# Python正则表达式系列

## 常用匹配规则

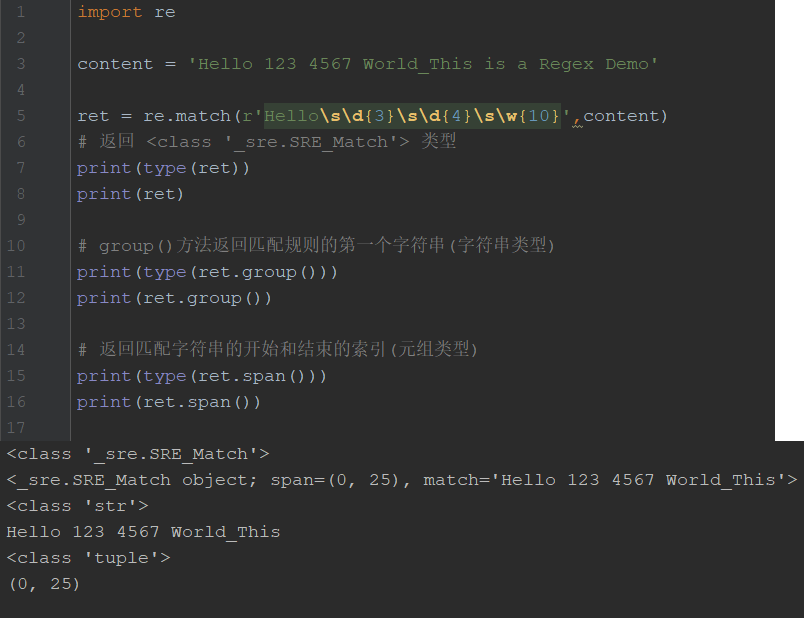
|  |  |
| --- | --- |
| 修饰符 | 描述 |
| \w | 匹配字母、数字及下划线 |
| \W | 匹配不是字母、数字及下划线的字符 |
| \s | 匹配任意空白字符，等价于[\t\n\r\f] |
| \S | 匹配任意非空字符 |
| \d | 匹配任意数字，等价于[0-9] |
| \D | 匹配任意非数字的字符 |
| \A | 匹配字符串开头 |
| \Z | 匹配字符串结尾，如果存在换行，只匹配换行前的结束字符串 |
| \z | 匹配字符串结尾，如果存在换行，同时还会匹配换行符 |
| \G | 匹配最后匹配完成的位置 |
| \n | 匹配一个换行符 |
| \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

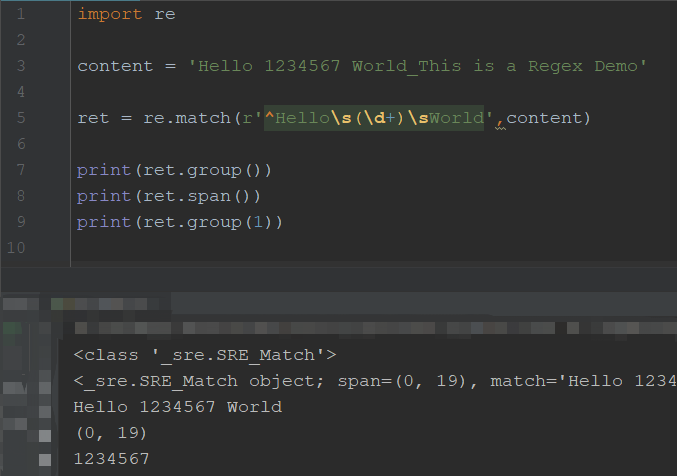
（即从字符串的开头开始匹配，一旦开头匹配不成功，那么整个匹配就失败了）

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



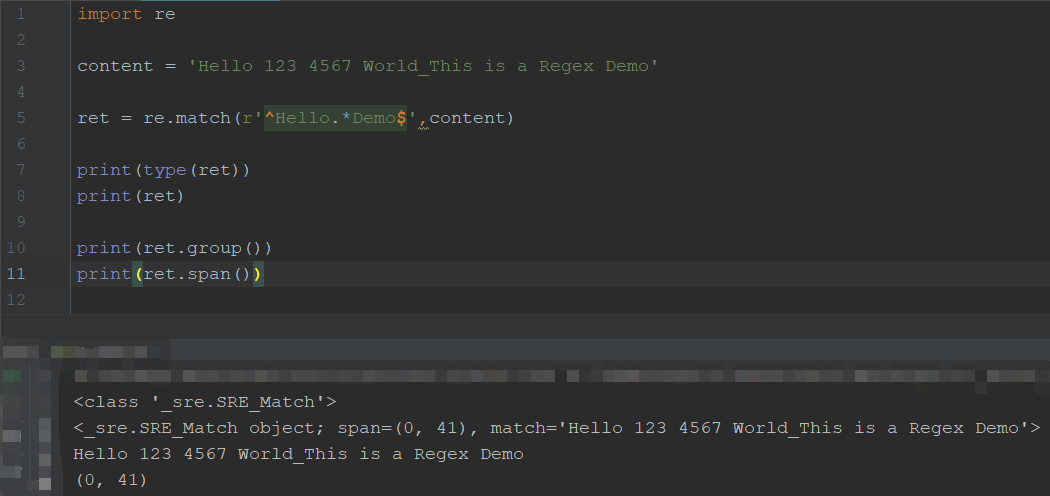
### 1.匹配目标

使用()来精准匹配目标



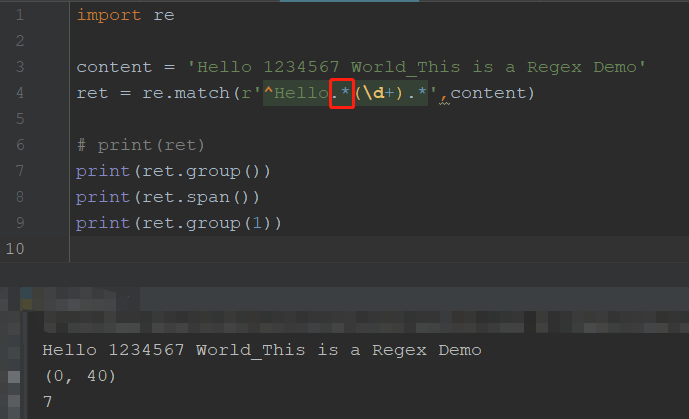
### 2.通用匹配

使用 **.\*** 来实现通用匹配

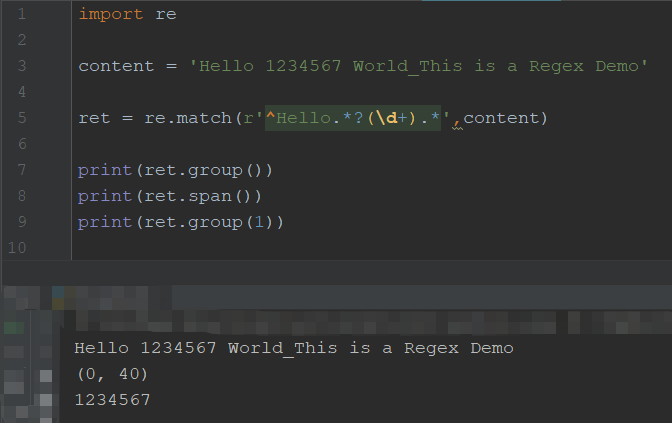


### 3.贪婪与非贪婪

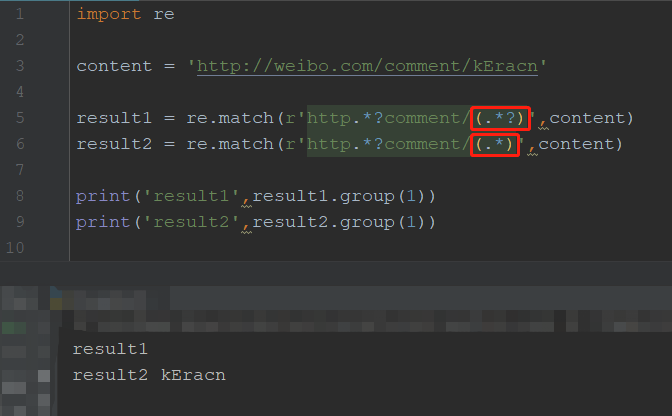
贪婪匹配：尽可能多的匹配字符 **.\***



非贪婪匹配：尽可能匹配少的字符 **.\*?**

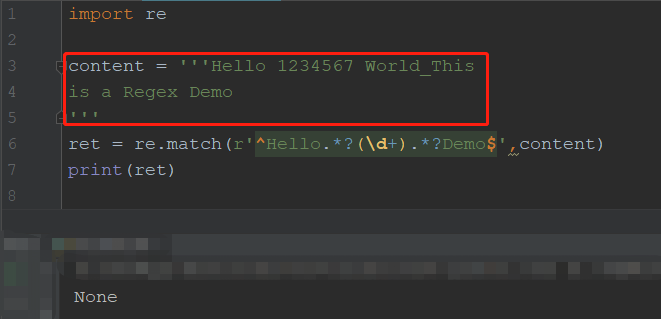


如果匹配的结果在字符串结尾, **.\*?** 可能匹配不到内容了，因为它会匹配尽可能少的内容

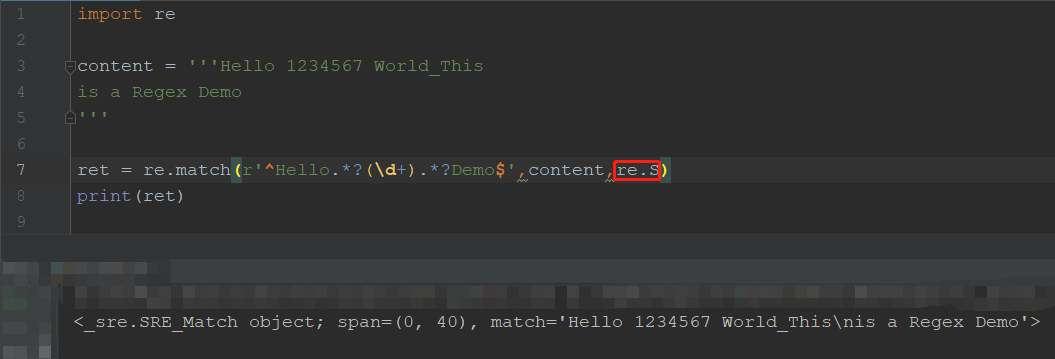


### 4.修饰符

字符串中加上换行符，正则表达式还是一样，结果返回None



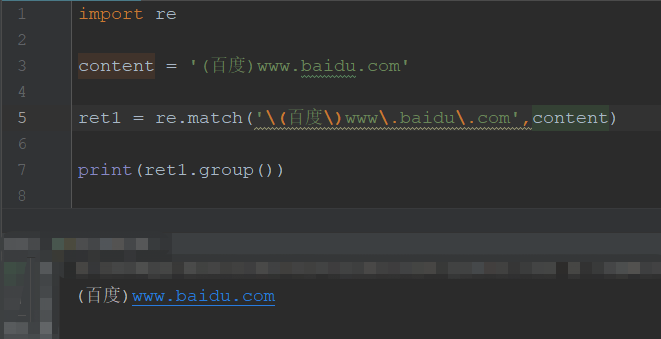
需要加上修饰符，修改如下



|  |  |
| --- | --- |
| 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

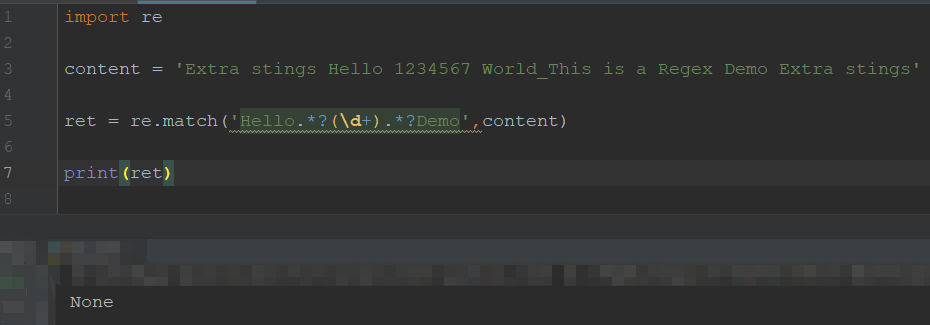
**.**可以匹配除换行符意外的任意字符，但如果目标字符串里面含有**.**那该怎么办



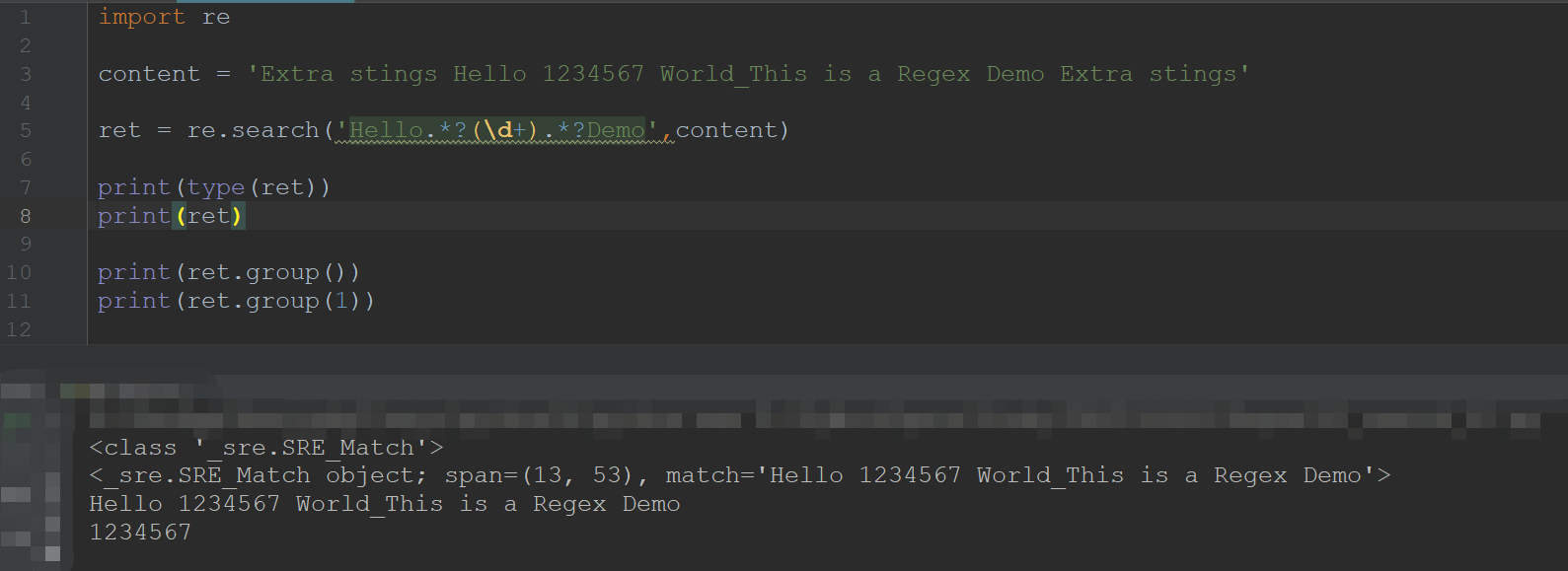
## search()方法

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



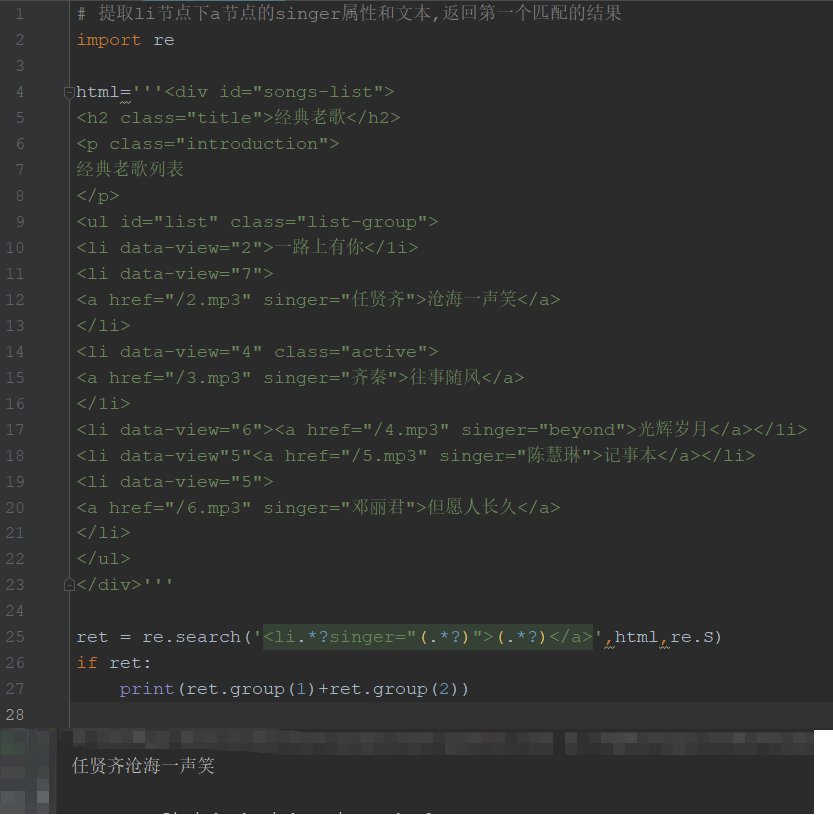
**以下有一段待匹配的HTML文本**

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

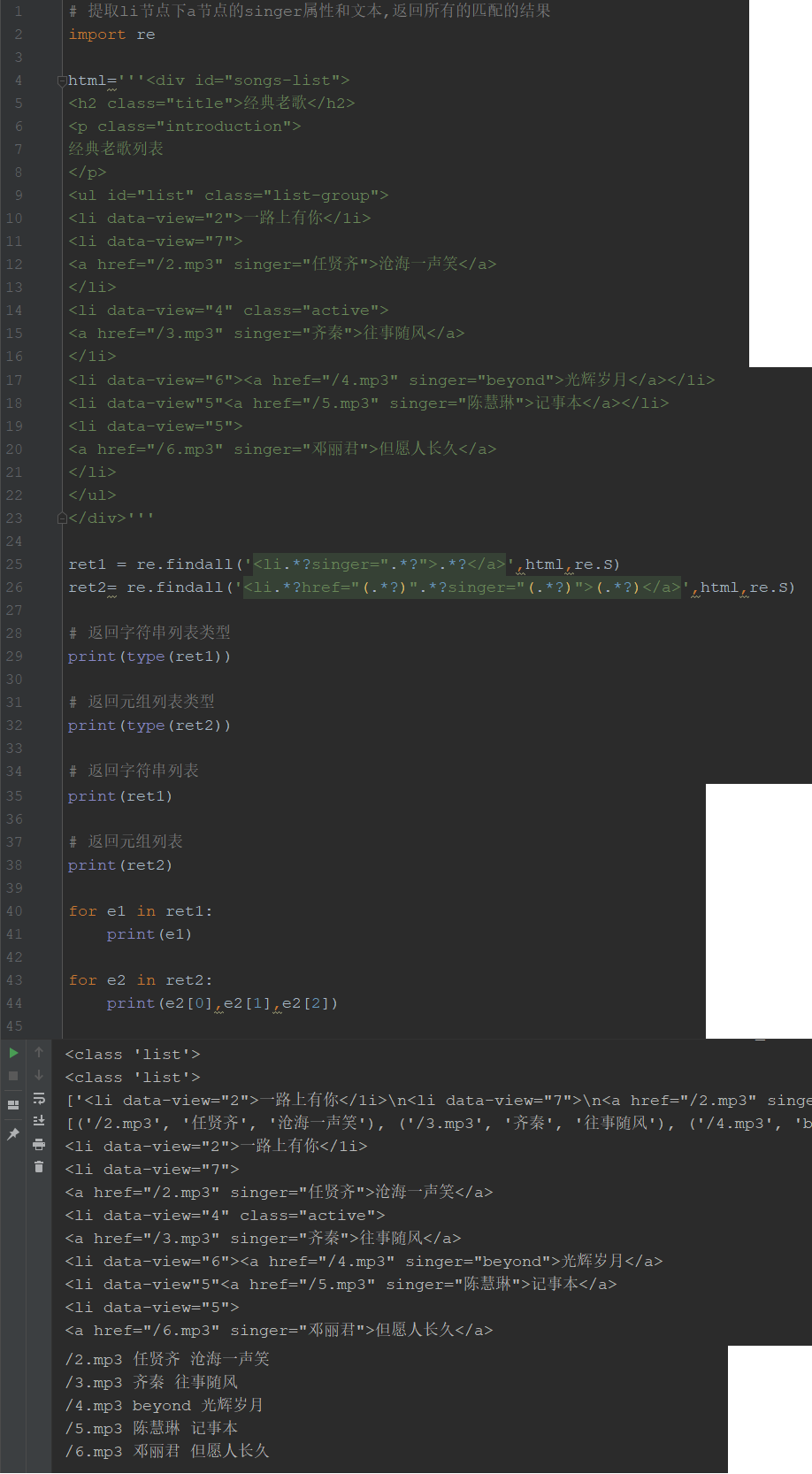


提取li节点下a节点的singer属性和文本,返回第一个匹配的结果



## 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

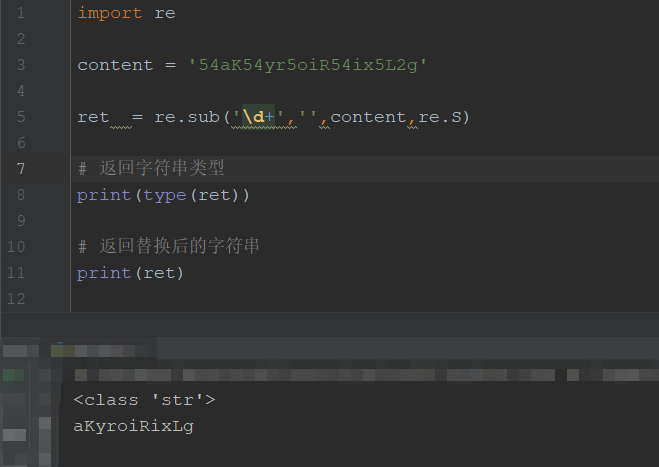


如果只需要获取一个内容可以用search方法，当需要提取多个内容时，可以用findall方法

## 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

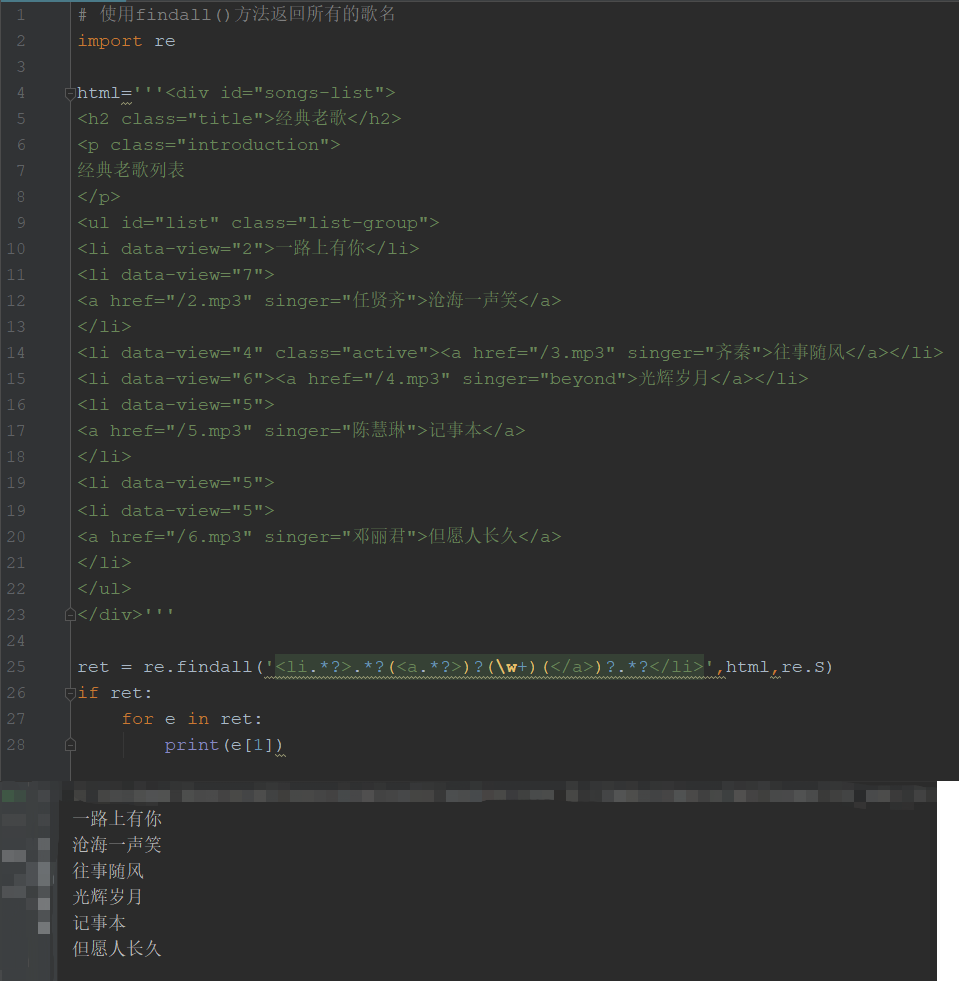


**注意：遇上换行符，sub可以使用修饰符，但必须指定参数名flags=re.S，否则将无效**

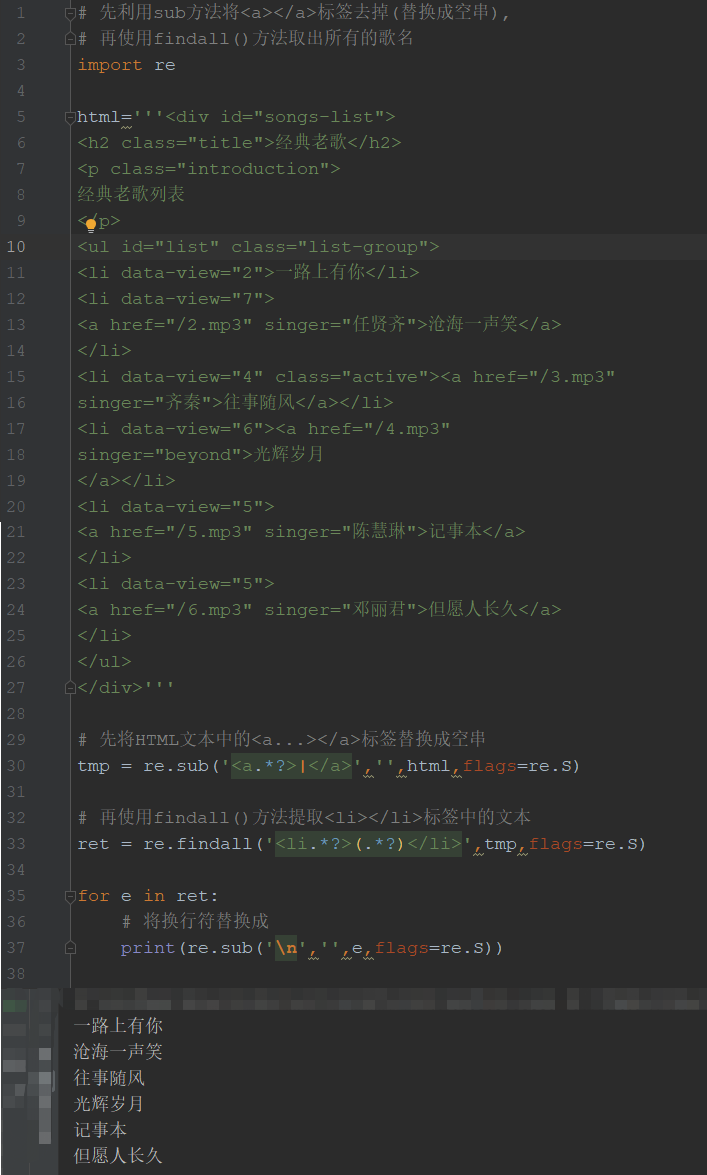
**如下：**

****

获取以上HTML文本中所有的歌名  
使用findall()方法

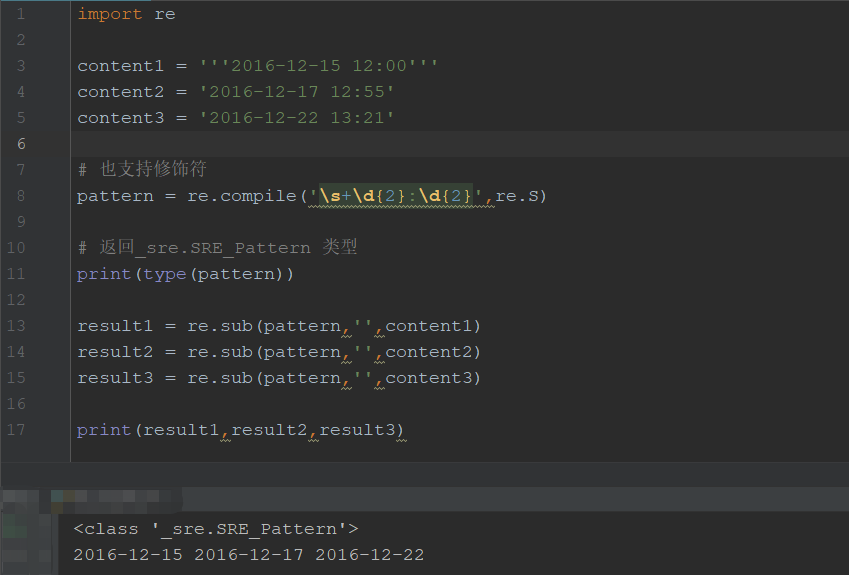


使用sub()方法



## 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()

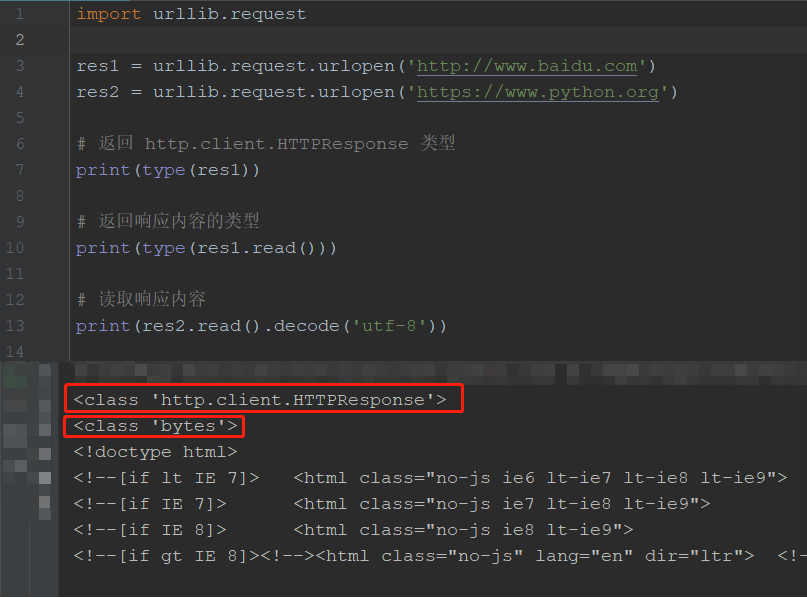
返回 **http.client.HTTPResponse** 类型的对象

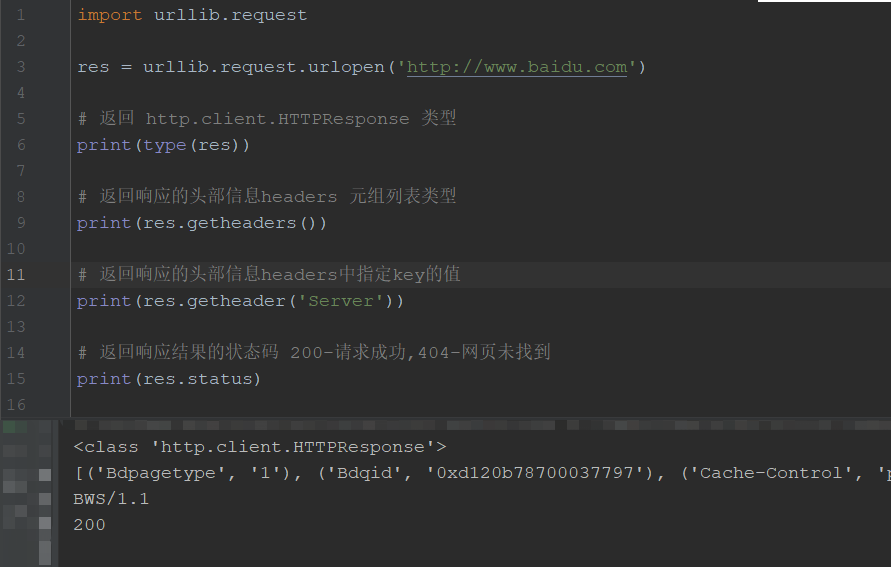
主要方法**read()、readinto()、getheader()、getheaders()、fileno()**方法、

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

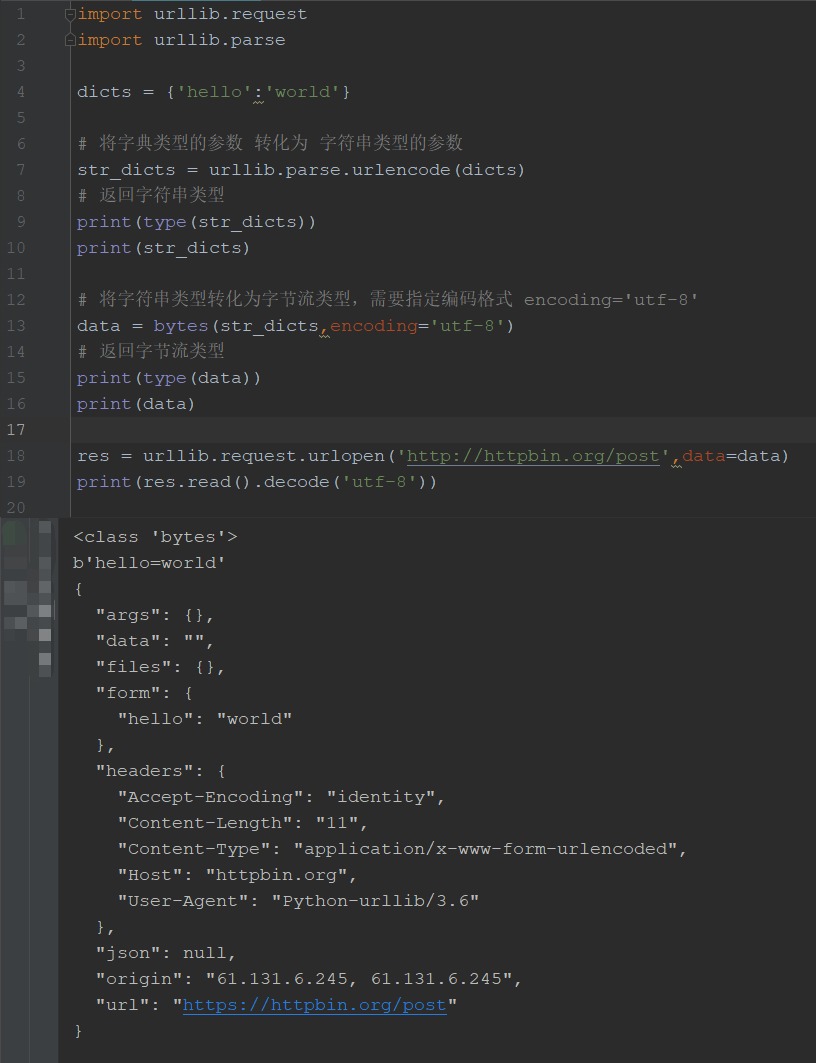
read()方法返回网页的内容，类型为Byte类型





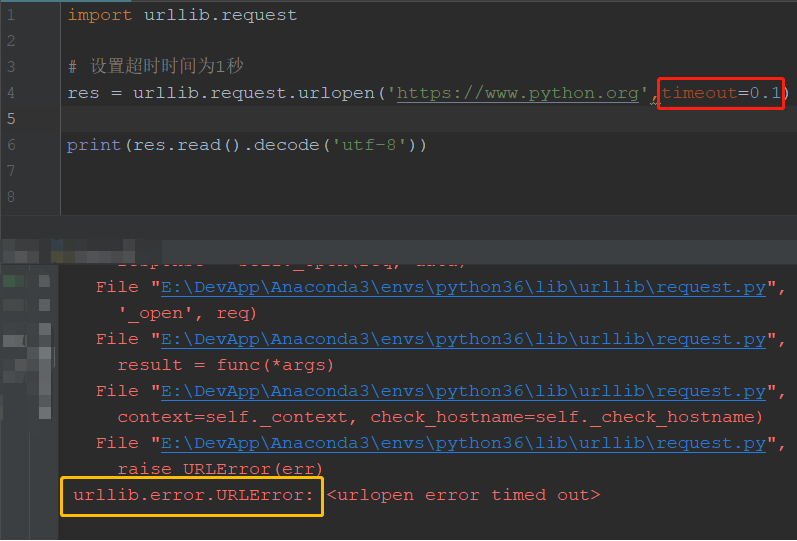
#### data参数

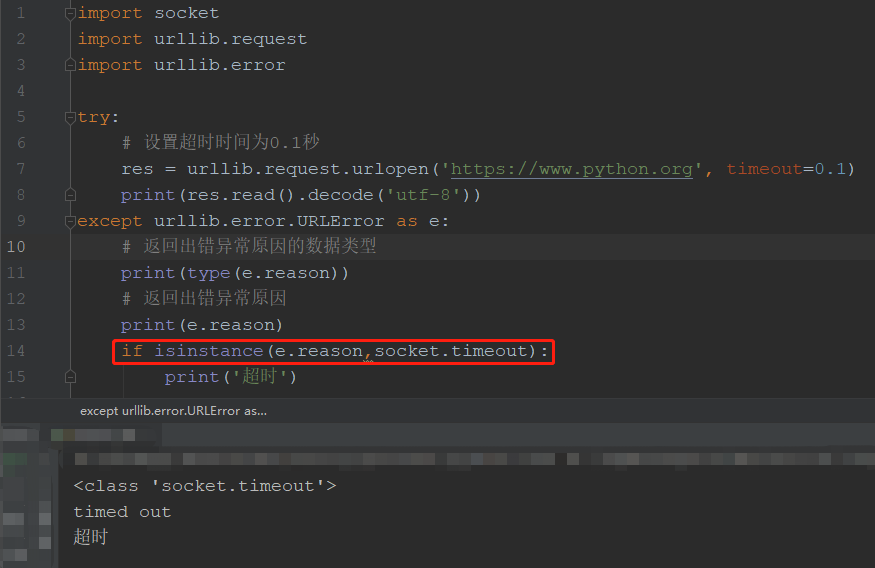
data参数是可选的，如果要添加该参数，需要使用bytes()方法将参数转化为字节流格式的内容，即bytes类型



#### timeout参数

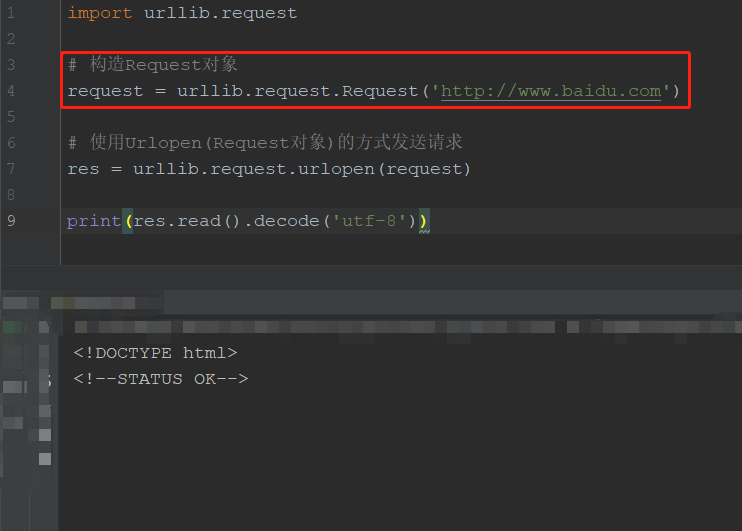
用于设置超时时间，单位为秒，如果请求超出设置的这个时间，还没有得到响应，就会抛出异常，如果不指定会使用全局默认时间，它支持Http、Https、FTP请求

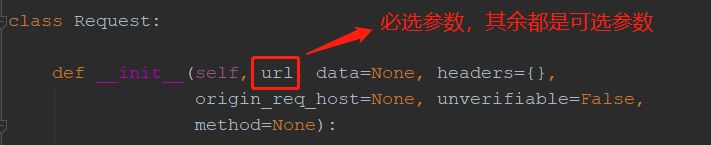




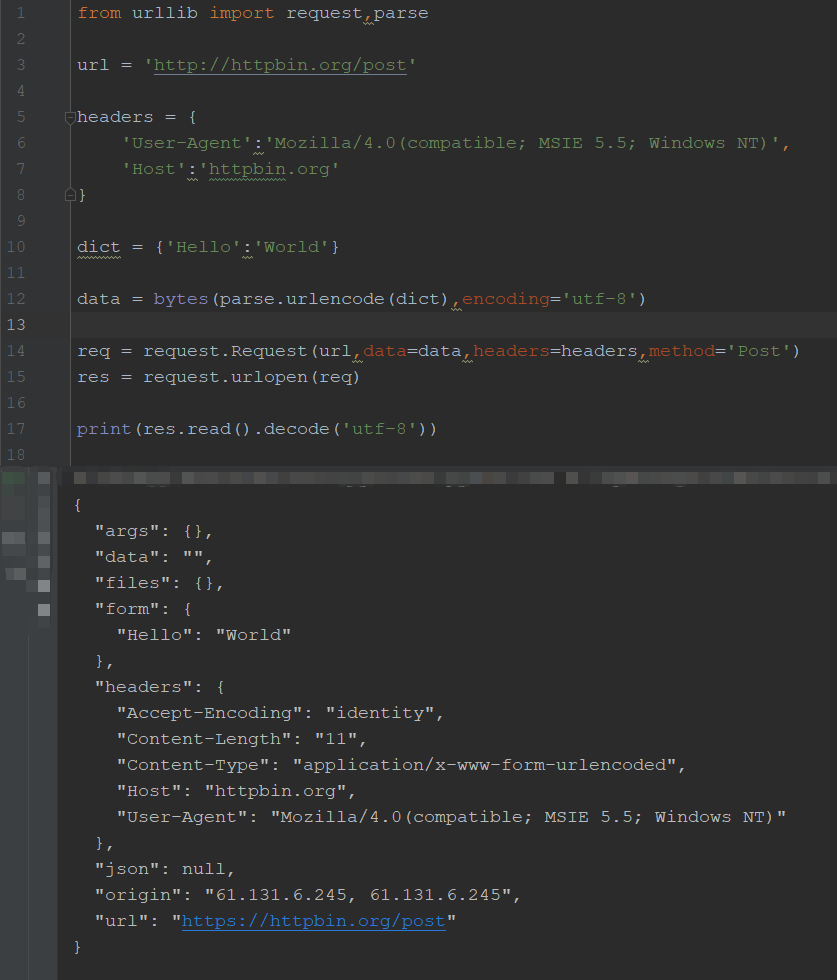
### Request类

以下先来感受下Request的用法

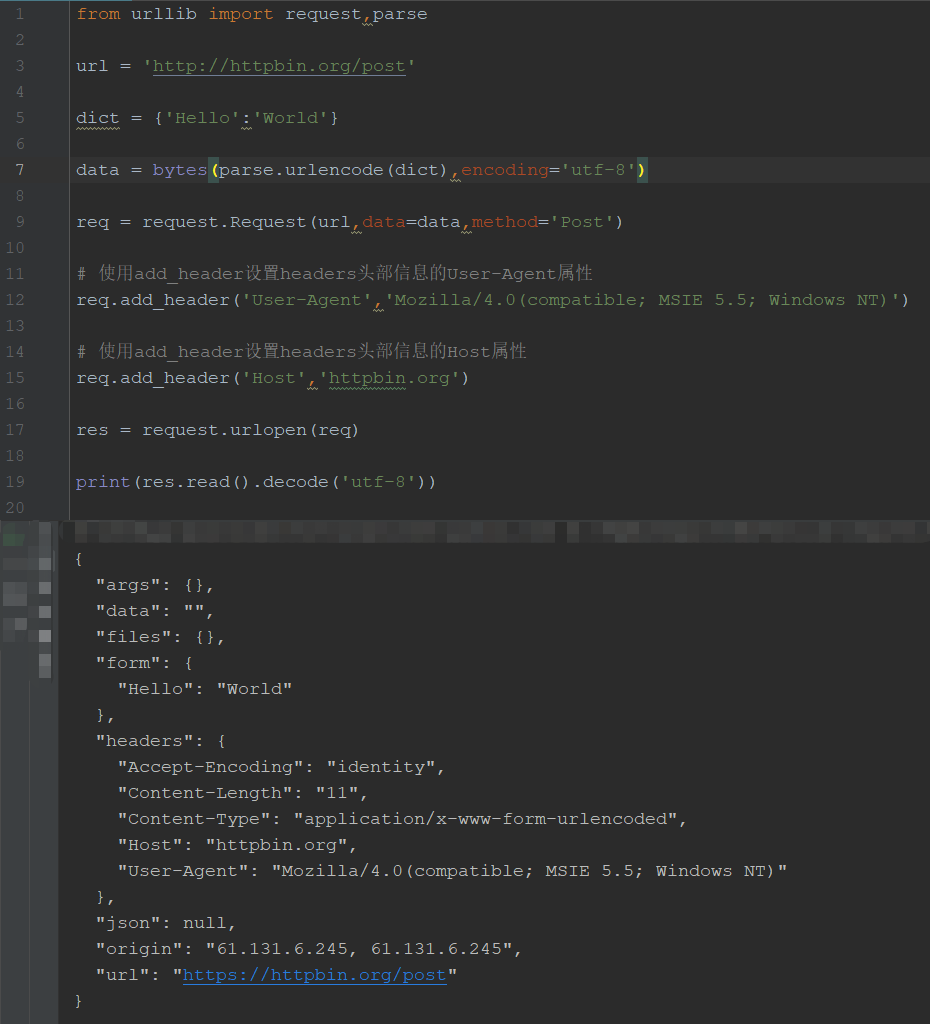




第1个参数url是必选参数，其余都是可选参数



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

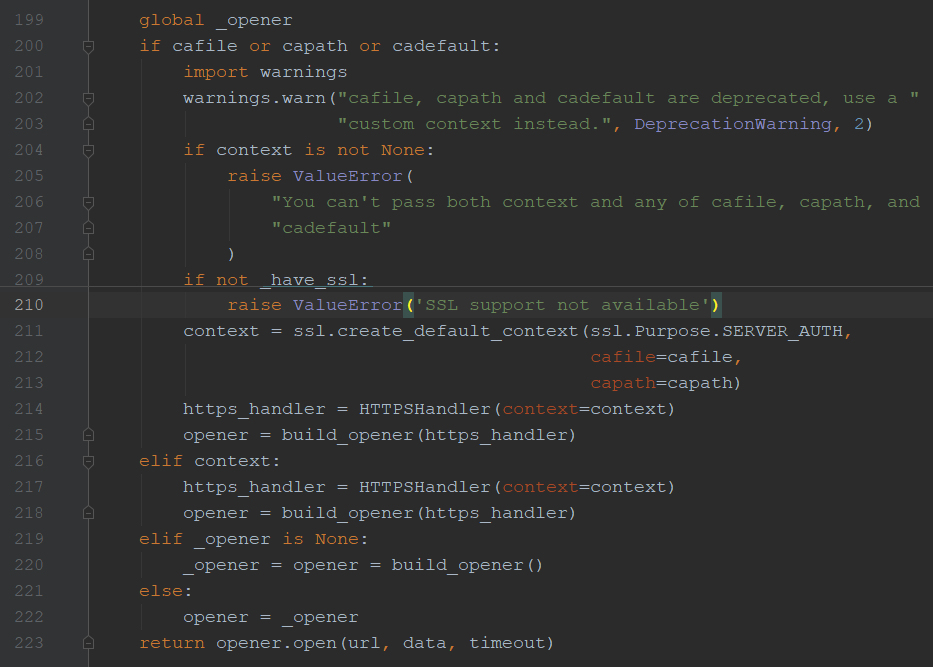
**HTTPPasswordMgr**：用于管理密码，它维护了用户名和密码的表

**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

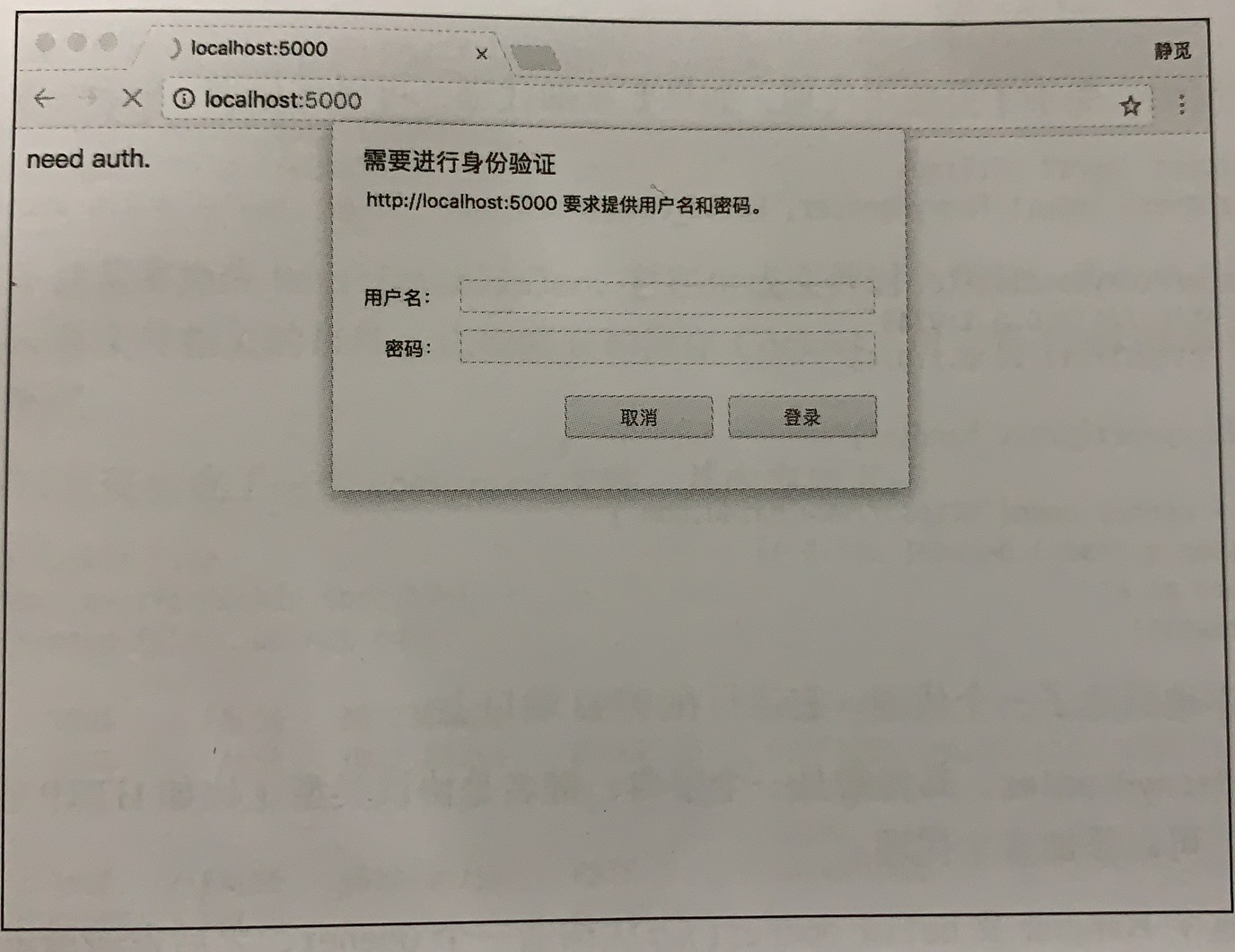
我们也可以称为Opener，urlopen()这个方法实际上就是urllib给我们提供的一个Opener。

可以查看源码代码片段

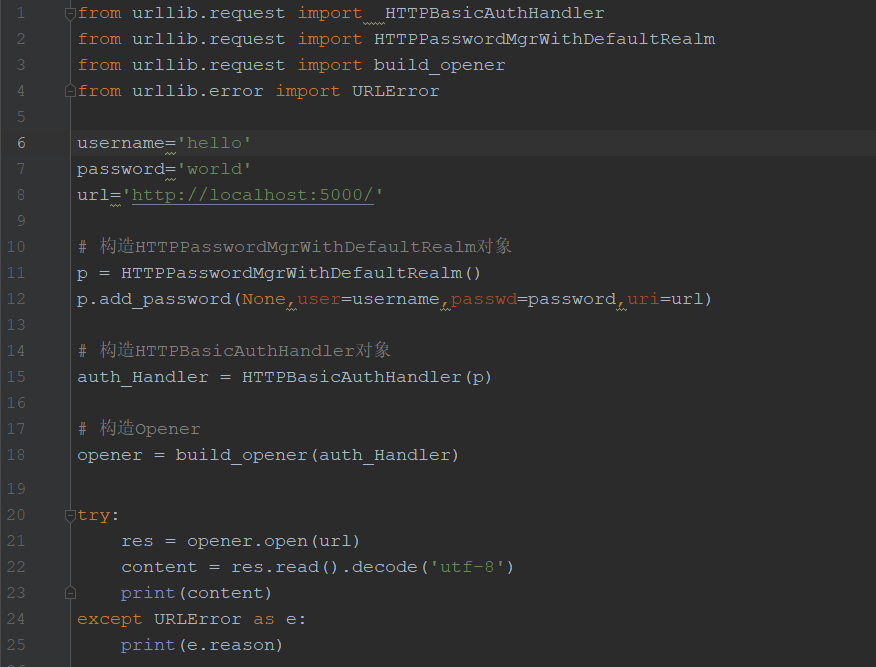


Opener可以使用Open()方法，返回的类型和urlopen()如出一辙，它和Handler有什么关系？**就是利用Handler来构建Opener**

##### 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：

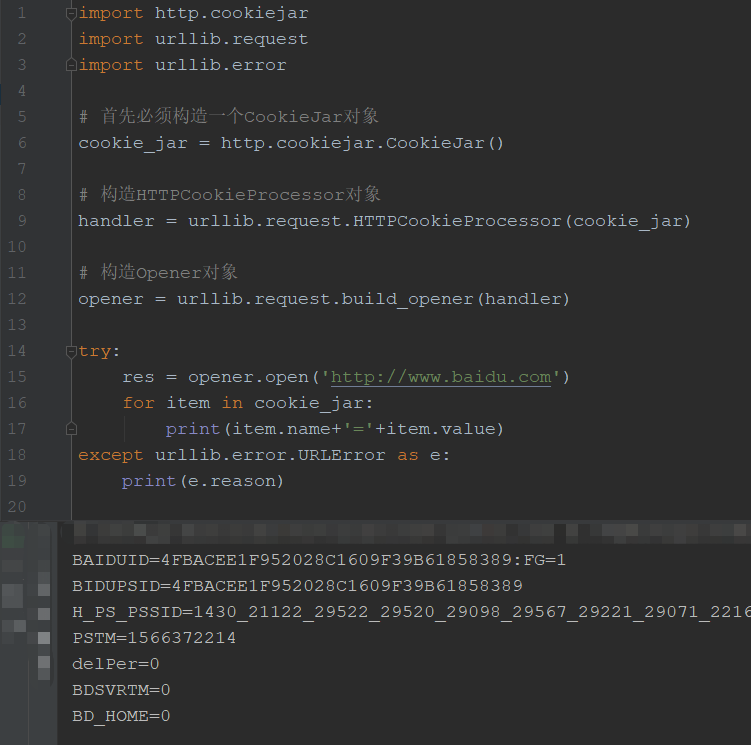


ProxyHandler其参数是一个字典，键名是协议类型(比如HTTP或者HTTPS)，键值是代理链接，可以添加多个代理

##### 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

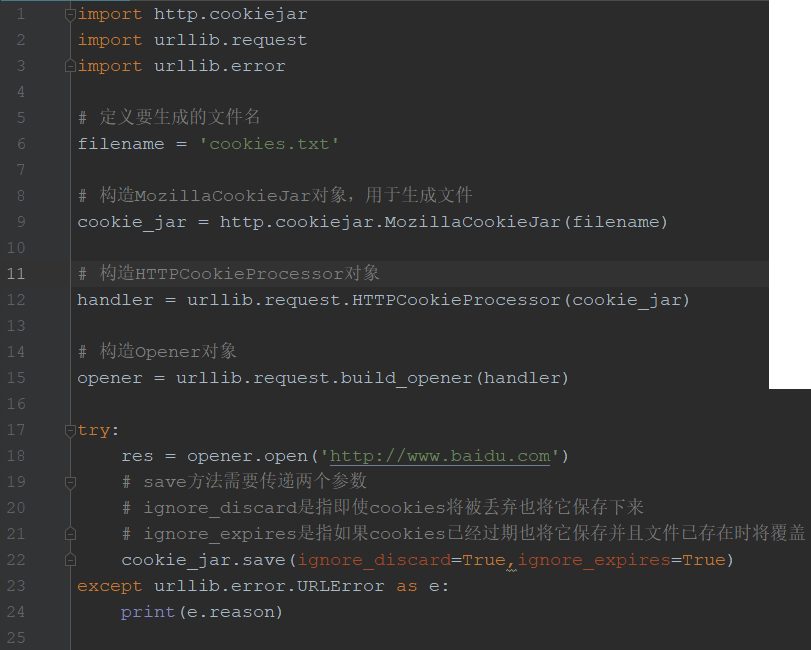


###### MozillaCookieJar对象

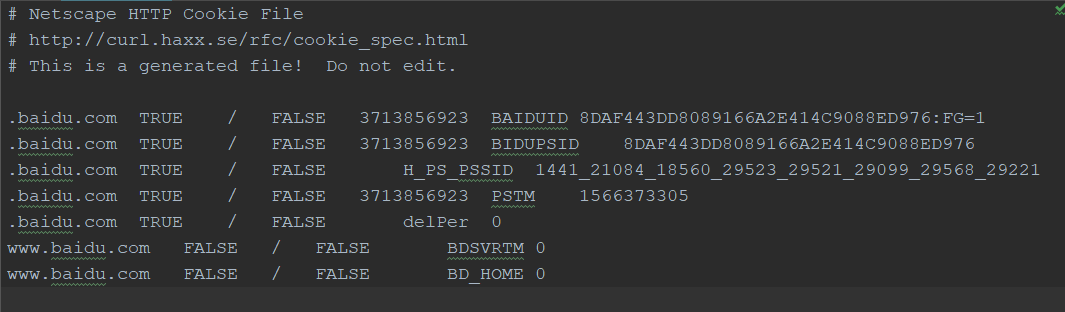
它是CookieJar的子类，可以用来处理Cookies和文件的相关事件，比如读取和保存Cookies，可以将Cookies保存呈Mozilla型浏览器的Cookies格式

**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

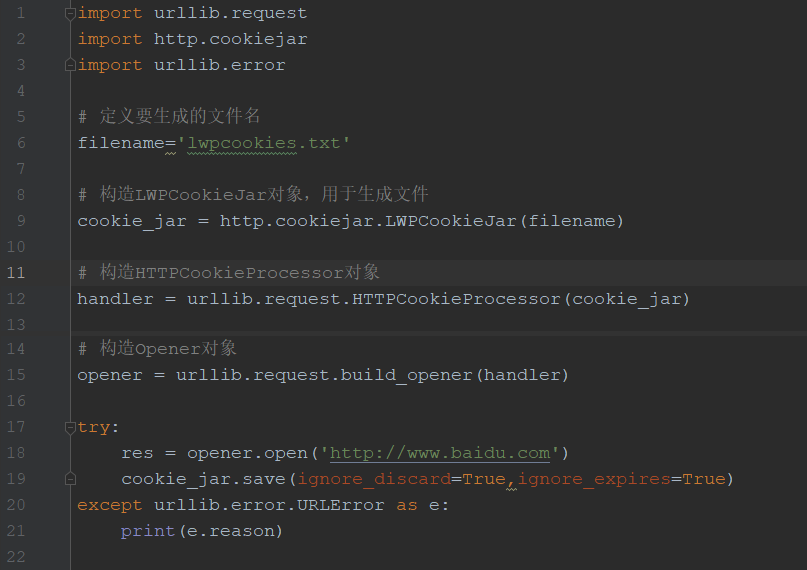


运行后生成的cookies.txt文件格式如下：

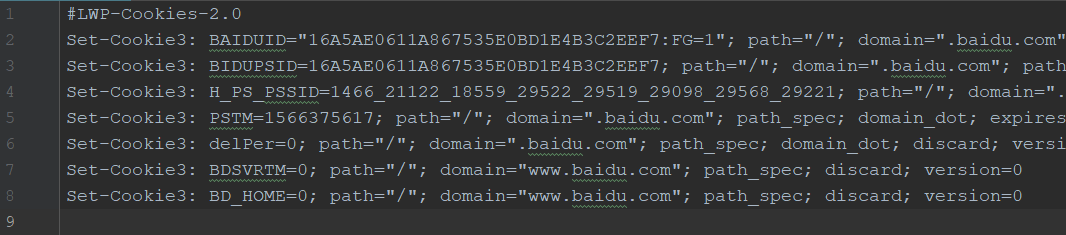


###### LWPCookieJar对象

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

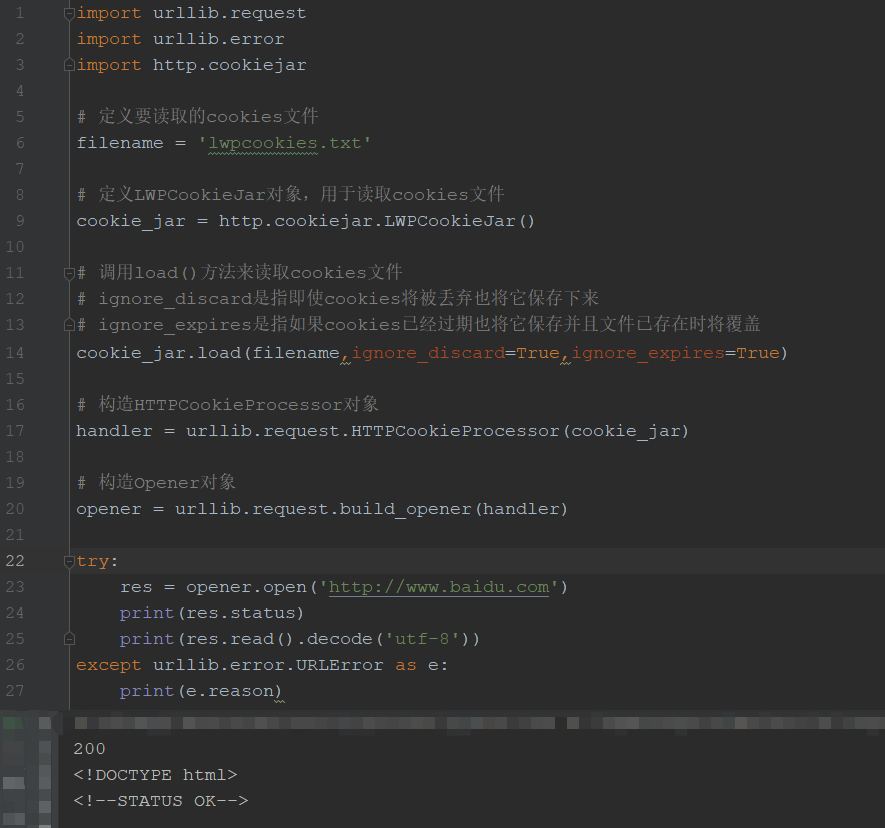


运行后生成的lwp格式的cookies.txt文件格式如下：



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**：异常处理模块（常用）

**parse**：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）

**robotparser**：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