

## 抓取网页的含义和URL基本构成

### 1、网络爬虫的定义

网络爬虫，即Web Spider，是一个很形象的名字。

把互联网比喻成一个蜘蛛网，那么Spider就是在网上爬来爬去的蜘蛛。

网络蜘蛛是通过网页的链接地址来寻找网页的。

从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其它链接地址，

然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页都抓取完为止。

如果把整个互联网当成一个网站，那么网络蜘蛛就可以用这个原理把互联网上所有的网页都抓取下来。

这样看来，网络爬虫就是一个爬行程序，一个抓取网页的程序。

网络爬虫的基本操作是抓取网页。

那么如何才能随心所欲地获得自己想要的页面？

我们先从URL开始。

### 2、浏览网页的过程

抓取网页的过程其实和读者平时使用IE浏览器浏览网页的道理是一样的。

比如说你在浏览器的地址栏中输入 `www.baidu.com` 这个地址。

打开网页的过程其实就是浏览器作为一个浏览的“客户端”，向服务器端发送了一次请求，把服务器端的文件“抓”到本地，再进行解释、展现。

HTML是一种标记语言，用标签标记内容并加以解析和区分。

浏览器的功能是将获取到的HTML代码进行解析，然后将原始的代码转变成我们直接看到的网页面。

### 3、URI的概念和举例

简单的来讲，URL就是在浏览器端输入的 `www.baidu.com` 这个字符串。

在理解URL之前，首先要理解URI的概念。

什么是URI？

Web上每种可用的资源，如 HTML文档、图像、视频片段、程序等都由一个通用资源标志符(Universal Resource Identifier, URI)进行定位。

URI通常由三部分组成：

- ①访问资源的命名机制；
- ②存放资源的主机名；
- ③资源自身的名称，由路径表示。

如下面的URI：

`http://www.why.com.cn/myhtml/html1223/`

我们可以这样解释它：

- ①这是一个可以通过HTTP协议访问的资源，
- ②位于主机 `www.webmonkey.com.cn`上，
- ③通过路径“/html/html40”访问。

### 4、URL的理解和举例

URL是URI的一个子集。它是Uniform Resource Locator的缩写，译为“统一资源定位符”。

通俗地说，URL是Internet上描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上。

采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。

URL的格式由三部分组成：

- ①第一部分是协议(或称为服务方式)。

②第二部分是存有该资源的主机IP地址(有时也包括端口号)。

③第三部分是主机资源的具体地址, 如目录和文件名等。

第一部分和第二部分用“://”符号隔开,

第二部分和第三部分用“/”符号隔开。

第一部分和第二部分是不可缺少的, 第三部分有时可以省略。

下面来看看两个URL的小例子。

### 1.HTTP协议的URL示例:

使用超级文本传输协议HTTP, 提供超级文本信息服务的资源。

例: <http://www.peopledaily.com.cn/channel/welcome.htm>

其计算机域名为www.peopledaily.com.cn。

超级文本文件(文件类型为.html)是在目录 /channel下的welcome.htm。

这是中国人民日报的一台计算机。

例: <http://www.rol.cn.net/talk/talk1.htm>

其计算机域名为www.rol.cn.net。

超级文本文件(文件类型为.html)是在目录/talk下的talk1.htm。

这是瑞得聊天室的地址, 可由此进入瑞得聊天室的第1室。

### 2. 文件的URL

用URL表示文件时, 服务器方式用file表示, 后面要有主机IP地址、文件的存取路 径(即目录)和文件名等信息。

有时可以省略目录和文件名, 但“/”符号不能省略。

例: <file://ftp.yoyodyne.com/pub/files/foobar.txt>

上面这个URL代表存放在主机ftp.yoyodyne.com上的pub/files/目录下的一个文件, 文件名是foobar.txt。

例: <file://ftp.yoyodyne.com/pub>

代表主机ftp.yoyodyne.com上的目录/pub。

例: <file://ftp.yoyodyne.com/>

代表主机ftp.yoyodyne.com的根目录。

爬虫最主要的处理对象就是URL, 它根据URL地址取得所需要的文件内容, 然后对它 进行进一步的处理。

因此, 准确地理解URL对理解网络爬虫至关重要。

## 利用urllib2通过指定的URL抓取网页内容

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读取出来，保存到本地。

类似于使用程序模拟IE浏览器的功能，把URL作为HTTP请求的内容发送到服务器端，然后读取服务器端的响应资源。

在Python中，我们使用urllib2这个组件来抓取网页。

urllib2是Python的一个获取URLs(Uniform Resource Locators)的组件。

它以urlopen函数的形式提供了一个非常简单的接口。

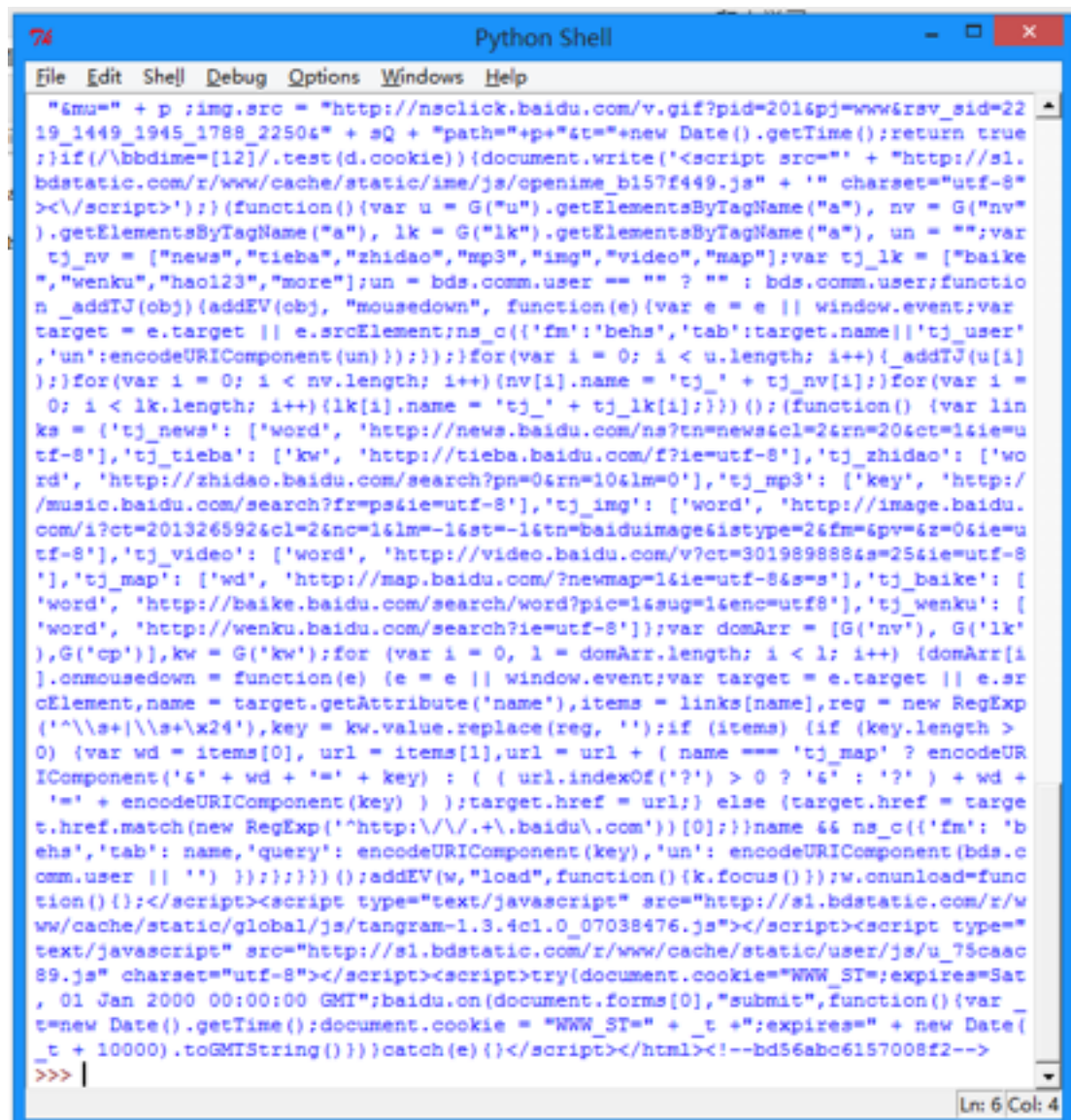
最简单的urllib2的应用代码只需要四行。

我们新建一个文件urllib2\_test01.py来感受一下urllib2的作用：

[python] view plaincopy

```
1. import urllib2
2. response = urllib2.urlopen('http://www.baidu.com/')
3. html = response.read()
4. print html
```

按下F5可以看到运行的结果：



```
File Edit Shell Debug Options Windows Help
"amu" + p ;img.src = "http://nsclick.baidu.com/v.gif?pid=2016pj=www&rsv_sid=22
19_1449_1945_1788_22504" + sQ + "path="+p+"&t="+new Date().getTime();return true
;if (/\/bbs\dime=[12]\/.test(d.cookie)){document.write("<script src=" + "http://sl
.bdstatic.com/r/www/cache/static/ime/js/openime_b157f449.js" + " charset=utf-8"
></script>");}(function(){var u = G("u").getElementsByName("a"), nv = G("nv"
).getElementsByName("a"), lk = G("lk").getElementsByName("a"), un = "";var
tj_nv = ["news","tieba","zhidao","mp3","img","video","map"];var tj_lk = ["baike
","wenku","hao123","more"];un = bds.comm.user == "" ? "" : bds.comm.user;functio
n _addTJ(obj){addEV(obj, "mousedown", function(e){var e = e || window.event;var
target = e.target || e.srcElement;ns_c({'fm':'behs','tab':target.name||'tj_user'
,'un':encodeURIComponent(un)});});for(var i = 0; i < u.length; i++){_addTJ(u[i]
);}for(var i = 0; i < nv.length; i++){nv[i].name = 'tj_' + tj_nv[i];}for(var i =
0; i < lk.length; i++){lk[i].name = 'tj_' + tj_lk[i];}}(function() {var lin
ks = {'tj_news': ['word', 'http://news.baidu.com/ns?tn=news&cl=2&rn=20&ct=1&ie=u
tf-8'],'tj_tieba': ['kw', 'http://tieba.baidu.com/f?ie=utf-8'],'tj_zhidao': ['wo
rd', 'http://zhidao.baidu.com/search?pn=0&rn=10&lm=0'],'tj_mp3': ['key', 'http:/
/music.baidu.com/search?fr=ps&ie=utf-8'],'tj_img': ['word', 'http://image.baidu
.com/i?ct=201326592&cl=2&nc=1&lm=-1&st=-1&tn=baiduimage&istype=2&fm=6pvr=6z=0&ie=u
tf-8'],'tj_video': ['word', 'http://video.baidu.com/v?ct=301989888&s=25&ie=utf-8
'],'tj_map': ['wd', 'http://map.baidu.com/?newmap=1&ie=utf-8&s=s'],'tj_baike': [
'word', 'http://baike.baidu.com/search?word?pic=1&sug=1&enc=utf8'],'tj_wenku': [
'word', 'http://wenku.baidu.com/search?ie=utf-8'];var domArr = [G('nv'), G('lk'
)],G('cp')],kw = G('kw');for (var i = 0, l = domArr.length; i < l; i++) {domArr[i
].onmousedown = function(e) {e = e || window.event;var target = e.target || e.sr
cElement,name = target.getAttribute('name'),items = links[name],reg = new RegExp
('^\s+|\s+\x24$'),key = kw.value.replace(reg, '');if (items) {if (key.length >
0) {var wd = items[0], url = items[1],url = url + ( name === 'tj_map' ? encodeUR
IComponent('&' + wd + '=' + key) : ( url.indexOf('?') > 0 ? '&' : '?' ) + wd +
'=' + encodeURIComponent(key) );target.href = url;} else {target.href = targe
t.href.match(new RegExp('^http:\/\/\.\.baidu\.com'))[0];}name && ns_c({'fm': 'b
ehs','tab': name,'query': encodeURIComponent(key),'un': encodeURIComponent(bds.c
omm.user || '' )});}}(function(){addEV(w,"load",function(){k.focus();});w.onunloa
d=function(){</script><script type="text/javascript" src="http://sl.bdstatic.com/r/w
ww/cache/static/global/js/tangram-1.3.4cl.0_07038476.js"></script><script type="
text/javascript" src="http://sl.bdstatic.com/r/www/cache/static/user/js/u_75caac
89.js" charset=utf-8"></script><script>try{document.cookie="WW ST=expires=Sat
, 01 Jan 2000 00:00:00 GMT";baidu.on(document.forms[0],"submit",function(){var
t=new Date().getTime();document.cookie = "WW ST=" + _t +":expires=" + new Date(
_t + 10000).toGMTString();})}catch(e){</script></html><!--bd56abc6157008f2-->
>>>
```

我们可以打开百度主页，右击，选择查看源代码（火狐OR谷歌浏览器均可），会发现也是完全一样的内容。

也就是说，上面这四行代码将我们访问百度时浏览器收到的代码们全部打印了出来。

这就是一个最简单的urllib2的例子。

除了"http:"，URL同样可以使用"ftp:"，"file:"等等来替代。

HTTP是基于请求和应答机制的：

客户端提出请求，服务端提供应答。

urllib2用一个Request对象来映射你提出的HTTP请求。

在它最简单的使用形式中你将用你要请求的地址创建一个Request对象，

通过调用urlopen并传入Request对象，将返回一个相关请求response对象，  
这个应答对象如同一个文件对象，所以你可以在Response中调用.read()。

我们新建一个文件urllib2\_test02.py来感受一下：

```
[python] view plaincopy
1. import urllib2
2. req = urllib2.Request('http://www.baidu.com')
3. response = urllib2.urlopen(req)
4. the_page = response.read()
5. print the_page
```

可以看到输出的内容和test01是一样的。  
urllib2使用相同的接口处理所有的URL头。例如你可以像下面那样创建一个ftp请求。

```
[python] view plaincopy
1. req = urllib2.Request('ftp://example.com/')
```

在HTTP请求时，允许你做额外的两件事。

## 1.发送data表单数据

这个内容相信做过Web端的都不会陌生，  
有时候你希望发送一些数据到URL(通常URL与CGI[通用网关接口]脚本，或其他WEB应用程序挂接)。

在HTTP中,这个经常使用熟知的POST请求发送。

这个通常在你提交一个HTML表单时由你的浏览器来做。

并不是所有的POSTs都来源于表单，你能够使用POST提交任意的数据到你自己的程序。

一般的HTML表单，data需要编码成标准形式。然后做为data参数传到Request对象。

编码工作使用urllib的函数而非urllib2。

我们新建一个文件urllib2\_test03.py来感受一下：

```
[python] view plaincopy
1. import urllib
2. import urllib2
3.
4. url = 'http://www.someserver.com/register.cgi'
5.
6. values = {'name' : 'WHY',
7.           'location' : 'SDU',
8.           'language' : 'Python' }
9.
```

```
10. data = urllib.urlencode(values) # 编码工作
11. req = urllib2.Request(url, data) # 发送请求同时传data表单
12. response = urllib2.urlopen(req) #接受反馈的信息
13. the_page = response.read() #读取反馈的内容
```

如果没有传送data参数，urllib2使用GET方式的请求。  
GET和POST请求的不同之处是POST请求通常有"副作用"，  
它们会由于某种途径改变系统状态(例如提交成堆垃圾到你的门口)。  
Data同样可以通过在Get请求的URL本身上面编码来传送。

[python] view plaincopy

```
1. import urllib2
2. import urllib
3.
4. data = {}
5.
6. data['name'] = 'WHY'
7. data['location'] = 'SDU'
8. data['language'] = 'Python'
9.
10. url_values = urllib.urlencode(data)
11. print url_values
12.
13. name=Somebody+Here&language=Python&location=Northampton
14. url = 'http://www.example.com/example.cgi'
15. full_url = url + '?' + url_values
16.
17. data = urllib2.open(full_url)
```

这样就实现了Data数据的Get传送。

## 2.设置Headers到http请求

有一些站点不喜欢被程序（非人为访问）访问，或者发送不同版本的内容到不同的浏览器。

默认的urllib2把自己作为“Python-urllib/x.y”(x和y是Python主版本和次版本号,例如Python-urllib/2.7),

这个身份可能会让站点迷惑，或者干脆不工作。

浏览器确认自己身份是通过User-Agent头，当你创建了一个请求对象，你可以给他一个包含头数据的字典。

下面的例子发送跟上面一样的内容，但把自身模拟成Internet Explorer。

[python] view plaincopy

```
1. import urllib
2. import urllib2
3.
4. url = 'http://www.someserver.com/cgi-bin/register.cgi'
5.
6. user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
7. values = {'name' : 'WHY',
8.           'location' : 'SDU',
9.           'language' : 'Python' }
10.
11. headers = { 'User-Agent' : user_agent }
12. data = urllib.urlencode(values)
13. req = urllib2.Request(url, data, headers)
14. response = urllib2.urlopen(req)
15. the_page = response.read()
```

## 异常的处理和HTTP状态码的分类

先来说一说HTTP的异常处理问题。

当urlopen不能够处理一个response时，产生urlError。

不过通常的Python APIs异常如ValueError,TypeError等也会同时产生。

HTTPError是urlError的子类，通常在特定HTTP URLs中产生。

### 1.URLError

通常，URLError在没有网络连接(没有路由到特定服务器)，或者服务器不存在的情况下产生。

这种情况下，异常同样会带有"reason"属性，它是一个tuple（可以理解为不可变的数组），

包含了一个错误号和一个错误信息。

我们建一个urllib2\_test06.py来感受一下异常的处理：

[python] view plaincopy

```
1. import urllib2
2.
3. req = urllib2.Request('http://www.baibai.com')
4.
5. try: urllib2.urlopen(req)
6.
7. except urllib2.URLError, e:
8.     print e.reason
```



按下F5，可以看到打印出来的内容是：

[Errno 11001] getaddrinfo failed

也就是说，错误号是11001，内容是getaddrinfo failed

## 2.HTTPError

服务器上每一个HTTP 应答对象response包含一个数字"状态码"。

有时状态码指出服务器无法完成请求。默认的处理程序会为你处理一部分这种应答。

例如:假如response是一个"重定向"，需要客户端从别的地址获取文档，urllib2将为你处理。

其他不能处理的，urlopen会产生一个HTTPError。

典型的错误包含"404"(页面无法找到)，"403"(请求禁止)，和"401"(带验证请求)。

HTTP状态码表示HTTP协议所返回的响应的状态。

比如客户端向服务器发送请求，如果成功地获得请求的资源，则返回的状态码为200，表示响应成功。

如果请求的资源不存在，则通常返回404错误。

HTTP状态码通常分为5种类型，分别以1~5五个数字开头，由3位整数组成：

-----  
----

200：请求成功 处理方式：获得响应的内容，进行处理

201：请求完成，结果是创建了新资源。新创建资源的URI可在响应的实体中得到 处理方式：爬虫中不会遇到

202：请求被接受，但处理尚未完成 处理方式：阻塞等待

204：服务器端已经实现了请求，但是没有返回新的信息。如果客户是用户代理，则无须为此更新自身的文档视图。 处理方式：丢弃

300：该状态码不被HTTP/1.0的应用程序直接使用，只是作为3XX类型回应的默认解释。存在多个可用的被请求资源。 处理方式：若程序中能够处理，则进行进一步处理，如果程序中不能处理，则丢弃

301：请求到的资源都会分配一个永久的URL，这样就可以在将来通过该URL来访问此资源 处理方式：重定向到分配的URL

302: 请求到的资源在一个不同的URL处临时保存 处理方式: 重定向到临时的URL

304 请求的资源未更新 处理方式: 丢弃

400 非法请求 处理方式: 丢弃

401 未授权 处理方式: 丢弃

403 禁止 处理方式: 丢弃

404 没有找到 处理方式: 丢弃

5XX 回应代码以“5”开头的状态码表示服务器端发现自己出现错误, 不能继续执行请求 处理方式: 丢弃

-----  
----

HTTPError实例产生后会有一个整型'code'属性, 是服务器发送的相关错误号。

Error Codes错误码

因为默认的处理程序处理了重定向(300以外号码), 并且100-299范围的号码指示成功, 所以你只能看到400-599的错误号码。

BaseHTTPServer.BaseHTTPRequestHandler.response是一个很有用的应答号码字典, 显示了HTTP协议使用的所有的应答号。

当一个错误号产生后, 服务器返回一个HTTP错误号, 和一个错误页面。

你可以使用HTTPError实例作为页面返回的应答对象response。

这表示和错误属性一样, 它同样包含了read,geturl,和info方法。

我们建一个urllib2\_test07.py来感受一下:

[python] view plaincopy

```
1. import urllib2
2. req = urllib2.Request('http://bbs.csdn.net/callmewhy')
3.
4. try:
5.     urllib2.urlopen(req)
6.
7. except urllib2.URLError, e:
8.
9.     print e.code
10.    #print e.read()
```

按下F5可以看见输出了404的错误码, 也就说没有找到这个页面。

### 3.Wrapping

所以如果你想为HTTPError或URLError做准备，将有两个基本的办法。推荐使用第二种。

我们建一个urllib2\_test08.py来示范一下第一种异常处理的方案：

```
[python] view plaincopy
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3. req = Request('http://bbs.csdn.net/callmewhy')
4.
5. try:
6.
7.     response = urlopen(req)
8.
9. except HTTPError, e:
10.
11.     print 'The server couldn\'t fulfill the request.'
12.
13.     print 'Error code: ', e.code
14.
15. except URLError, e:
16.
17.     print 'We failed to reach a server.'
18.
19.     print 'Reason: ', e.reason
20.
21. else:
22.     print 'No exception was raised.'
23.     # everything is fine
```

和其他语言相似，try之后捕获异常并且将其内容打印出来。

这里要注意的一点，except HTTPError 必须在第一个，否则except URLError将同样接受到HTTPError 。

因为HTTPError是URLError的子类，如果URLError在前面它会捕捉到所有的URLError（包括HTTPError）。

我们建一个urllib2\_test09.py来示范一下第二种异常处理的方案：

```
[python] view plaincopy
1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
```

```

3. req = Request('http://bbs.csdn.net/callmewhy')
4.
5. try:
6.
7.     response = urlopen(req)
8.
9. except URLError, e:
10.
11.     if hasattr(e, 'reason'):
12.
13.         print 'We failed to reach a server.'
14.
15.         print 'Reason: ', e.reason
16.
17.     elif hasattr(e, 'code'):
18.
19.         print 'The server couldn\'t fulfill the request.'
20.
21.         print 'Error code: ', e.code
22.
23. else:
24.     print 'No exception was raised.'
25.     # everything is fine

```

## Opener与Handler的介绍和实例应用

在开始后面的内容之前，先来解释一下urllib2中的两个方法：info and geturl

urlopen返回的应答对象response(或者HTTPError实例)有两个很有用的方法info()和geturl()

### 1.geturl():

这个返回获取的真实的URL，这个很有用，因为urlopen(或者opener对象使用的)或许会有重定向。获取的URL或许跟请求URL不同。

以人人中的一个超级链接为例，

我们建一个urllib2\_test10.py来比较一下原始URL和重定向的链接：

[python] view plaincopy

```

1. from urllib2 import Request, urlopen, URLError, HTTPError

```

```

2.
3.
4. old_url = 'http://rrurl.cn/b1UZuP'
5. req = Request(old_url)
6. response = urlopen(req)
7. print 'Old url :' + old_url
8. print 'Real url :' + response.geturl()

```

运行之后可以看到真正的链接指向的网址：

```

>>>
Old url :http://rrurl.cn/b1UZuP
Real url :http://www.polyu.edu.hk/polyuchallenge/best_of_the_best_elevator_pitch
award/bbca_voting_process.php?voted_team_id=670&section=1&bbca_year_id=3
>>> |

```

## 2.info():

这个返回对象的字典对象，该字典描述了获取的页面情况。通常是服务器发送的特定头headers。目前是httplib.HTTPMessage 实例。经典的headers包含"Content-length", "Content-type", 和其他内容。

我们建一个urllib2\_test11.py来测试一下info的应用：

[python] view plaincopy

```

1. from urllib2 import Request, urlopen, URLError, HTTPError
2.
3. old_url = 'http://www.baidu.com'
4. req = Request(old_url)
5. response = urlopen(req)
6. print 'Info():'
7. print response.info()

```

运行的结果如下，可以看到页面的相关信息：

```

>>> ===== RESTART =====
>>>
Info():
Date: Tue, 14 May 2013 06:10:01 GMT
Server: BWS/1.0
Content-Length: 10450
Content-Type: text/html; charset=utf-8
Cache-Control: private
Set-Cookie: BDSVRTM=4; path=/
Set-Cookie: H_PS_PSSID=2428_2362_1466_1945_1788_2249_2260_2251; path=/; domain=.
baidu.com
Set-Cookie: BAIDUID=5B39D8D917E85C3DEA80DAF96B7A3E9A:FG=1; expires=Tue, 14-May-4
3 06:10:01 GMT; path=/; domain=.baidu.com
Expires: Tue, 14 May 2013 06:10:01 GMT
P3P: CP=" OTI DSP COR IVA OUR IND COM "
Connection: Close

```

下面来说一说urllib2中的两个重要概念：Openers和Handlers。

## 1.Openers:

当你获取一个URL你使用一个opener(一个urllib2.OpenerDirector的实例)。

正常情况下，我们使用默认opener：通过urlopen。

但你能够创建个性的openers。

## **2.Handles:**

Openers使用处理器handlers，所有的“繁重”工作由handlers处理。

每个handlers知道如何通过特定协议打开URLs，或者如何处理URL打开时的各个方面。

例如HTTP重定向或者HTTP cookies。

如果你希望用特定处理器获取URLs你会想创建一个openers，例如获取一个能处理cookie的opener，或者获取一个不重定向的opener。

要创建一个 opener，可以实例化一个OpenerDirector，

然后调用.add\_handler(some\_handler\_instance)。

同样，可以使用build\_opener，这是一个更加方便的函数，用来创建 opener对象，他只需要一次函数调用。

build\_opener默认添加几个处理器，但提供快捷的方法来添加或更新默认处理器。

其他的处理器handlers你或许会希望处理代理，验证，和其他常用但有点特殊的情况。

install\_opener 用来创建（全局）默认opener。这个表示调用urlopen将使用你安装的opener。

Opener对象有一个open方法。

该方法可以像urlopen函数那样直接用来获取urls：通常不必调用install\_opener，除了为了方便。

说完了上面两个内容，下面我们来看一下基本认证的内容，这里会用到上面提及的Opener和Handler。

Basic Authentication 基本验证

为了展示创建和安装一个handler，我们将使用HTTPBasicAuthHandler。

当需要基础验证时，服务器发送一个header(401错误码) 请求验证。这个指定了scheme 和一个‘realm’，看起来像这样：Www-authenticate: SCHEME realm="REALM".

例如

Www-authenticate: Basic realm="cPanel Users"

客户端必须使用新的请求，并在请求头里包含正确的姓名和密码。这是“基础验证”，为了简化这个过程，我们可以创建一个HTTPBasicAuthHandler的实例，并让opener使用这个handler就可以啦。

HTTPBasicAuthHandler使用一个密码管理的对象来处理URLs和realms来映射用户名和密码。

如果你知道realm(从服务器发送来的头里)是什么，你就能使用HTTPPasswordMgr。

通常人们不关心realm是什么。那样的话，就能用方便的HTTPPasswordMgrWithDefaultRealm。

这个将在你为URL指定一个默认的用户名和密码。

这将在你为特定realm提供一个其他组合时得到提供。

我们通过给realm参数指定None提供给add\_password来指示这种情况。

最高层次的URL是第一个要求验证的URL。你传给.add\_password()更深层次的URLs将同样合适。

说了这么多废话，下面来用一个例子演示一下上面说到的内容。

我们建一个urllib2\_test12.py来测试一下info的应用：

[python] view plaincopy

```
1. # -*- coding: utf-8 -*-
2. import urllib2
3.
4. # 创建一个密码管理者
5. password_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
6.
7. # 添加用户名和密码
8.
9. top_level_url = "http://example.com/foo/"
10.
11. # 如果知道 realm, 我们可以使用他代替 ``None``.
```

```
12. # password_mgr.add_password(None, top_level_url, username, password)
13. password_mgr.add_password(None, top_level_url, 'why', '1223')
14.
15. # 创建了一个新的handler
16. handler = urllib2.HTTPBasicAuthHandler(password_mgr)
17.
18. # 创建 "opener" (OpenerDirector 实例)
19. opener = urllib2.build_opener(handler)
20.
21. a_url = 'http://www.baidu.com/'
22.
23. # 使用 opener 获取一个URL
24. opener.open(a_url)
25.
26. # 安装 opener.
27. # 现在所有调用 urllib2.urlopen 将用我们的 opener.
28. urllib2.install_opener(opener)
29.
30.
```

注意：以上的例子我们仅提供我们的HTTPBasicAuthHandler给build\_opener。

默认的openers有正常状况的handlers：ProxyHandler，UnknownHandler，HTTPHandler，HTTPDefaultErrorHandler，HTTPRedirectHandler，FTPHandler，FileHandler，HTTPErrorProcessor。

代码中的top\_level\_url 实际上可以是完整URL(包含"http:"，以及主机名及可选的端口号)。

例如：http://example.com/。

也可以是一个“authority”(即主机名和可选的包含端口号)。

例如：“example.com” or “example.com:8080”。

后者包含了端口号。

## urllib2的使用细节与抓站技巧

前面说到了urllib2的简单入门，下面整理了一部分urllib2的使用细节。



## 1.Proxy 的设置

urllib2 默认会使用环境变量 `http_proxy` 来设置 HTTP Proxy。

如果想在程序中明确控制 Proxy 而不受环境变量的影响，可以使用代理。

新建test14来实现一个简单的代理Demo：

```
[python] view plaincopy
1. import urllib2
2. enable_proxy = True
3. proxy_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
4. null_proxy_handler = urllib2.ProxyHandler({})
5. if enable_proxy:
6.     opener = urllib2.build_opener(proxy_handler)
7. else:
8.     opener = urllib2.build_opener(null_proxy_handler)
9. urllib2.install_opener(opener)
```

这里要注意的一个细节，使用 `urllib2.install_opener()` 会设置 `urllib2` 的全局 `opener`。

这样后面的使用会很方便，但不能做更细致的控制，比如想在程序中使用两个不同的 Proxy 设置等。

比较好的做法是不使用 `install_opener` 去更改全局的设置，而只是直接调用 `opener` 的 `open` 方法代替全局的 `urlopen` 方法。

## 2.Timeout 设置

在老版 Python 中（Python2.6前），`urllib2` 的 API 并没有暴露 Timeout 的设置，要设置 Timeout 值，只能更改 Socket 的全局 Timeout 值。

```
[python] view plaincopy
1. import urllib2
2. import socket
3. socket.setdefaulttimeout(10) # 10 秒钟后超时
4. urllib2.socket.setdefaulttimeout(10) # 另一种方式
```

在 Python 2.6 以后，超时可以通过 `urllib2.urlopen()` 的 `timeout` 参数直接设置。

```
[python] view plaincopy
1. import urllib2
2. response = urllib2.urlopen('http://www.google.com', timeout=10)
```

### 3.在 HTTP Request 中加入特定的 Header

要加入 header, 需要使用 Request 对象:

```
[python] view plaincopy
1. import urllib2
2. request = urllib2.Request('http://www.baidu.com/')
3. request.add_header('User-Agent', 'fake-client')
4. response = urllib2.urlopen(request)
5. print response.read()
```

对有些 header 要特别留意, 服务器会针对这些 header 做检查

User-Agent : 有些服务器或 Proxy 会通过该值来判断是否是浏览器发出的请求

Content-Type : 在使用 REST 接口时, 服务器会检查该值, 用来确定 HTTP Body 中的内容该怎样解析。常见的取值有:

application/xml : 在 XML RPC, 如 RESTful/SOAP 调用时使用

application/json : 在 JSON RPC 调用时使用

application/x-www-form-urlencoded : 浏览器提交 Web 表单时使用  
在使用服务器提供的 RESTful 或 SOAP 服务时, Content-Type 设置错误会导致服务器拒绝服务

### 4.Redirect

urllib2 默认情况下会针对 HTTP 3XX 返回码自动进行 redirect 动作, 无需人工配置。要检测是否发生了 redirect 动作, 只要检查一下 Response 的 URL 和 Request 的 URL 是否一致就可以了。

```
[python] view plaincopy
1. import urllib2
2. my_url = 'http://www.google.cn'
3. response = urllib2.urlopen(my_url)
4. redirected = response.geturl() == my_url
5. print redirected
6.
7. my_url = 'http://rrurl.cn/blUZuP'
8. response = urllib2.urlopen(my_url)
9. redirected = response.geturl() == my_url
10. print redirected
```

如果不想自动 redirect，除了使用更低层次的 httplib 库之外，还可以自定义 HTTPRedirectHandler 类。

```
[python] view plaincopy
1. import urllib2
2. class RedirectHandler(urllib2.HTTPRedirectHandler):
3.     def http_error_301(self, req, fp, code, msg, headers):
4.         print "301"
5.         pass
6.     def http_error_302(self, req, fp, code, msg, headers):
7.         print "303"
8.         pass
9.
10. opener = urllib2.build_opener(RedirectHandler)
11. opener.open('http://rrurl.cn/b1UZuP')
```

## 5.Cookie

urllib2 对 Cookie 的处理也是自动的。如果需要得到某个 Cookie 项的值，可以这么做：

```
[python] view plaincopy
1. import urllib2
2. import cookielib
3. cookie = cookielib.CookieJar()
4. opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
5. response = opener.open('http://www.baidu.com')
6. for item in cookie:
7.     print 'Name = '+item.name
8.     print 'Value = '+item.value
```

运行之后就会输出访问百度的Cookie值：

```
>>>
Name = BAIDUID
Value = 67F3AB790249CDB85175F05DB33EB998:FG=1
Name = H_PS_PSSID
Value = 2428_1455_1945_1788_2250
Name = BDSVRTM
Value = 17
```

## 6.使用 HTTP 的 PUT 和 DELETE 方法

urllib2 只支持 HTTP 的 GET 和 POST 方法，如果要使用 HTTP PUT 和 DELETE，只能使用比较低层的 httplib 库。虽然如此，我们还是能通过下面的方式，使 urllib2 能够发出 PUT 或DELETE 的请求：

```
[python] view plaincopy
1. import urllib2
2. request = urllib2.Request(uri, data=data)
3. request.get_method = lambda: 'PUT' # or 'DELETE'
4. response = urllib2.urlopen(request)
```

## 7.得到 HTTP 的返回码

对于 200 OK 来说，只要使用 urlopen 返回的 response 对象的 getcode() 方法就可以得到 HTTP 的返回码。但对其它返回码来说，urlopen 会抛出异常。这时候，就要检查异常对象的 code 属性了：

```
[python] view plaincopy
1. import urllib2
2. try:
3.     response = urllib2.urlopen('http://bbs.csdn.net/why')
4. except urllib2.HTTPError, e:
5.     print e.code
```

## 8.Debug Log

使用 urllib2 时，可以通过下面的方法把 debug Log 打开，这样收发包的内容就会在屏幕上打印出来，方便调试，有时可以省去抓包的工作

```
[python] view plaincopy
1. import urllib2
2. httpHandler = urllib2.HTTPHandler(debuglevel=1)
3. httpsHandler = urllib2.HTTPSHandler(debuglevel=1)
4. opener = urllib2.build_opener(httpHandler, httpsHandler)
5. urllib2.install_opener(opener)
6. response = urllib2.urlopen('http://www.google.com')
```

这样就可以看到传输的数据包内容了：

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
send: 'GET / HTTP/1.1\r\nAccept-Encoding: identity\r\nHost: www.google.com\r\nCo
nnection: close\r\nUser-Agent: Python-urllib/2.7\r\n\r\n'
reply: 'HTTP/1.1 302 Found\r\n'
header: Location: http://www.google.com.hk/url?sa=p&hl=zh-CN&pref=hkredirect&pva
l=yes&q=http://www.google.com.hk/&ust=1368520013687696&usq=AFQjCNGXt4R3CJfVhn6PA
sS7UgXi7AqX_Q
header: Cache-Control: private
header: Content-Type: text/html; charset=UTF-8
header: Set-Cookie: PREF=ID=f72cff61912d8358:FF=0:NW=1:TM=1368519983:LM=13685199
83:S=fTQMGf13MLM14v_: expires=Thu, 14-May-2015 08:26:23 GMT; path=/; domain=.go
ogle.com
header: Date: Tue, 14 May 2013 08:26:23 GMT
header: Server: gws
header: Content-Length: 376
header: X-XSS-Protection: 1; mode=block
header: X-Frame-Options: SAMEORIGIN
header: Connection: close
send: 'GET /url?sa=p&hl=zh-CN&pref=hkredirect&pval=yes&q=http://www.google.com.h
k/&ust=1368520013687696&usq=AFQjCNGXt4R3CJfVhn6PAasS7UgXi7AqX_Q HTTP/1.1\r\nAccep
t-Encoding: identity\r\nHost: www.google.com.hk\r\nConnection: close\r\nUser-Age
nt: Python-urllib/2.7\r\n\r\n'
reply: 'HTTP/1.1 302 Found\r\n'
header: X-Frame-Options: ALLOWALL
header: Location: http://www.google.com.hk/
header: Cache-Control: private
header: Content-Type: text/html; charset=UTF-8
header: Set-Cookie: PREF=ID=a37fa0c30a92618b:FF=2:LD=zh-CN:NW=1:TM=1368519983:LM
=1368519983:S=qqh_Q2YkTHXIY-GX; expires=Thu, 14-May-2015 08:26:23 GMT; path=/; d
omain=.google.com.hk
header: Set-Cookie: NID=67=GuQ_VcymquxCtv9Lg7UGZJP10rtqubT41FR6ss0iha7x2PyJsquevc
A8xoj76M9batchP0oOxnn192wuZuJKUpLMqr-38pMH3dw15C8PAWDANhNclY_A56cFscKGkFFhz; exp
ires=Wed, 13-Nov-2013 08:26:23 GMT; path=/; domain=.google.com.hk; HttpOnly
header: P3P: CP="This is not a P3P policy! See http://www.google.com/support/acc
counts/bin/answer.py?hl=en&answer=151657 for more info."
```

## 9.表单的处理

登录必要填表，表单怎么填？

首先利用工具截取所要填表的内容。

比如我一般用firefox+httpfox插件来看看自己到底发送了些什么包。

以verycd为例，先找到自己发的POST请求，以及POST表单项。

可以看到verycd的话需要填

username,password,continueURI,fk,login\_submit这几项，其中fk是随机生成的（其实不太随机，看上去像是把epoch时间经过简单的编码生成的），需要从网页获取，也就是说得先访问一次网页，用正则表达式等工具截取返回数据中的fk项。continueURI顾名思义可以随便写，

login\_submit是固定的，这从源码可以看出。还有username，password那就很显然了：

```
[python] view plaincopy
1. # -*- coding: utf-8 -*-
2. import urllib
3. import urllib2
4. postdata=urllib.urlencode({
5.     'username':'汪小光',
6.     'password':'why888',
7.     'continueURI':'http://www.verycd.com/',
8.     'fk':'',
9.     'login_submit':'登录'
10. })
11. req = urllib2.Request(
12.     url = 'http://secure.verycd.com/signin',
13.     data = postdata
14. )
15. result = urllib2.urlopen(req)
16. print result.read()
```

## 10.伪装成浏览器访问

某些网站反感爬虫的到访，于是对爬虫一律拒绝请求  
这时候我们需要伪装成浏览器，这可以通过修改http包中的header来实现

```
[python] view plaincopy
1. #...
2.
3. headers = {
4.     'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:
5.         1.9.1.6) Gecko/20091201 Firefox/3.5.6'
6. }
7. req = urllib2.Request(
8.     url = 'http://secure.verycd.com/signin/*/http://
9.         www.verycd.com/',
10.     data = postdata,
11.     headers = headers
12. )
13. #...
```

## 11.对付"反盗链"

某些站点有所谓的反盗链设置，其实说穿了很简单，  
就是检查你发送请求的header里面，referer站点是不是他自己，  
所以我们只需要像把headers的referer改成该网站即可，以cnbeta为例：

```
#...
```

```
headers = {
```

```
'Referer':'http://www.cnbeta.com/articles'
}
#...
```

headers是一个dict数据结构，你可以放入任何想要的header，来做一些伪装。

例如，有些网站喜欢读取header中的X-Forwarded-For来看看人家的真实IP，可以直接把X-Forwarded-For改了。

## 一个简单的百度贴吧的小爬虫

```
[python] view plaincopy
1. # -*- coding: utf-8 -*-
2. #-----
3. # 程序：百度贴吧爬虫
4. # 版本：0.1
5. # 作者：why
6. # 日期：2013-05-14
7. # 语言：Python 2.7
8. # 操作：输入带分页的地址，去掉最后面的数字，设置一下起始页数和终点页数。
9. # 功能：下载对应页码内的所有页面并存储为html文件。
10. #-----
11.
12. import string, urllib2
13.
14. #定义百度函数
15. def baidu_tieba(url, begin_page, end_page):
16.     for i in range(begin_page, end_page+1):
17.         sName = string.zfill(i,5) + '.html' #自动填充成六位的文件名
18.         print '正在下载第' + str(i) + '个网页，并将其存储
           为' + sName + '.....'
19.         f = open(sName, 'w+')
20.         m = urllib2.urlopen(url + str(i)).read()
21.         f.write(m)
22.         f.close()
23.
24.
25. #----- 在这里输入参数 -----
26.
27. # 这个是山东大学的百度贴吧中某一个帖子的地址
28. #bdurl = 'http://tieba.baidu.com/p/2296017831?pn='
29. #iPostBegin = 1
30. #iPostEnd = 10
31.
```

```
32.bdurl = str(raw_input(u'请输入贴吧的地址，去掉pn=后面的数字: \n'))
33.begin_page = int(raw_input(u'请输入开始的页数: \n'))
34.end_page = int(raw_input(u'请输入终点的页数: \n'))
35. #----- 在这里输入参数 -----
36.
37.
38. #调用
39.baidu_tieba.bdurl,begin_page,end_page)
```

## Python中的正则表达式教程

接下来准备用糗百做一个爬虫的小例子。

但是在这之前，先详细的整理一下Python中的正则表达式的相关内容。

正则表达式在Python爬虫中的作用就像是老师点名时用的花名册一样，是必不可少的神兵利器。

### 一、正则表达式基础

#### 1.1.概念介绍

正则表达式是用于处理字符串的强大工具，它并不是Python的一部分。

其他编程语言中也有正则表达式的概念，区别只在于不同的编程语言实现支持的语法数量不同。

它拥有自己独特的语法以及一个独立的处理引擎，在提供了正则表达式的语言里，正则表达式的语法都是一样的。

下图展示了使用正则表达式进行匹配的流程：





正则表达式的大致匹配过程是：

- 1.依次拿出表达式和文本中的字符比较，
- 2.如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。
- 3.如果表达式中有量词或边界，这个过程会稍微有一些不同。

下图列出了Python支持的正则表达式元字符和语法：

语法	说明	表达式实例	完整匹配的字符串
字符			
一般字符	匹配自身	abc	abc
.	匹配任意除换行符“\n”外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符“需要匹配，可以使用\“或者字符集[“]， 字符集（字符类），对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或 [a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是 abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字 符集中如果要使用[]、-或^，可以在前面加上反斜杠[]，或把[] 、-放在第一个字符，把^放在非第一个字符。	a\c a\ c	a.c a c
[...]		a[bcd]e	abe ace ade
预定义字符集（可以写在字符集[...]中）			
\d	数字：[0-9]	a\d c	a1c
\D	非数字：[^0-9]	a\D c	abc
\s	空白字符：[<空格> \f \n \r \t]	a\s c	a c
\S	非空白字符：[^<空格>]	a\S c	abc
\w	单词字符：[A-Za-z0-9_]	a\w c	abc
\W	非单词字符：[^A-Za-z0-9_]	a\W c	a c
数量词（用在字符或[...]之后）			
*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹 配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n} 变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗匹配字符串中的字符）			
^	匹配字符串开头。 在单行模式中匹配每一行的开头。	^abc	abc
\$	匹配字符串末尾。 在单行模式中匹配每一行的末尾。	abc\$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配w和W之间。	a\b bc	a bc
\B	[^b]	a Bbc	abc
圆括号、分组			
()	代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配 右边的表达式。 如果 没有被包括在()中，则它的范围是整个正则表达式。 被括起来的表达式将作为分组，从表达式左边开始每遇到一 个分组的左括号“(", 编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式 中的 仅在分组中有效。	abc def	abc def
(...)		(abc){2} a(123 456)c	abccabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abccabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?...)	(...)的不分组版本，用于使用 或后接数量词。	(?abc){2}	abccabc
(?:msux)	ilmsux的每个字符代表一个匹配模式，只能用在正则表达式 的开头，可选多个。匹配模式将在下文中介绍。	(?:)abc	Abc
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!=...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配 yes-pattern，否则需要匹配no-pattern。 no-pattern可以省略。	(\d)abc(?(1)\d abc)	1abc2 abccabc

## 1.2. 数量词的贪婪模式与非贪婪模式

正则表达式通常用于在文本中查找匹配的字符串。

贪婪模式，总是尝试匹配尽可能多的字符；

非贪婪模式则相反，总是尝试匹配尽可能少的字符。

Python里数量词默认是贪婪的。

例如：正则表达式"ab\*"如果用于查找"abbbc"，将找到"abbb"。

而如果使用非贪婪的数量词"ab\*?"，将找到"a"。

## 1.3. 反斜杠的问题

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。

假如你需要匹配文本中的字符"\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\\"：

第一个和第三个用于在编程语言里将第二个和第四个转义成反斜杠，转换成两个反斜杠\\后再在正则表达式里转义成一个反斜杠用来匹配反斜杠\。

这样显然是非常麻烦的。

Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r\"表示。

同样，匹配一个数字的\"d\"可以写成r\"d\"。

有了原生字符串，妈妈再也不用担心我的反斜杠问题~

## 二、介绍re模块

### 2.1. Compile

Python通过re模块提供对正则表达式的支持。

使用re的一般步骤是：

Step1：先将正则表达式的字符串形式编译为Pattern实例。

Step2：然后使用Pattern实例处理文本并获得匹配结果（一个Match实例）。

Step3：最后使用Match实例获得信息，进行其他的操作。

我们新建一个re01.py来试验一下re的应用：

[python] view plaincopy

```
1. # -*- coding: utf-8 -*-
2. #一个简单的re实例，匹配字符串中的hello字符串
3.
```

```

4. #导入re模块
5. import re
6.
7. # 将正则表达式编译成Pattern对象，注意hello前面的r的意思是“原生字符串”
8. pattern = re.compile(r'hello')
9.
10. # 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
11. match1 = pattern.match('hello world!')
12. match2 = pattern.match('helloo world!')
13. match3 = pattern.match('helllo world!')
14.
15. #如果match1匹配成功
16. if match1:
17.     # 使用Match获得分组信息
18.     print match1.group()
19. else:
20.     print 'match1匹配失败!'
21.
22.
23. #如果match2匹配成功
24. if match2:
25.     # 使用Match获得分组信息
26.     print match2.group()
27. else:
28.     print 'match2匹配失败!'
29.
30.
31. #如果match3匹配成功
32. if match3:
33.     # 使用Match获得分组信息
34.     print match3.group()
35. else:
36.     print 'match3匹配失败!'

```

可以看到控制台输出了匹配的三个结果：

```

# 将正则表达式编译成Pattern对象，注意hello前面的r的意思是
pattern = re.compile(r'hello')

# 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
match1 = pattern.match('hello world!')
match2 = pattern.match('helloo world!')
match3 = pattern.match('helllo world!')

#如果match1匹配成功

```

```

File Edit Shell Debug Options
Python 2.7.3 (default, Apr 1
32
Type "copyright", "credits"
>>> =====
>>>
hello
hello
match3匹配失败!

```

下面来具体看看代码中的关键方法。

★ `re.compile(strPattern[, flag])`:

这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。

第二个参数flag是匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如`re.I`或`re.M`。

另外，你也可以在regex字符串中指定模式，比如`re.compile('pattern', re.I | re.M)`与`re.compile('(?im)pattern')`是等价的。

可选值有：

- `re.I`(全拼：IGNORECASE): 忽略大小写（括号内是完整写法，下同）
- `re.M`(全拼：MULTILINE): 多行模式，改变'^'和'\$'的行为（参见上图）
- `re.S`(全拼：DOTALL): 点任意匹配模式，改变'.'的行为
- `re.L`(全拼：LOCALE): 使预定字符类 `\w \W \b \B \s \S` 取决于当前区域设定
- `re.U`(全拼：UNICODE): 使预定字符类 `\w \W \b \B \s \S \d \D` 取决于unicode定义的字符属性
- `re.X`(全拼：VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。

以下两个正则表达式是等价的：

[python] [view plaincopy](#)

```
1. # -*- coding: utf-8 -*-
2. #两个等价的re匹配,匹配一个小数
3. import re
4.
5. a = re.compile(r"""\d + # the integral part
6.                \.    # the decimal point
7.                \d *  # some fractional digits""", re.X)
8.
9. b = re.compile(r"\d+\.\d*")
10.
11. match11 = a.match('3.1415')
12. match12 = a.match('33')
13. match21 = b.match('3.1415')
14. match22 = b.match('33')
15.
16. if match11:
17.     # 使用Match获得分组信息
18.     print match11.group()
19. else:
20.     print u'match11不是小数'
21.
22. if match12:
23.     # 使用Match获得分组信息
24.     print match12.group()
```

```

25. else:
26.     print u'match12不是小数'
27.
28. if match21:
29.     # 使用Match获得分组信息
30.     print match21.group()
31. else:
32.     print u'match21不是小数'
33.
34. if match22:
35.     # 使用Match获得分组信息
36.     print match22.group()
37. else:
38.     print u'match22不是小数'

```

re提供了众多模块方法用于完成正则表达式的功能。这些方法可以使用Pattern实例的相应方法替代，唯一的好处是少写一行re.compile()代码，但同时也无法复用编译后的Pattern对象。这些方法将在Pattern类的实例方法部分一起介绍。如一开始的hello实例可以简写为：

[\[html\] view plain copy](#)

```

1. # -*- coding: utf-8 -*-
2. #一个简单的re实例，匹配字符串中的hello字符串
3. import re
4.
5. m = re.match(r'hello', 'hello world!')
6. print m.group()

```

re模块还提供了一个方法escape(string)，用于将string中的正则表达式元字符如\*/+/?等之前加上转义符再返回

## 2.2. Match

Match对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用Match提供的可读属性或方法来获取这些信息。

属性：

1. string: 匹配时使用的文本。
2. re: 匹配时使用的Pattern对象。
3. pos: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
4. endpos: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。

5. `lastindex`: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组, 将为`None`。
6. `lastgroup`: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组, 将为`None`。

方法:

1. `group([group1, ...])`:  
获得一个或多个分组截获的字符串; 指定多个参数时将以元组形式返回。`group1`可以使用编号也可以使用别名; 编号0代表整个匹配的子串; 不填写参数时, 返回`group(0)`; 没有截获字符串的组返回`None`; 截获了多次的组返回最后一次截获的子串。
2. `groups([default])`:  
以元组形式返回全部分组截获的字符串。相当于调用`group(1,2,...last)`。`default`表示没有截获字符串的组以这个值替代, 默认为`None`。
3. `groupdict([default])`:  
返回以有别名的组的别名为键、以该组截获的子串为值的字典, 没有别名的组不包含在内。`default`含义同上。
4. `start([group])`:  
返回指定的组截获的子串在string中的起始索引 (子串第一个字符的索引)。 `group`默认值为0。
5. `end([group])`:  
返回指定的组截获的子串在string中的结束索引 (子串最后一个字符的索引+1)。 `group`默认值为0。
6. `span([group])`:  
返回(`start(group)`, `end(group)`)。
7. `expand(template)`:  
将匹配到的分组代入`template`中然后返回。`template`中可以使用`\id`或`\g<id>`、`\g<name>`引用分组, 但不能使用编号0。`\id`与`\g<id>`是等价的; 但`\10`将被认为是第10个分组, 如果你想表达`\1`之后是字符'0', 只能使用`\g<1>0`。

下面来用一个py实例输出所有的内容加深理解:

[python] view plaincopy

```
1. # -*- coding: utf-8 -*-  
2. #一个简单的match实例
```

```

3.
4. import re
5. # 匹配如下内容: 单词+空格+单词+任意字符
6. m = re.match(r'(\w+) (\w+)(?P<sign>.*)', 'hello world!')
7.
8. print "m.string:", m.string
9. print "m.re:", m.re
10. print "m.pos:", m.pos
11. print "m.endpos:", m.endpos
12. print "m.lastindex:", m.lastindex
13. print "m.lastgroup:", m.lastgroup
14.
15. print "m.group():" , m.group()
16. print "m.group(1,2):" , m.group(1, 2)
17. print "m.groups():" , m.groups()
18. print "m.groupdict():" , m.groupdict()
19. print "m.start(2):" , m.start(2)
20. print "m.end(2):" , m.end(2)
21. print "m.span(2):" , m.span(2)
22. print r"m.expand(r'\g<2> \g<1>\g<3>'):" , m.expand(r'\2 \1\3')
23.
24. ### output ###
25. # m.string: hello world!
26. # m.re: <_sre.SRE_Pattern object at 0x016E1A38>
27. # m.pos: 0
28. # m.endpos: 12
29. # m.lastindex: 3
30. # m.lastgroup: sign
31. # m.group(1,2): ('hello', 'world')
32. # m.groups(): ('hello', 'world', '!')
33. # m.groupdict(): {'sign': '!'}
34. # m.start(2): 6
35. # m.end(2): 11
36. # m.span(2): (6, 11)
37. # m.expand(r'\2 \1\3'): world hello!

```

## 2.3. Pattern

Pattern对象是一个编译好的正则表达式，通过Pattern提供的一系列方法可以对文本进行匹配查找。

Pattern不能直接实例化，必须使用re.compile()进行构造，也就是re.compile()返回的对象。

Pattern提供了几个可读属性用于获取表达式的相关信息：

1. pattern: 编译时用的表达式字符串。
2. flags: 编译时用的匹配模式。数字形式。



3. groups: 表达式中分组的数量。
4. groupindex: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。

可以用下面这个例子查看pattern的属性：

[python] [view plaincopy](#)

```
1. # -*- coding: utf-8 -*-
2. #一个简单的pattern实例
3.
4. import re
5. p = re.compile(r'(\w+) (\w+)(?P<sign>.*)', re.DOTALL)
6.
7. print "p.pattern:", p.pattern
8. print "p.flags:", p.flags
9. print "p.groups:", p.groups
10. print "p.groupindex:", p.groupindex
11.
12. ### output ###
13. # p.pattern: (\w+) (\w+)(?P<sign>.*)
14. # p.flags: 16
15. # p.groups: 3
16. # p.groupindex: {'sign': 3}
```

下面重点介绍一下pattern的实例方法及其使用。

## 1.match

match(string[, pos[, endpos]]) | re.match(pattern, string[, flags]):

这个方法将从string的pos下标处起尝试匹配pattern；

如果pattern结束时仍可匹配，则返回一个Match对象；

如果匹配过程中pattern无法匹配，或者匹配未结束就已到达endpos，则返回None。

pos和endpos的默认值分别为0和len(string)；

re.match()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

注意：这个方法并不是完全匹配。

当pattern结束时若string还有剩余字符，仍然视为成功。

想要完全匹配，可以在表达式末尾加上边界匹配符'\$'。

下面来看一个Match的简单案例：

[python] [view plaincopy](#)

```
1. # encoding: UTF-8
2. import re
```

```

3.
4. # 将正则表达式编译成Pattern对象
5. pattern = re.compile(r'hello')
6.
7. # 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
8. match = pattern.match('hello world!')
9.
10. if match:
11.     # 使用Match获得分组信息
12.     print match.group()
13.
14. ### 输出 ###
15. # hello

```

## 2.search

`search(string[, pos[, endpos]])` | `re.search(pattern, string[, flags])`:

这个方法用于查找字符串中可以匹配成功的子串。

从string的pos下标处起尝试匹配pattern，

如果pattern结束时仍可匹配，则返回一个Match对象；

若无法匹配，则将pos加1后重新尝试匹配；

直到pos=endpos时仍无法匹配则返回None。

pos和endpos的默认值分别为0和len(string)；

`re.search()`无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

那么它和match有什么区别呢？

`match()`函数只检测re是不是在string的开始位置匹配，

`search()`会扫描整个string查找匹配，

`match ()` 只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，`match()`就返回none

例如：

```
print(re.match('super', 'superstition').span())
```

会返回(0, 5)

```
print(re.match('super', 'insuperable'))
```

则返回None

`search()`会扫描整个字符串并返回第一个成功的匹配

例如：

```
print(re.search('super', 'superstition').span())
```

返回(0, 5)

```
print(re.search('super', 'insuperable').span())
```

返回(2, 7)

看一个search的实例：

[python] [view plaincopy](#)

```
1. # -*- coding: utf-8 -*-
2. #一个简单的search实例
3.
4. import re
5.
6. # 将正则表达式编译成Pattern对象
7. pattern = re.compile(r'world')
8.
9. # 使用search()查找匹配的子串，不存在能匹配的子串时将返回None
10. # 这个例子中使用match()无法成功匹配
11. match = pattern.search('hello world!')
12.
13. if match:
14.     # 使用Match获得分组信息
15.     print match.group()
16.
17. ### 输出 ###
18. # world
```

### 3.split

split(string[, maxsplit]) | re.split(pattern, string[, maxsplit]):

按照能够匹配的子串将string分割后返回列表。

maxsplit用于指定最大分割次数，不指定将全部分割。

[python] [view plaincopy](#)

```
1. import re
2.
3. p = re.compile(r'\d+')
4. print p.split('one1two2three3four4')
5.
6. ### output ###
7. # ['one', 'two', 'three', 'four', '']
```

### 4.findall

findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags]):

搜索string，以列表形式返回全部能匹配的子串。

[python] [view plaincopy](#)

```
1. import re
```

```

2.
3. p = re.compile(r'\d+')
4. print p.findall('one1two2three3four4')
5.
6. ### output ###
7. # ['1', '2', '3', '4']

```

## 5.finditer

`finditer(string[, pos[, endpos]])` | `re.finditer(pattern, string[, flags])`:

搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

[\[html\] view plaincopy](#)

```

1. import re
2.
3. p = re.compile(r'\d+')
4. for m in p.finditer('one1two2three3four4'):
5.     print m.group(),
6.
7. ### output ###
8. # 1 2 3 4

```

## 6.sub

`sub(repl, string[, count])` | `re.sub(pattern, repl, string[, count])`:

使用repl替换string中每一个匹配的子串后返回替换后的字符串。

当repl是一个字符串时，可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。

当repl是一个方法时，这个方法应当只接受一个参数（Match对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。

count用于指定最多替换次数，不指定时全部替换。

[\[python\] view plaincopy](#)

```

1. import re
2.
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
5.
6. print p.sub(r'\2 \1', s)
7.
8. def func(m):
9.     return m.group(1).title() + ' ' + m.group(2).title()
10.
11. print p.sub(func, s)
12.
13. ### output ###
14. # say i, world hello!
15. # I Say, Hello World!

```

## 7.subn

subn(repl, string[, count]) 同 re.sub(pattern, repl, string[, count]):

返回 (sub(repl, string[, count]), 替换次数)。

[python] [view plaincopy](#)

```
1. import re
2.
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
5.
6. print p.subn(r'\2 \1', s)
7.
8. def func(m):
9.     return m.group(1).title() + ' ' + m.group(2).title()
10.
11. print p.subn(func, s)
12.
13. ### output ###
14. # ('say i, world hello!', 2)
15. # ('I Say, Hello World!', 2)
```

至此，Python的正则表达式基本介绍就算是完成了^\_^

## 糗事百科的网络爬虫（v0.2）源码及解析

### 项目内容：

用Python写的糗事百科的网络爬虫。

### 使用方法：

新建一个Bug.py文件，然后将代码复制到里面后，双击运行。

### 程序功能：

在命令提示行中浏览糗事百科。

### 原理解释：

首先，先浏览一下糗事百科的主页：<http://www.qiushibaike.com/hot/page/1>

可以看出来，链接中page/后面的数字就是对应的页码，记住这一点为以后的编写做准备。

然后，右击查看页面源码：

```
<div class="content" title="2013-05-15 13:05:55">
```

给一个白富美当伴娘。。。不要割。。。据说婚车司机曾追过新娘，

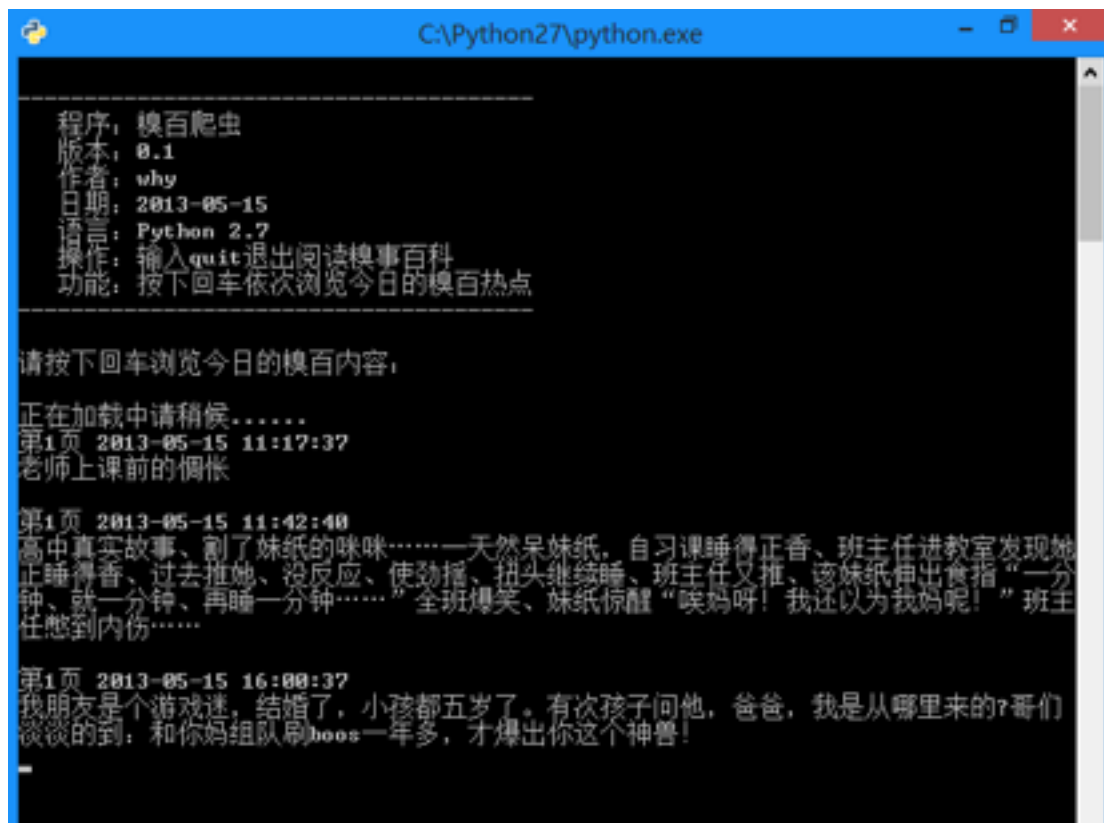
```
</div>
```

观察发现，每一个段子都用div标记，其中class必为content，title是发帖时间，我们只需要用正则表达式将其“扣”出来就可以了。

明白了原理之后，剩下的就是正则表达式的内容了，可以参照这篇博文：

<http://blog.csdn.net/wxg694175346/article/details/8929576>

运行效果：



[python] view plaincopy

```
1. # -*- coding: utf-8 -*-
2. #-----
3. # 程序: 摸百爬虫
4. # 版本: 0.2
5. # 作者: why
6. # 日期: 2013-05-15
7. # 语言: Python 2.7
```

```

8. # 操作: 输入quit退出阅读糗事百科
9. # 功能: 按下回车依次浏览今日的糗百热点
10. # 更新: 解决了命令提示行下乱码的问题
11. #-----
12.
13. import urllib2
14. import urllib
15. import re
16. import thread
17. import time
18.
19. #----- 处理页面上的各种标签 -----
20. class HTML_Tool:
21.     # 用非 贪婪模式 匹配 \t 或者 \n 或者 空格 或者 超链接 或者 图片
22.     BgnCharToNoneRex = re.compile("(\\t|\\n| |<a.*?>|<img.*?>)")
23.
24.     # 用非 贪婪模式 匹配 任意<>标签
25.     EndCharToNoneRex = re.compile("<.*?>")
26.
27.     # 用非 贪婪模式 匹配 任意<p>标签
28.     BgnPartRex = re.compile("<p.*?>")
29.     CharToNewLineRex = re.compile("<br/>|</p>|<tr>|<div>|</div>")
30.     CharToNextTabRex = re.compile("<td>")
31.
32.     # 将一些html的符号实体转变为原始符号
33.     replaceTab = [("<", "<"),(">", ">"),("&", "&"),("&", "&"),("&", "&"),
34.                  (" ", " ")]
35.
36.     def Replace_Char(self,x):
37.         x = self.BgnCharToNoneRex.sub("",x)
38.         x = self.BgnPartRex.sub("\n",x)
39.         x = self.CharToNewLineRex.sub("\n",x)
40.         x = self.CharToNextTabRex.sub("\t",x)
41.         x = self.EndCharToNoneRex.sub("",x)
42.
43.         for t in self.replaceTab:
44.             x = x.replace(t[0],t[1])
45.         return x
46.
47.
48. #----- 加载处理糗事百科 -----
49. class HTML_Model:
50.
51.     def __init__(self):
52.         self.page = 1
53.         self.pages = []
54.         self.myTool = HTML_Tool()
55.         self.enable = False
56.
57.     # 将所有的段子都扣出来, 添加到列表中并且返回列表

```

```

58.     def GetPage(self,page):
59.         myUrl = "http://m.qiushibaike.com/hot/page/" + page
60.         myResponse = urllib2.urlopen(myUrl)
61.         myPage = myResponse.read()
62.         #encode的作用是将unicode编码转换成其他编码的字符串
63.         #decode的作用是将其他编码的字符串转换成unicode编码
64.         unicodePage = myPage.decode("utf-8")
65.
66.         # 找出所有class="content"的div标记
67.         #re.S是任意匹配模式，也就是.可以匹配换行符
68.         myItems = re.findall('<div.*?class="content".*?
title="(.*?)">(.*?)</div>',unicodePage,re.S)
69.         items = []
70.         for item in myItems:
71.             # item 中第一个是div的标题，也就是时间
72.             # item 中第二个是div的内容，也就是内容
73.             items.append([item[0].replace("\n",""),item[1].replace("\n
","")])
74.         return items
75.
76.     # 用于加载新的段子
77.     def LoadPage(self):
78.         # 如果用户未输入quit则一直运行
79.         while self.enable:
80.             # 如果pages数组中的内容小于2个
81.             if len(self.pages) < 2:
82.                 try:
83.                     # 获取新的页面中的段子们
84.                     myPage = self.GetPage(str(self.page))
85.                     self.page += 1
86.                     self.pages.append(myPage)
87.                 except:
88.                     print '无法链接糗事百科！'
89.             else:
90.                 time.sleep(1)
91.
92.     def ShowPage(self,q,page):
93.         for items in q:
94.             print u'第%d页' % page , items[0]
95.             print self.myTool.Replace_Char(items[1])
96.             myInput = raw_input()
97.             if myInput == "quit":
98.                 self.enable = False
99.                 break
100.
101.     def Start(self):
102.         self.enable = True
103.         page = self.page
104.
105.         print u'正在加载中请稍候.....'
106.

```



```

107.         # 新建一个线程在后台加载段子并存储
108.         thread.start_new_thread(self.LoadPage,())
109.
110.         #----- 加载处理糗事百科 -----
111.         while self.enable:
112.             # 如果self的page数组中存有元素
113.             if self.pages:
114.                 nowPage = self.pages[0]
115.                 del self.pages[0]
116.                 self.ShowPage(nowPage,page)
117.                 page += 1
118.
119.
120. #----- 程序的入口处 -----
121. print u"""
122. -----
123. 程序：糗百爬虫
124. 版本：0.1
125. 作者：why
126. 日期：2013-05-15
127. 语言：Python 2.7
128. 操作：输入quit退出阅读糗事百科
129. 功能：按下回车依次浏览今日的糗百热点
130. -----
131. """
132.
133.
134. print u'请按下回车浏览今日的糗百内容： '
135. raw_input(' ')
136. myModel = HTML_Model()
137. myModel.Start()

```

## 百度贴吧的网络爬虫（v0.4）源码及解析

百度贴吧的爬虫制作和糗百的爬虫制作原理基本相同，都是通过查看源码扣出关键数据，然后将其存储到本地txt文件。

### 项目内容：

用Python写的百度贴吧的网络爬虫。

### 使用方法：

新建一个BugBaidu.py文件，然后将代码复制到里面后，双击运行。

### 程序功能：

将贴吧中楼主发布的内容打包txt存储到本地。

### 原理解释：

首先，先浏览一下某一条贴吧，点击只看楼主并点击第二页之后url发生了一点变化，变成了：

[http://tieba.baidu.com/p/2296712428?see\\_lz=1&pn=1](http://tieba.baidu.com/p/2296712428?see_lz=1&pn=1)

可以看出来，see\_lz=1是只看楼主，pn=1是对应的页码，记住这一点为以后的编写做准备。

这就是我们需要利用的url。

接下来就是查看页面源码。

首先把题目抠出来存储文件的时候会用到。

可以看到百度使用gbk编码，标题使用h1标记：

[html] view plain copy

1. `<h1 class="core_title_txt" title="【原创】时尚首席（关于时尚，名利，事业，爱情，励志）">【原创】时尚首席（关于时尚，名利，事业，爱情，励志）</h1>`

同样，正文部分用div和class综合标记，接下来要做的只是用正则表达式来匹配即可。

运行截图：

```
C:\Python27\python.exe
# 程序： 百度贴吧爬虫
# 版本： 0.4
# 作者： why
# 日期： 2013-05-16
# 语言： Python 2.7
# 操作： 输入网址后自动只看楼主并保存到本地文件
# 功能： 将楼主发布的内容打包txt存储到本地。
#
格式： http://tieba.baidu.com/p/xxxxxxxxxx
请输入贴吧的地址：
http://tieba.baidu.com/p/1672347448
已经启动百度贴吧爬虫，咔嚓咔嚓
爬虫报告：发现楼主共有13页的原创内容
文章名称：【转载】我是一名杀手，仅此而已。
爬虫报告：爬虫1号正在加载中...
爬虫报告：爬虫2号正在加载中...
爬虫报告：爬虫3号正在加载中...
爬虫报告：爬虫4号正在加载中...
爬虫报告：爬虫5号正在加载中...
爬虫报告：爬虫6号正在加载中...
爬虫报告：爬虫7号正在加载中...
爬虫报告：爬虫8号正在加载中...
爬虫报告：爬虫9号正在加载中...
```

生成的txt文件：



[python] view plaincopy

```
1. # -*- coding: utf-8 -*-
2. #-----
3. # 程序: 百度贴吧爬虫
4. # 版本: 0.5
5. # 作者: why
6. # 日期: 2013-05-16
7. # 语言: Python 2.7
8. # 操作: 输入网址后自动只看楼主并保存到本地文件
9. # 功能: 将楼主发布的内容打包txt存储到本地。
10. #-----
11.
12. import string
13. import urllib2
14. import re
15.
16. #----- 处理页面上的各种标签 -----
17. class HTML_Tool:
18.     # 用非贪婪模式匹配 \t 或者 \n 或者 空格 或者 超链接 或者 图片
19.     BgnCharToNoneRex = re.compile("(\\t|\\n| |<a.*?>|<img.*?>)")
20.
21.     # 用非贪婪模式匹配 任意<>标签
22.     EndCharToNoneRex = re.compile("<.*?>")
23.
24.     # 用非贪婪模式匹配 任意<p>标签
25.     BgnPartRex = re.compile("<p.*?>")
26.     CharToNewLineRex = re.compile("<br/>|</p>|<tr>|<div>|</div>")
27.     CharToNextTabRex = re.compile("<td>")
28.
29.     # 将一些html的符号实体转变为原始符号
```

```

30.     replaceTab = [("<", "<"),(">", ">"),("&", "&"),("&", "&"),("&", "&"),
    (" ", " ")
31.
32.     def Replace_Char(self,x):
33.         x = self.BgnCharToNoneRex.sub("",x)
34.         x = self.BgnPartRex.sub("\n",x)
35.         x = self.CharToNewLineRex.sub("\n",x)
36.         x = self.CharToNextTabRex.sub("\t",x)
37.         x = self.EndCharToNoneRex.sub("",x)
38.
39.         for t in self.replaceTab:
40.             x = x.replace(t[0],t[1])
41.         return x
42.
43. class Baidu_Spider:
44.     # 申明相关的属性
45.     def __init__(self,url):
46.         self.myUrl = url + '?see_lz=1'
47.         self.datas = []
48.         self.myTool = HTML_Tool()
49.         print u'已经启动百度贴吧爬虫，咔嚓咔嚓'
50.
51.     # 初始化加载页面并将其转码储存
52.     def baidu_tieba(self):
53.         # 读取页面的原始信息并将其从gbk转码
54.         myPage = urllib2.urlopen(self.myUrl).read().decode("gbk")
55.         # 计算楼主发布内容一共有多少页
56.         endPage = self.page_counter(myPage)
57.         # 获取该帖的标题
58.         title = self.find_title(myPage)
59.         print u'文章名称: ' + title
60.         # 获取最终的数据
61.         self.save_data(self.myUrl,title,endPage)
62.
63.     #用来计算一共有多少页
64.     def page_counter(self,myPage):
65.         # 匹配 "共有<span class="red">12</span>页" 来获取一共有多少页
66.         myMatch = re.search(r'<class="red">(\d+?)</
span>', myPage, re.S)
67.         if myMatch:
68.             endPage = int(myMatch.group(1))
69.             print u'爬虫报告: 发现楼主共有%d页的原创内容' % endPage
70.         else:
71.             endPage = 0
72.             print u'爬虫报告: 无法计算楼主发布内容有多少页！'
73.         return endPage
74.
75.     # 用来寻找该帖的标题
76.     def find_title(self,myPage):
77.         # 匹配 <h1 class="core_title_txt" title="">xxxxxxxxxx</h1> 找出
标题

```

```

78.         myMatch = re.search(r'<h1.*?>(.*?)</h1>', myPage, re.S)
79.         title = u'暂无标题'
80.         if myMatch:
81.             title = myMatch.group(1)
82.         else:
83.             print u'爬虫报告: 无法加载文章标题!'
84.             # 文件名不能包含以下字符: \ / : * ? " < > |
85.             title = title.replace('\
            \',').replace('/',').replace(':',').replace('*',').replace('?',').replace(
            '\',').replace('>',').replace('<',').replace('|',')
86.             return title
87.
88.
89.     # 用来存储楼主发布的内容
90.     def save_data(self,url,title,endPage):
91.         # 加载页面数据到数组中
92.         self.get_data(url,endPage)
93.         # 打开本地文件
94.         f = open(title+'.txt','w+')
95.         f.writelines(self.datas)
96.         f.close()
97.         print u'爬虫报告: 文件已下载到本地并打包成txt文件'
98.         print u'请按任意键退出...'
99.         raw_input();
100.
101.     # 获取页面源码并将其存储到数组中
102.     def get_data(self,url,endPage):
103.         url = url + '&pn='
104.         for i in range(1,endPage+1):
105.             print u'爬虫报告: 爬虫%d号正在加载中...' % i
106.             myPage = urllib2.urlopen(url + str(i)).read()
107.             # 将myPage中的html代码处理并存储到datas里面
108.             self.deal_data(myPage.decode('gbk'))
109.
110.
111.     # 将内容从页面代码中抠出来
112.     def deal_data(self,myPage):
113.         myItems = re.findall('id="post_content.*?>(.*?)</
            div>',myPage,re.S)
114.         for item in myItems:
115.             data = self.myTool.Replace_Char(item.replace("\n","").enc
            ode('gbk'))
116.             self.datas.append(data+'\n')
117.
118.
119.
120. #----- 程序入口处 -----
121. print u""#-----
122. # 程序: 百度贴吧爬虫
123. # 版本: 0.5
124. # 作者: why

```

```

125.#    日期: 2013-05-16
126.#    语言: Python 2.7
127.#    操作: 输入网址后自动只看楼主并保存到本地文件
128.#    功能: 将楼主发布的内容打包txt存储到本地。
129.#-----
130. """
131.
132.# 以某小说贴吧为例子
133.# bdurl = 'http://tieba.baidu.com/p/2296712428?see_lz=1&pn=1'
134.
135.print u'请输入贴吧的地址最后的数字串: '
136.bdurl = 'http://tieba.baidu.com/p/' + str(raw_input(u'http://
    tieba.baidu.com/p/'))
137.
138.#调用
139.mySpider = Baidu_Spider(bdurl)
140.mySpider.baidu_tieba()

```

## 5步将py文件打包成exe文件

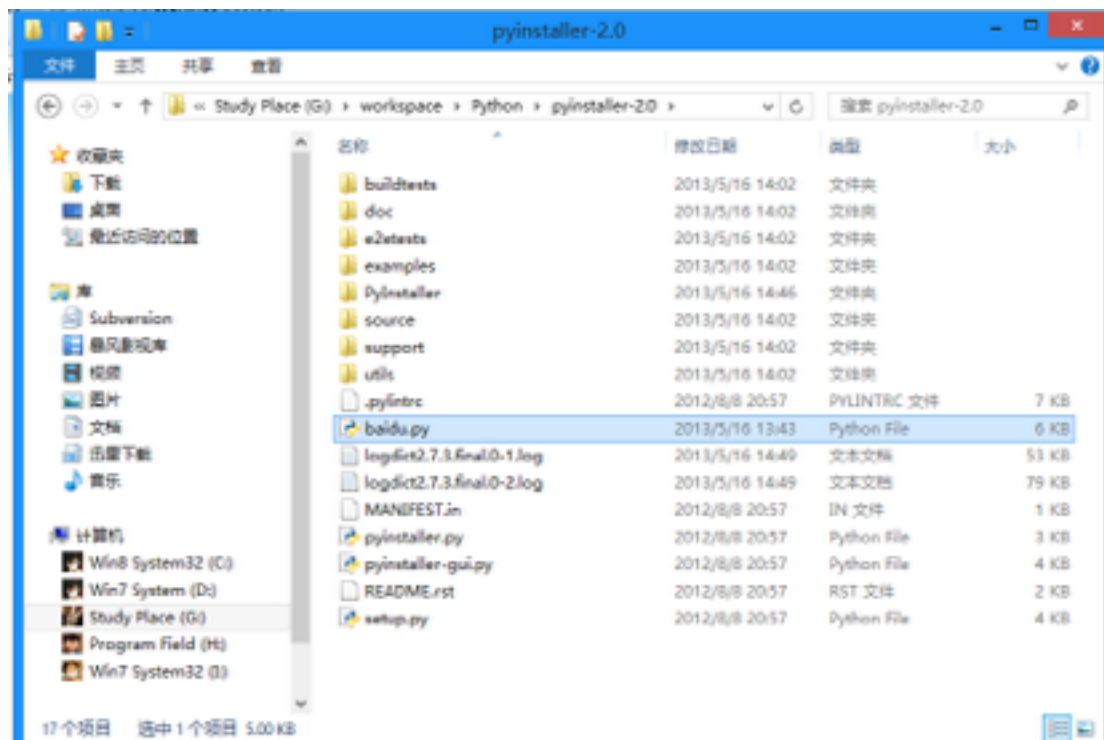
1.下载pyinstaller并解压（可以去官网下载最新版）：

<http://nchc.dl.sourceforge.net/project/pyinstaller/2.0/pyinstaller-2.0.zip>

2.下载pywin32并安装（注意版本，我的是python2.7）：

<http://download.csdn.net/download/lanlandechong/4367925>

3.将项目文件放到pyinstaller文件夹下面（我的是baidu.py）：



4.按住shift键右击，在当前路径打开命令提示行，输入以下内容（最后的是文件名）：

```
python pyinstaller.py -F baidu.py
```

5.生成的exe文件，在baidu文件夹下的dist文件夹中，双击即可运行：

