



Python网络爬虫

极客学院出版

前言

为什么用 Python 写网络爬虫？Python 这种稍严谨而流行库又非常多的语言，大大弱化了计算机运行速度对程序的影响，强化了为程序员容易思考而打造的特性，所以选择 Python 来实现网络爬虫。

本教程简单的介绍如何使用 Python 的相关模块如 urllib2 来实现网络爬虫。

适用人群

想要效率的截取网络素材，有批量抓取需求的开发者。

学习前提

学习本教程前，我们假设你已经了解 Python 这门语言。

版本信息

书中演示代码基于以下版本：

语言/框架	版本信息
Python	2.7.5

鸣谢：<http://blog.csdn.net/column/details/why-bug.html>

目录

前言	1
第 1 章 抓取网页的含义和 URL 基本构成	5
#	6
#	6
#	6
#	6
#	6
第 2 章 利用 urllib2 通过指定的 URL 抓取网页内容	12
#	6
#	6
第 3 章 异常的处理和 HTTP 状态码的分类	19
#	6
#	6
#	6
第 4 章 Opener 与 Handler 的介绍和实例应用	26
#	6
#	6
第 5 章 urllib2 的使用细节与抓站技巧	33
#	6
#	6
#	6
#	6
#	6

	#	6
	#	6
	#	6
	#	6
	#	6
	#	6
第 6 章	一个简单的百度贴吧的小爬虫	46
第 7 章	Python 中的正则表达式教程	49
	#	6
	#	6
第 8 章	糗事百科的网络爬虫（v0.3）源码及解析(简化更新)	65
	#	6
第 9 章	百度贴吧的网络爬虫（v0.4）源码及解析	71
第 10 章	一个爬虫的诞生全过程（以山东大学绩点运算为例）	79
	#	6
	#	6
	#	6
	#	6
	#	6
	#	6
	#	6
	#	6
	#	6
第 11 章	亮剑！爬虫框架小抓抓 Scrapy 闪亮登场！	99
	#	6
	#	6
	#	6

#	6
#	6
#	6
#	6
#	6
#	6

第 12 章 爬虫框架 Scrapy 的第一个爬虫示例入门教程 110

#	6
#	6
#	6
#	6



T



1



抓取网页的含义和 URL 基本构成



#

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。

#

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站页面。

#

URI 和 URL 的概念和举例

简单的来讲，URL 就是在浏览器端输入的 `wiki.jikexueyuan.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。

#

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 的一般格式为(带方括号[]的为可选项)：

`protocol :// hostname[:port] / path / [;parameters][?query]#fragment`

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。

#

URL 和 URI 简单比较

URI 属于 URL 更低层次的抽象，一种字符串文本标准。

换句话说，URI 属于父类，而 URL 属于 URI 的子类。URL 是 URI 的一个子集。

URI 的定义是：统一资源标识符；

URL 的定义是：统一资源定位符。

二者的区别在于，URI 表示请求服务器的路径，定义这么一个资源。

而 URL 同时说明要如何访问这个资源（http://）。

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。

1. 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。

2

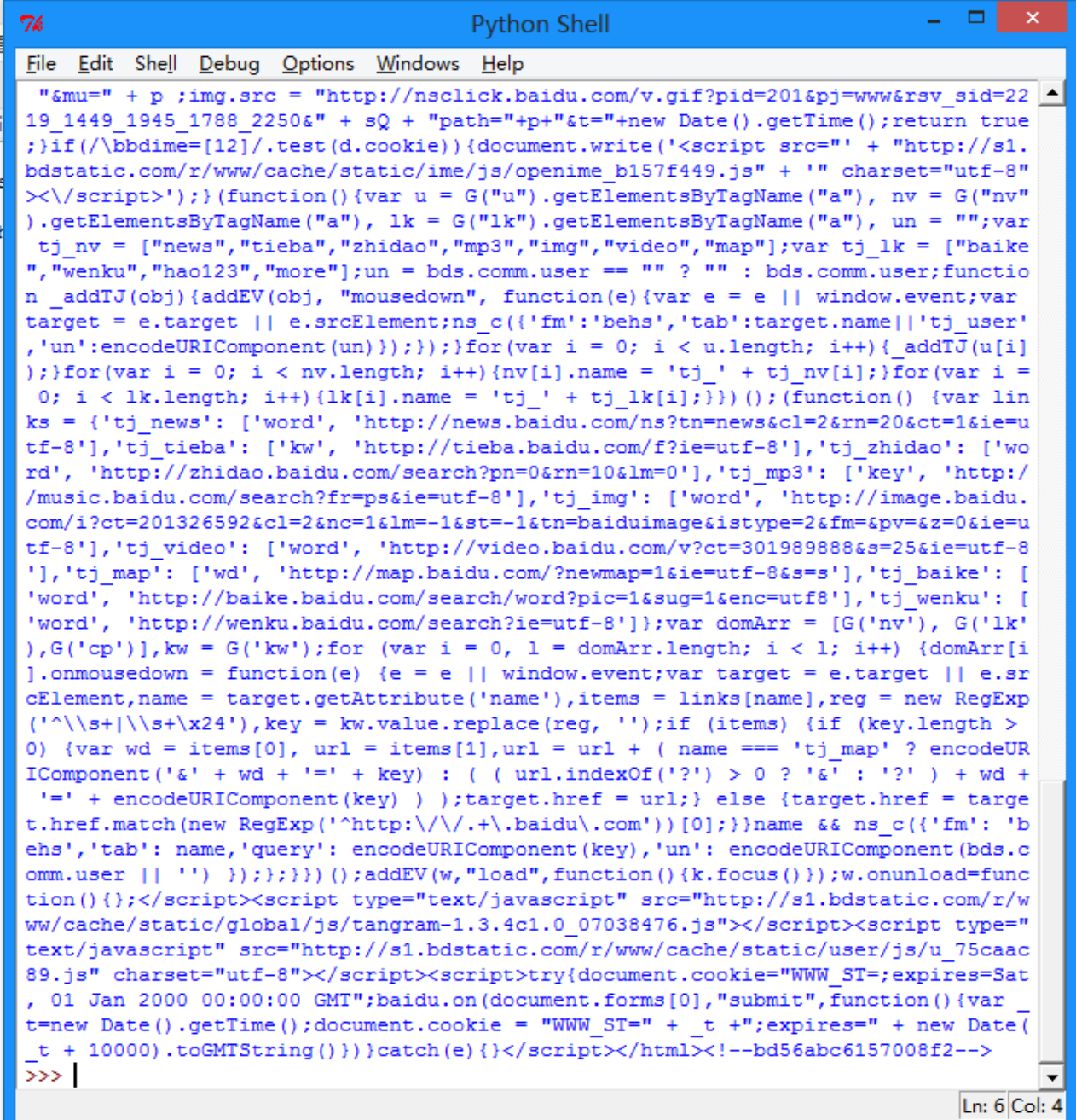
利用 urllib2 通过指定的 URL 抓取网页内容

版本号：Python2.7.5，Python3 改动较大，各位另寻教程。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：

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

按下 F5 可以看到运行的结果：



```

"&mu=" + p ;img.src = "http://nsclick.baidu.com/v.gif?pid=201&pj=www&rsv_sid=22
19_1449_1945_1788_2250&" + sQ + "path="+p+"&t="+new Date().getTime();return true
;};if(/\bbdime=[12]/.test(d.cookie)){document.write('<script src="' + "http://s1.
bdstatic.com/r/www/cache/static/ime/js/openime_b157f449.js" + '" charset="utf-8"
></script>');}(function(){var u = G("u").getElementsByTagName("a"), nv = G("nv"
).getElementsByTagName("a"), lk = G("lk").getElementsByTagName("a"), un = "";var
tj_nv = ["news","tieba","zhidao","mp3","img","video","map"];var tj_lk = ["baike
","wenku","haol23","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.baid
u.com/i?ct=201326592&cl=2&nc=1&lm=-1&st=-1&tn=baiduimage&istype=2&fm=&pv=&z=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\.'))[0];}name && ns_c({'fm': 'b
ehs','tab': name,'query': encodeURIComponent(key),'un': encodeURIComponent(bds.c
omm.user || '')});});addEV(w,"load",function(){k.focus()});w.onunload=func
tion(){}/</script><script type="text/javascript" src="http://s1.bdstatic.com/r/w
ww/cache/static/global/js/tangram-1.3.4c1.0_07038476.js"></script><script type="
text/javascript" src="http://s1.bdstatic.com/r/www/cache/static/user/js/u_75caac
89.js" charset="utf-8"></script><script>try{document.cookie="WWW_ST=;expires=Sat
, 01 Jan 2000 00:00:00 GMT";baidu.on(document.forms[0],"submit",function(){var _
t=new Date().getTime();document.cookie = "WWW_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：

```
import urllib2
req = urllib2.Request('http://www.baidu.com')
response = urllib2.urlopen(req)
the_page = response.read()
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。

```
req = urllib2.Request('ftp://example.com/')
```

在 HTTP 请求时，允许你做额外的两件事。

#

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：

```
import urllib
import urllib2

url = 'http://www.someserver.com/register.cgi'

values = {'name': 'WHY',
          'location': 'SDU',
          'language': 'Python' }

data = urllib.urlencode(values) # 编码工作
req = urllib2.Request(url, data) # 发送请求同时传data表单
response = urllib2.urlopen(req) #接受反馈的信息
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。

```
import urllib2
import urllib

data = {}

data['name'] = 'WHY'
data['location'] = 'SDU'
data['language'] = 'Python'

url_values = urllib.urlencode(data)
print url_values

name=Somebody+Here&language=Python&location=Northampton
url = 'http://www.example.com/example.cgi'
```

```
full_url = url + '?' + url_values  
  
data = urllib2.open(full_url)
```

这样就实现了 Data 数据的 Get 传送。

#

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。（多谢大家的提醒，现在这个 Demo 已经不可用了，不过原理还是那样的）。

```
import urllib
import urllib2

url = 'http://www.someserver.com/cgi-bin/register.cgi'

user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
values = {'name': 'WHY',
          'location': 'SDU',
          'language': 'Python' }

headers = { 'User-Agent': user_agent }
data = urllib.urlencode(values)
req = urllib2.Request(url, data, headers)
response = urllib2.urlopen(req)
the_page = response.read()
```



3



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。

#

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：

```
import urllib2

req = urllib2.Request('http://www.baibai.com')

try: urllib2.urlopen(req)

except urllib2.URLError, e:
    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。

#

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: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: 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：

```
import urllib2
req = urllib2.Request('http://bbs.csdn.net/callmewhy')
```

```
try:
    urllib2.urlopen(req)

except urllib2.URLError, e:

    print e.code
    #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。

#

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：

```
from urllib2 import Request, urlopen, URLError, HTTPError

req = Request('http://bbs.csdn.net/callmewhy')

try:

    response = urlopen(req)

except HTTPError, e:

    print 'The server couldn\'t fulfill the request.'

    print 'Error code: ', e.code

except URLError, e:

    print 'We failed to reach a server.'

    print 'Reason: ', e.reason

else:

    print 'No exception was raised.'
    # 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：

```
from urllib2 import Request, urlopen, URLError, HTTPError

req = Request('http://bbs.csdn.net/callmewhy')
```

```
try:

    response = urlopen(req)

except URLError, e:

    if hasattr(e, 'code'):

        print 'The server couldn\'t fulfill the request.'

        print 'Error code: ', e.code

    elif hasattr(e, 'reason'):

        print 'We failed to reach a server.'

        print 'Reason: ', e.reason

else:

    print 'No exception was raised.'
    # everything is fine
```



4

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()

#

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：

```
from urllib2 import Request, urlopen, URLError, HTTPError

old_url = 'http://rrurl.cn/b1UZuP'
req = Request(old_url)
response = urlopen(req)
print 'Old url : ' + old_url
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
>>> |
```

#

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：

```
from urllib2 import Request, urlopen, URLError, HTTPError

old_url = 'http://www.baidu.com'
req = Request(old_url)
response = urlopen(req)
print 'Info():'
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。

#

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。

#

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：

```
\# -*- coding: utf-8 -*-
import urllib2

\# 创建一个密码管理者
password_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()

\# 添加用户名和密码

top_level_url = "http://example.com/foo/"

\# 如果知道 realm, 我们可以使用他代替 ``None``.
\# password_mgr.add_password(None, top_level_url, username, password)
password_mgr.add_password(None, top_level_url, 'why', '1223')

\# 创建了一个新的handler
handler = urllib2.HTTPBasicAuthHandler(password_mgr)

\# 创建 "opener" (OpenerDirector 实例)
opener = urllib2.build_opener(handler)

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

\# 使用 opener 获取一个URL
opener.open(a_url)

\# 安装 opener.
\# 现在所有调用 urllib2.urlopen 将用我们的 opener.
urllib2.install_opener(opener)
```

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。



T

5



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。

#

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：

```
import urllib2
enable_proxy = True
proxy_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
null_proxy_handler = urllib2.ProxyHandler({})
if enable_proxy:
    opener = urllib2.build_opener(proxy_handler)
else:
    opener = urllib2.build_opener(null_proxy_handler)
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。

#

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。

```
import urllib2
import socket
socket.setdefaulttimeout(10) # 10 秒钟后超时
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。

```
import urllib2
response = urllib2.urlopen('http://www.google.com', timeout=10)
```

#

在 HTTP Request 中加入特定的 Header

要加入 header，需要使用 Request 对象：

```
import urllib2
request = urllib2.Request('http://www.baidu.com/')
request.add_header('User-Agent', 'fake-client')
response = urllib2.urlopen(request)
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

#

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。

```
import urllib2
my_url = 'http://www.google.cn'
response = urllib2.urlopen(my_url)
redirected = response.geturl() == my_url
print redirected

my_url = 'http://rrurl.cn/b1UZuP'
response = urllib2.urlopen(my_url)
redirected = response.geturl() == my_url
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。

```
import urllib2
class RedirectHandler(urllib2.HTTPRedirectHandler):
    def http_error_301(self, req, fp, code, msg, headers):
        print "301"
        pass
    def http_error_302(self, req, fp, code, msg, headers):
        print "303"
        pass

opener = urllib2.build_opener(RedirectHandler)
opener.open('http://rrurl.cn/b1UZuP')
```

#

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：

```
import urllib2
import cookielib
cookie = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
response = opener.open('http://www.baidu.com')
for item in cookie:
    print 'Name = '+item.name
    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
```


#

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 httpplib 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：

```
import urllib2
request = urllib2.Request(uri, data=data)
request.get_method = lambda: 'PUT' # or 'DELETE'
response = urllib2.urlopen(request)
```

#

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：

```
import urllib2
try:
    response = urllib2.urlopen('http://bbs.csdn.net/why')
except urllib2.HTTPError, e:
    print e.code
```

#

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

```
import urllib2
httpHandler = urllib2.HTTPHandler(debuglevel=1)
httpsHandler = urllib2.HTTPSHandler(debuglevel=1)
opener = urllib2.build_opener(httpHandler, httpsHandler)
urllib2.install_opener(opener)
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&usg=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=ftQHGF13WlmI4v-; 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&usg=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=a37fa0c30a92615b: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_VcymquxCtv9Lg7UGZJP10rtqubT41FR6ss0iha7x2PyJsqevc
A8xoj76M9batchPOoOxnn192wuZuJKUpLWqr-38pMH3dwl5C8PAwDANhNclY_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
ounts/bin/answer.py?hl=en&answer=151657 for more info."
```

#

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：

```
# -*- coding: utf-8 -*-

import urllib
import urllib2
postdata=urllib.urlencode({
    'username':'汪小光',
    'password':'why888',
    'continueURI':'http://www.verycd.com/',
    'fk':"",
    'login_submit':'登录'
})
req = urllib2.Request(
    url = 'http://secure.verycd.com/signin',
    data = postdata
)
result = urllib2.urlopen(req)
print result.read()
```

#

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。

```
\#...

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6'
}
req = urllib2.Request(
    url = 'http://secure.verycd.com/signin/*/http://www.verycd.com/',
    data = postdata,
    headers = headers
)
\#...
```

#

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。



T

6



一个简单的百度贴吧的小爬虫



```

#-*- coding: utf-8 -*-

#-----

# 程序： 百度贴吧爬虫

# 版本： 0.1

# 作者： why

# 日期： 2013-05-14

# 语言： Python 2.7

# 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。

# 功能： 下载对应页码内的所有页面并存储为html文件。

#-----

import string, urllib2

#定义百度函数

def baidu_tieba(url,begin_page,end_page):
    for i in range(begin_page, end_page+1):
        sName = string.zfill(i,5) + '.html'#自动填充成六位的文件名
        print '正在下载第' + str(i) + '个网页，并将其存储为' + sName + '.....'
        f = open(sName,'w+')
        m = urllib2.urlopen(url + str(i)).read()
        f.write(m)
        f.close()

#----- 在这里输入参数 -----

# 这个是山东大学的百度贴吧中某一个帖子的地址

#bdurl = 'http://tieba.baidu.com/p/2296017831?pn='

#iPostBegin = 1

#iPostEnd = 10

```



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

#调用

baidu_tieba(bdurl,begin_page,end_page)
```



7



Python 中的正则表达式教程



HTML



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。

#

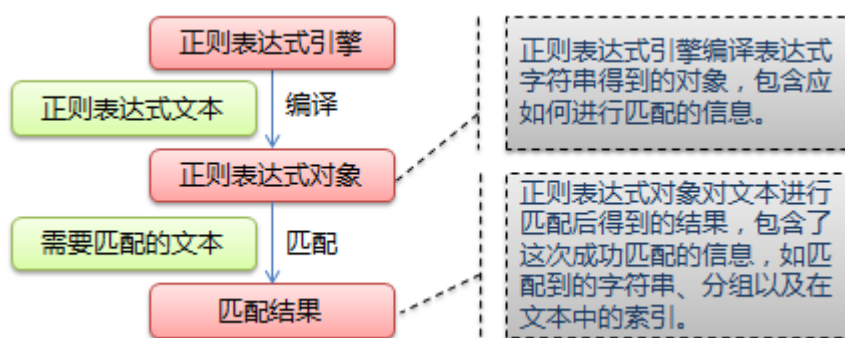
正则表达式基础

#

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[*]。	a\.c a\\c	a.c a\c
[...]	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[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：[^ \d]	a\D c	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\s c	a c
\S	非空白字符：[^ \s]	a\S c	abc
\w	单词字符：[A-Za-z0-9_]	a\w c	abc
\W	非单词字符：[^ \w]	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（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
\$	匹配字符串末尾。 在多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
逻辑、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。	abc def	abc def
(...)	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 仅在该组中有效。	(abc){2} a(123 456)c	abcaabc a456c

#

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`。

#

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\"`。有了原生字符串，妈妈再也不用担心我的反斜杠问题~

#

介绍 re 模块

#

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：

```
``` # -- coding: utf-8 --  

#一个简单的re实例，匹配字符串中的hello字符串

#导入re模块

import re

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”

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

if match1:
 # 使用Match获得分组信息
 print match1.group()
```

```

else:
 print 'match1匹配失败！'

#如果match2匹配成功

if match2:
 # 使用Match获得分组信息
 print match2.group()
else:
 print 'match2匹配失败！'

#如果match3匹配成功

if match3:
 # 使用Match获得分组信息
 print match3.group()
else:
 print 'match3匹配失败！'

```

可以看到控制台输出了匹配的三个结果：

```

```

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 | 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：

```
-- coding: utf-8 --

#两个等价的re匹配,匹配一个小数

import re

a = re.compile(r"""\d + # the integral part . # the decimal point \d * # some fractional digits""", re.X)

b = re.compile(r"\d+.\d*")

match11 = a.match('3.1415')
match12 = a.match('33')
match21 = b.match('3.1415')
match22 = b.match('33')

if match11:
 # 使用Match获得分组信息
 print match11.group()
else:
 print u'match11不是小数'

if match12:
 # 使用Match获得分组信息
 print match12.group()
else:
 print u'match12不是小数'

if match21:
 # 使用Match获得分组信息
 print match21.group()
else:
 print u'match21不是小数'

if match22:
 # 使用Match获得分组信息
 print match22.group()
else:
 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.com

如一开始的 hello 实例可以简写为：

```
-- coding: utf-8 --
```

```
#一个简单的re实例，匹配字符串中的hello字符串
```

```
import re
```

```
m = re.match(r'hello', 'hello world!')
```

```
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

```
####
```

```
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\*\*：**

- string: 匹配时使用的文本。
- re: 匹配时使用的 Pattern 对象。
- 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。
- 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。
- 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
- 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捕获的字符串；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

2. groups([default]):

以元组形式返回全部分组捕获的字符串。相当于调用 group(1,2,...last)。default 表示没有捕获字符串的组以这个值替代，默认为 N

3. groupdict([default]):

返回以有别名的组的别名为键、以该组捕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]):

返回指定的组捕获的子串在 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]):

返回指定的组捕获的子串在 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

下面来用一个 py 实例输出所有的内容加深理解：

```
-- coding: utf-8 --

#一个简单的match实例

import re

匹配如下内容： 单词+空格+单词+任意字符

m = re.match(r'(\w+) (\w+)(?P.*)', 'hello world!')

print "m.string:", m.string
print "m.re:", m.re
print "m.pos:", m.pos
print "m.endpos:", m.endpos
print "m.lastindex:", m.lastindex
print "m.lastgroup:", m.lastgroup

print "m.group():", m.group()
print "m.group(1,2):", m.group(1, 2)
print "m.groups():", m.groups()
print "m.groupdict():", m.groupdict()
print "m.start(2):", m.start(2)
print "m.end(2):", m.end(2)
print "m.span(2):", m.span(2)
print r"m.expand(r'\g<2> \g<1>\g<3>'):", m.expand(r'\2 \1\3')

output

m.string: hello world!

m.re: <_sre.SRE_Pattern object at 0x016E1A38>

m.pos: 0

m.endpos: 12

m.lastindex: 3

m.lastgroup: sign
```

```
m.group(1,2): ('hello', 'world')

m.groups(): ('hello', 'world', '!')

m.groupdict(): {'sign': '!'}

m.start(2): 6

m.end(2): 11

m.span(2): (6, 11)

m.expand(r'\2 \1\3'): world hello!
```

```
####
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：

- pattern: 编译时用的表达式字符串。
- flags: 编译时用的匹配模式。数字形式。
- groups: 表达式中分组的数量。
- 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：

```
-- coding: utf-8 --

一个简单的pattern实例

import re
p = re.compile(r'(\w+) (\w+)(?P.*)', re.DOTALL)

print "p.pattern:", p.pattern
print "p.flags:", p.flags
print "p.groups:", p.groups
print "p.groupindex:", p.groupindex

output

p.pattern: (\w+) (\w+)(?P.*)
```

```
p.flags: 16

p.groups: 3

p.groupindex: {'sign': 3}
```

下面重点介绍一下 pattern 的实例方法及其使用。

```
#####
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 p

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：

```
encoding: UTF-8

import re

将正则表达式编译成Pattern对象

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

match = pattern.match('hello world!')

if match:
 # 使用Match获得分组信息
 print match.group()

输出

hello
```

```
#####
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 `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

`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：

```
-- coding: utf-8 --
```

```
#一个简单的search实例
```

```
import re
```

```
将正则表达式编译成Pattern对象
```

```
pattern = re.compile(r'world')
```

# 使用search()查找匹配的子串，不存在能匹配的子串时将返回None

# 这个例子中使用match()无法成功匹配

```
match = pattern.search('hello world!')
```

```
if match:
```

# 使用Match获得分组信息

```
print match.group()
```

### 输出 ###

# world

```
#####
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。

```
import re
```

```
p = re.compile(r'\d+')
```

```
print p.split('one1two2three3four4')
```

### output ###

```
['one', 'two', 'three', 'four', '']
```

```
#####
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。

```
import re
```

```
p = re.compile(r'\d+')
```

```
print p.findall('one1two2three3four4')
```

```
output
```

```
['1', '2', '3', '4']
```

```
#####
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。

```
import re
```

```
p = re.compile(r'\d+')

```

```
for m in p.finditer('one1two2three3four4'):

```

```
 print m.group(),

```

```
output
```

```
1 2 3 4
```

```
#####
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。

```
import re
```

```
p = re.compile(r'(\w+) (\w+)')

```

```
s = 'i say, hello world!'

```

```
print p.sub(r'\2 \1', s)

```

```
def func(m):

```

```
 return m.group(1).title() + ' ' + m.group(2).title()

```

```
print p.sub(func, s)

```

```
output
```



```
say i, world hello!
```

```
I Say, Hello World!
```

```
#####
subn
```

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

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

```
import re
```

```
p = re.compile(r'(\w+) (\w+)')
```

```
s = 'i say, hello world!'
```

```
print p.subn(r'\2 \1', s)
```

```
def func(m):
```

```
 return m.group(1).title() + ' ' + m.group(2).title()
```

```
print p.subn(func, s)
```

```
output
```

```
('say i, world hello!', 2)
```

```
('I Say, Hello World!', 2)
```

```
...
```

8

糗事百科的网络爬虫（v0.3）源码及解析(简化更新)

#

---

Q&A:

#

为什么有段时间显示糗事百科不可用？

答：前段时间因为糗事百科添加了 Header 的检验，导致无法爬取，需要在代码中模拟Header。现在代码已经作了修改，可以正常使用。

#

为什么需要单独新建个线程？

答：基本流程是这样的：爬虫在后台新起一个线程，一直爬取两页的糗事百科，如果剩余不足两页，则再爬一页。用户按下回车只是从库存中获取最新的内容，而不是上网获取，所以浏览更顺畅。也可以把加载放在主线程，不过这样会导致爬取过程中等待时间过长的的问题。

#

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：



```
-*- coding: utf-8 -*-
```

```
import urllib2
import urllib
import re
import thread
import time
```

```
#----- 加载处理糗事百科 -----
```

```
class Spider_Model:
```

```
 def __init__(self):
 self.page = 1
 self.pages = []
 self.enable = False
```

```
 # 将所有的段子都扣出来，添加到列表中并且返回列表
 def GetPage(self,page):
```

```

myUrl = "http://m.qiushibaike.com/hot/page/" + page
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }
req = urllib2.Request(myUrl, headers = headers)
myResponse = urllib2.urlopen(req)
myPage = myResponse.read()
#encode的作用是将unicode编码转换成其他编码的字符串
#decode的作用是将其他编码的字符串转换成unicode编码
unicodePage = myPage.decode("utf-8")

找出所有class="content"的div标记
#re.S是任意匹配模式, 也就是.可以匹配换行符
myItems = re.findall('<div.*?class="content".*?title="(.*?)">(.*?)</div>',unicodePage,re.S)
items = []
for item in myItems:
 # item 中第一个是div的标题, 也就是时间
 # item 中第二个是div的内容, 也就是内容
 items.append([item[0].replace("\n",""),item[1].replace("\n","")])
return items

用于加载新的段子
def LoadPage(self):
 # 如果用户未输入quit则一直运行
 while self.enable:
 # 如果pages数组中的内容小于2个
 if len(self.pages) < 2:
 try:
 # 获取新的页面中的段子们
 myPage = self.GetPage(str(self.page))
 self.page += 1
 self.pages.append(myPage)
 except:
 print '无法链接糗事百科! '
 else:
 time.sleep(1)

def ShowPage(self,nowPage,page):
 for items in nowPage:
 print u'第%d页' % page , items[0] , items[1]
 myInput = raw_input()
 if myInput == "quit":
 self.enable = False
 break

def Start(self):

```

```

self.enable = True
page = self.page

print u'正在加载中请稍候.....'

新建一个线程在后台加载段子并存储
thread.start_new_thread(self.LoadPage,())

#----- 加载处理糗事百科 -----
while self.enable:
 # 如果self的page数组中存有元素
 if self.pages:
 nowPage = self.pages[0]
 del self.pages[0]
 self.ShowPage(nowPage,page)
 page += 1

#----- 程序的入口处 -----

print u""" 程序：糗百爬虫
版本：0.3
作者：why
日期：2014-06-03
语言：Python 2.7
操作：输入quit退出阅读糗事百科
功能：按下回车依次浏览今日的糗百热点 """

print u'请按下车浏览今日的糗百内容：'
raw_input(' ')
myModel = Spider_Model()
myModel.Start()

```



9

## 百度贴吧的网络爬虫（v0.4）源码及解析





更新：百度贴吧现在已经改成 utf-8 编码了吧，需要把代码中的 `decode('gbk')` 改成 `decode('utf-8')`。

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。

#

源码下载：

<http://download.csdn.net/detail/wxg694175346/6925583>

#

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：

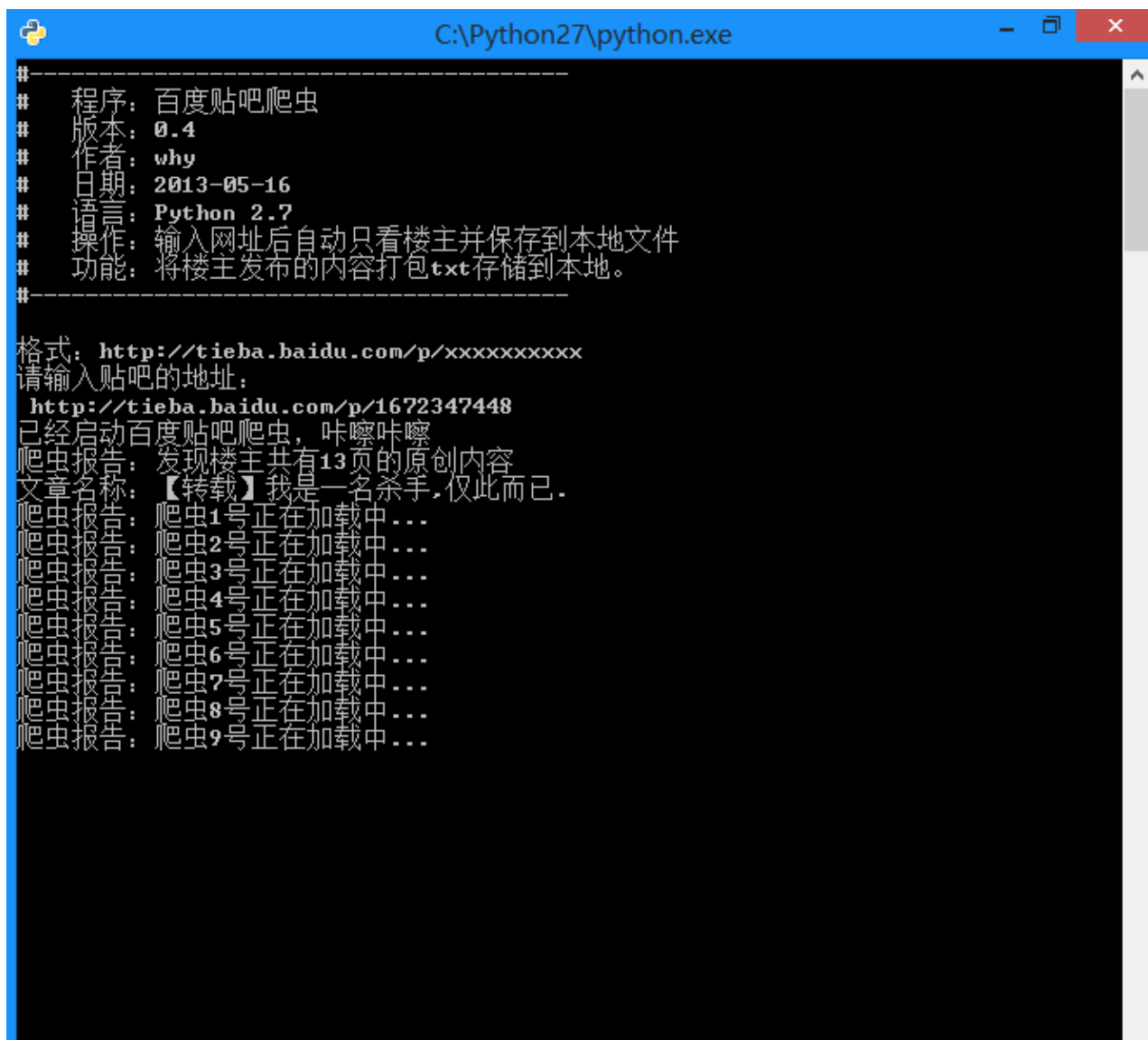
首先，先浏览一下某一条贴吧，点击只看楼主并点击第二页之后 url 发生了一点变化，变成了：

[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：

```
<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，
```

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：



```

#-----
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：

【转载】我是一名杀手,仅此而已..txt	2013/5/16 14:05	文本文档	317 KB
百度贴吧爬虫v0.1.py	2013/5/16 9:24	Python File	2 KB
百度贴吧爬虫v0.2.py	2013/5/16 11:04	Python File	3 KB
百度贴吧爬虫v0.3.py	2013/5/16 12:51	Python File	5 KB
百度贴吧爬虫v0.4.py	2013/5/16 13:43	Python File	6 KB

【转载】我是一名杀手,仅此而已..txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

很不错的一篇文章,转过来给没看的基友们看。作者:天生killer。在SD吧没看到有相关的帖子,自己觉得很棒,就给大家转过来了。

在这里,我发誓,我写下的每一句话都是真话。我是一名杀手,对,就像你们在电影里看到的那样,拿人钱财,与人消灾的杀手,但是,我不能幸福的看完片子再对电影中的情节评头论足,因为我就是那样,我不是电影看多了在这里意淫,也不是得了狂妄幻想症,我,就是一个杀手,仅此而已。我不知道我在这里写下我事会给自己或者公司带来怎样的后果,但是无论是生命还是人的忍耐都是有极限的,随便吧。大家感兴趣的只当是看了一篇文笔很烂的作家写下的故事,而我,也只能和陌生人讲心里话。我叫T,28岁,我没有双重职业,我只有一个赖以谋生的饭碗就是替公司的雇主杀人,当我端上这个饭碗的第一天开始,我就不可能再把它放下了,因为放下它的那天就是我死的那天。以前的我是一名搏击运动员,过着很正常的生活,有家人,有朋友,因为一场拳赛认识了H,因为他的出现,我的世界彻底颠覆了。后来想起和H认识后的一幕幕,就像看着一个又一个的圈套,H在前面回头对我笑着,我跟在后面一个一个的钻,等醒悟的时候,这条路已经不可能再回头了。认识H的那年我才17岁,这也是后来为什么我匿名给公司订单,用行价3倍的钱买H的命的原因。我不知道大家是否能想象所有的事我忍了17年,17年,不能对任何人讲真话,不能痛痛快快的和知己倾诉,我唯一的倾诉对象就是我养的那只猫,只是后来它也死了。进公司的那年我18岁,就是认识H后的一年,人犯错误只有两个原因,一个是情,一个是钱,天底下所有的错误都源于这两个东西。我且因为钱,因为太多的诱惑和不懂世事。我进的这家公司在行业内规模最大的也是

```
-*- coding: utf-8 -*-
```

```
#-----
```

```
程序: 百度贴吧爬虫
```

```
版本: 0.5
```

```
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。
```

```
#-----
```

```
import string
import urllib2
import re
```

```

#----- 处理页面上的各种标签 -----

class HTML_Tool:
 # 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
 BgnCharToNoneRex = re.compile("(\\t\\n| |<a.*?>|<img.*?>)")

 # 用非贪婪模式 匹配 任意<>标签
 EndCharToNoneRex = re.compile("<.*?>")

 # 用非贪婪模式 匹配 任意<p>标签
 BgnPartRex = re.compile("<p.*?>")
 CharToNewLineRex = re.compile("
|</p>|<tr>|<div>|</div>")
 CharToNextTabRex = re.compile("<td>")

 # 将一些html的符号实体转变为原始符号
 replaceTab = [("<", "<"), (">", ">"), ("&", "&"), ("\"", "\""), (" ", " ")]

 def Replace_Char(self, x):
 x = self.BgnCharToNoneRex.sub("", x)
 x = self.BgnPartRex.sub("\n ", x)
 x = self.CharToNewLineRex.sub("\n", x)
 x = self.CharToNextTabRex.sub("\t", x)
 x = self.EndCharToNoneRex.sub("", x)

 for t in self.replaceTab:
 x = x.replace(t[0], t[1])
 return x

class Baidu_Spider:
 # 申明相关的属性
 def __init__(self, url):
 self.myUrl = url + '?see_lz=1'
 self.datas = []
 self.myTool = HTML_Tool()
 print u'已经启动百度贴吧爬虫，咔嚓咔嚓'

 # 初始化加载页面并将其转码储存
 def baidu_tieba(self):
 # 读取页面的原始信息并将其从gbk转码
 myPage = urllib2.urlopen(self.myUrl).read().decode("gbk")
 # 计算楼主发布内容一共有多少页
 endPage = self.page_counter(myPage)
 # 获取该帖的标题
 title = self.find_title(myPage)
 print u'文章名称: ' + title

```

```
获取最终的数据
self.save_data(self.myUrl,title,endPage)

#用来计算一共有多少页
def page_counter(self,myPage):
 # 匹配 "共有12页" 来获取一共有多少页
 myMatch = re.search(r'class="red">(\d+?)', myPage, re.S)
 if myMatch:
 endPage = int(myMatch.group(1))
 print u'爬虫报告：发现楼主共有%d页的原创内容' % endPage
 else:
 endPage = 0
 print u'爬虫报告：无法计算楼主发布内容有多少页！'
 return endPage

用来寻找该帖的标题
def find_title(self,myPage):
 # 匹配 <h1 class="core_title_txt" title="">xxxxxxxxxx</h1> 找出标题
 myMatch = re.search(r'<h1.*?>(.*?)</h1>', myPage, re.S)
 title = u'暂无标题'
 if myMatch:
 title = myMatch.group(1)
 else:
 print u'爬虫报告：无法加载文章标题！'
 # 文件名不能包含以下字符： \ / : * ? " < > |
 title = title.replace("\\", "").replace('/', "").replace(':', "").replace('*', "").replace('?', "").replace('"', "").replace('>', "").replace('<', "").replace('|', "")
 return title

用来存储楼主发布的内容
def save_data(self,url,title,endPage):
 # 加载页面数据到数组中
 self.get_data(url,endPage)
 # 打开本地文件
 f = open(title+'.txt','w+')
 f.writelines(self.datas)
 f.close()
 print u'爬虫报告：文件已下载到本地并打包成txt文件'
 print u'请按任意键退出...'
 raw_input();

获取页面源码并将其存储到数组中
def get_data(self,url,endPage):
 url = url + '&pn='
 for i in range(1,endPage+1):
```

```

print u'爬虫报告：爬虫%d号正在加载中...' % i
myPage = urllib2.urlopen(url + str(i)).read()
将myPage中的html代码处理并存储到datas里面
self.deal_data(myPage.decode('gbk'))

将内容从页面代码中抠出来
def deal_data(self, myPage):
 myItems = re.findall('id="post_content.*?>(.*?)</div>', myPage, re.S)
 for item in myItems:
 data = self.myTool.Replace_Char(item.replace("\n", "").encode('gbk'))
 self.datas.append(data + '\n')

#----- 程序入口处 -----

print u'#####'
程序：百度贴吧爬虫

版本：0.5

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。

#-----

'''

以某小说贴吧为例子

bdurl = 'http://tieba.baidu.com/p/2296712428?see_lz=1&pn=1'

print u'请输入贴吧的地址最后的数字串：'
bdurl = 'http://tieba.baidu.com/p/' + str(raw_input(u'http://tieba.baidu.com/p/'))

#调用

```

```
mySpider = Baidu_Spider(bdurl)
mySpider.baidu_tieba()
```



10

一个爬虫的诞生全过程（以山东大学绩点运算为例）





先来说一下我们学校的网站：

[http://jwxt.sdu.edu.cn:7777/zhxt\\_bks/zhxt\\_bks.html](http://jwxt.sdu.edu.cn:7777/zhxt_bks/zhxt_bks.html)

查询成绩需要登录，然后显示各学科成绩，但是只显示成绩而没有绩点，也就是加权平均分。

本科生综合查询

必修课成绩表					
课程号	课程名	序号	学分	考试时间	成绩
0063109410	动画史与经典赏析	0	2	20130107	95
0063105410	色彩构成	0	1.5	20130107	89
0131700310	Web技术导论(工程)	601	2	20130107	75
0063104310	平面构成	0	2	20130107	90
0311001820	大学英语视听说(2级起点3)	337	1	20130107	78
0303100311	计算机图形学(双语)	0	4	20130107	100
0291000310	体育(3)	309	1	20130107	90
0311001310	大学英语读写(2级起点3)	337	2	20130107	84
0051100210	传统文学修养(国学)	601	2	20120628	90

显然这样手动计算绩点是一件非常麻烦的事情。所以我们可以用python做一个爬虫来解决这个问题。

#

## 决战前夜

先来准备一下工具：HttpFox 插件。这是一款 http 协议分析插件，分析页面请求和响应的时间、内容、以及浏览器用到的 COOKIE 等。以我为例，安装在火狐上即可，效果如图：

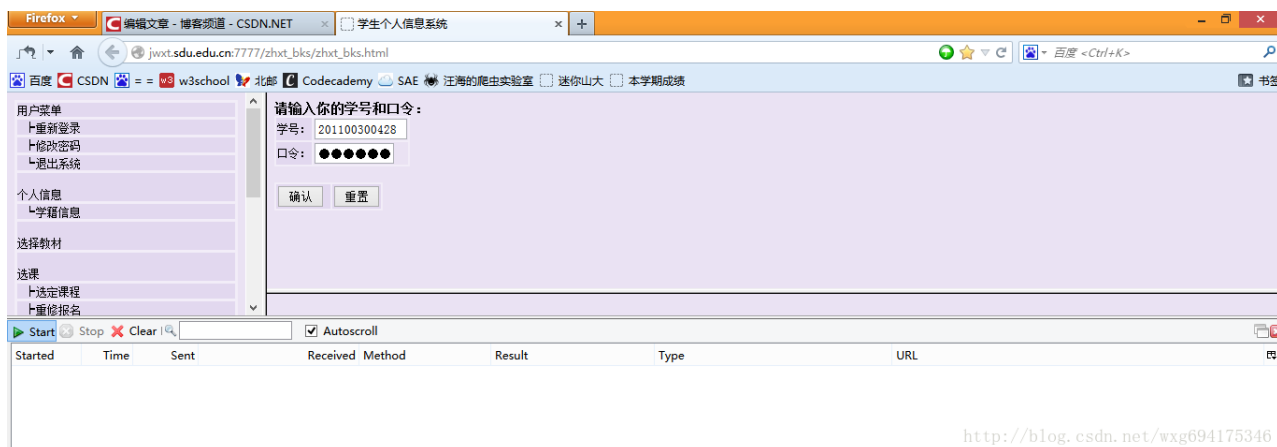


可以非常直观的查看相应的信息。点击 start 是开始检测，点击 stop 暂停检测，点击 clear 清除内容。一般在使用之前，点击 stop 暂停，然后点击 clear 清屏，确保看到的是访问当前页面获得的数据。

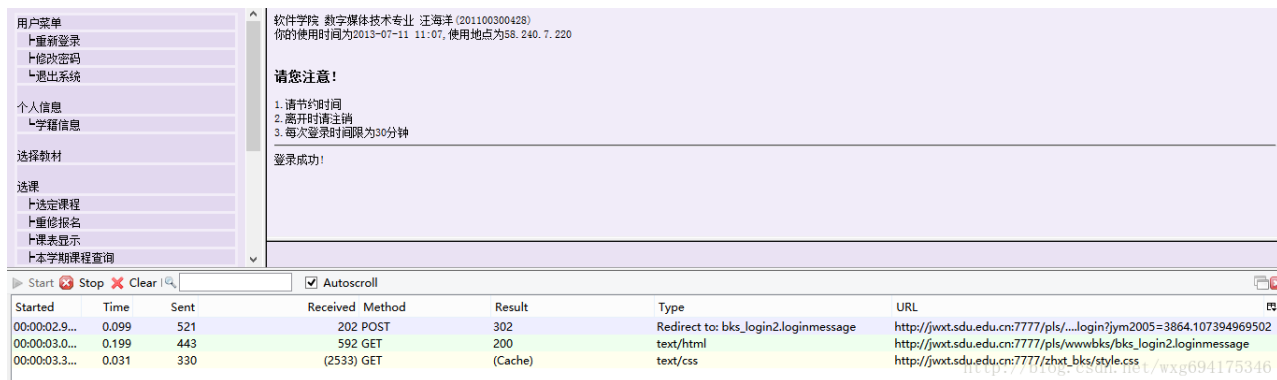
## #

## 深入敌后

下面就去山东大学的成绩查询网站，看一看在登录的时候，到底发送了那些信息。先来到登录页面，把 httpfox 打开，clear 之后，点击 start 开启检测：



输入完了个人信息，确保 httpfox 处于开启状态，然后点击确定提交信息，实现登录。这个时候可以看到，httpfox 检测到了三条信息：

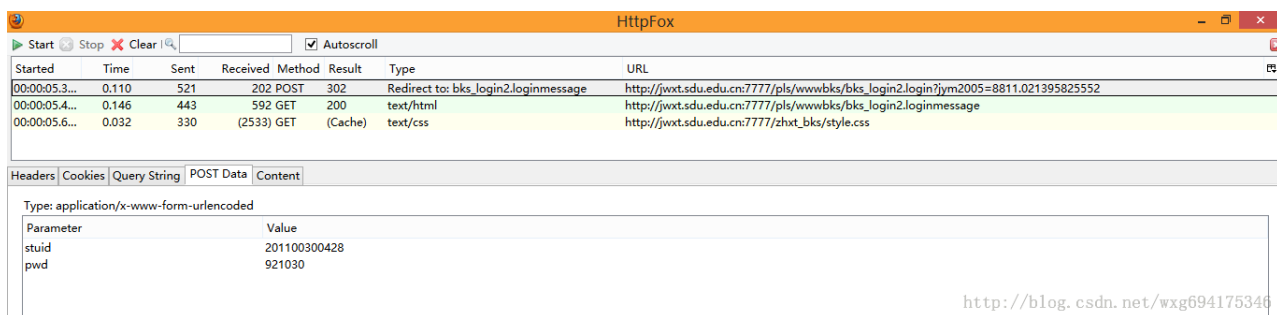


这时点击 stop 键，确保捕获到的是访问该页面之后反馈的数据，以便我们做爬虫的时候模拟登陆使用。

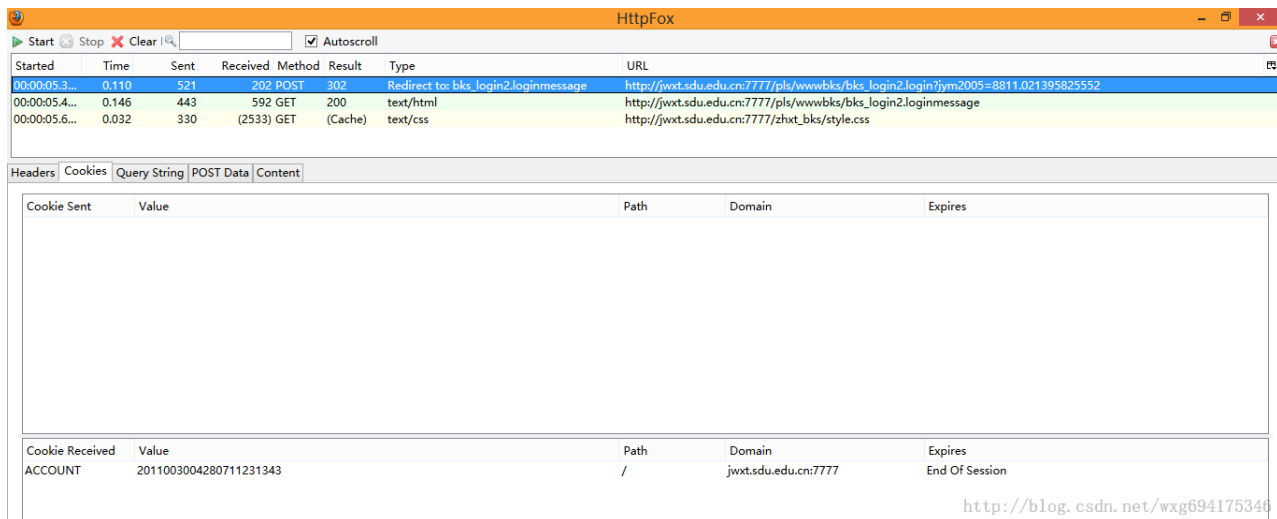
#

## 庖丁解牛

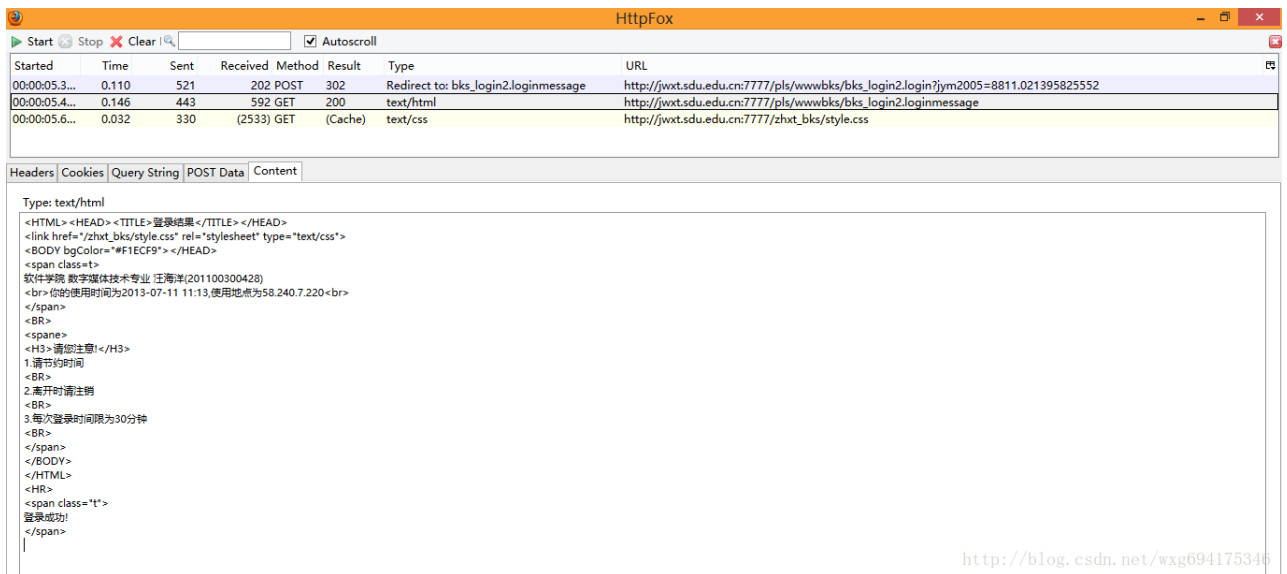
乍一看我们拿到了三个数据，两个是 GET 的一个是 POST 的，但是它们到底是什么，应该怎么用，我们还一无所知。所以，我们需要挨个查看一下捕获到的内容。先看 POST 的信息：



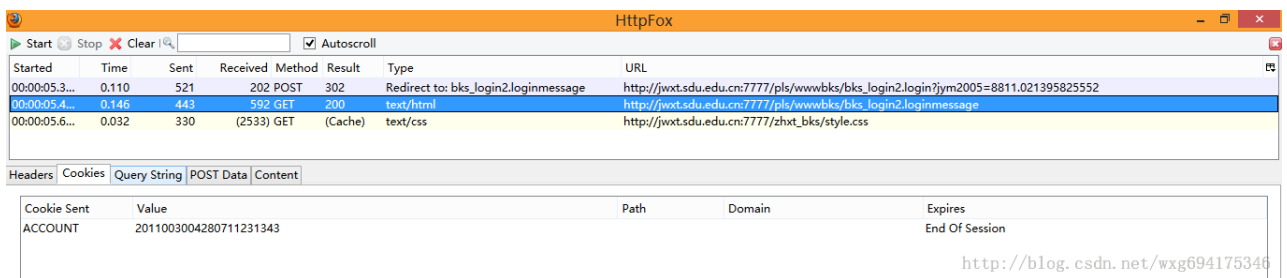
既然是 POST 的信息，我们就直接看 PostData 即可。可以看到一共 POST 两个数据，stuid 和 pwd。并且从 Type 的 Redirect to 可以看出，POST 完毕之后跳转到了 bks\_login2.loginmessage 页面。由此看出，这个数据是点击确定之后提交的表单数据。点击 cookie 标签，看看 cookie 信息：



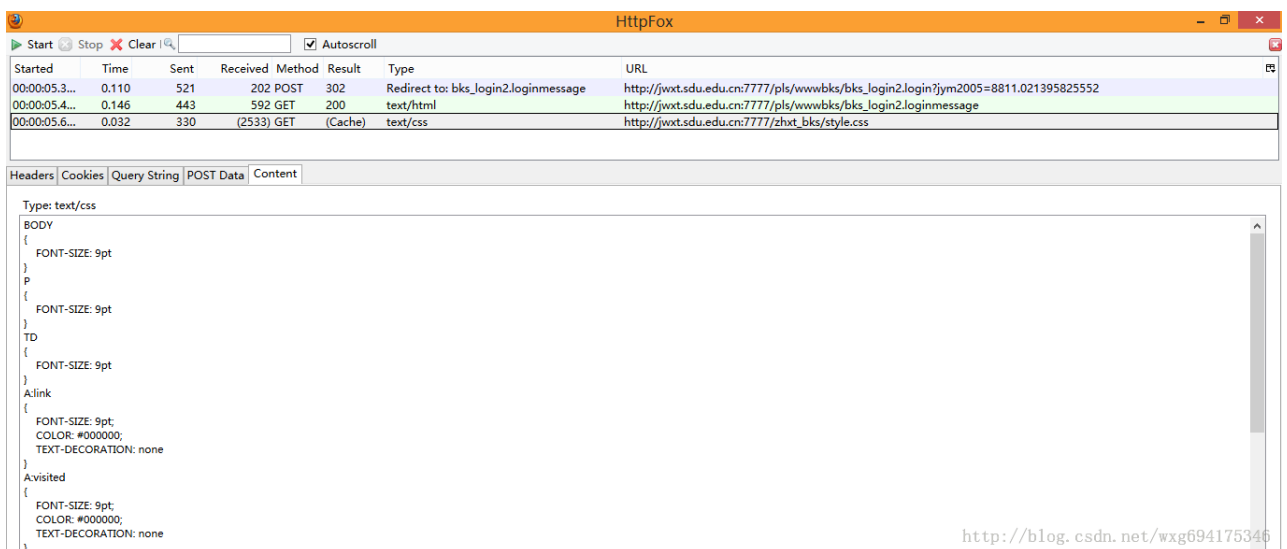
没错，收到了一个 ACCOUNT 的 cookie，并且在 session 结束之后自动销毁。那么提交之后收到了哪些信息呢？我们来看看后面的两个 GET 数据。先看第一个，我们点击 content 标签可以查看收到的内容，是不是有一种生吞活剥的快感——HTML 源码暴露无疑了：



看来这个只是显示页面的 html 源码而已，点击 cookie，查看 cookie 的相关信息：



啊哈，原来 html 页面的内容是发送了 cookie 信息之后才接受到的。再来看看最后一个接收到的信息：



大致看了一下应该只是一个叫做 style.css 的 css 文件，对我们没有太大的作用。

## #

## 冷静应战

既然已经知道了我们向服务器发送了什么数据，也知道了我们接收到了什么数据，基本的流程如下：

- 首先，我们 POST 学号和密码--->然后返回 cookie 的值
- 然后发送 cookie 给服务器--->返回页面信息。
- 获取到成绩页面的数据，用正则表达式将成绩和学分单独取出并计算加权平均数。

OK，看上去好像很简单的样纸。那下面我们就来试试看吧。但是在实验之前，还有一个问题没有解决，就是 POST 的数据到底发送到了哪里？再来看一下当初的页面：

The screenshot shows a web interface for a university system. On the left is a sidebar menu with categories like '用户菜单' (User Menu), '个人信息' (Personal Information), '选择教材' (Select Textbook), '选课' (Select Course), and '成绩查询' (Query Grades). The main area on the right is titled '请输入你的学号和口令:' (Please enter your student ID and password:). It contains two input fields: '学号:' (Student ID) and '口令:' (Password). Below these fields are two buttons: '确认' (Confirm) and '重置' (Reset). At the bottom right of the main area, there is a URL: <http://blog.csdn.net/wxg694175346>.

很明显是用一个 html 框架来实现的，也就是说，我们在地址栏看到的地址并不是右边提交表单的地址。那么怎样才能获得真正的地址。右击查看页面源代码：嗯没错，那个 name="w\_right" 的就是我们要的登录页面。网站的原来的地址是：

[http://jwxt.sdu.edu.cn:7777/zhxt\\_bks/zhxt\\_bks.html](http://jwxt.sdu.edu.cn:7777/zhxt_bks/zhxt_bks.html)

所以，真正的表单提交的地址应该是：

[http://jwxt.sdu.edu.cn:7777/zhxt\\_bks/xk\\_login.html](http://jwxt.sdu.edu.cn:7777/zhxt_bks/xk_login.html)

输入一看，果不其然：



居然是清华大学的选课系统。。。目测是我校懒得做页面了就直接借了。。。结果连标题都不改一下。。。但是这个页面依旧不是我们需要的页面，因为我们的 POST 数据提交到的页面，应该是表单 form 的 ACTION 中提交到的页面。也就是说，我们需要查看源码，来知道 POST 数据到底发送到了哪里：



嗯，目测这个才是提交 POST 数据的地址。

整理到地址栏中，完整的地址应该如下：

`http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login`

（获取的方式很简单，在火狐浏览器中直接点击那个链接就能看到这个链接的地址了）

#

---

小试牛刀

接下来的任务就是：用 python 模拟发送一个 POST 的数据并取到返回的 cookie 值。

关于 cookie 的操作可以看看这篇博文：

<http://blog.csdn.net/wxg694175346/article/details/8925978>

我们先准备一个 POST 的数据，再准备一个 cookie 的接收，然后写出源码如下：

```
-*- coding: utf-8 -*-

#-----

程序：山东大学爬虫

版本：0.1

作者：why

日期：2013-07-12

语言：Python 2.7

操作：输入学号和密码

功能：输出成绩的加权平均值也就是绩点

#-----

import urllib
import urllib2
import cookielib

cookie = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))

#需要POST的数据#

postdata=urllib.urlencode({
 'stuid':'201100300428',
 'pwd':'921030'
```



```

})

#自定义一个请求#

req = urllib2.Request(
 url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login',
 data = postdata
)

#访问该链接#

result = opener.open(req)

#打印返回的内容#

print result.read()

```

如此这般之后，再看看运行的效果：

```

Python Shell
File Edit Shell Debug Options Windows Help
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 =====
>>>
<HTML><HEAD><TITLE>登录结果</TITLE></HEAD>
<link href="/zhxt_bks/style.css" rel="stylesheet" type="text/css">
<BODY bgColor="#F1ECF9"></HEAD>

软件学院 数字媒体技术专业 汪海洋(2011100300428)

你的使用时间为2013-07-12 11:29,使用地点为222.49.196.41

<H3>请您注意!</H3>
1.请节约时间

2.离开时请注销

3.每次登录时间限为30分钟

</BODY>
</HTML>
<HR>

登录成功!

>>> |

```

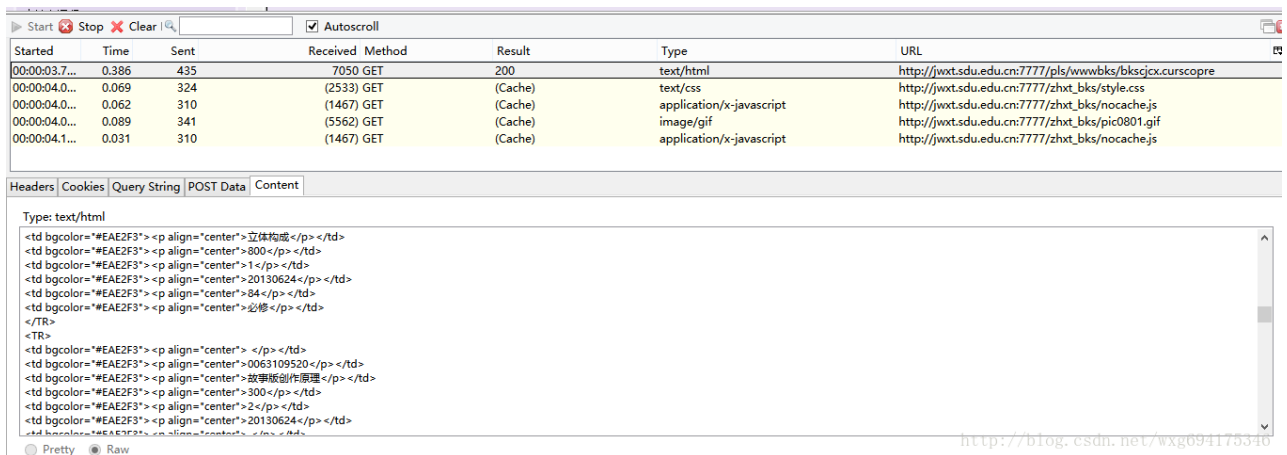
<http://blog.csdn.net/wxg694175346>

ok，如此这般，我们就算模拟登陆成功了。

## #

## 偷天换日

接下来的任务就是用爬虫获取到学生的成绩。再来看看源网站。开启 HTTPFOX 之后，点击查看成绩，发现捕获到了如下的数据：



点击第一个 GET 的数据，查看内容可以发现 Content 就是获取到的成绩的内容。

而获取到的页面链接，从页面源代码中右击查看元素，可以看到点击链接之后跳转的页面（火狐浏览器只需要右击，“查看此框架”，即可）：



从而可以得到查看成绩的链接如下：

<http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscope>

#

---

万事俱备

现在万事俱备啦，所以只需要把链接应用到爬虫里面，看看能否查看到成绩的页面。从 httpfox 可以看到，我们发送了一个 cookie 才能返回成绩的信息，所以我们就用 python 模拟一个 cookie 的发送，以此来请求成绩的信息：

```
-*- coding: utf-8 -*-

#-----

程序：山东大学爬虫

版本：0.1

作者：why

日期：2013-07-12

语言：Python 2.7

操作：输入学号和密码

功能：输出成绩的加权平均值也就是绩点

#-----

import urllib
import urllib2
import cookielib

#初始化一个CookieJar来处理Cookie的信息#

cookie = cookielib.CookieJar()

#创建一个新的opener来使用我们的CookieJar#

opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))

#需要POST的数据#
```

```

postdata=urllib.urlencode({
 'stuid':'201100300428',
 'pwd':'921030'
})

#自定义一个请求#

req = urllib2.Request(
 url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login',
 data = postdata
)

#访问该链接#

result = opener.open(req)

#打印返回的内容#

print result.read()

#打印cookie的值

for item in cookie:
 print 'Cookie: Name = '+item.name
 print 'Cookie: Value = '+item.value

#访问该链接#

result = opener.open('http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre')

#打印返回的内容#

print result.read()

```

按下 F5 运行即可，看看捕获到的数据吧：

```

<td bgcolor="#EAE2F3"><p align="center">数据结构课程设计(双语)</p></td>
<td bgcolor="#EAE2F3"><p align="center">4</p></td>
<td bgcolor="#EAE2F3"><p align="center">2</p></td>
<td bgcolor="#EAE2F3"><p align="center">20130624</p></td>
<td bgcolor="#EAE2F3"><p align="center">优秀</p></td>
<td bgcolor="#EAE2F3"><p align="center">必修</p></td>
</TR>
<TR>
<td bgcolor="#EAE2F3"><p align="center"> </p></td>
<td bgcolor="#EAE2F3"><p align="center">0901000410</p></td>
<td bgcolor="#EAE2F3"><p align="center">形势政策与社会实践(4)</p></td>
<td bgcolor="#EAE2F3"><p align="center">300</p></td>
<td bgcolor="#EAE2F3"><p align="center">1</p></td>
<td bgcolor="#EAE2F3"><p align="center">20130624</p></td>
<td bgcolor="#EAE2F3"><p align="center"></p></td>
<td bgcolor="#EAE2F3"><p align="center">必修</p></td>
</TR>
</table>
</td>
</tr>
</table>

<INPUT TYPE="submit" VALUE="显示排名">
<INPUT TYPE="reset" VALUE=" 复 位 ">
</FORM>
</CENTER>
</body>
</html>

```

<http://blog.csdn.net/wxg694175346>

>>> |

既然这样就没有什么问题了吧，用正则表达式将数据稍稍处理一下，取出学分和相应的分数就可以了。

#

手到擒来

这么一大堆 html 源码显然是不利于我们处理的，下面要用正则表达式来抠出必须的数据。

关于正则表达式的教程可以看看这个博文：

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

我们来看看成绩的源码：

```

85 <td bgcolor="#EAE2F3"><p align="center"> </p></td>
86 <td bgcolor="#EAE2F3"><p align="center">限选</p></td>
87 </TR>
88 <TR>
89 <td bgcolor="#EAE2F3"><p align="center"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="006330239002013062486 数字音乐制作"></p></td>
90 <td bgcolor="#EAE2F3"><p align="center">0063302310</p></td>
91 <td bgcolor="#EAE2F3"><p align="center">数字音乐制作</p></td>
92 <td bgcolor="#EAE2F3"><p align="center">800</p></td>
93 <td bgcolor="#EAE2F3"><p align="center">1</p></td>
94 <td bgcolor="#EAE2F3"><p align="center">20130624</p></td>
95 <td bgcolor="#EAE2F3"><p align="center">86</p></td>
96 <td bgcolor="#EAE2F3"><p align="center">限选</p></td>
97 </TR>
98 <TR>
99 <td bgcolor="#EAE2F3"><p align="center"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="014120017002013062491 自然探索与创新(创新)"></p></td>
100 <td bgcolor="#EAE2F3"><p align="center">0141200110</p></td>
101 <td bgcolor="#EAE2F3"><p align="center">自然探索与创新(创新)</p></td>
102 <td bgcolor="#EAE2F3"><p align="center">600</p></td>
103 <td bgcolor="#EAE2F3"><p align="center">2</p></td>
104 <td bgcolor="#EAE2F3"><p align="center">20130624</p></td>
105 <td bgcolor="#EAE2F3"><p align="center">91</p></td>
106 <td bgcolor="#EAE2F3"><p align="center">必修</p></td>
107 </TR>
108 <TR>
109 <td bgcolor="#EAE2F3"><p align="center"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="028100024122013062466 马克思主义原理"></p></td>
110 <td bgcolor="#EAE2F3"><p align="center">0281000210</p></td>
111 <td bgcolor="#EAE2F3"><p align="center">马克思主义原理</p></td>
112 <td bgcolor="#EAE2F3"><p align="center">312</p></td>
113 <td bgcolor="#EAE2F3"><p align="center">3</p></td>

```

<http://blog.csdn.net/wxg694175346>

既然如此，用正则表达式就易如反掌了。

我们将代码稍稍整理一下，然后用正则来取出数据：

```
-*- coding: utf-8 -*-
```

```
#-----
```

```
程序：山东大学爬虫
```

```
版本：0.1
```

```
作者：why
```

```
日期：2013-07-12
```

```
语言：Python 2.7
```

```

操作：输入学号和密码

功能：输出成绩的加权平均值也就是绩点

#-----

import urllib
import urllib2
import cookielib
import re

class SDU_Spider:
 # 申明相关的属性
 def __init__(self):
 self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login' # 登录的url
 self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 显示成绩的url
 self.cookieJar = cookielib.CookieJar() # 初始化一个CookieJar来处理Cookie的信息
 self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'}) # POST的数据
 self.weights = [] #存储权重，也就是学分
 self.points = [] #存储分数，也就是成绩
 self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookieJar))

 def sdu_init(self):
 # 初始化链接并且获取cookie
 myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata) # 自定义一个请求
 result = self.opener.open(myRequest) # 访问登录页面，获取到必须的cookie的值
 result = self.opener.open(self.resultUrl) # 访问成绩页面，获得成绩的数据
 # 打印返回的内容
 # print result.read()
 self.deal_data(result.read().decode('gbk'))
 self.print_data(self.weights);
 self.print_data(self.points);

 # 将内容从页面代码中抠出来
 def deal_data(self,myPage):
 myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?<p.*?<p.*?>(.*?)</p>.*?<p.*?<p.*?>(.*?)</p>.*?</TR>',myPage,re.S)
 for item in myItems:
 self.weights.append(item[0].encode('gbk'))
 self.points.append(item[1].encode('gbk'))

 # 将内容从页面代码中抠出来
 def print_data(self,items):

```



```

for item in items:
 print item

#调用

mySpider = SDU_Spider()
mySpider.sdu_init()

```

水平有限，正则是有点丑。运行的效果如图：

```

File Edit Shell Debug Options Windows Help
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 =====
>>>
2
1
2
2
1
2
3
1
4.5
2
1
75
84
86
91
66
76
80
优秀
>>> |

```

ok，接下来的只是数据的处理问题了。。

#

凯旋而归

完整的代码如下，至此一个完整的爬虫项目便完工了。

```
-*- coding: utf-8 -*-

#-----

程序：山东大学爬虫

版本：0.1

作者：why

日期：2013-07-12

语言：Python 2.7

操作：输入学号和密码

功能：输出成绩的加权平均值也就是绩点

#-----

import urllib
import urllib2
import cookielib
import re
import string

class SDU_Spider:
 # 申明相关的属性
 def __init__(self):
 self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login' # 登录的url
 self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 显示成绩的url
 self.cookieJar = cookielib.CookieJar() # 初始化一个CookieJar来处理Cookie的信息
 self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'}) # POST的数据
 self.weights = [] # 存储权重，也就是学分
 self.points = [] # 存储分数，也就是成绩
```

```

self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookieJar))

def sdu_init(self):
 # 初始化链接并且获取cookie
 myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata) # 自定义一个请求
 result = self.opener.open(myRequest) # 访问登录页面，获取到必须的cookie的值
 result = self.opener.open(self.resultUrl) # 访问成绩页面，获得成绩的数据
 # 打印返回的内容
 # print result.read()
 self.deal_data(result.read().decode('gbk'))
 self.calculate_date();

将内容从页面代码中抠出来
def deal_data(self,myPage):
 myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?<p.*?<p.*?>(.*?)</p>.*?<p.*?<p.*?>(.*?)</p>.*?</TR>',myPage,re.S)
 for item in myItems:
 self.weights.append(item[0].encode('gbk'))
 self.points.append(item[1].encode('gbk'))

#计算绩点，如果成绩还没出来，或者成绩是优秀良好，就不运算该成绩
def calculate_date(self):
 point = 0.0
 weight = 0.0
 for i in range(len(self.points)):
 if(self.points[i].isdigit()):
 point += string.atof(self.points[i])*string.atof(self.weights[i])
 weight += string.atof(self.weights[i])
 print point/weight

#调用

mySpider = SDU_Spider()
mySpider.sdu_init()

```



11



亮剑！爬虫框架小抓抓 Scrapy 闪亮登场！



前面十章爬虫笔记陆陆续续记录了一些简单的 Python 爬虫知识，用来解决简单的贴吧下载，绩点运算自然不在话下。不过要想批量下载大量的内容，比如知乎的所有的问答，那便显得游刃有余不有点了。于是乎，爬虫框架 Scrapy 就这样出场了！Scrapy = Scrach+Python，Scrach 这个单词是抓取的意思，暂且可以叫它：小抓抓吧。

小抓抓的官网地址：[点我点我](#)。

那么下面来简单的演示一下小抓抓 Scrapy 的安装流程。

具体流程参照：[官网教程](#)

友情提醒：一定要按照 Python 的版本下载，要不然安装的时候会提醒找不到 Python。建议大家安装 32 位是因为有些版本的必备软件 64 位不好找。

#

---

安装 Python（建议 32 位）

建议安装 Python2.7.x，3.x 貌似还不支持。安装完了记得配置环境，将 python 目录和 python 目录下的 Scripts 目录添加到系统环境变量的 Path 里。在 cmd 中输入 python 如果出现版本信息说明配置完毕。

#

---

## 安装 lxml

lxml 是一种使用 Python 编写的库，可以迅速、灵活地处理 XML。点击[这里](#)选择对应的 Python 版本安装。

#

---

安装 setuptools

用来安装 egg 文件，点击[这里](#)下载 python2.7 的对应版本的 setuptools。



#

---

安装 zope.interface

可以使用第三步下载的 setuptools 来安装 egg 文件，现在也有 exe 版本，点击[这里](#)下载。

#

---

安装 Twisted

Twisted 是用 Python 实现的基于事件驱动的网络引擎框架，点击[这里](#)下载。

#

---

安装 pyOpenSSL

pyOpenSSL 是 Python 的 OpenSSL 接口，点击[这里](#)下载。

#

---

安装 win32py

提供 win32api，点击[这里](#)下载

#

---

安装 Scrapy

终于到了激动人心的时候了！安装了那么多小部件之后终于轮到主角登场。直接在 cmd 中输入 `easy_install scrapy` 回车即可。

#

---

### 检查安装

打开一个 cmd 窗口，在任意位置执行 scrapy 命令，得到下列页面，表示环境配置成功。

```
C:\Users\SmallLight>Scrapy
Scrapy 0.22.2 - no active project

Usage:
 scrapy <command> [options] [args]

Available commands:
 bench Run quick benchmark test
 fetch Fetch a URL using the Scrapy downloader
 runspider Run a self-contained spider (without creating a project)
 settings Get settings values
 shell Interactive scraping console
 startproject Create new project
 version Print Scrapy version
 view Open URL in browser, as seen by Scrapy

 [more] More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

<http://blog.csdn.net/pleasecallmewhy>



12

爬虫框架 Scrapy 的第一个爬虫示例入门教程



我们使用 dmoz.org 这个网站来作为小抓抓一展身手的对象。

首先先要回答一个问题。

问：把网站装进爬虫里，总共分几步？

答案很简单，四步：

- 新建项目 (Project)：新建一个新的爬虫项目
- 明确目标 (Items)：明确你想要抓取的目标
- 制作爬虫 (Spider)：制作爬虫开始爬取网页
- 存储内容 (Pipeline)：设计管道存储爬取内容

好的，基本流程既然确定了，那接下来就一步一步的完成就可以了。



#

---

新建项目（Project）

在空目录下按住 Shift 键右击，选择“在此处打开命令窗口”，输入一下命令：

```
scrapy startproject tutorial
```

其中，tutorial 为项目名称。

可以看到将会创建一个 tutorial 文件夹，目录结构如下：

```
tutorial/
 scrapy.cfg
 tutorial/
 __init__.py
 items.py
 pipelines.py
 settings.py
 spiders/
 __init__.py
 ...
```

下面来简单介绍一下各个文件的作用：

- scrapy.cfg：项目的配置文件
- tutorial/：项目的 Python 模块，将会从这里引用代码
- tutorial/items.py：项目的 items 文件
- tutorial/pipelines.py：项目的 pipelines 文件
- tutorial/settings.py：项目的设置文件
- tutorial/spiders/：存储爬虫的目录

## #

---

### 明确目标（Item）

在 Scrapy 中，items 是用来加载抓取内容的容器，有点像 Python 中的 Dic，也就是字典，但是提供了一些额外的保护减少错误。

一般来说，item 可以用 scrapy.item.Item 类来创建，并且用 scrapy.item.Field 对象来定义属性（可以理解成类似于 ORM 的映射关系）。

接下来，我们开始来构建 item 模型（model）。

首先，我们想要的内容有：

- 名称（name）
- 链接（url）
- 描述（description）

修改 tutorial 目录下的 items.py 文件，在原本的 class 后面添加我们自己的 class。因为要抓 dmoz.org 网站的内容，所以我们可以将其命名为 DmozItem：

```
Define here the models for your scraped items

#

See documentation in:

http://doc.scrapy.org/en/latest/topics/items.html

from scrapy.item import Item, Field

class TutorialItem(Item):
 # define the fields for your item here like:
 # name = Field()
 pass

class DmozItem(Item):
 title = Field()
 link = Field()
 desc = Field()
```

刚开始看起来可能会有些看不懂，但是定义这些 item 能让你用其他组件的时候知道你的 items 到底是什么。可以把 Item 简单的理解成封装好的类对象。

## #

## 制作爬虫 (Spider)

制作爬虫，总体分两步：先爬再取。

也就是说，首先你要获取整个网页的所有内容，然后再取出其中对你有用的部分。

## #

## 爬

Spider 是用户自己编写的类，用来从一个域（或域组）中抓取信息。他们定义了用于下载的 URL 列表、跟踪链接的方案、解析网页内容的方式，以此来提取 items。要建立一个 Spider，你必须用 scrapy.spider.BaseSpider 创建一个子类，并确定三个强制的属性：

- name: 爬虫的识别名称，必须是唯一的，在不同的爬虫中你必须定义不同的名字。
- start\_urls: 爬取的 URL 列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些 urls 开始。其他子 URL 将会从这些起始 URL 中继承性生成。
- parse(): 解析的方法，调用的时候传入从每一个 URL 传回的 Response 对象作为唯一参数，负责解析并匹配抓取的数据(解析为 item)，跟踪更多的 URL。

这里可以参考宽度爬虫教程中提及的思想来帮助理解，教程传送：[\[Java\] 知乎下巴第5集：使用HttpClient工具包和宽度爬虫](#)。

也就是把 Url 存储下来并依此为起点逐步扩散开去，抓取所有符合条件的网页 Url 存储起来继续爬取。

下面我们来写第一只爬虫，命名为 dmoz\_spider.py，保存在 tutorial/spiders 目录下。

dmoz\_spider.py 代码如下：

```
from scrapy.spider import Spider

class DmozSpider(Spider):
 name = "dmoz"
 allowed_domains = ["dmoz.org"]
 start_urls = [
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
]
```

```
def parse(self, response):
 filename = response.url.split("/")[-2]
 open(filename, 'wb').write(response.body)
```

allow\_domains 是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页。从 parse 函数可以看出，将链接的最后两个地址取出作为文件名进行存储。然后运行一下看看，在 tutorial 目录下按住 shift 右击，在此处打开命令窗口，输入：

```
scrapy crawl dmoz
```

运行结果如图：

```
ImportError: cannot import name ResponseTypes
File "C:\Python27\lib\site-packages\scrapy-0.22.2-py2.7.egg\scrapy\responsetyp
es.py", line 113, in <module>
 responsetypes = ResponseTypes()
File "C:\Python27\lib\site-packages\scrapy-0.22.2-py2.7.egg\scrapy\responsetyp
es.py", line 34, in __init__
 self.mimetypes = MimeTypes()
File "C:\Python27\lib\mimetypes.py", line 66, in __init__
 init()
File "C:\Python27\lib\mimetypes.py", line 358, in init
 db.read_windows_registry()
File "C:\Python27\lib\mimetypes.py", line 258, in read_windows_registry
 for subkeyname in enum_types(hkcr):
File "C:\Python27\lib\mimetypes.py", line 249, in enum_types
 ctype = ctype.encode(default_encoding) # omit in 3.x!
UnicodeDecodeError: 'ascii' codec can't decode byte 0xb0 in position 1: ordinal
not in range(128)
```

报错了：

```
UnicodeDecodeError: 'ascii' codec can't decode byte 0xb0 in position 1: ordinal not in range(128)
```

运行第一个 Scrapy 项目就报错，真是命运多舛。应该是出了编码问题，谷歌了一下找到了解决方案：

在 python 的 Lib\site-packages 文件夹下新建一个 sitecustomize.py：

```
import sys
sys.setdefaultencoding('gb2312')
```

再次运行，OK，问题解决了，看一下结果：

```

2014-02-22 15:59:20+0800 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Books/> (referer: None)
2014-02-22 15:59:20+0800 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/> (referer: None)
2014-02-22 15:59:20+0800 [dmoz] INFO: Closing spider (finished)
2014-02-22 15:59:20+0800 [dmoz] INFO: Dumping Scrapy stats:
 {'downloader/request_bytes': 516,
 'downloader/request_count': 2,
 'downloader/request_method_count/GET': 2,
 'downloader/response_bytes': 15897,
 'downloader/response_count': 2,
 'downloader/response_status_count/200': 2,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2014, 2, 22, 7, 59, 20, 930000),
 'log_count/DEBUG': 4,
 'log_count/INFO': 7,
 'response_received_count': 2,
 'scheduler/dequeued': 2,
 'scheduler/dequeued/memory': 2,
 'scheduler/enqueued': 2,
 'scheduler/enqueued/memory': 2,
 'start_time': datetime.datetime(2014, 2, 22, 7, 59, 19, 182000)}
2014-02-22 15:59:20+0800 [dmoz] INFO: Spider closed (finished)

```

最后一句 INFO: Closing spider (finished)表明爬虫已经成功运行并且自行关闭了。包含[dmoz]的行，那对应着我们的爬虫运行的结果。可以看到 start\_urls 中定义每个URL都有日志行。还记得我们的 start\_urls 吗？

```
http://www.dmoz.org/Computers/Programming/Languages/Python/Books
```

```
http://www.dmoz.org/Computers/Programming/Languages/Python/Resources
```

因为这些 URL 是起始页面，所以他们没有引用(referrers)，所以在它们的每行末尾你会看到 ( referer: <None> )。在 parse 方法的作用下，两个文件被创建：分别是 Books 和 Resources，这两个文件中有 URL 的页面内容。

那么在刚刚的电闪雷鸣之中到底发生了什么呢？首先，Scrapy 为爬虫的 start\_urls 属性中的每个 URL 创建了一个 scrapy.http.Request 对象，并将爬虫的 parse 方法指定为回调函数。然后，这些 Request 被调度并执行，之后通过 parse()方法返回 scrapy.http.Response 对象，并反馈给爬虫。

#

取

爬取整个网页完毕，接下来的就是取过程了。光存储一整个网页还是不够用的。在基础的爬虫里，这一步可以用正则表达式来抓。在 Scrapy 里，使用一种叫做 XPath selectors 的机制，它基于 XPath 表达式。如果你了解更多 selectors 和其他机制你可以查阅资料：[点我点我](#)

这是一些 XPath 表达式的例子和他们的含义

- /html/head/title: 选择 HTML 文档 `<head>` 元素下面的 `<title>` 标签。
- /html/head/title/text(): 选择前面提到的 `<title>` 元素下面的文本内容
- //td: 选择所有 `<td>` 元素
- //div[@class="mine"]: 选择所有包含 `class="mine"` 属性的 div 标签元素

以上只是几个使用 XPath 的简单例子，但是实际上 XPath 非常强大。可以参照 W3C 教程：[点我点我](#)。

为了方便使用 XPaths，Scrapy 提供 XPathSelector 类，有两种可以选择，HtmlXPathSelector(HTML 数据解析)和 XmlXPathSelector(XML 数据解析)。必须通过一个 Response 对象对他们进行实例化操作。你会发现 Selector 对象展示了文档的节点结构。因此，第一个实例化的 selector 必与根节点或者是整个目录有关。

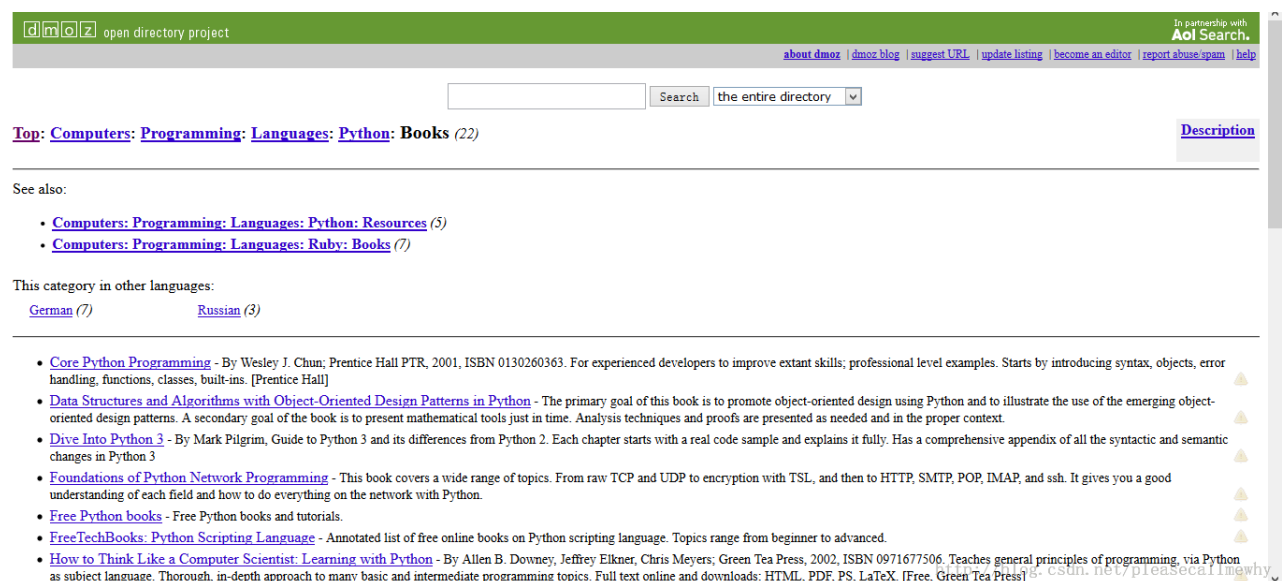
在 Scrapy 里面，Selectors 有四种基础的方法（点击查看 API 文档）：

- xpath(): 返回一系列的 selectors，每一个 select 表示一个 xpath 参数表达式选择的节点
- css(): 返回一系列的 selectors，每一个 select 表示一个 css 参数表达式选择的节点
- extract(): 返回一个 unicode 字符串，为选中的数据
- re(): 返回一串一个 unicode 字符串，为使用正则表达式抓取出来的内容

## #

### xpath 实验

下面我们在 Shell 里面尝试一下 Selector 的用法。实验的网址：<http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>

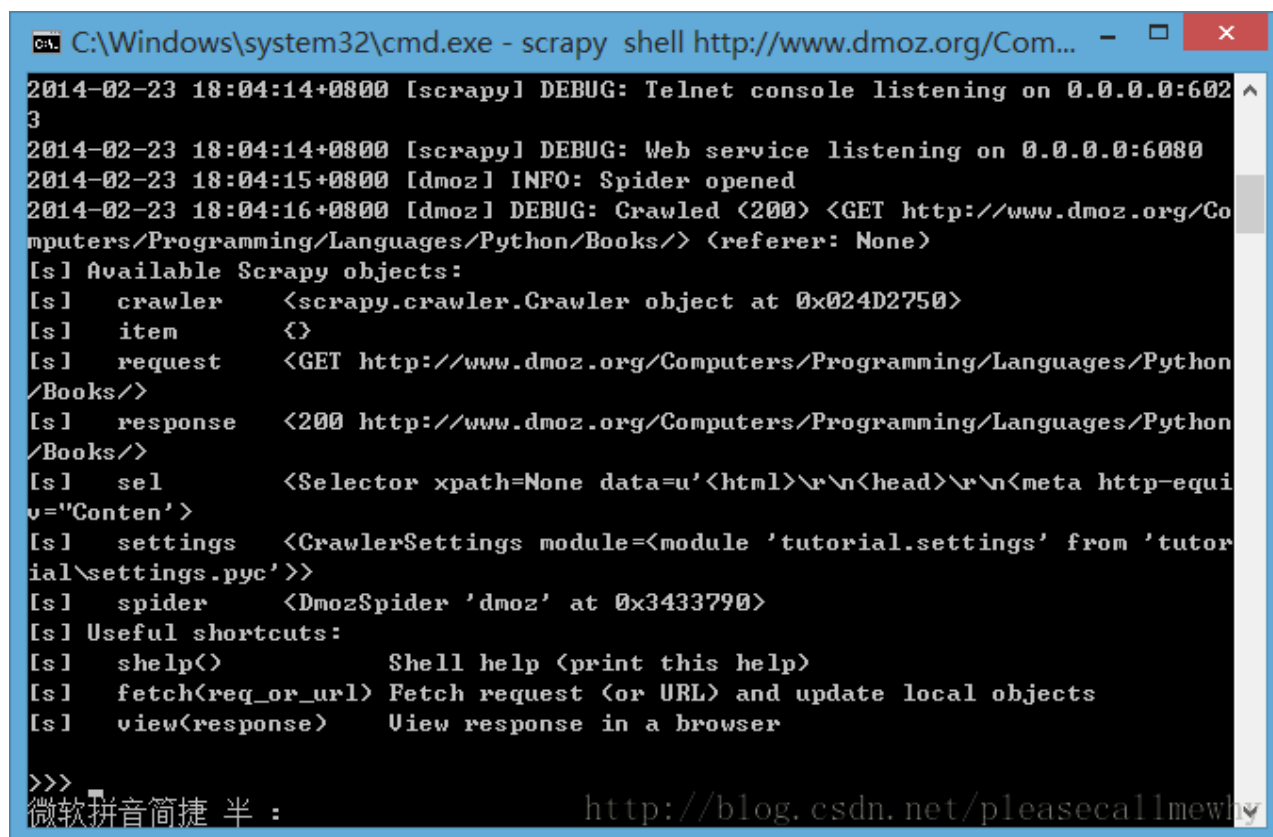


The screenshot shows the DMOZ directory page for "Computers: Programming: Languages: Python: Books". The page has a green header with the DMOZ logo and navigation links. Below the header is a search bar and a dropdown menu. The main content area shows the category path "Top: Computers: Programming: Languages: Python: Books (22)" and a "Description" link. Under "See also:", there are links to "Computers: Programming: Languages: Python: Resources (5)" and "Computers: Programming: Languages: Ruby: Books (7)". Under "This category in other languages:", there are links for "German (7)" and "Russian (3)". The bottom section lists several books with their titles, authors, and brief descriptions, each followed by a small yellow icon.

熟悉完了实验的小白鼠，接下来就是用 Shell 爬取网页了。进入到项目的顶层目录，也就是第一层 tutorial 文件夹下，在 cmd 中输入：

```
scrapy shell http://www.dmoz.org/Computers/Programming/Languages/Python/Books/
```

回车后可以看到如下的内容：



```
C:\Windows\system32\cmd.exe - scrapy shell http://www.dmoz.org/Com...
2014-02-23 18:04:14+0800 [scrapy] DEBUG: Telnet console listening on 0.0.0.0:6023
2014-02-23 18:04:14+0800 [scrapy] DEBUG: Web service listening on 0.0.0.0:6080
2014-02-23 18:04:15+0800 [dmoz] INFO: Spider opened
2014-02-23 18:04:16+0800 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Books/> (referer: None)
[scrapy] Available Scrapy objects:
[scrapy] crawler <scrapy.crawler.Crawler object at 0x024D2750>
[scrapy] item {}
[scrapy] request <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
[scrapy] response <200 http://www.dmoz.org/Computers/Programming/Languages/Python/Books/>
[scrapy] sel <Selector xpath=None data=u'<html>\r\n<head>\r\n<meta http-equiv='Content'>'>
[scrapy] settings <CrawlerSettings module=<module 'tutorial.settings' from 'tutorial\settings.pyc'>>
[scrapy] spider <DmozSpider 'dmoz' at 0x3433790>
[scrapy] Useful shortcuts:
[scrapy] shelp() Shell help <print this help>
[scrapy] fetch(req_or_url) Fetch request <or URL> and update local objects
[scrapy] view(response) View response in a browser

>>>
```

在 Shell 载入后，你将获得 response 回应，存储在本地变量 response 中。所以如果你输入 response.body，你将会看到 response 的 body 部分，也就是抓取到的页面内容：





现在就像是一大堆沙子握在手里，里面藏着我们想要的金子，所以下一步，就是用筛子摇两下，把杂质出去，选出关键的内容。

selector 就是这样一个筛子。在旧的版本中，Shell 实例化两种 selectors，一个是解析 HTML 的 hxs 变量，一个是解析 XML 的 xxs 变量。

而现在的 Shell 为我们准备好的 selector 对象，sel，可以根据返回的数据类型自动选择最佳的解析方案(XML or HTML)。

然后我们来捣弄一下！~

要彻底搞清楚这个问题，首先先要知道，抓到的页面到底是个什么样子。比如，我们要抓取网页的标题，也就是

`<title>` 这个标签：

```

<title>
Open Directory - Computers: Programming: Languages: Python: Books
</title>
```

可以输入：

```
sel.xpath('//title')
```

结果就是：

```
>>> sel.xpath('//title')
[<Selector xpath='//title' data=u'<title>Open Directory - Computers: Progr'>]
```

这样就能把这个标签取出来了，用 `extract()` 和 `text()` 还可以进一步做处理。

备注：简单的罗列一下有用的 xpath 路径表达式：

表达式	描述
nodename	选取此节点的所有子节点。
/	从根节点选取。
//	从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。
.	选取当前节点。
..	选取当前节点的父节点。
@	选取属性。

全部的实验结果如下，In[i]表示第 i 次实验的输入，Out[i]表示第 i 次结果的输出（建议大家参照：[W3C 教程](#)）：

```
In [1]: sel.xpath('//title')
Out[1]: [<Selector xpath='//title' data=u'<title>Open Directory - Computers: Progr'>]
```

```

In [2]: sel.xpath('//title').extract()
Out[2]: [u'<title>Open Directory – Computers: Programming: Languages: Python: Books</title>']

In [3]: sel.xpath('//title/text()')
Out[3]: [<Selector xpath='//title/text()' data=u'Open Directory – Computers: Programming:'>]

In [4]: sel.xpath('//title/text()').extract()
Out[4]: [u'Open Directory – Computers: Programming: Languages: Python: Books']

In [5]: sel.xpath('//title/text()').re('(\w+):')
Out[5]: [u'Computers', u'Programming', u'Languages', u'Python']

```

当然 title 这个标签对我们来说没有太多的价值，下面我们就来真正抓取一些有意义的东西。使用火狐的审查元素我们可以清楚地看到，我们需要的东西如下：



```

<ul class="directory-url" style="margin-left:0;">


```

By Wesley ...

```

<div class="flag"></div>


```

<http://blog.csdn.net/pleasecallmewhy>

我们可以用如下代码来抓取这个 `<li>` 标签：

```
sel.xpath('//ul/li')
```

从 `<li>` 标签中，可以这样获取网站的描述：

```
sel.xpath('//ul/li/text()').extract()
```

可以这样获取网站的标题：

```
sel.xpath('//ul/li/a/text()').extract()
```

可以这样获取网站的超链接：

```
sel.xpath('//ul/li/a/@href').extract()
```

当然，前面的这些例子是直接获取属性的方法。我们注意到 xpath 返回了一个对象列表，那么我们也可以直接调用这个列表中对象的属性挖掘更深的节点。

（参考：[Nesting selectors](#) and [Working with relative XPath](#)s in the [Selectors](#)）：

```

sites = sel.xpath('//ul/li')
for site in sites:
 title = site.xpath('a/text()').extract()
 link = site.xpath('a/@href').extract()

```

```
desc = site.xpath('text()').extract()
print title, link, desc
```

## #

### xpath 实战

我们用 shell 做了这么久的实战，最后我们可以把前面学习到的内容应用到 dmoz\_spider 这个爬虫中。

在原爬虫的 parse 函数中做如下修改：

```
from scrapy.spider import Spider
from scrapy.selector import Selector

class DmozSpider(Spider):
 name = "dmoz"
 allowed_domains = ["dmoz.org"]
 start_urls = [
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
]

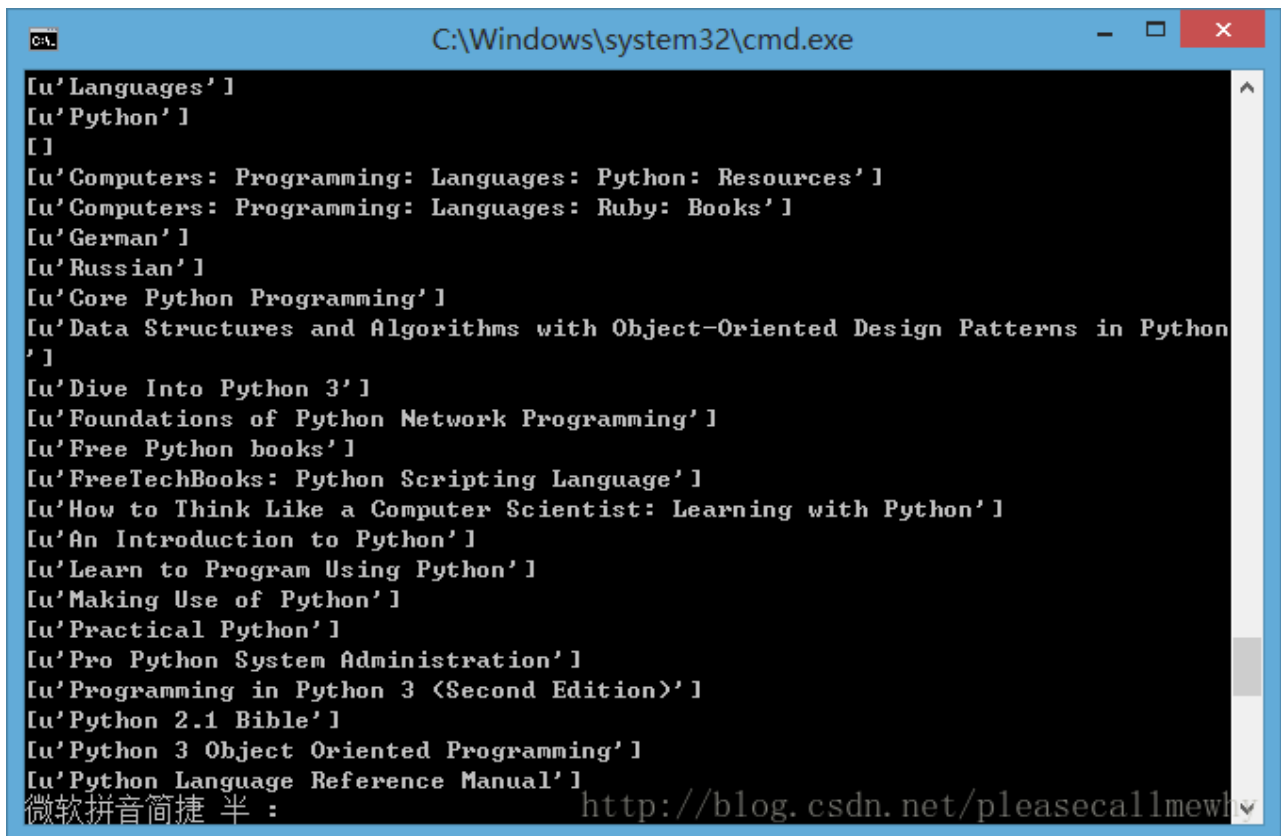
 def parse(self, response):
 sel = Selector(response)
 sites = sel.xpath('//ul/li')
 for site in sites:
 title = site.xpath('a/text()').extract()
 link = site.xpath('a/@href').extract()
 desc = site.xpath('text()').extract()
 print title
```

注意，我们从 scrapy.selector 中导入了 Selector 类，并且实例化了一个新的 Selector 对象。这样我们就可以像 Shell 中一样操作 xpath 了。

我们来试着输入一下命令运行爬虫（在 tutorial 根目录里面）：

```
scrapy crawl dmoz
```

运行结果如下：



```

C:\Windows\system32\cmd.exe

[u'Languages']
[u'Python']
[]
[u'Computers: Programming: Languages: Python: Resources']
[u'Computers: Programming: Languages: Ruby: Books']
[u'German']
[u'Russian']
[u'Core Python Programming']
[u'Data Structures and Algorithms with Object-Oriented Design Patterns in Python']
[u'Dive Into Python 3']
[u'Foundations of Python Network Programming']
[u'Free Python books']
[u'FreeTechBooks: Python Scripting Language']
[u'How to Think Like a Computer Scientist: Learning with Python']
[u'An Introduction to Python']
[u'Learn to Program Using Python']
[u'Making Use of Python']
[u'Practical Python']
[u'Pro Python System Administration']
[u'Programming in Python 3 (Second Edition)']
[u'Python 2.1 Bible']
[u'Python 3 Object Oriented Programming']
[u'Python Language Reference Manual']
微软拼音简捷 半 : http://blog.csdn.net/pleasecallmewh

```

果然，成功的抓到了所有的标题。但是好像不太对啊，怎么 Top, Python 这种导航栏也抓取出来了呢？我们只需要红圈中的内容：

**Top: Computers: Programming: Languages: Python: Books (22)**

See also:

- [Computers: Programming: Languages: Python: Resources \(5\)](#)
- [Computers: Programming: Languages: Ruby: Books \(7\)](#)

This category in other languages:

[German \(7\)](#)      [Russian \(3\)](#)

- [Core Python Programming](#) - By Wesley J. Chun; Prentice Hall PTR, 2001, ISBN 0130260363. For experienced developers to improve extant skills; pro handling, functions, classes, built-ins. [Prentice Hall]
- [Data Structures and Algorithms with Object-Oriented Design Patterns in Python](#) - The primary goal of this book is to promote object-oriented design patterns. A secondary goal of the book is to present mathematical tools just in time. Analysis techniques and proofs are presented as needed
- [Dive Into Python 3](#) - By Mark Pilgrim, Guide to Python 3 and its differences from Python 2. Each chapter starts with a real code sample and explains it changes in Python 3
- [Foundations of Python Network Programming](#) - This book covers a wide range of topics. From raw TCP and UDP to encryption with TLS, and then understanding of each field and how to do everything on the network with Python.
- [Free Python books](#) - Free Python books and tutorials.
- [FreeTechBooks: Python Scripting Language](#) - Annotated list of free online books on Python scripting language. Topics range from beginner to advanced
- [How to Think Like a Computer Scientist: Learning with Python](#) - By Allen B. Downey, Jeffrey Elkner, Chris Meyers; Green Tea Press, 2002; ISBN 0-975-1206-9-3. Thorough, in-depth approach to many basic and intermediate programming topics. Full text online and downloadable: HTML, PDF, PS, T

看来是我们的 xpath 语句有点问题，没有仅仅把我们需要的项目名称抓取出来，也抓了一些无辜的但是 xpath 语法相同的元素。审查元素我们发现我们需要的 `<ul>` 具有 `class='directory-url'` 的属性，那么只要把 xpath 语句改成 `sel.xpath('//ul[@class="directory-url"]/li')` 即可将 xpath 语句做如下调整：

```
from scrapy.spider import Spider
from scrapy.selector import Selector

class DmozSpider(Spider):
 name = "dmoz"
 allowed_domains = ["dmoz.org"]
 start_urls = [
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
]

 def parse(self, response):
 sel = Selector(response)
 sites = sel.xpath('//ul[@class="directory-url"]/li')
 for site in sites:
 title = site.xpath('a/text()').extract()
 link = site.xpath('a/@href').extract()
 desc = site.xpath('text()').extract()
 print title
```

成功抓出了所有的标题，绝对没有滥杀无辜：

```

C:\Windows\system32\cmd.exe

[u' Social Bug']
2014-02-23 20:32:07+0800 [dmoz] DEBUG: Crawled (200) <GET http://www.dmoz.org/Computers/Programming/Languages/Python/Books/> <referer: None>
[u' Core Python Programming']
[u' Data Structures and Algorithms with Object-Oriented Design Patterns in Python']
[u' Dive Into Python 3']
[u' Foundations of Python Network Programming']
[u' Free Python books']
[u' FreeTechBooks: Python Scripting Language']
[u' How to Think Like a Computer Scientist: Learning with Python']
[u' An Introduction to Python']
[u' Learn to Program Using Python']
[u' Making Use of Python']
[u' Practical Python']
[u' Pro Python System Administration']
[u' Programming in Python 3 (Second Edition)']
[u' Python 2.1 Bible']
[u' Python 3 Object Oriented Programming']
[u' Python Language Reference Manual']
[u' Python Programming Patterns']
[u' Python Programming with the Java Class Libraries: A Tutorial for Building Web and Enterprise Applications with Jython']
[u' Python: Visual QuickStart Guide']
微软拼音简捷 半 : http://blog.csdn.net/pleasecallmew

```

#

使用 Item

接下来我们来看一看如何使用 Item。前面我们说过，Item 对象是自定义的 python 字典，可以使用标准字典语法获取某个属性的值：

```

>>> item = DmozItem()
>>> item['title'] = 'Example title'
>>> item['title']
'Example title'

```

作为一只爬虫，Spiders 希望能将其抓取的数据存放到 Item 对象中。为了返回我们抓取数据，spider 的最终代码应当是这样：

```

from scrapy.spider import Spider
from scrapy.selector import Selector

from tutorial.items import DmozItem

class DmozSpider(Spider):
 name = "dmoz"

```

```
allowed_domains = ["dmoz.org"]
start_urls = [
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
 "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
]

def parse(self, response):
 sel = Selector(response)
 sites = sel.xpath('//ul[@class="directory-url"]/li')
 items = []
 for site in sites:
 item = DmozItem()
 item['title'] = site.xpath('a/text()').extract()
 item['link'] = site.xpath('a/@href').extract()
 item['desc'] = site.xpath('text()').extract()
 items.append(item)
 return items
```



#

存储内容 (Pipeline)

保存信息的最简单的方法是通过 [Feed exports](#)，主要有四种：JSON，JSON lines，CSV，XML。我们将结果用最常用的 JSON 导出，命令如下：

```
scrapy crawl dmoz -o items.json -t json
```

-o 后面是导出文件名，-t 后面是导出类型。然后来看一下导出的结果，用文本编辑器打开 json 文件即可（为了方便显示，在 item 中删去了除了 title 之外的属性）：

```
{
 "title": ["Free Python and Zope Hosting Directory"],
 "title": ["O'Reilly Python Center"],
 "title": ["Python Developer's Guide"],
 "title": ["Social Bug"],
 "title": ["Core Python Programming"],
 "title": ["Data Structures and Algorithms with Object-Oriented Design Patterns in Python"],
 "title": ["Dive Into Python 3"],
 "title": ["Foundations of Python Network Programming"],
 "title": ["Free Python books"],
 "title": ["FreeTechBooks: Python Scripting Language"],
 "title": ["How to Think Like a Computer Scientist: Learning with Python"],
 "title": ["An Introduction to Python"],
 "title": ["Learn to Program Using Python"],
 "title": ["Making Use of Python"],
 "title": ["Practical Python"],
 "title": ["Pro Python System Administration"],
 "title": ["Programming in Python 3 (Second Edition)"],
 "title": ["Python 2.1 Bible"],
 "title": ["Python 3 Object Oriented Programming"],
 "title": ["Python Language Reference Manual"],
 "title": ["Python Programming Patterns"],
 "title": ["Python Programming with the Java Class Libraries: A Tutorial for Building Web and Enterpri"],
 "title": ["Python: Visual QuickStart Guide"],
 "title": ["Sams Teach Yourself Python in 24 Hours"],
 "title": ["Text Processing in Python"],
 "title": ["XML Processing with Python"]
}
```

<http://blog.csdn.net/pleasecallmewhy>

因为这个只是一个小型的例子，所以这样简单的处理就可以了。如果你想用抓取的 items 做更复杂的事情，你可以写一个 Item Pipeline(条目管道)。这个我们以后再慢慢玩^\_^

# 极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/python-crawler/>