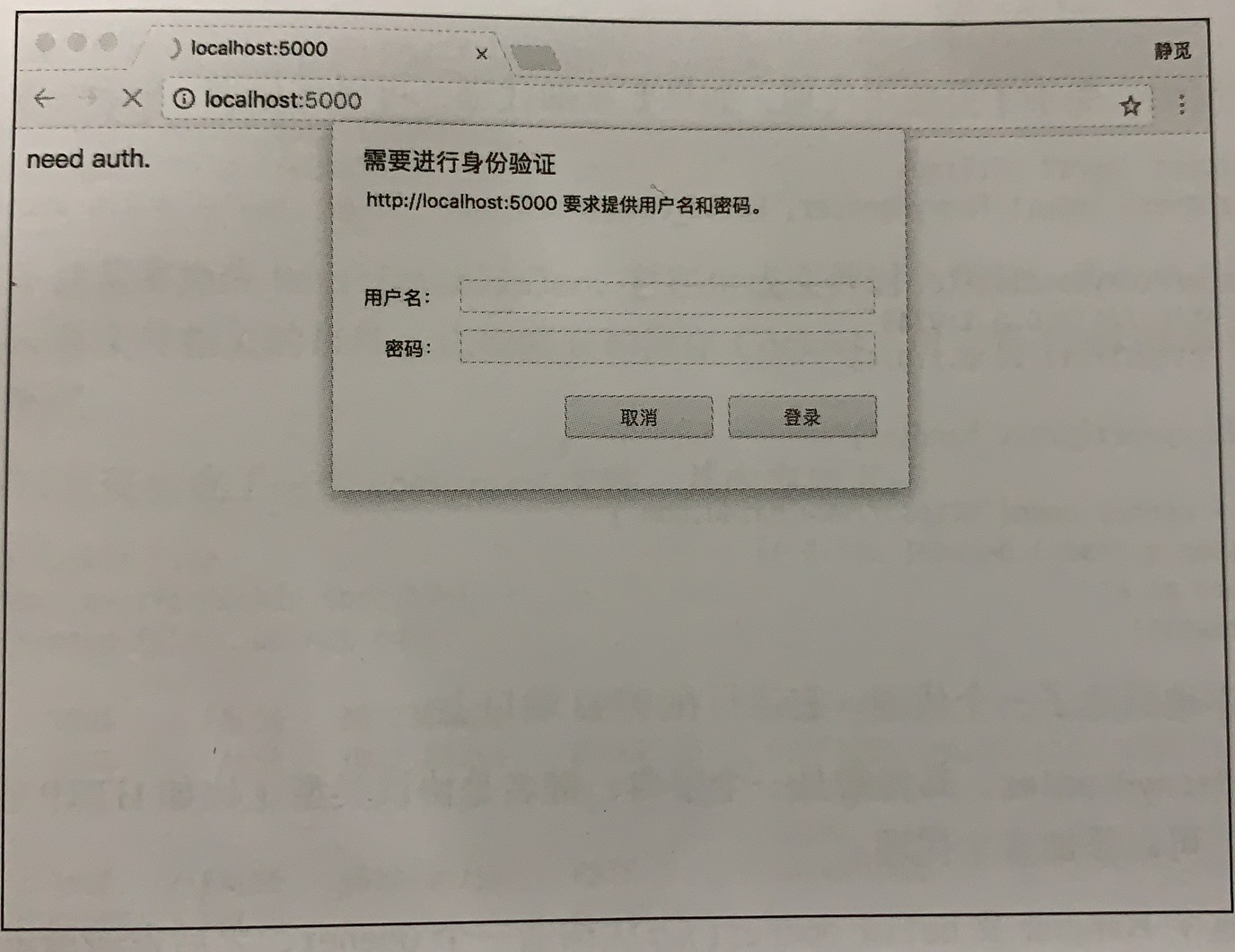
我们也可以称为Opener，urlopen()这个方法实际上就是urllib给我们提供的一个Opener。

可以查看源码代码片段

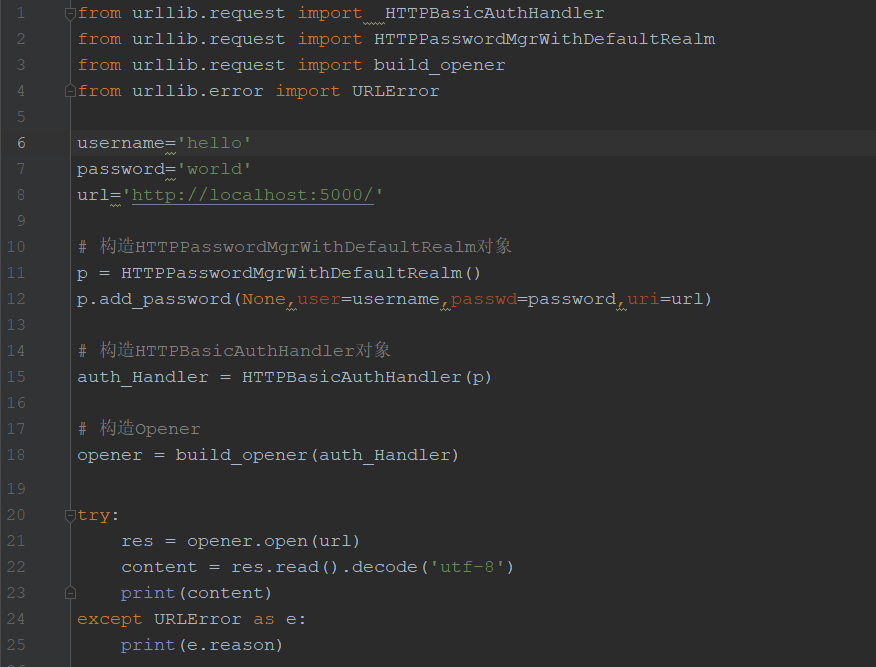


Opener可以使用Open()方法，返回的类型和urlopen()如出一辙，它和Handler有什么关系？**就是利用Handler来构建Opener**

##### 验证



这种情况可以借助HTTPBasicAuthHandler类来构造Opener，代码如下：



##### 代理

做爬虫的时候，免不了要使用代理，如果要添加代理，可以这样做：

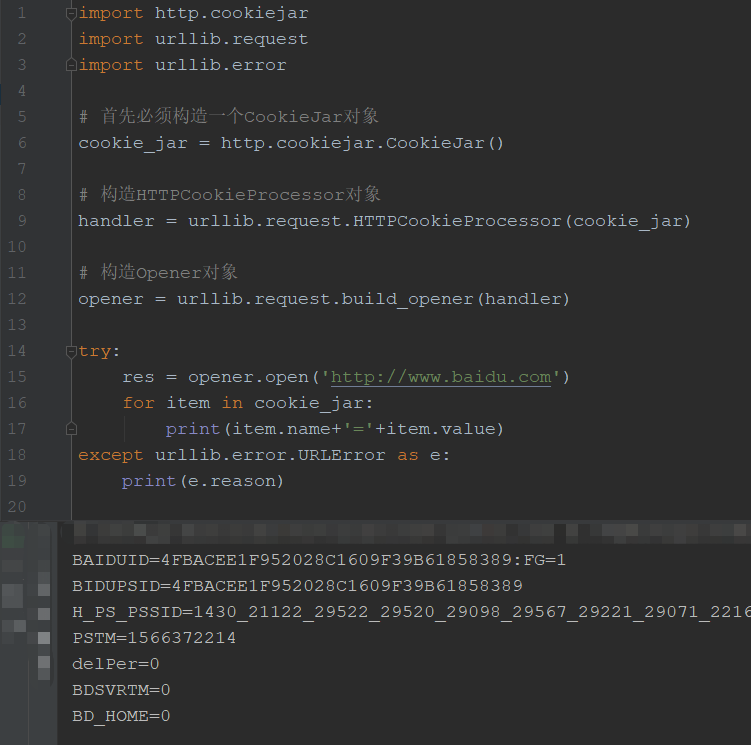


ProxyHandler其参数是一个字典，键名是协议类型(比如HTTP或者HTTPS)，键值是代理链接，可以添加多个代理

##### Cookies

###### CookieJar对象

下面先来看看如何将网站的Cookies获取下来，相关代码如下

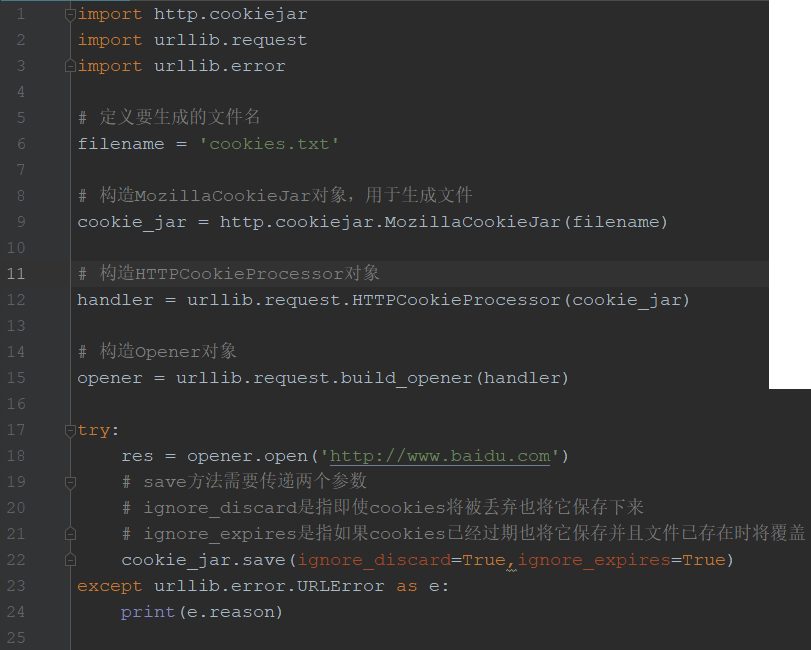


###### MozillaCookieJar对象

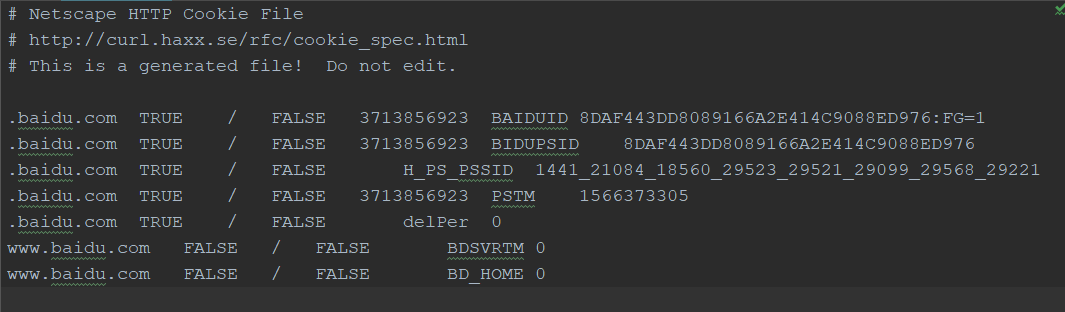
它是CookieJar的子类，可以用来处理Cookies和文件的相关事件，比如读取和保存Cookies，可以将Cookies保存呈Mozilla型浏览器的Cookies格式

**ignore\_discard**是指即使cookies将被丢弃也将它保存下来

**ignore\_expires**是指如果cookies已经过期也将它保存并且文件已存在时将覆盖

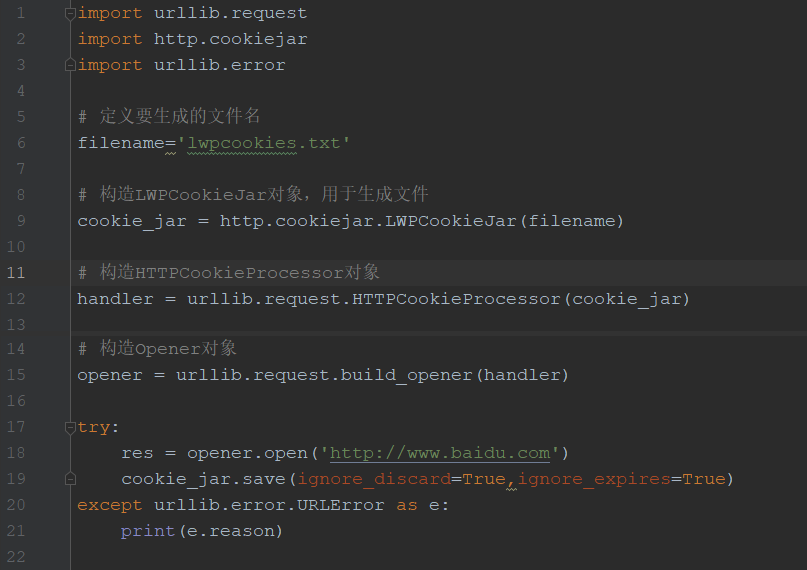


运行后生成的cookies.txt文件格式如下：

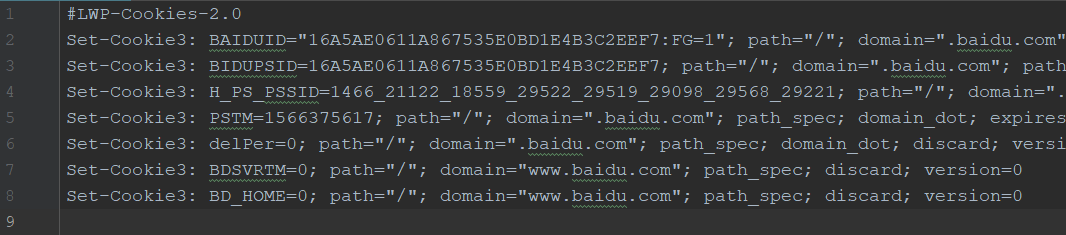


###### LWPCookieJar对象

LWPCookieJar同样可以读取和保存Cookies，但是保存的格式和MozillaCookieJar不一样，它会保存成libwww-perl(LWP)格式的Cookies文件



运行后生成的lwp格式的cookies.txt文件格式如下：



生成cookies文件后，如何从文件中读取并利用呢？

以下以LWPCookieJar格式文件为例，相关代码如下



可以看到这里调用load()方法来读取cookies文件，获取到了cookies文件的内容

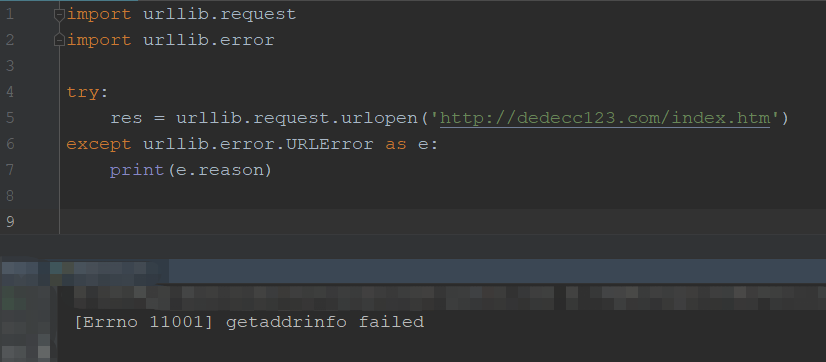
## error模块

异常处理模块（常用），urllib的error模块定义了由urllib的request模块产生的异常。

### URLError类

URLError类来自urllib库的error模块，它继承自OSError类，是error异常模块的基类，由request模块产生的异常都可以通过这个类来捕获异常。

它有一个属性**reason**，即返回错误的原因。相关代码如下：



### HTTPError类

它是URLError类的子类，专门处理HTTP请求错误，比如认证失败等，它有3个属性：

#### code

返回HTTP状态码，比如404表示网页不存在，500表示服务器内部错误等

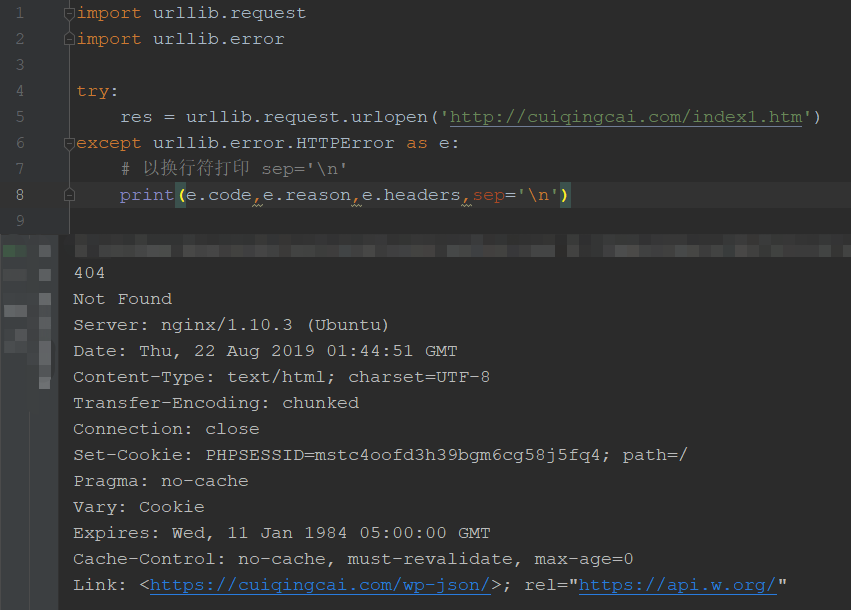
#### reason

同父类一样，用于返回错误的原因

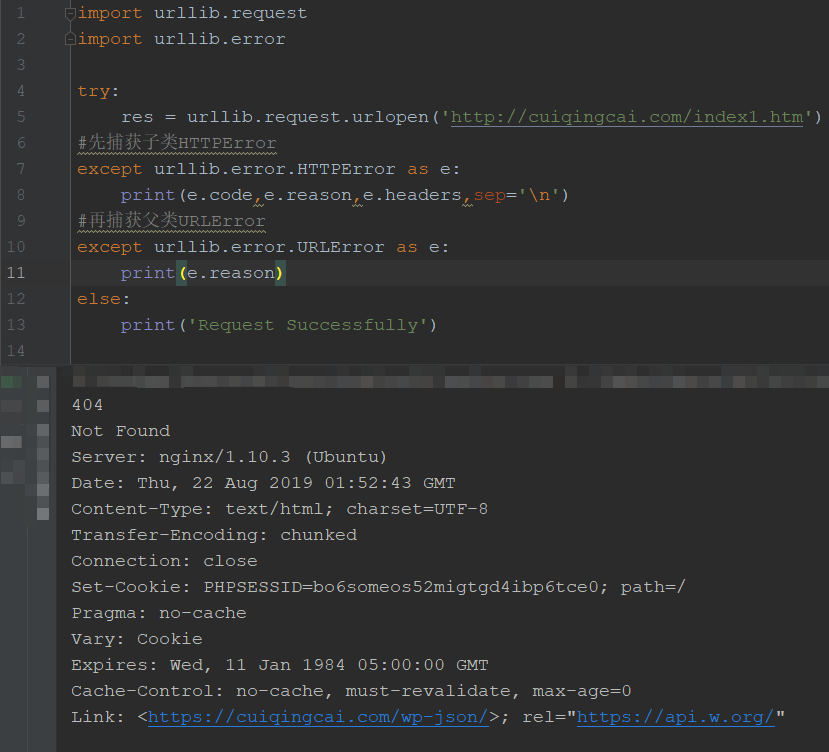
#### headers

返回请求头

相关代码如下

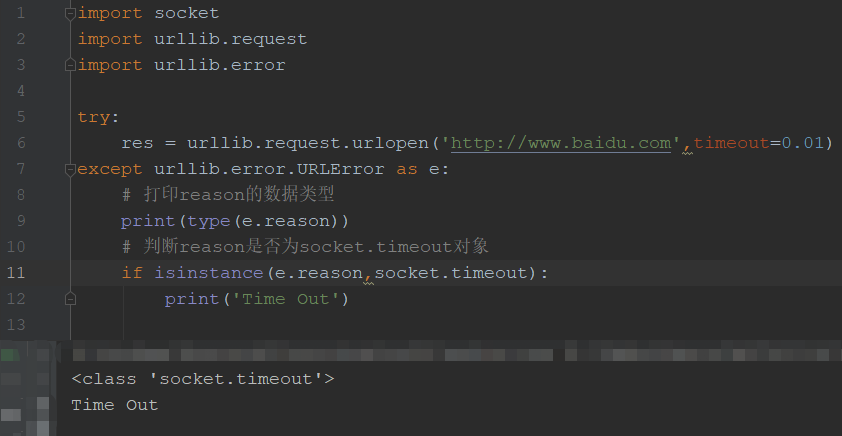


**因为URLError是HTTPError的父类**，所以可以先捕获子类的错误，再去捕获父类的错误，所以上述代码更好(更健壮)的写法如下：



这样就可以做到先捕获HTTPError，获取它的错误状态码，原因，headers信息，如果不是HTTPError异常，就会捕获URLError异常，输出错误原因，最后用**else**来处理正常的逻辑。

但是有时候，reason属性返回不一定是字符串，也可能是一个对象，相关代码如下：



可以发现，**reason**的属性的结果是**socket.timeout**类，所以我们可以用**isinstance()**方法来判断它的类型

## Parse模块

它是一个工具模块，提供了许多URL处理的方法，比如拆分、解析、合并等（常用）

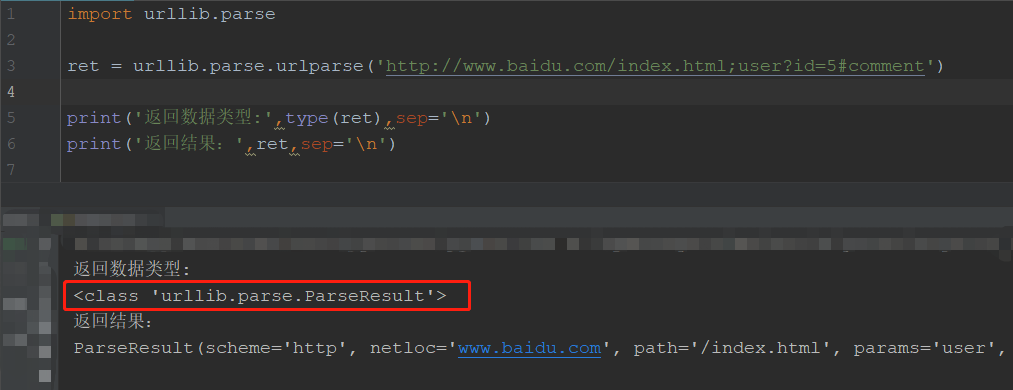
它定义了处理URL的标准接口，例如实现URL各部分的抽取，合并以及链接转换。它支持如下协议的URL处理：

**file、ftp、gopher、hdl、http、https、imap、mailto、mms、news、nntp、prospero、**

**rsync、stsp、rtspu、sftp、sip、sips、snews、svn、svn+ssh、telnet、wais**

### urlparse()方法

可以实现url识别和分段，相关代码如下



可以看到，返回**urllib.parse.ParseResult** 类型的对象

返回结果：

ParseResult(scheme='http', netloc='www.baidu.com', path='/index.html', params='user', query='id=5', fragment='comment')

包含6部分：

**schema** **://** 的前面就是schema，代表协议

**netloc**  第一个**/**符号便是netloc，即域名

**path** 访问路径

**params** 分号后面代表参数

**query** 问号?后面是查询条件

**fragment** 井号#后面是锚点，用于直接定位页面内部的下拉位置

#### API用法



##### url

必填项，即待解析的URL

##### schema

它是默认的协议，如果这个链接没有带协议信息，可以将这个作为默认协议，相关代码如下



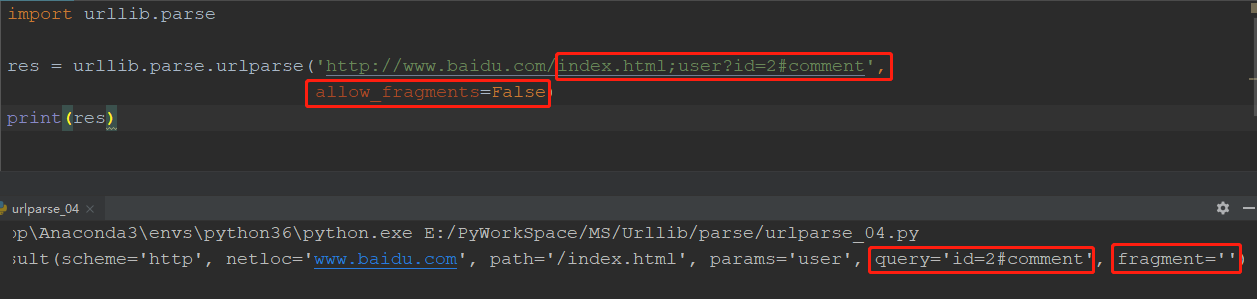
假设待解析的URL带上了schema(协议)，并且也指定了schema参数，相关代码如下



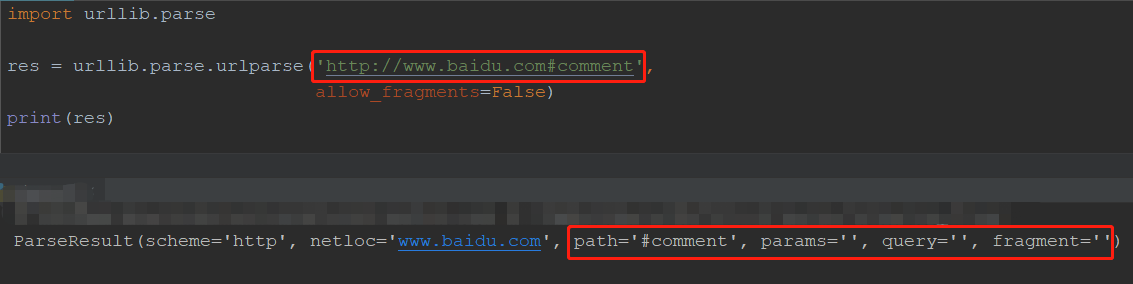
可见，schema参数只有在URL中不包含schema信息时才生效，如果URL中有schema信息，就会返回解析出的schema

##### allow\_fragment

即是否忽略fragment，若被设置为False，fragment部分就会被忽略，解析后，fragment部分将为空，它会被解析成path、params或者query的一部分，相关代码如下：



假设URL中不包含params和query，相关代码如下



可以发现，当URL中不包含params和query时，fragment便会被解析为path的一部分

ParseResult结果实际上是一个元组，我们可以通过索引顺序来获取，也可以通过属性名来获取，相关代码如下：

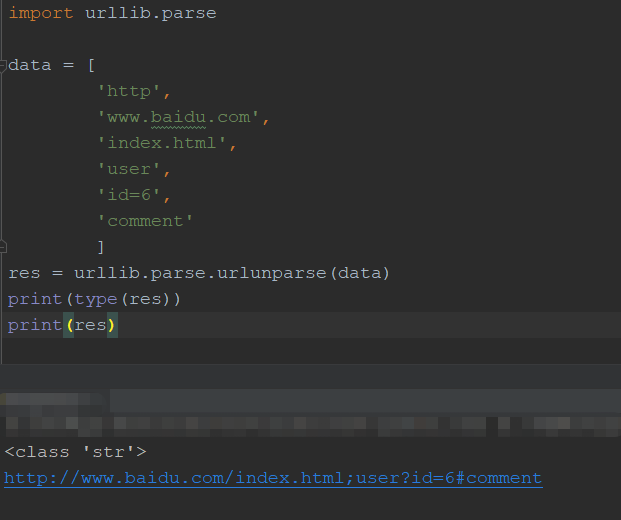


### urlunparse()方法

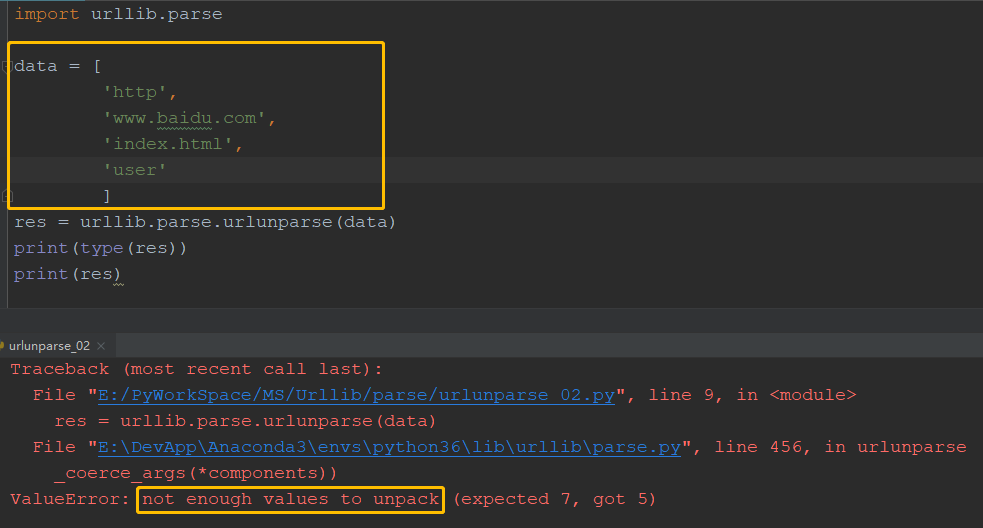
有了urlparse()，相应就有了它的对应方法urlunparse()，它接受的参数是一个可迭代的对象（列表或者元组），但是它的长度是6，并且参数的顺序要按如下顺序

scheme, netloc, url, params, query, fragment

代码如下：



否则会抛出参数数量不足或者过多的异常，相关代码如下：



### urlsplit ()方法

这个方法和urlparse()非常相似，只不过它不再单独解析params这一部分，将原先params部分的内容合并到path部分中，示例如下：



返回结果是**urllib.parse.SplitResult**类型，它其实也是一个元组类型，既可以用属性名获取值，也可以用索引来获取



### urlunsplit ()方法

与urlunparse()类似，它也是将链接各个部分组合成完整的链接方法，传入的参数也是一个可迭代的对象（列表或元组类型），唯一的区别是长度必须为5，并且参数的顺序要严格按照如下



相关代码如下



### urljoin()方法

有了urlunparse()和urlunsplit()方法，我们可以完成链接的合并，不过前提必须要有特定长度的对象，链接的每部分都要清晰的分开

生成链接还有另外一种方法，就是urljoin()方法，我们可以提供一个base\_url(基础链接)作为第1个参数，将新的链接作为第2个参数，

**该方法会分析base\_url的scheme、netloc、和path这3部分内容，并对新链接缺失部分进行补充** ，最后返回结果，相关代码如下



**总结：可以发现，base\_url提供了三项内容scheme、netloc和path，如果这3项在新的链接里不存在，就予以补充，如果新链接存在，就是用新链接的部分，而base\_url中的params、query、fragment是不起作用的，**

**通过urljoin()方法可以轻松实现链接的解析、拼合与生成。**

### urlencode()方法

它在构造GET请求参数的时非常有用，**将字典类型的参数序列化成GET请求的参数**

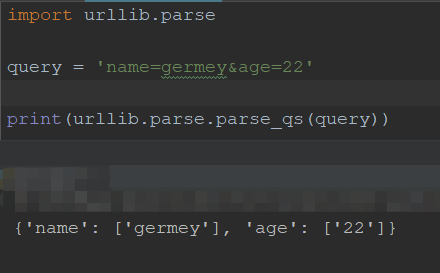
相关代码如下



这个方法非常有用，有时候为了更加方便的构造参数，会事先用字典来表示，然后再使用该方法序列化为GET参数

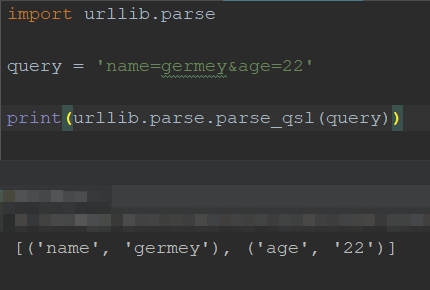
### parse\_qs()方法

有了序列化，必定有反序列化，如果我们有一串GET请求参数，利用parse\_qs()方法可以将它转化为字典类型的数据，代码如下



### parse\_qsl()方法

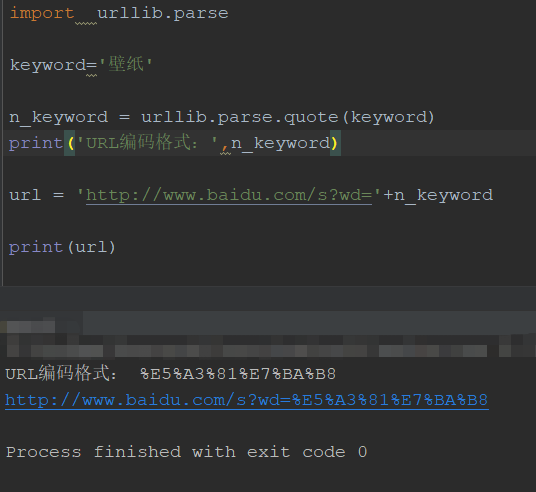
它可以将一串GET请求参数转化(反序列化)为元组组成的列表，代码如下



可以看到，运行的结果是一个列表，列表中的每一个元素是一个元组，元组的第1个内容是参数名，第2个内容是参数值

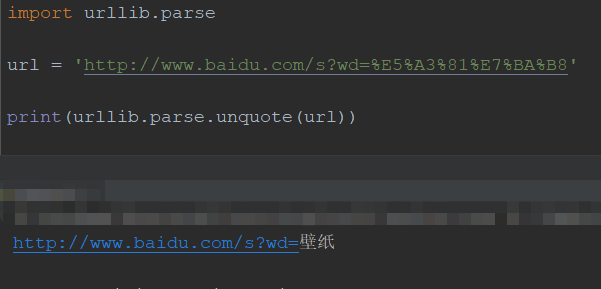
### quote()方法

该方法可以将内容转化为URL编码的格式，URL中带有中文参数时，有时候可能会导致乱码的问题，此时需要用这个方法将中文字符转化为URL编码，代码如下：



### unquote()方法

它可以进行**URL解码**，相关代码如下：



## robotparser模块

主要是用来识别网站的robot.txt文件，然后判断哪些网站可以爬，哪些网站不可以爬

### Robots协议

也称作爬虫协议，全名叫做网络爬虫排除标准，通常是一个叫做**robots.txt**文件，**一般放在网站的根目录底下**

当搜索爬虫访问一个站点时，它首先会检查这个站点的根目录下是否存在robots.txt，如果存在，搜索爬虫会根据其中定义的爬取范围来爬取，如果没有这个文件，搜索爬虫便会访问所有可直接访问的页面

**robots文件样例：**

**User-Agent： \***

**Disallow: /**

**Allow: /public/**

表示对于所有的爬虫只能爬取/public目录的功能

#### User-Agent属性

描述了搜索爬虫的名称，设置为 **\*** 代表该协议对任何爬取爬虫有效，至少指定1个User-Agent属性

#### Disallow属性

指定了不允许抓取的目录

#### Allow属性

一般与Disallow一起使用，代表允许抓取的目录

### 爬虫名称

|  |  |  |
| --- | --- | --- |
| 爬虫名称 | 名称 | 网站 |
| BaiduSpider | 百度 | www.baidu.com |
| Googlebot | 谷歌 | www.google.com |
| 360Spider | 360搜索 | www.so.com |
| YoudaoBot | 有道 | www.youdao.com |
| ia\_archiver | Alexa | www.alexa.com |
| Scooter | altavista | www. Altavista.com |

### robotparser模块

该模块提供了一个类RobotFileParser，它可以根据某网站的robots.txt文件来判断一个爬取的爬虫是否有权限来爬取这个网页了

#### RobotFileParser类

声明如下：默认url参数为空

class urllib.robotparser. RobotFileParser ( url='' )

这个类常用的方法如下

##### set\_url()方法

用来设置robot.txt文件的链接，如果创建RobotFileParser对象时传入了链接，那么就不需要再使用这个方法设置了

##### read()方法

读取robots.txt文件并分析。只能读取一次，这个方法不会返回任何内容，但是执行了读取的操作

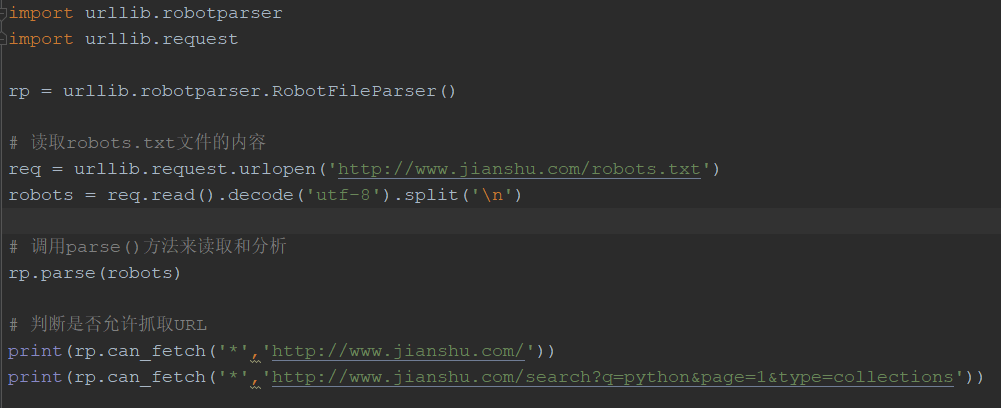
使用read()方式来读取



##### parse()方法

用来解析robots.txt文件，传入的参数是robots.txt某些行的内容，它会按照robots.txt规则来分析这些内容

使用parse()方式来读取



##### can\_fetch()方法

该方法传入两个参数，第一个是User-Agent，第二个是要抓取的URL，结果返回True或False

##### mtime()方法

返回上次抓取和分析robots.txt的时间

##### modified()方法

将当前时间设置为上次抓取和分析robots.txt文件的时间

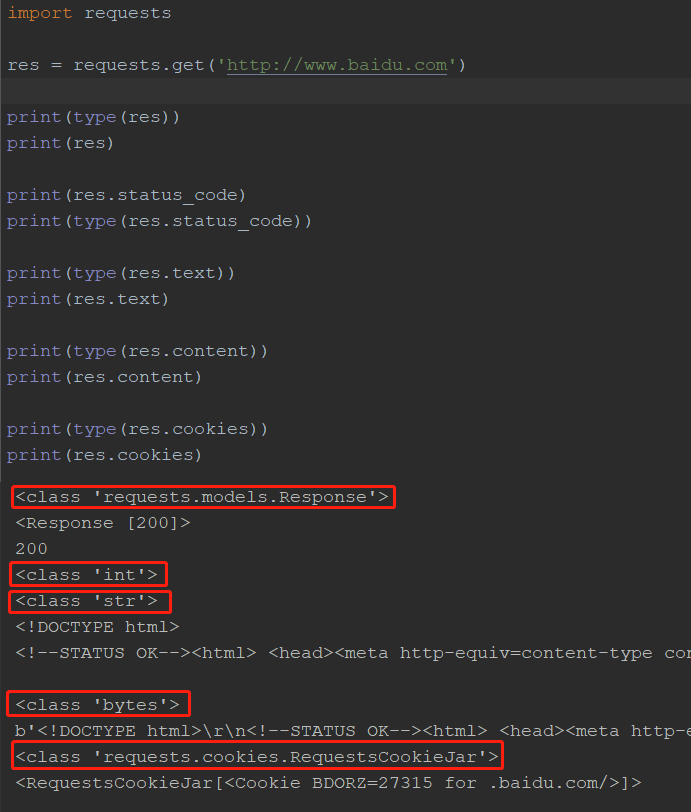
# requests库的使用

## 基本用法

在使用之前，请确保安装好了requests库

pip install requests

## 实例引入



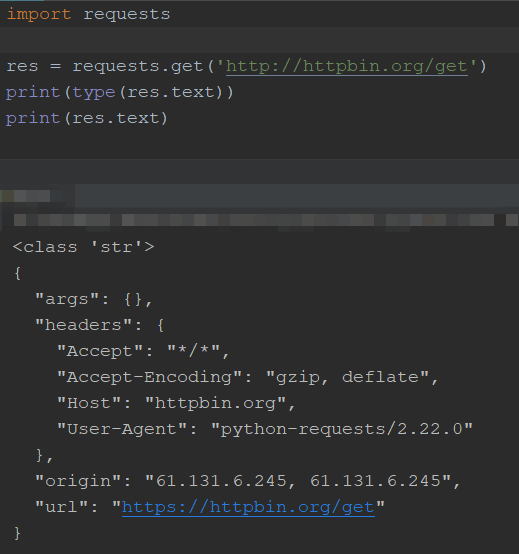
Get()方法实现与urlopen()相同的操作，返回requests.models.Response 类型

分别输出Response的类型，状态码，响应体类型，Cookies等

## GET请求方法

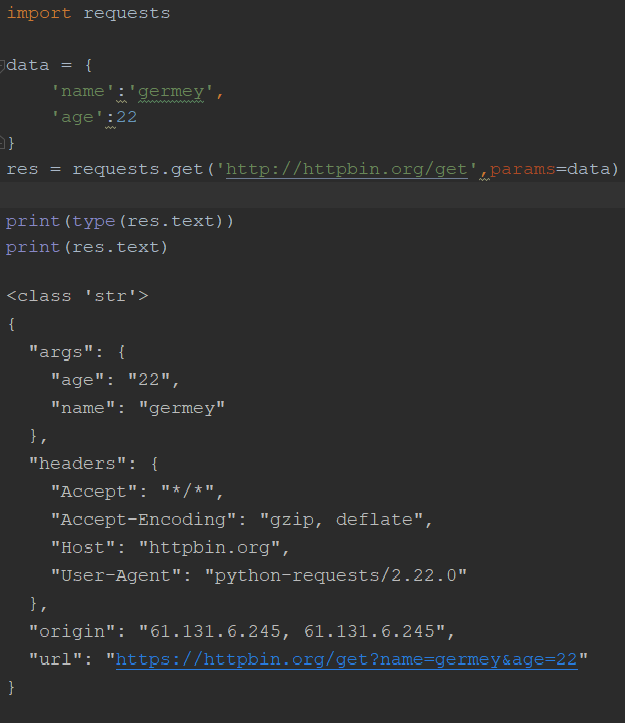
### 基本实例

构建一个简单的GET请求，请求的链接为<http://httpbin.org/get>

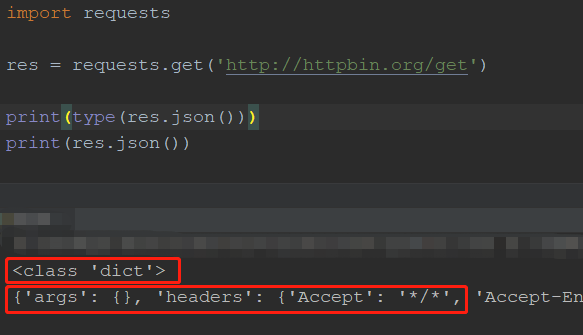


### params参数

对于Get请求，如果要附加额外的参数信息，一般用字典来存储，代码如下：



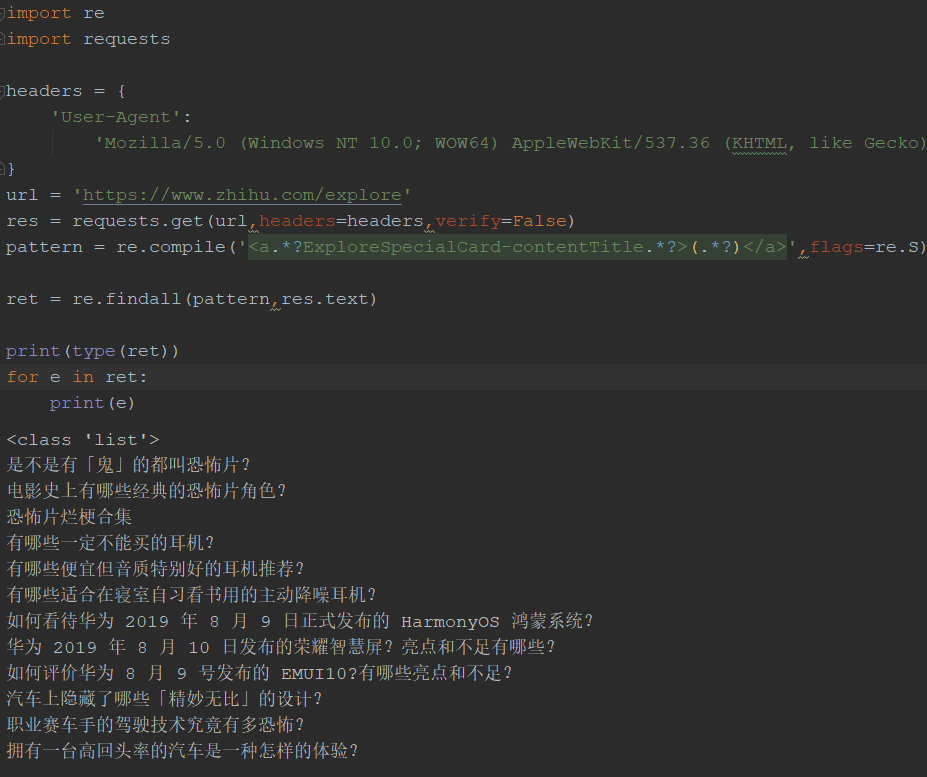
如上网页的返回类型实际上是str类型，但是它很特殊，是Json格式的，所以如果想直接解析返回结果，得到一个字典类型的话，可以直接调用**json()**方法,代码如下



### 抓取网页

上面请求链接返回的是Json形式的字符串，那么如果请求普通网页，则肯定能获取相应的内容了。下面以“知乎”-“发现”页面，记得设置verify=False,否则将会报错

CERTIFICATE\_VERIFY\_FAILED



### 抓取二进制数据

在上面的例子中，我们抓取的是知乎的一个页面，实际上返回的是一个HTML文档，如果想抓取图片、音频、视频等文件，那该怎么办？

先抓取二进制数据，再转化为相应的格式的文件，相关代码如下

**Response对象的Content属性是返回Bytes(字节流类型)的内容**

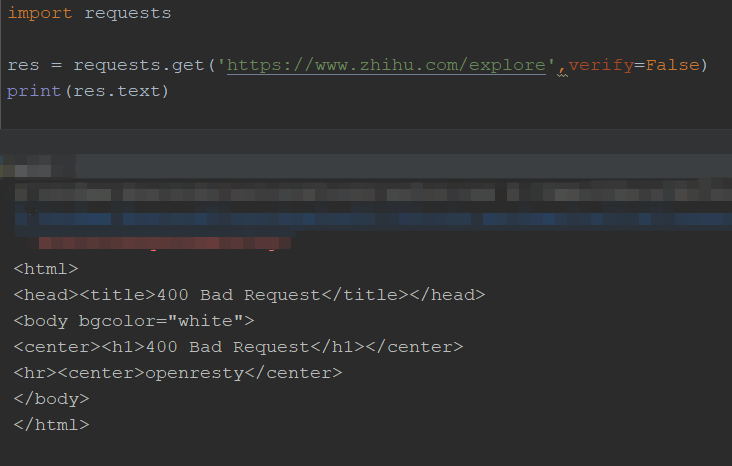
下面以抓取GitHub的站点图标为例来看一下



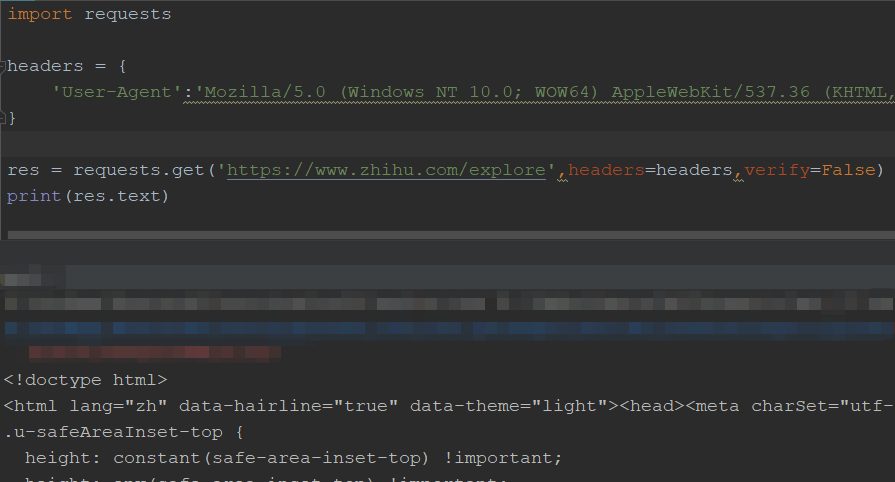
同样的抓取音视频文件也是类似的做法

### 添加headers

比如，在上面的“知乎”的例子中，如果不传递headers，就不能正常请求了

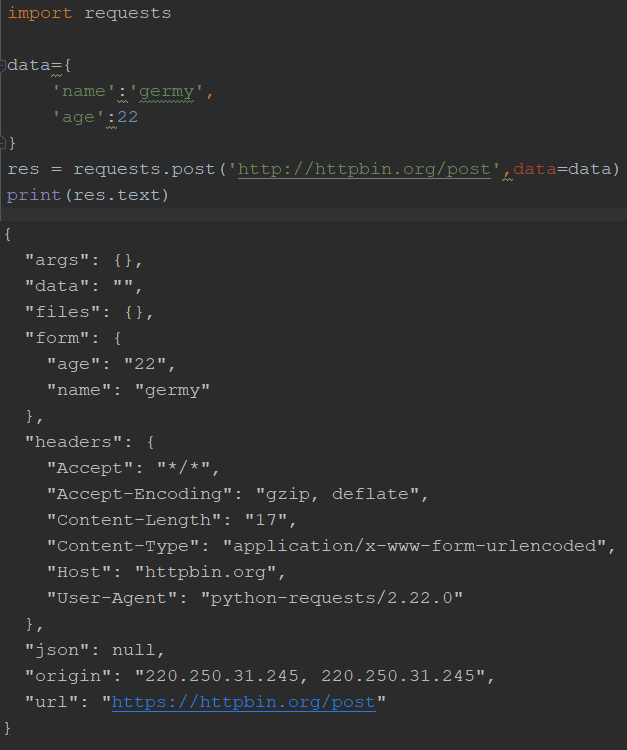


但如果加上headers并加上User-Agent信息，那就没问题了



## POST请求方法

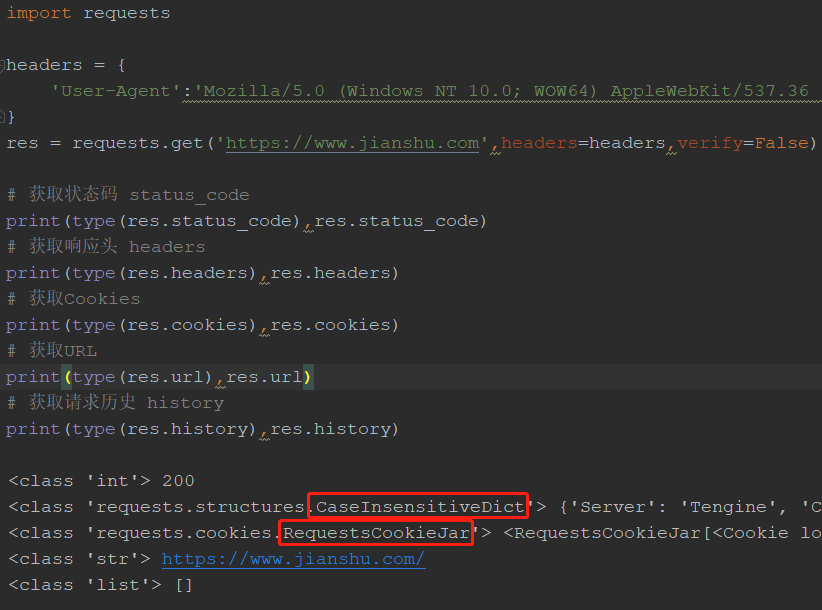
使用requests实现post请求同样非常简单，示例如下：



## 响应

使用text和content获取了响应的内容，此外还有很多属性和方法来获取其他信息，比如

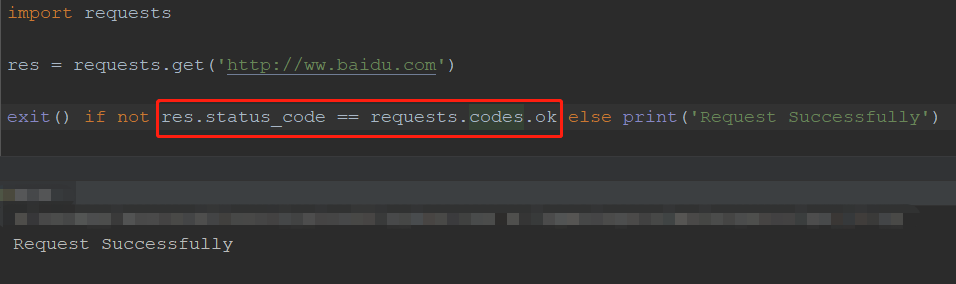
状态码、响应头、Cookies等，相关代码如下：



### 状态码

常用来判断请求是否成功，requests还提供了一个内置的状态码查询对象，**requests.codes**

相关代码如下：

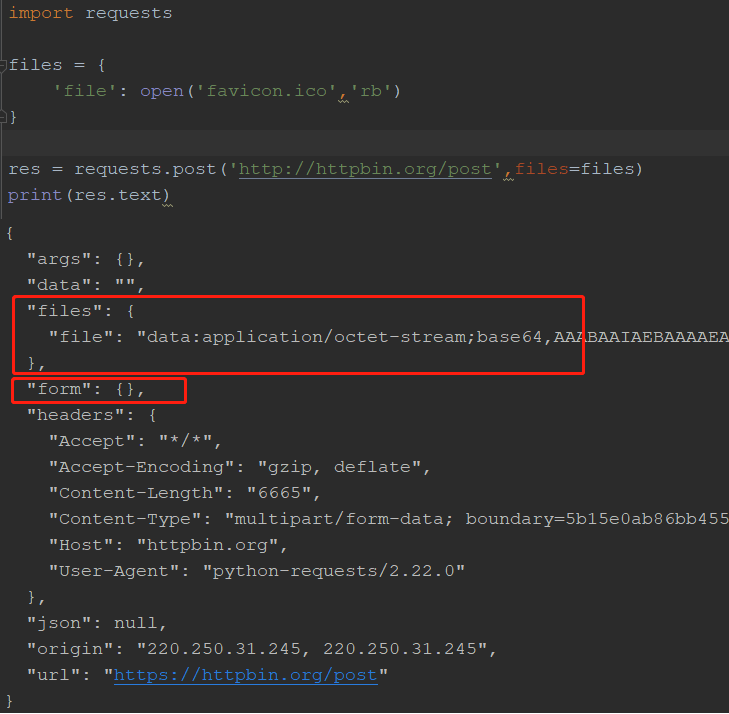


### 状态码对应表

状态返回码和相应的查询条件：

## 高级用法

### 文件上传



这个网站会返回响应，里面包含files字段 ，而form字段为空，说明文件上传部分会单独有一个files字段来标识

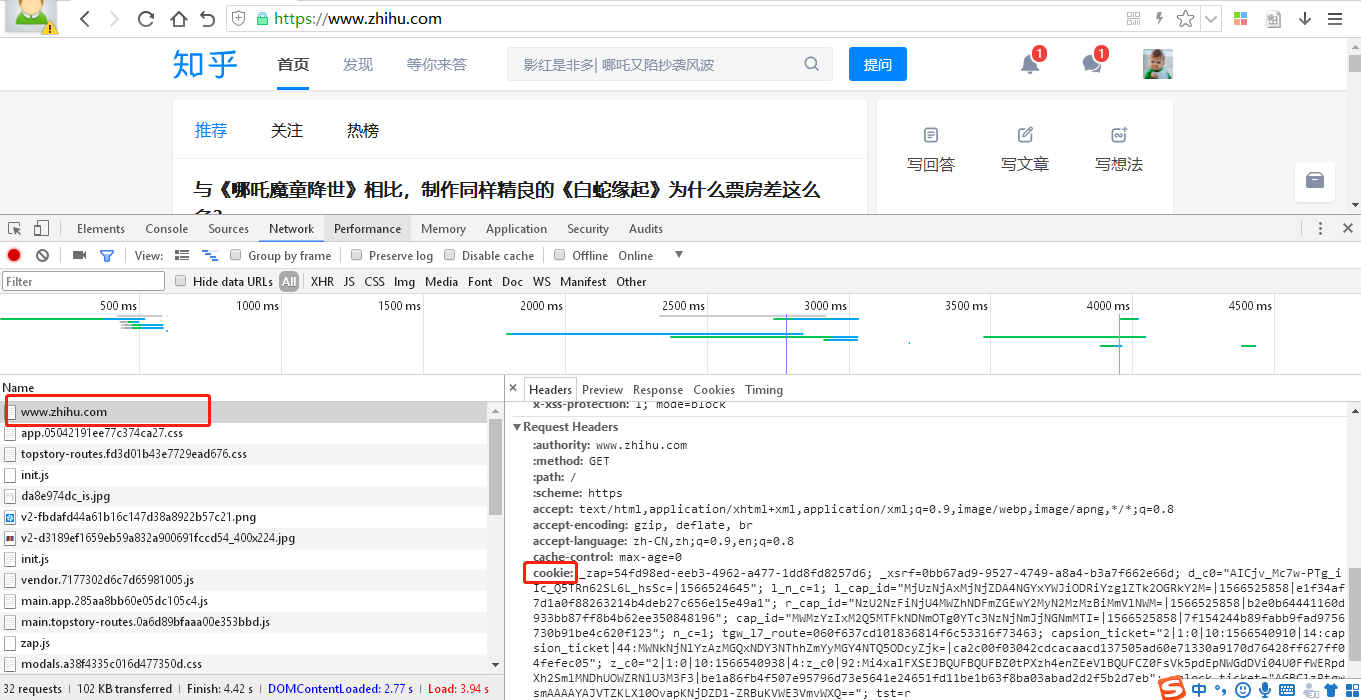
### Cookies

获取Cookies的过程，相关代码如下：

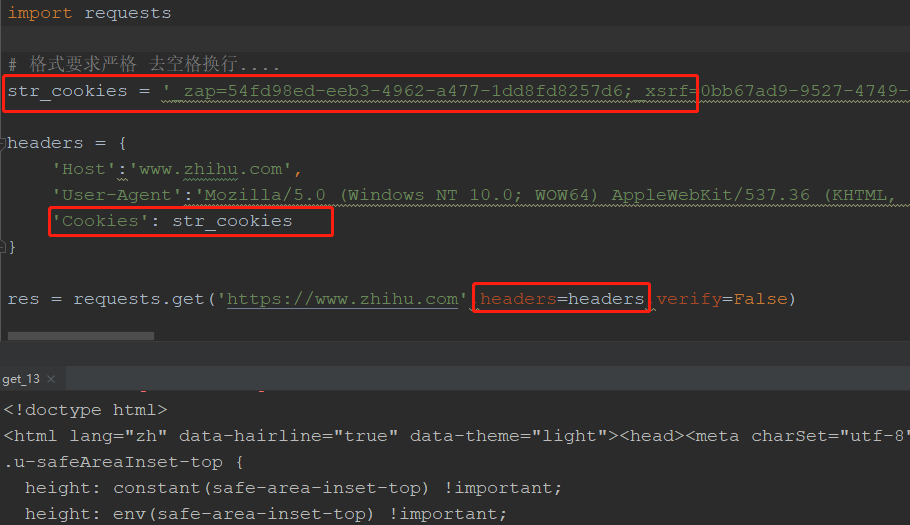


首先调用cookies属性即可得到Cookies，可以发现它是RequestsCookieJar类型，然后用items()方法将其转化为由元组组成的列表，遍历输出每一个Cookie的名称和值

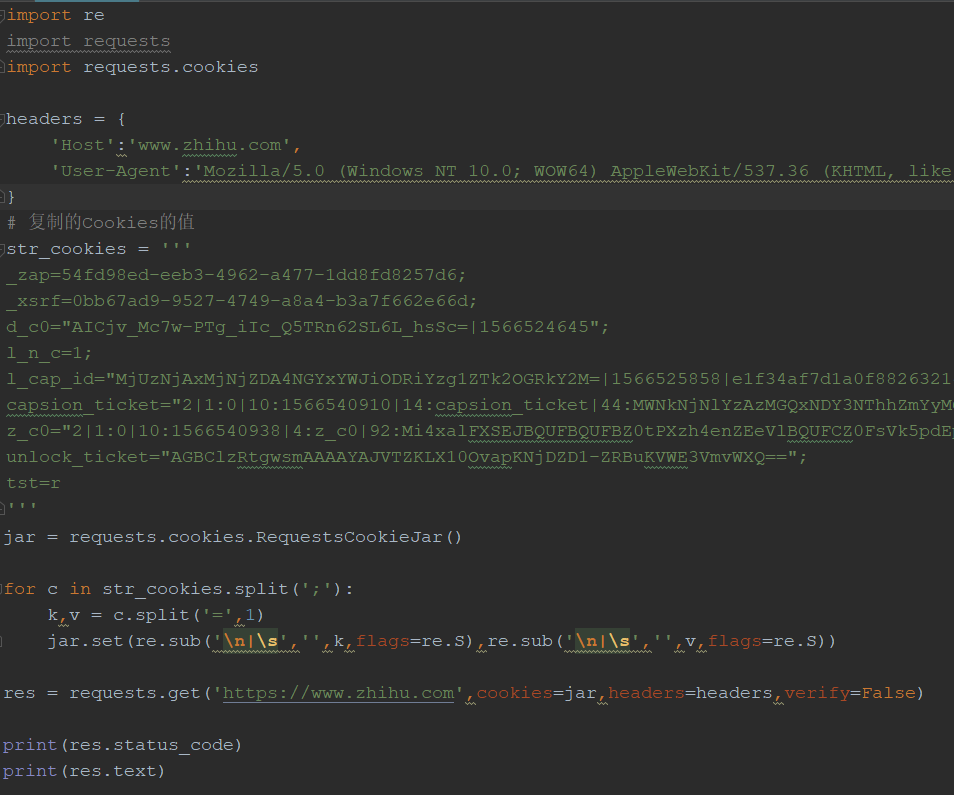
我们也可以直接用Cookie来维持登录状态，首先登录知乎，将headers中的Cookie复制下来，将其设置到headers中，然后发送请求，相关代码如下



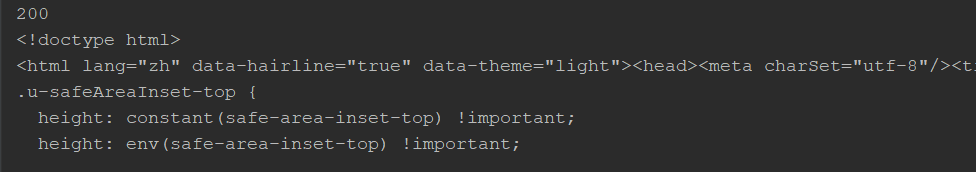
方法1：直接设置headers中Cookies的值



方法2：你也可以通过设置cookie参数，不过就需要构造requests.cookies.RequestsCookieJar对象



结果输出如下：



### 会话维持

利用Session可以做到模拟同一个会话而不用担心Cookies的问题，它通常用于模拟登录成功之后再进行下一步的操作，

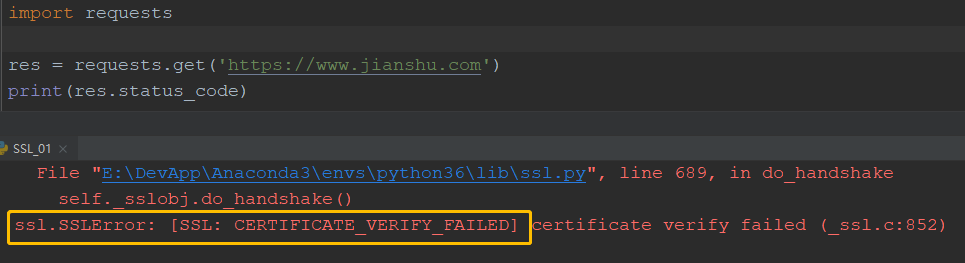
Session平常用的很广泛，可以用于模拟在一个浏览器中打开同一站点中不同的页面。



### SSL证书验证

Requests还提供了证书验证的功能，当发送HTTP请求的时候，它会检查SSL证书，我们可以使用verify参数控制是否检查此证书，如果不加verify参数的话，默认值为True，会自动验证。

相关代码如下

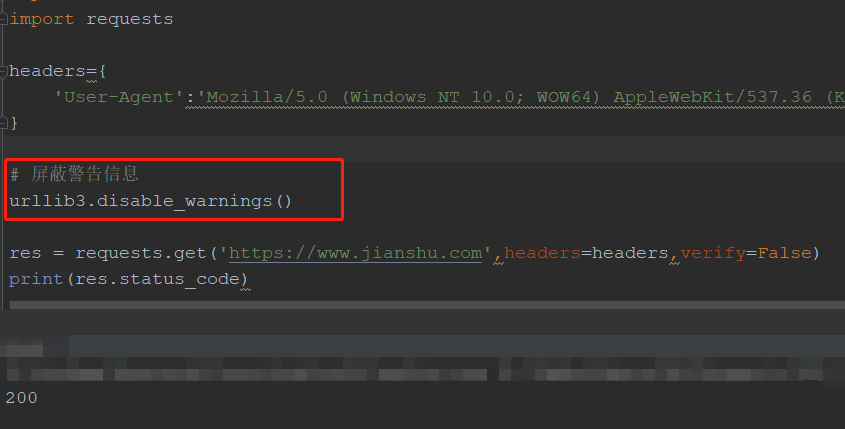


把verify参数设置为False即可，如下

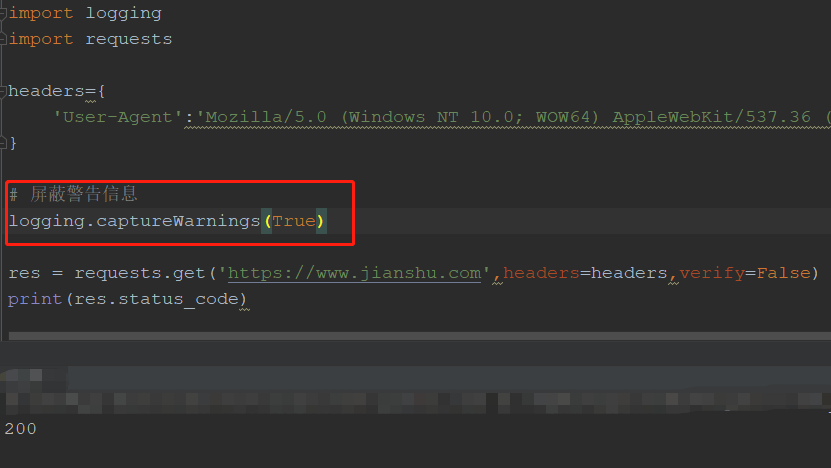


屏蔽警告信息的方式有以下：

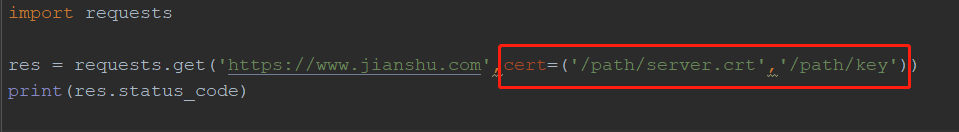
方法1：**设置urllib3.disable\_warnings()**



方法2：**通过捕获到警告到日志的方式忽略警告**



方法3：也可以指定本地证书用作客户端证书，这可以是**单个文件(包含密钥和证书)**或一个包含两个文件路径的**元组**



上面的代码是演示实例，我们需要有crt文件和key文件，并且指定他们的路径，注意，本地的私有证书的key必须是解密状态，加密状态下的key是不支持的

### 代理设置

对于某些网站，在测试的时候，请求几次能正常返回内容，但是一旦开始大规模爬取，对于大规模且频繁的请求，网站会弹出验证码，或者跳转至登录认证页面，更甚者可能会直接封禁客户端IP，导致一定时间段内无法访问，

我们需要设置代理来解决这个问题，这就需要用到proxies参数，如下所示：



**若代理使用HTTP Basic Auth**，可以使用类似[**http://user:password@host:port**](http://user:password@host:port)这样的语法来设置代理

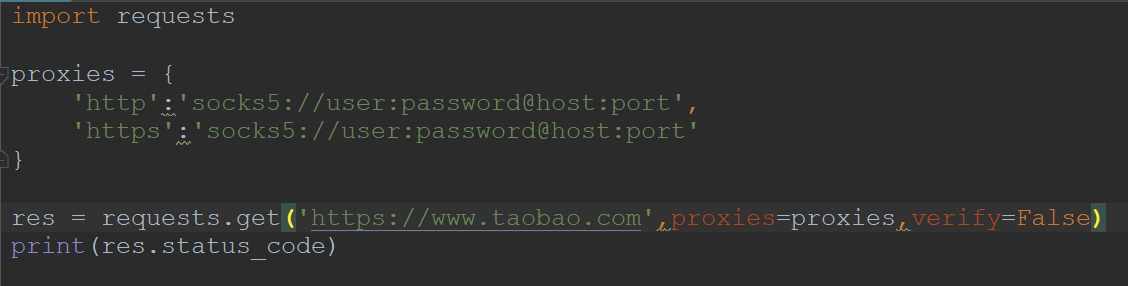


**requests还支持socks协议代理**

首先，需要安装socks库



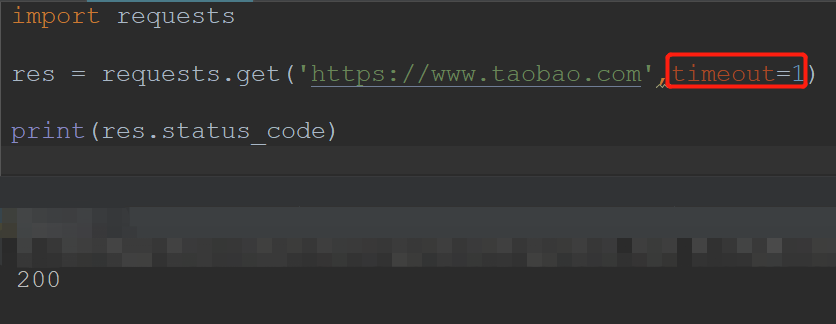
然后就可以使用socks协议代理了，相关代码如下



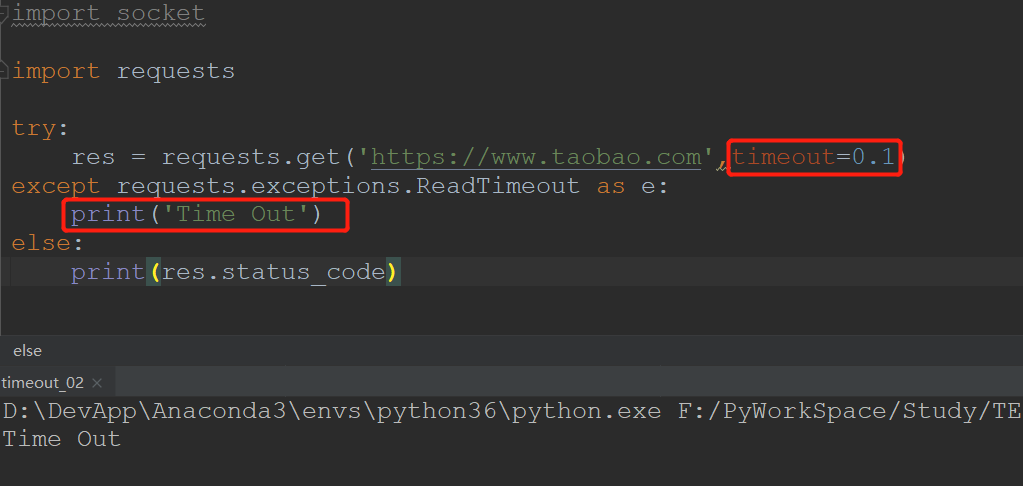
### 超时设置

在本机网络状况不好或者服务器网络响应太慢甚至无响应时,我们可能会等待特别久的时间才可能收到响应,甚至到最后收不到响应而报错。为了防止服务器不能及时响应,应该设置一个超时时间,

即超过了这个时间还没有得到响应,那就报错。这需要用到 **timeout**参数。这个时间的计算是**发出请求到服务器返回响应的时间**。示例如下



异常情况

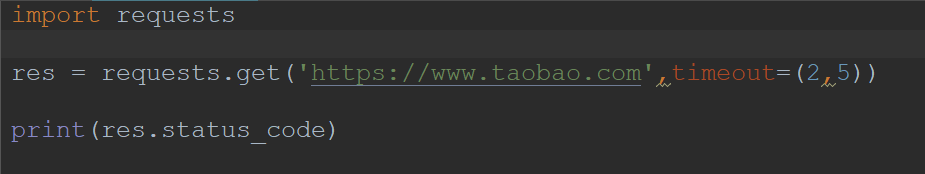


通过这样的方式，我们可以将timeout设置为1秒，如果1秒内没有响应，那就抛出异常

**实际上，请求分为两个阶段：即 连接 和 读取**

**上面设置的timeout将用作连接和读取这二者的timeout总和**

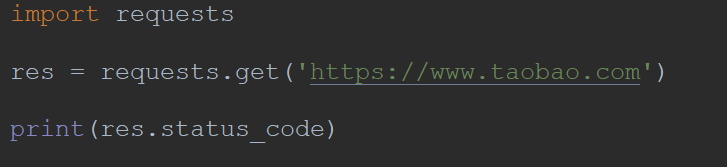
如果要分别指定连接和读取的时间，就可以传入一个**元组**，代码如下



如果想永久等待，可以直接将timeout设置为None，或者**不设置**,因为默认值为None



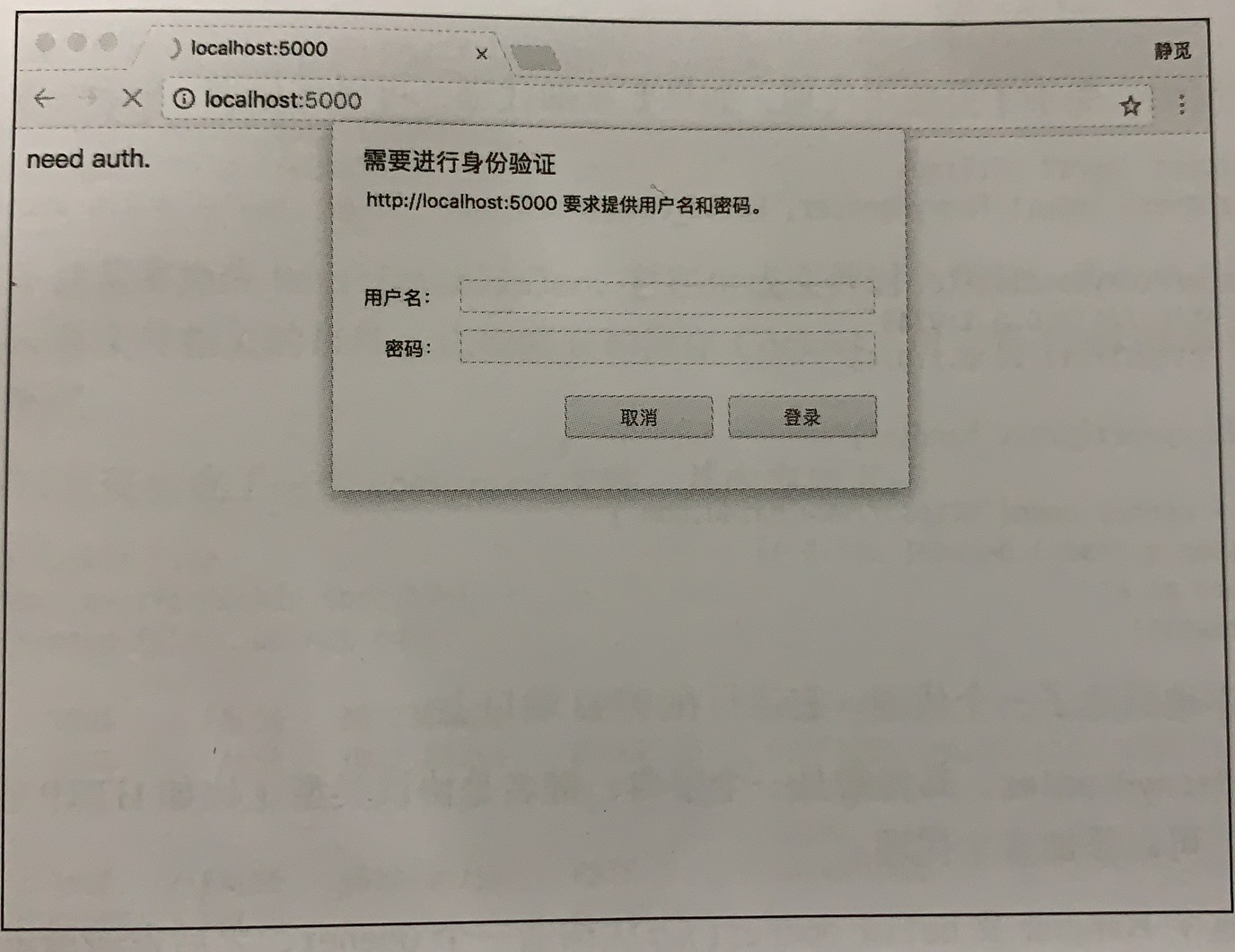
或者



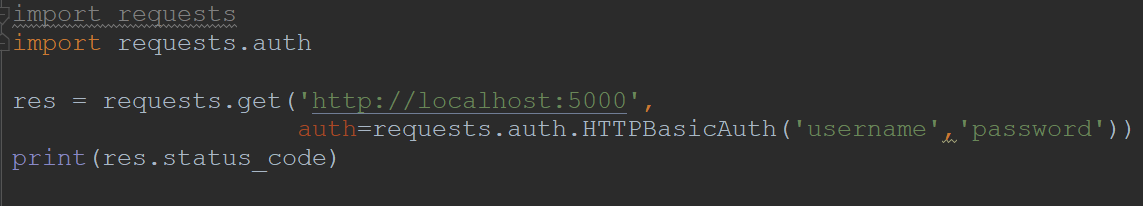
### 身份认证

#### HTTPBasicAuth类认证

在访问网站时，我们可能会遇到这样的认证页面



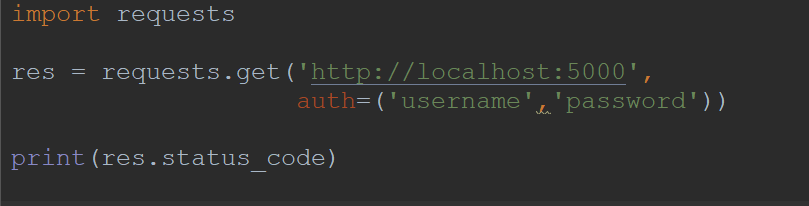
我们可以使用requests自带的**HTTPBasicAuth类**实现身份认证功能，示例如下：



如果用户名和密码正确的话，请求时就会自动认证成功，会返回200状态码；如果认证失败，则返回401状态码。

当然，如果参数传递一个**HTTPBasicAuth类**，就显得有点烦琐了，所以requests提供了一个更简单的写法，可以直接传递一个**元组**，它会默认使用**HTTPBasicAuth**这个类来认证。

相关代码如下

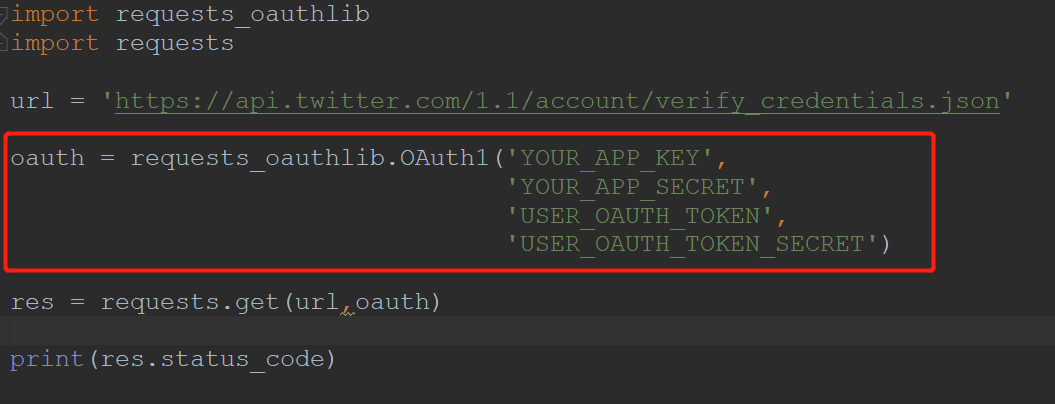


#### OAuth认证

此外，requests还提供了其他认证的方式，如OAuth认证，不过此时要安装oauth包，安装命令如下：

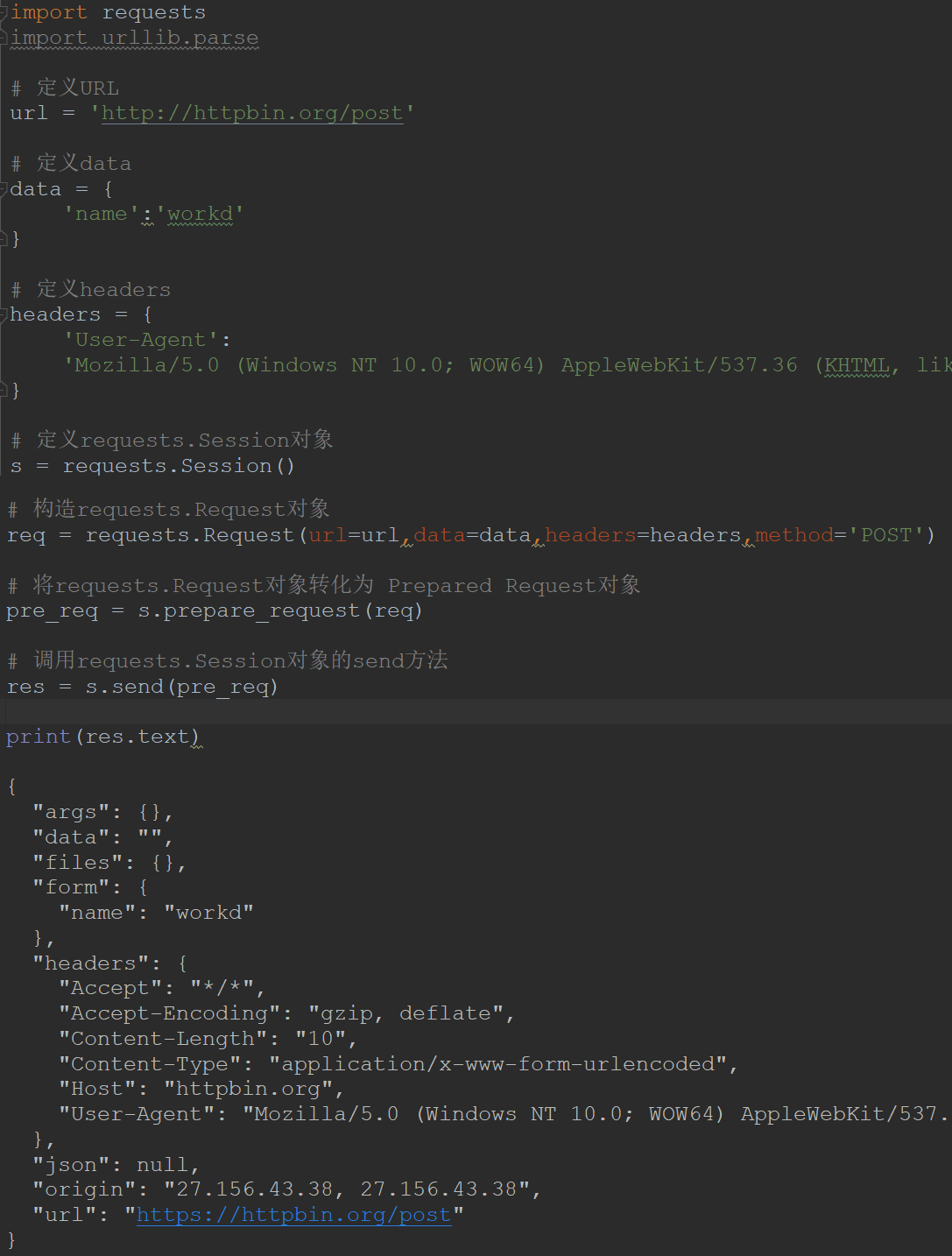


使用OAuth1认证的方法如下：



### Prepared Request

前面介绍的urllib库时，我们可以将请求表示为**数据结构**，其中各个参数都可以通过构造**Request对象**来表示。在requests库里同样可以做到，这个**数据结构**就是**Prepared Request 对象,**示例如下：

****

可以看到，以上达到同样的requests的POST请求效果

有了Request对象，就可以将请求当做**独立的对象**来看待，这样在进行队列调度时会非常方便，后面我们会用它来构造一个**Request队列**