Python爬虫学习系列教程

极客学院出版

前言

网络爬虫，即 Web Spider，是一个很形象的名字。目前爬虫开发的语言的主要是 Python，本教程是作者实际开发使用的心得总结，还附加几个小的爬虫案例，帮助读者更好的学习 Python 开发爬虫。

适用人群

适用于爬虫初学者，如果你对高效抓取数据有兴趣，那么本教程将会是你不错的选择。

学习前提

学习本教程前，你需要对 Python 语言有一定的了解。

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

> 致谢
>
> http://cuiqingcai.com/1052.html

目录

1 综述

大家好哈，最近博主在学习 Python，学习期间也遇到一些问题，获得了一些经验，在此将自己的学习系统地整理下来，如果大家有兴趣学习爬虫的话，可以将这些文章作为参考，也欢迎大家一共分享学习经验。

Python 版本:2.7，Python 3 请另寻其他博文。

首先爬虫是什么?

> 网络爬虫（又被称为网页蜘蛛，网络机器人，在 FOAF 社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动的抓取万维网信息的程序或者脚本。 根据我的经验，要学习 Python 爬虫，我们要学习的共有以下几点:

- Python 基础知识
- Python 中 urllib 和 urllib2 库的用法
- Python 正则表达式
- Python 爬虫框架 Scrapy
- Python 爬虫更高级的功能

# #

Python 基础学习

首先，我们要用 Python 写爬虫，肯定要了解 Python 的基础吧，万丈高楼平地起，不能忘啦那地基，哈哈，那么我就分享一下自己曾经看过的一些 Python 教程，小伙伴们可以作为参考。

#

极客学院 Python 教程

曾经有一些基础的语法是在慕课网上看的，上面附有一些练习，学习完之后可以作为练习，感觉效果还是蛮不错的，不过稍微遗憾的是内容基本上都是最基础的，入门开始的话，就这个吧

学习网址：极客学院 Python 教程

#

廖雪峰 Python 教程

后来，我发现了廖老师的 Python 教程，讲的那是非常通俗易懂哪，感觉也是非常不错，大家如果想进一步了解 Python 就看一下这个吧。

学习网址：廖雪峰 Python 教程

#

简明 Python 教程

还有一个我看过的，简明 Python 教程，感觉讲的也不错

学习网址：简明 Python 教程

# #

Python urllib 和 urllib2 库的用法

urllib 和 urllib2 库是学习 Python 爬虫最基本的库，利用这个库我们可以得到网页的内容，并对内容用正则表达式提取分析，得到我们想要的结果。这个在学习过程中我会和大家分享的。

# #

Python 正则表达式

Python 正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。这个在后面的博文会分享的。

#

爬虫框架 Scrapy

如果你是一个 Python 高手，基本的爬虫知识都已经掌握了，那么就寻觅一下 Python 框架吧，我选择的框架是 Scrapy 框架。这个框架有什么强大的功能呢？下面是它的官方介绍：

> HTML, XML 源数据 选择及提取 的内置支持 提供了一系列在 spider 之间共享的可复用的过滤器(即 Item Loaders)，对智能处理爬取数据提供了内置支持。 通过 feed 导出 提供了多格式(JSON、CSV、XML)，多存储后端(FTP、S3、本地文件系统)的内置支持 提供了 media pipeline，可以 自动下载 爬取到的数据中的图片(或者其他资源)。 高扩展性。您可以通过使用 signals ，设计好的 API(中间件, extensions, pipelines)来定制实现您的功能。

内置的中间件及扩展为下列功能提供了支持:

- cookies and session 处理
- HTTP 压缩
- HTTP 认证
- HTTP 缓存
- user-agent模拟
- robots.txt

爬取深度限制

针对非英语语系中不标准或者错误的编码声明, 提供了自动检测以及健壮的编码支持。

支持根据模板生成爬虫。在加速爬虫创建的同时，保持在大型项目中的代码更为一致。详细内容请参阅 genspider 命令。

针对多爬虫下性能评估、失败检测，提供了可扩展的 状态收集工具 。

提供 交互式 shell 终端 , 为您测试 XPath 表达式，编写和调试爬虫提供了极大的方便

提供 System service, 简化在生产环境的部署及运行

内置 Web service, 使您可以监视及控制您的机器

内置 Telnet 终端 ，通过在 Scrapy 进程中钩入 Python 终端，使您可以查看并且调试爬虫

Logging 为您在爬取过程中捕捉错误提供了方便

支持 Sitemaps 爬取

具有缓存的 DNS 解析器

官方文档：http://doc.scrapy.org/en/latest/

等我们掌握了基础的知识，再用这个 Scrapy 框架吧！

2

爬虫基础了解

# #

什么是爬虫

爬虫，即网络爬虫，大家可以理解为在网络上爬行的一直蜘蛛，互联网就比作一张大网，而爬虫便是在这张网上爬来爬去的蜘蛛咯，如果它遇到资源，那么它就会抓取下来。想抓取什么？这个由你来控制它咯。

比如它在抓取一个网页，在这个网中他发现了一条道路，其实就是指向网页的超链接，那么它就可以爬到另一张网上来获取数据。这样，整个连在一起的大网对这之蜘蛛来说触手可及，分分钟爬下来不是事儿。

#

浏览网页的过程

在用户浏览网页的过程中，我们可能会看到许多好看的图片，比如 http://image.baidu.com/ ，我们会看到几张的图片以及百度搜索框，这个过程其实就是用户输入网址之后，经过 DNS **服务器**，找到服务器主机，向服务器发出一个请求，服务器经过解析之后，发送给用户的浏览器 HTML、JS、CSS 等文件，浏览器解析出来，用户便可以看到形形色色的图片了。

因此，用户看到的网页实质是由 HTML 代码构成的，爬虫爬来的便是这些内容，通过分析和过滤这些 HTML 代码，实现对图片、文字等资源的获取。

#

URL 的含义

URL，即统一资源定位符，也就是我们说的网址，统一资源定位符是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。

> 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。

爬虫爬取数据时必须要有一个目标的URL才可以获取数据，因此，它是爬虫获取数据的基本依据，准确理解它的含义对爬虫学习有很大帮助。

# #

环境的配置

学习 Python，当然少不了环境的配置，最初我用的是 Notepad++，不过发现它的提示功能实在是太弱了，于是，在 Windows 下我用了 PyCharm，在 Linux 下我用了 Eclipse for Python，另外还有几款比较优秀的 IDE，大家可以参考这篇文章 学习 Python 推荐的 IDE 。好的**开发工具**是前进的**推进器**，希望大家可以找到适合自己的 IDE

3

Urllib 库的基本使用

# #

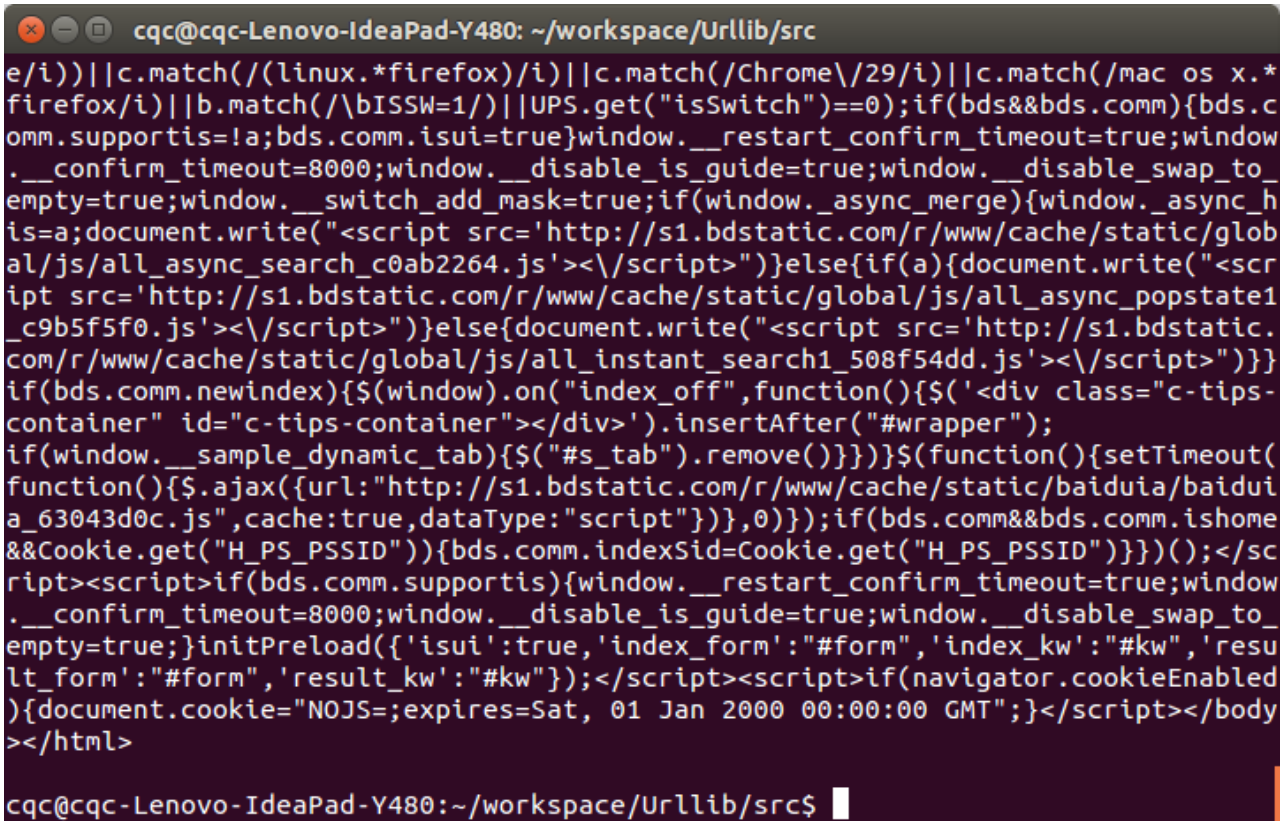
分分钟扒一个网页下来

怎样扒网页呢？其实就是根据URL来获取它的网页信息，虽然我们在浏览器中看到的是一幅幅优美的画面，但是其实是由浏览器解释才呈现出来的，实质它是一段 HTML 代码，加 JS、CSS，如果把网页比作一个人，那么 HTML 便是他的骨架，JS 便是他的肌肉，CSS 便是它的衣服。所以最重要的部分是存在于 HTML 中的，下面我们就写个例子来扒一个网页下来。

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

是的你没看错，真正的程序就两行，把它保存成 demo.py，进入该文件的目录，执行如下命令查看运行结果，感受一下。

```
python demo.py
```



看，这个网页的源码已经被我们扒下来了，是不是很酸爽？

#

分析扒网页的方法

那么我们来分析这两行代码，第一行

```
response = urllib2.urlopen("http://www.baidu.com")
```

首先我们调用的是 urllib2 库里面的 urlopen 方法，传入一个 URL，这个网址是百度首页，协议是 HTTP 协议，当然你也可以把 HTTP 换做 FTP,FILE,HTTPS 等等，只是代表了一种**访问控制**协议，urlopen 一般接受三个参数，它的参数如下：

```
urlopen(url, data, timeout)
```

第一个参数 url 即为 URL，第二个参数 data 是访问 URL 时要传送的数据，第三个 timeout是设置超时时间。

第二三个参数是可以不传送的，data 默认为空 None，timeout 默认为 socket._GLOBAL_DEFAULT_TIMEOUT

第一个参数 URL 是必须要传送的，在这个例子里面我们传送了百度的 URL，执行 urlopen 方法之后，返回一个 response 对象，返回信息便保存在这里面。

```
print response.read()
```

response 对象有一个 read 方法，可以返回获取到的网页内容。

如果不加 read 直接打印会是什么？答案如下：

```
<addinfourl at 139728495260376 whose fp = <socket._fileobject object at 0x7f1513fb3ad0>>
```

直接打印出了该对象的描述，所以记得一定要加 read 方法，否则它不出来内容可就不怪我咯！

#

构造 Requset

其实上面的 urlopen 参数可以传入一个 request 请求,它其实就是一个 Request 类的实例，构造时需要传入 Url,Data 等等的内容。比如上面的两行代码，我们可以这么改写

```
import urllib2
request = urllib2.Request("http://www.baidu.com")
response = urllib2.urlopen(request)
print response.read()
```

运行结果是完全一样的，只不过中间多了一个 request 对象，推荐大家这么写，因为在构建请求时还需要加入好多内容，通过构建一个 request，服务器响应请求得到应答，这样显得逻辑上清晰明确。

#

POST 和 GET 数据传送

上面的程序演示了最基本的网页抓取，不过，现在大多数网站都是动态网页，需要你动态地传递参数给它，它做出对应的响应。所以，在访问时，我们需要传递数据给它。最常见的情况是什么？对了，就是登录注册的时候呀。

把数据用户名和密码传送到一个 URL，然后你得到服务器处理之后的响应，这个该怎么办？下面让我来为小伙伴们揭晓吧！

数据传送分为 POST 和 GET 两种方式，两种方式有什么区别呢?

最重要的区别是 GET 方式是直接以链接形式访问，链接中包含了所有的参数，当然如果包含了密码的话是一种不安全的选择，不过你可以直观地看到自己提交了什么内容。POST 则不会在网址上显示所有的参数，不过如果你想直接查看提交了什么就不太方便了，大家可以酌情选择。

POST 方式：

上面我们说了 data 参数是干嘛的？对了，它就是用在这里的，我们传送的数据就是这个参数data，下面演示一下 POST 方式。

```
import urllib
import urllib2
values = {"username":"1016903103@qq.com","password":"XXXX"}
data = urllib.urlencode(values)
url = "https://passport.csdn.net/account/login?from=http://my.csdn.net/my/mycsdn"
request = urllib2.Request(url,data)
response = urllib2.urlopen(request)
print response.read()
```

我们引入了 urllib 库，现在我们模拟登陆 CSDN，当然上述代码可能登陆不进去，因为还要做一些设置头部 header 的工作，或者还有一些参数没有设置全，还没有提及到在此就不写上去了，在此只是说明登录的原理。我们需要定义一个字典，名字为 values，参数我设置了 username 和 password，下面利用 urllib 的 urlencode 方法将字典编码，命名为 data，构建 request 时传入两个参数，url 和 data，运行程序，即可实现登陆，返回的便是登陆后呈现的页面内容。当然你可以自己搭建一个**服务器**来测试一下。

注意上面字典的定义方式还有一种，下面的写法是等价的

```
import urllib
import urllib2
```

```
values = {}
values['username'] = "1016903103@qq.com"
values['password'] = "XXXX"
data = urllib.urlencode(values)
url = "http://passport.csdn.net/account/login?from=http://my.csdn.net/my/mycsdn"
request = urllib2.Request(url,data)
response = urllib2.urlopen(request)
print response.read()
```

以上方法便实现了 POST 方式的传送

GET 方式：

至于 GET 方式我们可以直接把参数写到网址上面，直接构建一个带参数的 URL 出来即可。

```
import urllib
import urllib2
values = {}
values['username'] = "1016903103@qq.com"
values['password'] = "XXXX"
data = urllib.urlencode(values)
url = "http://passport.csdn.net/account/login?from=http://my.csdn.net/my/mycsdn"
request = urllib2.Request(url,data)
response = urllib2.urlopen(request)
print response.read()
```

你可以 print geturl，打印输出一下 url，发现其实就是原来的 url 加？然后加编码后的参数

```
http://passport.csdn.net/account/login?username=1016903103%40qq.com&password=XXXX
```

和我们平常 GET 访问方式一模一样，这样就实现了数据的 GET 方式传送。

4

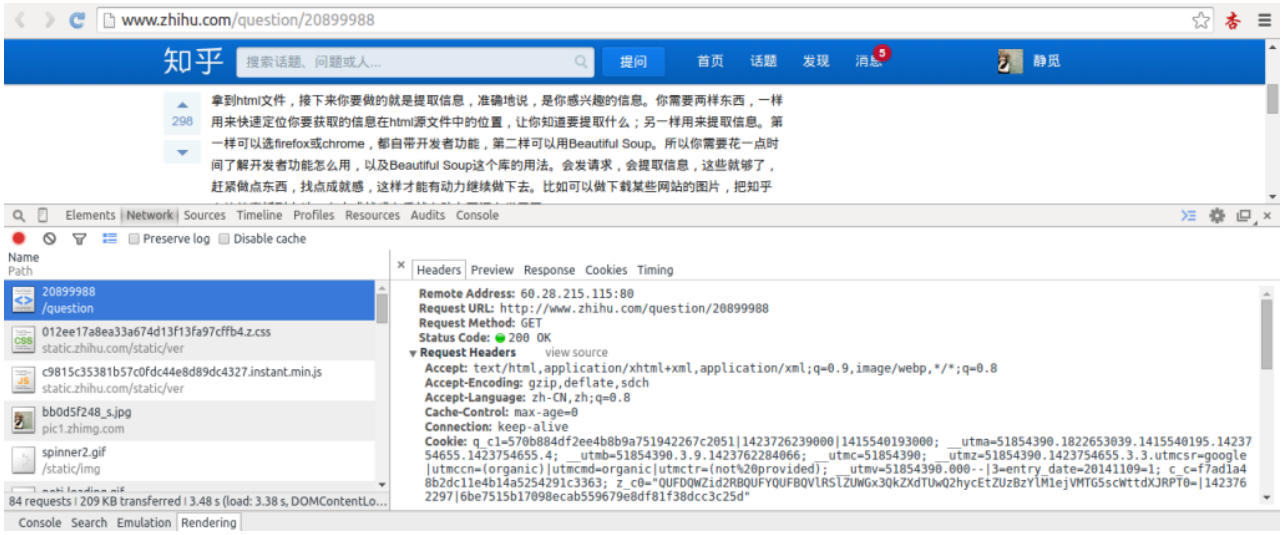
Urllib 库的高级用法

#

设置 Headers

有些网站不会同意程序直接用上面的方式进行访问，如果识别有问题，那么站点根本不会响应，所以为了完全模拟浏览器的工作，我们需要设置一些 Headers 的属性。

首先，打开我们的浏览器，调试浏览器 F12，我用的是 Chrome，打开网络监听，示意如下，比如知乎，点登录之后，我们会发现登陆之后界面都变化了，出现一个新的界面，实质上这个页面包含了许许多多的内容，这些内容也不是一次性就加载完成的，实质上是执行了好多次请求，一般是首先请求HTML文件，然后加载 JS，CSS 等等，经过多次请求之后，网页的骨架和肌肉全了，整个网页的效果也就出来了。



拆分这些请求，我们只看一第一个请求，你可以看到，有个 Request URL，还有 headers，下面便是 response，图片显示得不全，小伙伴们可以亲身实验一下。那么这个头中包含了许许多多是信息，有文件编码啦，压缩方式啦，请求的 agent 啦等等。

其中，agent 就是请求的身份，如果没有写入请求身份，那么服务器不一定会响应，所以可以在 headers 中设置 agent,例如下面的例子，这个例子只是说明了怎样设置的 headers，小伙伴们看一下设置格式就好。

```
import urllib
import urllib2
url = 'http://www.server.com/login'
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
values = {'username' : 'cqc',  'password' : 'XXXX' }
headers = { 'User-Agent' : user_agent }
data = urllib.urlencode(values)
request = urllib2.Request(url, data, headers)
```

```
response = urllib2.urlopen(request)
page = response.read()
```

这样，我们设置了一个 headers，在构建 request 时传入，在请求时，就加入了 headers 传送，服务器若识别了是浏览器发来的请求，就会得到响应。

另外，我们还有**对付”反盗链”**的方式，对付防盗链，服务器会识别 headers 中的 referer 是不是它自己，如果不是，有的服务器不会响应，所以我们还可以在 headers 中加入 referer

例如我们可以构建下面的headers

```
headers = { 'User-Agent' : 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)' ,
                    'Referer':'http://www.zhihu.com/articles' }
```

同上面的方法，在传送请求时把 headers 传入 Request 参数里，这样就能应付防盗链了。

另外 headers 的一些属性，下面的需要特别注意一下：

- 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。
- 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

其他的有必要的可以审查浏览器的 headers 内容，在构建时写入同样的数据即可。

#

Proxy（代理）的设置

urllib2 默认会使用**环境变量** http_proxy 来设置 HTTP Proxy。假如一个网站它会**检测**某一段时间某个IP 的访问次数，如果访问次数过多，它会禁止你的访问。所以你可以设置一些**代理服务器**来帮助你做工作，每隔一段时间换一个代理，网站君都不知道是谁在捣鬼了，这酸爽！

下面一段代码说明了**代理**的设置用法

```
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)
```

#

Timeout 设置

上一节已经说过 urlopen 方法了，第三个参数就是 timeout 的设置，可以设置等待多久超时，为了解决一些网站实在响应过慢而造成的影响。

例如下面的代码,如果第二个参数 data 为空那么要特别指定是 timeout 是多少，写明形参，如果data已经传入，则不必**声明**。

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

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

#

使用 HTTP 的 PUT 和 DELETE 方法

http 协议有六种请求方法，get,head,put,delete,post,options，我们有时候需要用到 PUT 方式或者 DELETE 方式请求。

> PUT：这个方法比较少见。HTML 表单也不支持这个。本质上来讲， PUT 和 POST 极为相似，都是向服务器发送数据，但它们之间有一个重要区别，PUT 通常指定了资源的存放位置，而 POST 则没有，POST 的数据存放位置由服务器自己决定。 DELETE：删除某一个资源。基本上这个也很少见，不过还是有一些地方比如amazon的S3云服务里面就用的这个方法来删除资源。 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，不过用的次数的确是少，在这里提一下。

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

#

使用 DebugLog

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，这个也不太常用，仅提一下

```
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.baidu.com')
```

以上便是一部分高级特性，前三个是重要内容，在后面，还有 cookies 的设置还有异常的处理，小伙伴们加油！

5

URLError 异常处理

大家好，本节在这里主要说的是 URLError 还有 HTTPError，以及对它们的一些处理。

第 5 章 3 URLError

首先解释下 URLError 可能产生的原因：

- 网络无连接，即本机无法上网
- 连接不到特定的服务器
- 服务器不存在

在代码中，我们需要用 try-except 语句来包围并捕获相应的异常。下面是一个例子，先感受下它的风骚

```
import urllib2
requset = urllib2.Request('http://www.xxxxx.com')
try:
    urllib2.urlopen(requset)
except urllib2.URLError, e:
    print e.reason
```

我们利用了 urlopen 方法访问了一个不存在的网址，运行结果如下：

```
[Errno 11004] getaddrinfo failed
```

它说明了错误代号是11004，错误原因是 getaddrinfo failed

#

HTTPError

HTTPError 是 URLError 的子类，在你利用 urlopen 方法发出一个请求时，服务器上都会对应一个应答对象 response，其中它包含一个数字”状态码”。举个例子，假如 response 是一个”重定向”，需定位到别的地址获取文档，urllib2 将对此进行处理。

其他不能处理的，urlopen 会产生一个 HTTPError，对应相应的状态吗，HTTP 状态码表示HTTP 协议所返回的响应的状态。下面将状态码归结如下：

```
100：继续  客户端应当继续发送请求。客户端应当继续发送请求的剩余部分，或者如果请求已经完成，忽略这个响应。
101： 转换协议  在发送完这个响应最后的空行后，服务器将会切换到在 Upgrade 消息头中定义的那些协议。只有在切换新的协议更有
102：继续处理   由 WebDAV（RFC 2518）扩展的状态码，代表处理将被继续执行。
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
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
500：服务器内部错误  服务器遇到了一个未曾预料的状况，导致了它无法完成对请求的处理。一般来说，这个问题都会在**服务器端**
501：服务器无法识别  服务器不支持当前请求所需要的某个功能。当服务器无法识别请求的方法，并且无法支持其对任何资源的请求。
502：错误网关  作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应。
503：服务出错   由于临时的**服务器**维护或者过载，服务器当前无法处理请求。这个状况是临时的，并且将在一段时间以后恢复。
```

HTTPError 实例产生后会有一个 code 属性，这就是是服务器发送的相关错误号。 因为 urllib2 可以为你处理重定向，也就是 3 开头的代号可以被处理，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。

下面我们写一个例子来感受一下，捕获的异常是 HTTPError，它会带有一个 code 属性，就是错误代号，另外我们又打印了 reason 属性，这是它的父类 URLError 的属性。

```
import urllib2
req = urllib2.Request('http://blog.csdn.net/cqcre')
try:
   urllib2.urlopen(req)
```

```
except urllib2.HTTPError, e:
    print e.code
    print e.reason
```

运行结果如下

```
403
Forbidden
```

错误代号是 403，错误原因是 Forbidden，说明服务器禁止访问。

我们知道，HTTPError 的父类是 URLError，根据编程经验，父类的异常应当写到子类异常的后面，如果子类捕获不到，那么可以捕获父类的异常，所以上述的代码可以这么改写

```
import urllib2
req = urllib2.Request('http://blog.csdn.net/cqcre')
try:
    urllib2.urlopen(req)
except urllib2.HTTPError, e:
    print e.code
except urllib2.URLError, e:
    print e.reason
else:
    print "OK"
```

如果捕获到了 HTTPError，则输出 code，不会再处理 URLError 异常。如果发生的不是HTTPError，则会去捕获 URLError 异常，输出错误原因。

另外还可以加入 hasattr 属性提前对属性进行判断，代码改写如下

```
import urllib2
req = urllib2.Request('http://blog.csdn.net/cqcre')
try:
    urllib2.urlopen(req)
except urllib2.URLError, e:
    if hasattr(e,"code"):
        print e.code
    if hasattr(e,"reason"):
        print e.reason
else:
    print "OK"
```

首先对异常的属性进行判断，以免出现属性输出报错的现象。

以上，就是对 URLError 和 HTTPError 的相关介绍，以及相应的错误处理办法，小伙伴们加油!

6

Cookie 的使用

大家好哈，上一节我们研究了一下爬虫的异常处理问题，那么接下来我们一起来看一下 Cookie 的使用。

为什么要使用 Cookie 呢?

Cookie，指某些网站为了辨别用户身份、进行 session 跟踪而储存在用户本地终端上的数据（通常经过加密）

比如说有些网站需要登录后才能访问某个页面，在登录之前，你想抓取某个页面内容是不允许的。那么我们可以利用 Urllib2 库保存我们登录的 Cookie，然后再抓取其他页面就达到目的了。

在此之前呢，我们必须先介绍一个 opener 的概念。

#

Opener

当你获取一个 URL 你使用一个 opener(一个 urllib2.OpenerDirector 的实例)。在前面，我们都是使用的默认的 opener，也就是 urlopen。它是一个特殊的 opener，可以理解成opener 的一个特殊实例，传入的参数仅仅是 url，data，timeout。

如果我们需要用到 Cookie，只用这个 opener 是不能达到目的的，所以我们需要创建更一般的opener 来实现对 Cookie 的设置。

#

Cookielib

cookielib 模块的主要作用是提供可存储 cookie 的对象，以便于与 urllib2 模块配合使用来访问 Internet 资源。Cookielib 模块非常强大，我们可以利用本模块的 CookieJar 类的对象来捕获 cookie 并在后续连接请求时重新发送，比如可以实现模拟登录功能。该模块主要的对象有 CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

它们的关系：CookieJar ——派生——>FileCookieJar ——派生——>MozillaCookieJar 和LWPCookieJar

#

获取 Cookie 保存到变量

首先，我们先利用 CookieJar 对象实现获取 cookie 的功能，存储到变量中，先来感受一下

```
import urllib2
import cookielib
\#声明一个CookieJar对象实例来保存cookie
cookie = cookielib.CookieJar()
\#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器
handler=urllib2.HTTPCookieProcessor(cookie)
\#通过handler来构建opener
opener = urllib2.build_opener(handler)
\#此处的open方法同urllib2的urlopen方法，也可以传入request
response = opener.open('http://www.baidu.com')
for item in cookie:
    print 'Name = '+item.name
    print 'Value = '+item.value
```

我们使用以上方法将 cookie 保存到变量中，然后打印出了 cookie 中的值，运行结果如下

```
Name = BAIDUID
Value = B07B663B645729F11F659C02AAE65B4C:FG=1
Name = BAIDUPSID
Value = B07B663B645729F11F659C02AAE65B4C
Name = H\_PS\_PSSID
Value = 12527\_11076\_1438\_10633
Name = BDSVRTM
Value = 0
```

```
Name = BD_HOME
Value = 0
```

#

保存 Cookie 到文件

在上面的方法中，我们将 cookie 保存到了 cookie 这个变量中，如果我们想将 cookie 保存到文件中该怎么做呢？这时，我们就要用到

FileCookieJar 这个对象了，在这里我们使用它的子类 MozillaCookieJar 来实现 Cookie的保存

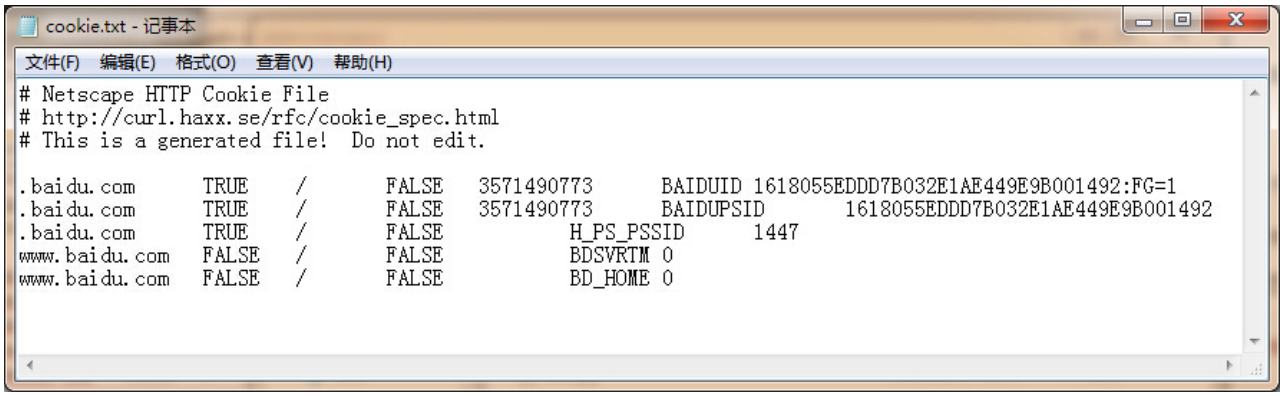
```
import cookielib
import urllib2

\#设置保存cookie的文件，同级目录下的cookie.txt
filename = 'cookie.txt'
\#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件
cookie = cookielib.MozillaCookieJar(filename)
\#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器
handler = urllib2.HTTPCookieProcessor(cookie)
\#通过handler来构建opener
opener = urllib2.build_opener(handler)
\#创建一个请求，原理同urllib2的urlopen
response = opener.open("http://www.baidu.com")
\#保存cookie到文件
cookie.save(ignore_discard=True, ignore_expires=True)
```

关于最后 save 方法的两个参数在此说明一下：

官方解释如下：

> ignore_discard: save even cookies set to be discarded. ignore_expires: save even cookies that have expiredThe file is overwritten if it already exists

由此可见，ignore_discard 的意思是即使 cookies 将被丢弃也将它保存下来，ignore_expires 的意思是如果在该文件中 cookies 已经存在，则覆盖原文件写入，在这里，我们将这两个全部设置为 True。运行之后，cookies 将被保存到 cookie.txt文件中，我们查看一下内容，附图如下



#

从文件中获取 Cookie 并访问

那么我们已经做到把 Cookie 保存到文件中了，如果以后想使用，可以利用下面的方法来读取cookie 并访问网站，感受一下

```
import cookielib
import urllib2

\#创建MozillaCookieJar实例对象
cookie = cookielib.MozillaCookieJar()
\#从文件中读取cookie内容到变量
cookie.load('cookie.txt', ignore_discard=True, ignore_expires=True)
\#创建请求的request
req = urllib2.Request("http://www.baidu.com")
\#利用urllib2的build_opener方法创建一个opener
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
response = opener.open(req)
print response.read()
```

设想，如果我们的 cookie.txt 文件中保存的是某个人登录百度的 cookie，那么我们提取出这个 cookie 文件内容，就可以用以上方法模拟这个人的账号登录百度。

#

利用 cookie 模拟网站登录

下面我们以我们学校的教育系统为例，利用 cookie 实现模拟登录，并将 cookie 信息保存到文本文件中，来感受一下 cookie 大法吧！

```
import urllib
import urllib2
import cookielib

filename = 'cookie.txt'
\#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件
cookie = cookielib.MozillaCookieJar(filename)
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
postdata = urllib.urlencode({
        'stuid':'201200131012',
        'pwd':'23342321'
    })
\#登录教务系统的URL
loginUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks\_login2.login'
\#模拟登录，并把cookie保存到变量
result = opener.open(loginUrl,postdata)
\#保存cookie到cookie.txt中
cookie.save(ignore\_discard=True, ignore\_expires=True)
\#利用cookie请求访问另一个网址，此网址是成绩查询网址
gradeUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bkscjcx.curscopre'
\#请求访问成绩查询网址
result = opener.open(gradeUrl)
print result.read()
```

以上程序的原理如下

创建一个带有 cookie 的 opener，在访问登录的 URL 时，将登录后的 cookie 保存下来，然后利用这个 cookie 来访问其他网址。

如登录之后才能查看的**成绩**查询呀，本学期课表呀等等网址，模拟登录就这么实现啦，是不是很酷炫?

好，小伙伴们要加油哦！我们现在可以顺利获取网站信息了，接下来就是把网站里面有效内容提取出来，下一节我们去会会正则表达式!

7

正则表达式

在前面我们已经搞定了怎样获取页面的内容，不过还差一步，这么多杂乱的代码夹杂文字我们怎样把它提取出来整理呢？下面就开始介绍一个十分强大的工具，正则表达式！

# #

了解正则表达式

正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。 正则表达式是用来匹配字符串非常强大的工具，在其他编程语言中同样有正则表达式的概念，Python同样不例外，利用了正则表达式，我们想要从返回的页面内容提取出我们想要的内容就易如反掌了。

> 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。

# #

正则表达式的语法规则

下面是 Python 中正则表达式的一些匹配规则，图片资料来自 CSDN

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\dc	a1c
\D	非数字：[^\d]	a\Dc	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\sc	a c
\S	非空白字符：[^\s]	a\Sc	abc
\w	单词字符：[A-Za-z0-9_]	a\wc	abc
\W	非单词字符：[^\w]	a\Wc	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	abcabc a456c

# #

正则表达式相关注解

#

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。Python 里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；非贪婪的则相反，总是尝试匹配尽可能少的字符。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”。

注：我们一般使用非贪婪模式来提取。

#

反斜杠问题

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”\\”：前两个和后两个分别用于在编程语言里转义成反斜杠，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。

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，妈妈也不用担心是不是漏写了反斜杠，写出来的表达式也更直观勒。

# #

Python Re 模块

Python 自带了 re 模块，它提供了对正则表达式的支持。主要用到的方法列举如下

```
 #返回pattern对象

re.compile(string[,flag])
 #以下为匹配所用函数

re.match(pattern, string[, flags])
re.search(pattern, string[, flags])
re.split(pattern, string[, maxsplit])
re.findall(pattern, string[, flags])
re.finditer(pattern, string[, flags])
re.sub(pattern, repl, string[, count])
re.subn(pattern, repl, string[, count])
```

在介绍这几个方法之前，我们先来介绍一下 pattern 的概念，pattern 可以理解为一个匹配模式，那么我们怎么获得这个匹配模式呢？很简单，我们需要利用 re.compile 方法就可以。例如

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

在参数中我们传入了原生字符串对象，通过 compile 方法编译生成一个 pattern 对象，然后我们利用这个对象来进行进一步的匹配。

另外大家可能注意到了另一个参数 flags，在这里解释一下这个参数的含义：

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。

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。

在刚才所说的另外几个方法例如 re.match 里我们就需要用到这个 pattern 了，下面我们一一介绍。

> 注：以下七个方法中的 flags 同样是代表匹配模式的意思，如果在 pattern 生成时已经指明了 flags，那么在下面的方法中就不需要传入这个参数了。

#

re.match(pattern, string[, flags])

这个方法将会从 string（我们要匹配的字符串）的开头开始，尝试匹配 pattern，一直向后匹配，如果遇到无法匹配的字符，立即返回 None，如果匹配未结束已经到达 string 的末尾，也会返回 None。两个结果均表示匹配失败，否则匹配 pattern 成功，同时匹配终止，不再对string 向后匹配。下面我们通过一个例子理解一下

```
__author__ = 'CQC'
\# -*- coding: utf-8 -*-

\#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re.match匹配文本，获得匹配结果，无法匹配时将返回None
result1 = re.match(pattern,'hello')
result2 = re.match(pattern,'helloo CQC!')
result3 = re.match(pattern,'helo CQC!')
result4 = re.match(pattern,'hello CQC!')

\#如果1匹配成功
if result1:
    # 使用Match获得分组信息
    print result1.group()
else:
    print '1匹配失败！'


\#如果2匹配成功
if result2:
    # 使用Match获得分组信息
    print result2.group()
else:
    print '2匹配失败！'


\#如果3匹配成功
```

```
if result3:
    # 使用Match获得分组信息
    print result3.group()
else:
    print '3匹配失败！'

\#如果4匹配成功
if result4:
    # 使用Match获得分组信息
    print result4.group()
else:
    print '4匹配失败！'
```

运行结果

```
hello
hello
3匹配失败!
hello
```

匹配分析

1. 第一个匹配，pattern 正则表达式为 ’hello’，我们匹配的目标字符串 string 也为hello，从头至尾完全匹配，匹配成功。
2. 第二个匹配，string 为 helloo CQC，从 string 头开始匹配 pattern 完全可以匹配，pattern 匹配结束，同时匹配终止，后面的 o CQC 不再匹配，返回匹配成功的信息。
3. 第三个匹配，string为helo CQC，从 string 头开始匹配 pattern，发现到 ‘o’ 时无法完成匹配，匹配终止，返回 None
4. 第四个匹配，同第二个匹配原理，即使遇到了空格符也不会受影响。

我们还看到最后打印出了 result.group()，这个是什么意思呢？下面我们说一下关于 match 对象的的属性和方法 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
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、\g引用分组，但不能使用编号0。\id与\g是等价的；但\10将被

下面我们用一个例子来体会一下

```
\# −*− 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 \g\g'):", m.expand(r'\2 \1\3')

\### output ###
\# m.string: hello world!
\# m.re:
\# 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!
```

#

re.search(pattern, string[, flags])

search 方法与 match 方法极其类似，区别在于 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。同样，search 方法的返回对象同样 match() 返回对象的方法和属性。我们用一个例子感受一下

```
\#导入re模块
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 = re.search(pattern,'hello world!')
if match:
    \# 使用Match获得分组信息
    print match.group()
\### 输出 ###
\# world
```

#

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。我们通过下面的例子感受一下。

```
import re

pattern = re.compile(r'\d+')
print re.split(pattern,'one1two2three3four4')
```

```
\### 输出 ###
\# ['one', 'two', 'three', 'four', '']
```

#

re.findall(pattern, string[, flags])

搜索 string，以列表形式返回全部能匹配的子串。我们通过这个例子来感受一下

```
import re

pattern = re.compile(r'\d+')
print re.findall(pattern,'one1two2three3four4')

\### 输出 ###
\# ['1', '2', '3', '4']
```

#

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。我们通过下面的例子来感受一下

```
import re

pattern = re.compile(r'\d+')
for m in re.finditer(pattern,'one1two2three3four4'):
    print m.group(),

\### 输出 ###
\# 1 2 3 4
```

#

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、\g 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。

```
import re

pattern = re.compile(r'(\w+) (\w+)')
s = 'i say, hello world!'

print re.sub(pattern,r'\2 \1', s)

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print re.sub(pattern,func, s)

\### output ###
\# say i, world hello!
\# I Say, Hello World!
```

#

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

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

```
import re

pattern = re.compile(r'(\w+) (\w+)')
s = 'i say, hello world!'

print re.subn(pattern,r'\2 \1', s)

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print re.subn(pattern,func, s)

\### output ###
\# ('say i, world hello!', 2)
\# ('I Say, Hello World!', 2)
```

#

Python Re 模块的另一种使用方式

在上面我们介绍了7个工具方法，例如 match，search 等等，不过调用方式都是 re.match，re.search 的方式，其实还有另外一种调用方式，可以通过 pattern.match，pattern.search 调用，这样调用便不用将 pattern 作为第一个参数传入了，大家想怎样调用皆可。

函数 API 列表

```
match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])
search(string[, pos[, endpos]]) | re.search(pattern, string[, flags])
split(string[, maxsplit]) | re.split(pattern, string[, maxsplit])
findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags])
finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags])
sub(repl, string[, count]) | re.sub(pattern, repl, string[, count])
subn(repl, string[, count]) |re.sub(pattern, repl, string[, count])
```

具体的调用方法不必详说了，原理都类似，只是参数的变化不同。小伙伴们尝试一下吧~

小伙伴们加油，即使这一节看得云里雾里的也没关系，接下来我们会通过一些实战例子来帮助大家熟练掌握正则表达式的。

参考文章：此文章部分内容出自 CNBlogs

8

Beautiful Soup 的用法

上一节我们介绍了正则表达式，它的内容其实还是蛮多的，如果一个正则匹配稍有差池，那可能程序就处在永久的循环之中，而且有的小伙伴们也对写正则表达式的写法用得不熟练，没关系，我们还有一个更强大的工具，叫 Beautiful Soup，有了它我们可以很方便地提取出 HTML 或 XML标签中的内容，实在是方便，这一节就让我们一起来感受一下 Beautiful Soup 的强大吧。

# #

Beautiful Soup 的简介

简单来说，Beautiful Soup 是 python 的一个库，最主要的功能是从网页抓取数据。官方解释如下：

> Beautiful Soup 提供一些简单的、python 式的函数用来处理导航、搜索、修改分析树等功能。它是一个工具箱，通过解析文档为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序。
> Beautiful Soup 自动将输入文档转换为 Unicode 编码，输出文档转换为 utf-8 编码。你不需要考虑编码方式，除非文档没有指定一个编码方式，这时，Beautiful Soup 就不能自动识别编码方式了。然后，你仅仅需要说明一下原始编码方式就可以了。
> Beautiful Soup 已成为和 lxml、html6lib 一样出色的 python 解释器，为用户灵活地提供不同的解析策略或强劲的速度。

废话不多说，我们来试一下吧~

#

Beautiful Soup 安装

Beautiful Soup 3 目前已经停止开发，推荐在现在的项目中使用 Beautiful Soup 4，不过它已经被移植到 BS4 了，也就是说导入时我们需要 import bs4 。所以这里我们用的版本是 Beautiful Soup 4.3.2 (简称BS4)，另外据说 BS4 对 Python3 的支持不够好，不过我用的是 Python2.7.7，如果有小伙伴用的是 Python3 版本，可以考虑下载 BS3 版本。

如果你用的是新版的 Debain 或 Ubuntu,那么可以通过系统的软件包管理来安装，不过它不是最新版本，目前是4.2.1版本

```
sudo apt-get install Python-bs4
```

如果想安装最新的版本，请直接下载安装包来手动安装，也是十分方便的方法。在这里我安装的是 Beautiful Soup 4.3.2

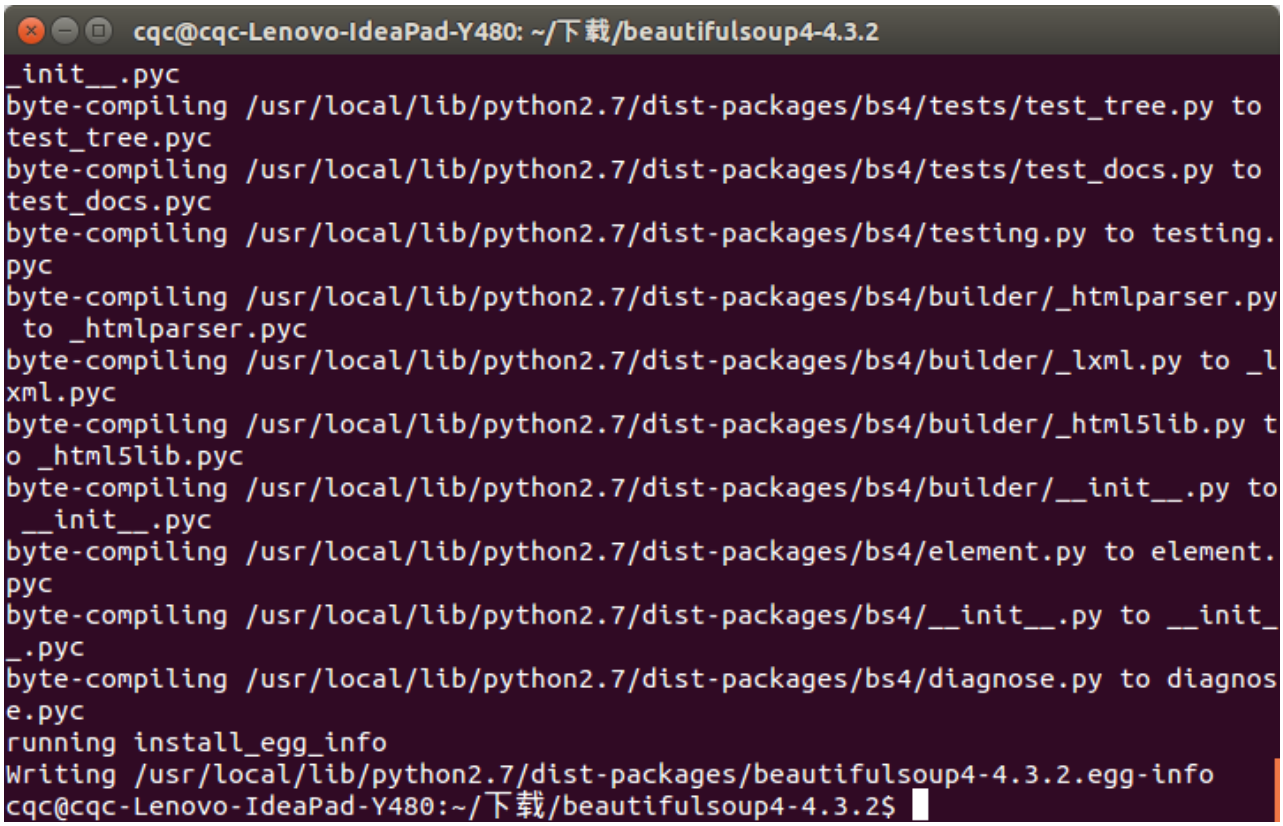
Beautiful Soup 3.2.1
Beautiful Soup 4.3.2

下载完成之后解压

运行下面的命令即可完成安装

```
sudo python setup.py install
```

如下图所示，证明安装成功了



然后需要安装 lxml

```
sudo apt-get install Python-lxml
```

Beautiful Soup 支持 Python 标准库中的 HTML 解析器,还支持一些第三方的解析器，如果我们不安装它，则 Python 会使用 Python 默认的解析器，lxml 解析器更加强大，速度更快，推荐安装。

#

开启 Beautiful Soup 之旅

在这里先分享官方文档链接，不过内容是有些多，也不够条理，在此本文章做一下整理方便大家参考。

官方文档

# #

创建 Beautiful Soup 对象

首先必须要导入 bs4 库

```
from bs4 import BeautifulSoup
```

我们创建一个字符串，后面的例子我们便会用它来演示

```
html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""
```

创建 beautifulsoup 对象

```
soup = BeautifulSoup(html)
```

另外，我们还可以用本地 HTML 文件来创建对象，例如

```
soup = BeautifulSoup(open('index.html'))
```

上面这句代码便是将本地 index.html 文件打开，用它来创建 soup 对象

下面我们来打印一下 soup 对象的内容，格式化输出

```
print soup.prettify()
```

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
```

以上便是输出结果，格式化打印出了它的内容，这个**函数**经常用到，小伙伴们要记好咯。

# #

四大对象种类

Beautiful Soup 将复杂HTML文档转换成一个复杂的树形结构,每个节点都是 Python 对象,所有对象可以归纳为4种:

- Tag
- NavigableString
- BeautifulSoup
- Comment

下面我们进行一一介绍

#

Tag

Tag 是什么？通俗点讲就是 HTML 中的一个个标签，例如

```
<title>The Dormouse's story</title>
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

上面的 title a 等等 HTML 标签加上里面包括的内容就是 Tag，下面我们来感受一下怎样用 Beautiful Soup 来方便地获取 Tags

下面每一段代码中注释部分即为运行结果

```
print soup.title
\#<title>The Dormouse's story</title>
```

```` print soup.head #

```

print soup.a #

```
```

print soup.p #

The Dormouse's story

我们可以利用 soup 加标签名轻松地获取这些标签的内容，是不是感觉比正则表达式方便多了？不过有一点是，它查找的是在所有内容中

我们可以验证一下这些对象的类型

print type(soup.a) #

对于 Tag，它有两个重要的属性，是 name 和 attrs，下面我们分别来感受一下

**name**

print soup.name print soup.head.name #[document] #head

soup 对象本身比较特殊，它的 name 即为 [document]，对于其他内部标签，输出的值便为标签本身的名称。

**attrs**

print soup.p.attrs #{'class': ['title'], 'name': 'dromouse'}

在这里，我们把 p 标签的所有属性打印输出了出来，得到的类型是一个字典。

如果我们想要单独获取某个属性，可以这样，例如我们获取它的 class 叫什么

print soup.p['class'] #['title']

还可以这样，利用 get 方法，传入属性的名称，二者是等价的

print soup.p.get('class') #['title']

我们可以对这些属性和内容等等进行修改，例如

soup.p['class']="newClass" print soup.p #

The Dormouse's story

还可以对这个属性进行删除，例如

del soup.p['class'] print soup.p #

The Dormouse's story

不过，对于修改删除的操作，不是我们的主要用途，在此不做详细介绍了，如果有需要，请查看前面提供的官方文档

####
NavigableString

既然我们已经得到了标签的内容，那么问题来了，我们要想获取标签内部的文字怎么办呢？很简单，用 .string 即可，例如

print soup.p.string #The Dormouse's story

这样我们就轻松获取到了标签里面的内容，想想如果用正则表达式要多麻烦。它的类型是一个 NavigableString，翻译过来叫 可以遍历

来检查一下它的类型

print type(soup.p.string) #

####
BeautifulSoup

BeautifulSoup 对象表示的是一个文档的全部内容.大部分时候,可以把它当作 Tag 对象，是一个特殊的 Tag，我们可以分别获取它的类

print type(soup.name) # print soup.name # [document] print soup.attrs #{} 空字典

####
Comment

Comment 对象是一个特殊类型的 NavigableString 对象，其实输出的内容仍然不包括注释符号，但是如果不好好处理它，可能会对我

我们找一个带注释的标签

print soup.a print soup.a.string print type(soup.a.string)

运行结果如下

Elsie

a 标签里的内容实际上是注释，但是如果我们利用 .string 来输出它的内容，我们发现它已经把注释符号去掉了，所以这可能会给我们带

另外我们打印输出下它的类型，发现它是一个 Comment 类型，所以，我们在使用前最好做一下判断，判断代码如下

if type(soup.a.string)==bs4.element.Comment: print soup.a.string

上面的代码中，我们首先判断了它的类型，是否为 Comment 类型，然后再进行其他操作，如打印输出。

###
遍历文档树

####
直接子节点

>要点：.contents .children 属性

**.contents**

**tag** 的 .content 属性可以将tag的子节点以列表的方式输出

print soup.head.contents #[]

输出方式为列表，我们可以用列表索引来获取它的某一个元素

print soup.head.contents[0] #

**.children **

它返回的不是一个 list，不过我们可以通过**遍历**获取所有子节点。

我们打印输出 .children 看一下，可以发现它是一个 list 生成器对象

print soup.head.children #

我们怎样获得里面的内容呢？很简单，遍历一下就好了，代码及结果如下

for child in soup.body.children: print child

The Dormouse's story

<

p class="story">Once upon a time there were three little sisters; and their names were , Lacie and

```
####
 所有子孙节点

>知识点：.descendants 属性

**.descendants**

.contents 和 .children 属性仅包含tag的直接子节点，.descendants 属性可以对所有tag的子孙节点进行递归循环，和 children类似，
```

for child in soup.descendants: print child

```
运行结果如下，可以发现，所有的节点都被打印出来了，先生最外层的 HTML 标签，其次从 head 标签一个个剥离，以此类推。
```

The Dormouse's story

<

p class="story">Once upon a time there were three little sisters; and their names were ,

```
####
 节点内容

>知识点：.string 属性

如果 tag 只有一个 NavigableString 类型子节点,那么这个 tag 可以使用 .string 得到子节点。如果一个 tag 仅有一个子节点,那么这个

通俗点说就是：如果一个标签里面没有标签了，那么 .string 就会返回标签里面的内容。如果标签里面只有唯一的一个标签了，那么 .str
```

print soup.head.string #The Dormouse's story print soup.title.string #The Dormouse's story

```
如果 tag 包含了多个子节点,tag 就无法确定，string 方法应该调用哪个子节点的内容, .string 的输出结果是 None
```

print soup.html.string # None

```
####
 多个内容

>知识点： .strings  .stripped\_strings 属性

**.strings**

获取多个内容，不过需要遍历获取，比如下面的例子
```

for string in soup.strings: print(repr(string)) # u"The Dormouse's story" # u'\n\n' # u"The Dormouse's story" # u'\n\n' # u'Once upon a time there were three little sisters; and their names were\n' # u'Elsie' # u',\n' # u'Lacie' # u' and\n' # u'Tillie' # u';\nand they lived at the bottom of a well.' # u'\n\n' # u'...' # u'\n' .stripped_strings

输出的字符串中可能包含了很多空格或空行,使用 .stripped_strings 可以去除多余空白内容

for string in soup.stripped_strings: print(repr(string)) # u"The Dormouse's story" # u"The Dormouse's story" # u'Once upon a time there were three little sisters; and their names were' # u'Elsie' # u',' # u'Lacie' # u'and' # u'Tillie' # u';\nand they lived at the bottom of a well.' # u'...'

####
父节点

>知识点： .parent 属性

p = soup.p print p.parent.name #body

content = soup.head.title.string print content.parent.name #title

####
全部父节点

>知识点：.parents 属性
>
通过元素的 .parents 属性可以递归得到元素的所有父辈节点，例如

content = soup.head.title.string for parent in content.parents: print parent.name

title head html [document]

####
兄弟节点

>知识点：.next_sibling .previous_sibling 属性

兄弟节点可以理解为和本节点处在统一级的节点，.next_sibling 属性获取了该节点的下一个兄弟节点，.previous_sibling 则与之相反

注意：实际文档中的tag的 .next_sibling 和 .previous_sibling 属性通常是字符串或空白，因为空白或者换行也可以被视作一个节点，

print soup.p.next_sibling # 实际该处为空白 print soup.p.prev_sibling #None 没有前一个兄弟节点，返回 None print soup.p.next_sibling.next_sibling #

Once upon a time there were three little sisters; and their names were \#, \#Lacie and \#Tillie; \#and they lived at the bottom of a well.

#下一个节点的下一个兄弟节点是我们可以看到的节点

####
全部兄弟节点

>知识点：.next_siblings  .previous_siblings 属性

通过 .next_siblings 和 .previous_siblings 属性可以对当前节点的兄弟节点迭代输出

for sibling in soup.a.next_siblings: print(repr(sibling)) # u',\n' # Lacie # u' and\n' # Tillie # u'; and they lived at the bottom of a well.' # None

####
前后节点

>知识点：.next_element  .previous_element 属性

与 .next_sibling  .previous_sibling 不同，它并不是针对于兄弟节点，而是在所有节点，不分层次

比如 head 节点为

那么它的下一个节点便是 title，它是不分层次关系的

print soup.head.next_element #

####
所有前后节点

>知识点：.next_elements  .previous_elements 属性

通过 .next_elements 和 .previous_elements 的迭代器就可以向前或向后访问文档的解析内容,就好像文档正在被解析一样

for element in last_a_tag.next_elements: print(repr(element)) # u'Tillie' # u';\nand they lived at the bott om of a well.' # u'\n\n' #

...

# u'...' # u'\n' # None

```
###
 搜索文档树

####
 find\_all( name , attrs , recursive , text , \*\*kwargs )
find\_all() 方法搜索当前tag的所有tag子节点,并判断是否符合过滤器的条件

#####
 name 参数

name 参数可以查找所有名字为 name 的 tag,字符串对象会被自动忽略掉

######
 传字符串

最简单的过滤器是字符串.在搜索方法中传入一个字符串参数,Beautiful Soup 会查找与字符串完整匹配的内容,下面的例子用于查找文档
```

soup.find_all('b') # [The Dormouse's story]

```

```

print soup.find_all('a') #[, Lacie, Tillie]

```
######
 传正则表达式

如果传入正则表达式作为参数,Beautiful Soup 会通过正则表达式的 match() 来匹配内容.下面例子中找出所有以b开头的标签,这表示 <
```

import re for tag in soup.find_all(re.compile("^b")): print(tag.name) # body # b

```
######
 传列表

如果传入列表参数,Beautiful Soup 会将与列表中任一元素匹配的内容返回.下面代码找到文档中所有 <a\> 标签和 <b\> 标签
```

soup.find_all(["a", "b"]) # [The Dormouse's story, # Elsie, # Lacie, # Tillie]

```
######
 传 True

True 可以匹配任何值,下面代码查找到所有的 tag,但是不会返回字符串节点
```

for tag in soup.find_all(True): print(tag.name) # html # head # title # body # p # b # p # a # a

```
######
 传方法

如果没有合适过滤器,那么还可以定义一个方法,方法只接受一个元素参数 [4] ,如果这个方法返回 True 表示当前元素匹配并且被找到,如果

下面方法校验了当前元素,如果包含 class 属性却不包含 id 属性,那么将返回 True:
```

def has_class_but_no_id(tag): return tag.has_attr('class') and not tag.has_attr('id')

```
将这个方法作为参数传入 find\_all() 方法,将得到所有 <p> 标签:
```

soup.find_all(has_class_but_no_id) # [

The Dormouse's story

, #

Once upon a time there were...

, #

...

]

```
#####
 keyword 参数

注意：如果一个指定名字的参数不是搜索内置的参数名,搜索时会把该参数当作指定名字 tag 的属性来搜索,如果包含一个名字为 id 的参
```

soup.find_all(id='link2') # [Lacie]

```
如果传入 href 参数,Beautiful Soup 会搜索每个 tag 的 ” href” 属性
```

soup.find_all(href=re.compile("elsie")) # [Elsie]

使用多个指定名字的参数可以同时过滤 tag 的多个属性

soup.find_all(href=re.compile("elsie"), id='link1') # [three]

在这里我们想用 class 过滤，不过 class 是 python 的关键词，这怎么办？加个下划线就可以

soup.find_all("a", class_="sister") # [Elsie, # Lacie, # Tillie]

有些 tag 属性在搜索不能使用,比如 HTML5 中的 data-* 属性

data_soup = BeautifulSoup('

foo!

') data_soup.find_all(data-foo="value") # SyntaxError: keyword can't be an expression

但是可以通过 find_all() 方法的 attrs 参数定义一个字典参数来搜索包含特殊属性的tag

data_soup.find_all(attrs={"data-foo": "value"}) # [

foo!

]

#####
text 参数

通过 text 参数可以搜搜文档中的字符串内容.与 name 参数的可选值一样, text 参数接受 字符串 , 正则表达式 , 列表, True

soup.find_all(text="Elsie") # [u'Elsie']

soup.find_all(text=["Tillie", "Elsie", "Lacie"]) # [u'Elsie', u'Lacie', u'Tillie']

soup.find_all(text=re.compile("Dormouse")) [u"The Dormouse's story", u"The Dormouse's story"]

#####
limit 参数

find_all() 方法返回全部的搜索结构,如果文档树很大那么搜索会很慢.如果我们不需要全部结果,可以使用 limit 参数限制返回结果的数量

文档树中有3个 tag 符合搜索条件,但结果只返回了2个,因为我们限制了返回数量

soup.find_all("a", limit=2) # [Elsie, # Lacie]

#####
 recursive 参数

调用 tag 的 find_all() 方法时,Beautiful Soup 会检索当前 tag 的所有子孙节点,如果只想搜索 tag 的直接子节点,可以使用参数 recurs

一段简单的文档:

<html>
<head>
<title>
The Dormouse's story
</title>
</head>
...

是否使用 recursive 参数的搜索结果:

soup.html.find_all("title") # []

soup.html.find_all("title", recursive=False) # []

####
 find( name , attrs , recursive , text , **kwargs )

它与 find_all() 方法唯一的区别是 find_all() 方法的返回结果是值包含一个元素的列表,而 find() 方法直接返回结果

####
 find_parents()  fin_parent()

find_all() 和 find() 只搜索当前节点的所有子节点,孙子节点等. find_parents() 和 find_parent() 用来搜索当前节点的父辈节点,搜索方

####
 find_next_siblings()  find_next_sibling()

这2个方法通过 .next_siblings 属性对当 tag 的所有后面解析的兄弟 tag 节点进行迭代, find_next_siblings() 方法返回所有符合条件的

####
 find_previous_siblings()  find_previous_sibling()

这2个方法通过 .previous_siblings 属性对当前 tag 的前面解析的兄弟 tag 节点进行迭代, find_previous_siblings() 方法返回所有符

####
find_all_next()  find_next()

这2个方法通过 .next_elements 属性对当前 tag 的之后的 tag 和字符串进行迭代, find_all_next() 方法返回所有符合条件的节点, fi

####
find_all_previous() 和 find_previous()

这2个方法通过 .previous_elements 属性对当前节点前面的 tag 和字符串进行迭代, find_all_previous() 方法返回所有符合条件的节

注：以上（2）（3）（4）（5）（6）（7）方法参数用法与 find_all() 完全相同，原理均类似，在此不再赘述。

###
CSS选择器

我们在写 CSS 时，标签名不加任何修饰，类名前加点，id名前加 \#，在这里我们也可以利用类似的方法来筛选元素，用到的方法是 sc

####
通过标签名查找

```
print soup.select('title') #[]
```

```
print soup.select('a') #[, Lacie, Tillie]
```

```
print soup.select('b') #[The Dormouse's story]
```

####
通过类名查找

```
print soup.select('.sister') #[, Lacie, Tillie]
```

####
通过 id 名查找

```
print soup.select('#link1') #[]
```

####

组合查找

组合查找即和写 class 文件时，标签名与类名、id 名进行的组合原理是一样的，例如查找 p 标签中，id 等于 link1 的内容，二者需要用

print soup.select('p #link1') #[]

直接子标签查找

print soup.select("head > title") #[]

####
属性查找

查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。

print soup.select('a[class="sister"]') #[, Lacie, Tillie]

print soup.select('a[href="http://example.com/elsie"]') #[]

同样，属性仍然可以与上述查找方式组合，不在同一节点的空格隔开，同一节点的不加空格

print soup.select('p a[href="http://example.com/elsie"]') #[]
```

好，这就是另一种与 find_all 方法有异曲同工之妙的查找方法，是不是感觉很方便?
```

#

总结

本篇内容比较多，把 Beautiful Soup 的方法进行了大部分整理和总结，不过这还不算完全，仍然有 Beautiful Soup 的修改删除功能，不过这些功能用得比较少，只整理了查找提取的方法，希望对大家有帮助！小伙伴们加油！

熟练掌握了 Beautiful Soup，一定会给你带来太多方便，加油吧！

# 9

# 爬取糗事百科段子

大家好，前面入门已经说了那么多基础知识了，下面我们做几个实战项目来挑战一下吧。那么这次为大家带来，Python 爬取糗事百科的小段子的例子。

首先，糗事百科大家都听说过吧？糗友们发的搞笑的段子一抓一大把，这次我们尝试一下用爬虫把他们抓取下来。

本篇目标

1. 抓取糗事百科热门段子
2. 过滤带有图片的段子
3. 实现每按一次回车显示一个段子的发布时间，发布人，段子内容，点赞数。

糗事百科是不需要登录的，所以也没必要用到 Cookie，另外糗事百科有的段子是附图的，我们把图抓下来图片不便于显示，那么我们就尝试过滤掉有图的段子吧。

好，现在我们尝试抓取一下糗事百科的热门段子吧，每按下一次回车我们显示一个段子。

#

确定 URL 并抓取页面代码

首先我们确定好页面的 URL 是 http://www.qiushibaike.com/hot/page/1，其中最后一个数字1代表页数，我们可以传入不同的值来获得某一页的段子内容。

我们初步构建如下的代码来打印页面代码内容试试看，先构造最基本的页面抓取方式，看看会不会成功

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


page = 1
url = 'http://www.qiushibaike.com/hot/page/' + str(page)
try:
    request = urllib2.Request(url)
    response = urllib2.urlopen(request)
    print response.read()
except urllib2.URLError, e:
    if hasattr(e,"code"):
        print e.code
    if hasattr(e,"reason"):
        print e.reason
```

运行程序，哦不，它竟然报错了，真是时运不济，命途多舛啊

```
line 373, in _read_status
 raise BadStatusLine(line)
httplib.BadStatusLine: ''
```

好吧，应该是 headers 验证的问题，我们加上一个 headers 验证试试看吧，将代码修改如下

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

page = 1
url = 'http://www.qiushibaike.com/hot/page/' + str(page)
user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
headers = { 'User-Agent' : user_agent }
try:
```

```
    request = urllib2.Request(url,headers = headers)
    response = urllib2.urlopen(request)
    print response.read()
except urllib2.URLError, e:
    if hasattr(e,"code"):
        print e.code
    if hasattr(e,"reason"):
        print e.reason
```

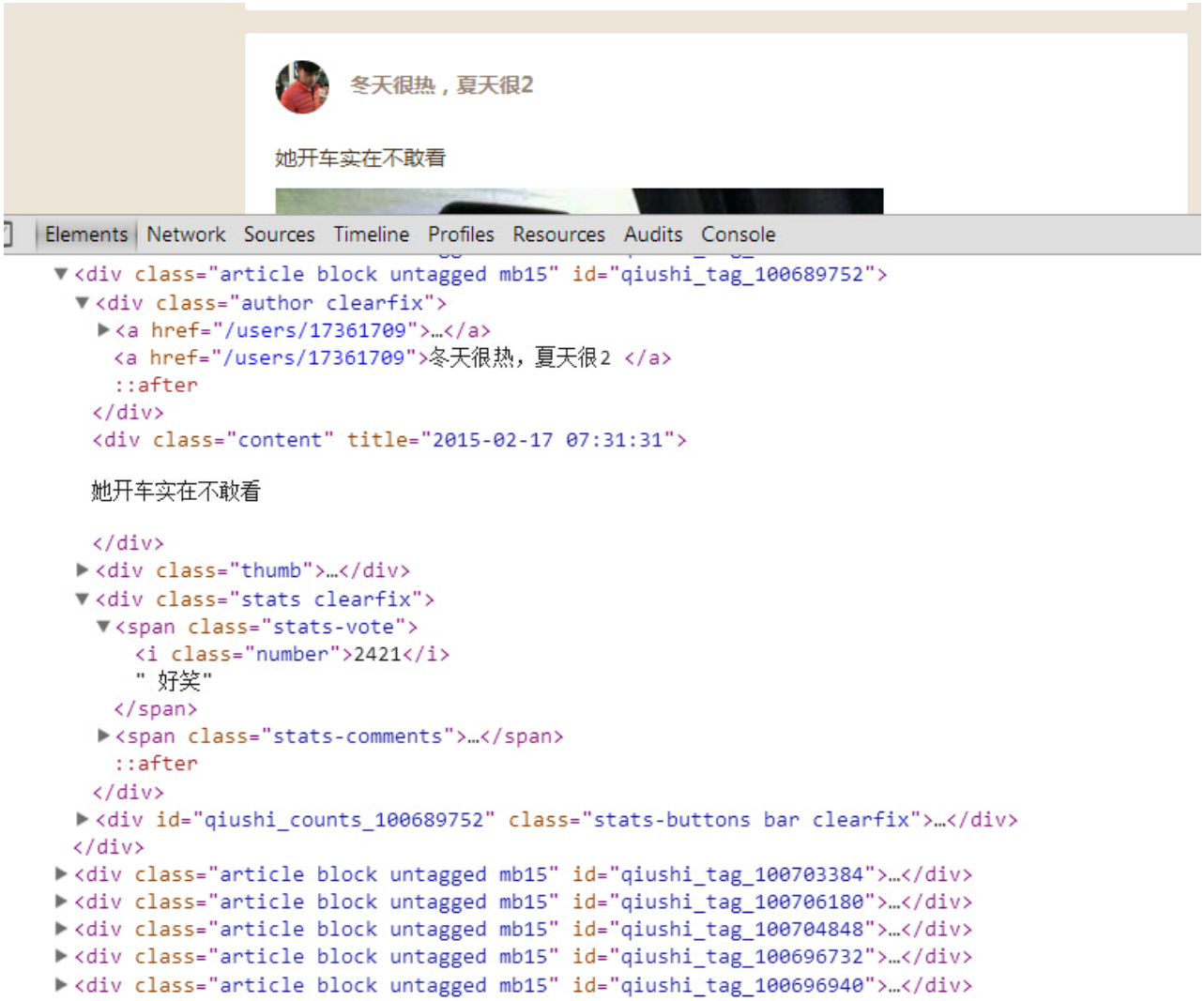
嘿嘿，这次运行终于正常了，打印出了第一页的 HTML 代码，大家可以运行下代码试试看。在这里运行结果太长就不贴了。

#

提取某一页的所有段子

好，获取了 HTML 代码之后，我们开始分析怎样获取某一页的所有段子。

首先我们审查元素看一下，按浏览器的 F12，截图如下




我们可以看到，每一个段子都是

…

包裹的内容。

现在我们想获取发布人，发布日期，段子内容，以及点赞的个数。不过另外注意的是，段子有些是带图片的，如果我们想在控制台显示图片是不现实的，所以我们直接把带有图片的段子给它剔除掉，只保存仅含文本的段子。

所以我们加入如下正则表达式来匹配一下，用到的方法是 re.findall 是找寻所有匹配的内容。方法的用法详情可以看前面说的正则表达式的介绍。

好，我们的正则表达式匹配语句书写如下，在原来的基础上追加如下代码

```
content = response.read().decode('utf-8')
pattern = re.compile('<div.?class="author.?>.?<a.?.?<a.?>(.?).?<div.?class'+ '="content".?title="(.?)">(.?)
```

(.*?)

<

div class="stats.*?class="number">(.?)',re.S) items = re.findall(pattern,content) for item in items: print it em[0],item[1],item[2],item[3],item[4]

现在正则表达式在这里稍作说明

1. .*? 是一个固定的搭配，.和*代表可以匹配任意无限多个字符，加上? 表示使用非贪婪模式进行匹配，也就是我们会尽可能短地做匹配

2. (.*?)代表一个分组，在这个正则表达式中我们匹配了五个分组，在后面的遍历item中，item[0]就代表第一个(.*?)所指代的内容，ite

3. re.S 标志代表在匹配时为点任意匹配模式，点 . 也可以代表换行符。

现在我们可以看一下部分运行结果

儒雅男神 2015-02-17 14:34:42 小时候一个一个拆着放的举个爪…
<\div class=”thumb”>
”糗事#100705418”
7093
奇怪的名字啊 2015-02-17 14:49:16
回家的路，你追我赶，回家的心情和窗外的阳光一样灿烂。一路向前，离亲人越来越近了。哪里有爸妈哪里才是家，希望所有糗友的爸爸妈妈都身体健康…….
4803

这是其中的两个段子，分别打印了发布人，发布时间，发布内容，附加图片以及点赞数。

其中，附加图片的内容我把图片代码整体抠了出来，这个对应 item[3]，所以我们只需要进一步判断 item[3]里面是否含有 img 这个字

好，我们再把上述代码中的 for 循环改为下面的样子

for item in items: haveImg = re.search("img",item[3]) if not haveImg: print item[0],item[1],item[2],item[4]

现在，整体的代码如下

# -- *coding:utf-8* -- import urllib import urllib2 import re

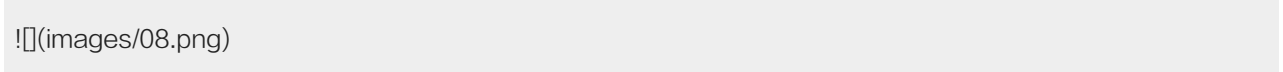
page = 1 url = 'http://www.qiushibaike.com/hot/page/' + str(page) user_agent = 'Mozilla/4.0 (compatibl e; MSIE 5.5; Windows NT)' headers = { 'User-Agent' : user_agent } try: request = urllib2.Request(ur

l,headers = headers) response = urllib2.urlopen(request) content = response.read().decode('utf-8') p attern = re.compile('<div.*?class="author.*?>.*?<a.*?.*?<a.*?>(.*?).*?<div.*?class'+ '="content".*?titl e="(.*?)">(.*?)(.*?)

<

div class="stats.*?class="number">(.*?)',re.S) items = re.findall(pattern,content) for item in items: haveI mg = re.search("img",item[3]) if not haveImg: print item[0],item[1],item[2],item[4] except urllib2.URLErr or, e: if hasattr(e,"code"): print e.code if hasattr(e,"reason"): print e.reason

运行一下看下效果



恩，带有图片的段子已经被剔除啦。是不是很开森?

###
完善交互，设计面向对象模式

好啦，现在最核心的部分我们已经完成啦，剩下的就是修一下边边角角的东西，我们想达到的目的是：

按下回车，读取一个段子，显示出段子的发布人，发布日期，内容以及点赞个数。

另外我们需要设计面向对象模式，引入类和方法，将代码做一下优化和封装，最后，我们的代码如下所示

author = 'CQC' # -- *coding:utf-8* -- import urllib import urllib2 import re import thread import time

#糗事百科爬虫类 class QSBK:

```
\#初始化方法，定义一些变量
def __init__(self):
    self.pageIndex = 1
    self.user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
    \#初始化headers
    self.headers = { 'User-Agent' : self.user_agent }
    \#存放段子的变量，每一个元素是每一页的段子们
    self.stories = []
    \#存放程序是否继续运行的变量
    self.enable = False
\#传入某一页的索引获得页面代码
def getPage(self,pageIndex):
    try:
        url = 'http://www.qiushibaike.com/hot/page/' + str(pageIndex)
```

```
        \#构建请求的request
        request = urllib2.Request(url,headers = self.headers)
        \#利用urlopen获取页面代码
        response = urllib2.urlopen(request)
        \#将页面转化为UTF-8编码
        pageCode = response.read().decode('utf-8')
        return pageCode

    except urllib2.URLError, e:
        if hasattr(e,"reason"):
            print u"连接糗事百科失败,错误原因",e.reason
            return None


\#传入某一页代码，返回本页不带图片的段子列表
def getPageItems(self,pageIndex):
    pageCode = self.getPage(pageIndex)
    if not pageCode:
        print "页面加载失败...."
        return None
    pattern = re.compile('<div.*?class="author.*?>.*?<a.*?</a>.*?<a.*?>(.*?)</a>.*?<div.*?class'+
                '="content".*?title="(.*?)">(.*?)</div>(.*?)<div class="stats.*?class="number">(.*?)</i>',re.S)
    items = re.findall(pattern,pageCode)
    \#用来存储每页的段子们
    pageStories = []
    \#遍历正则表达式匹配的信息
    for item in items:
        \#是否含有图片
        haveImg = re.search("img",item[3])
        \#如果不含有图片，把它加入list中
        if not haveImg:
            \#item[0]是一个段子的发布者，item[1]是发布时间,item[2]是内容，item[4]是点赞数
            pageStories.append([item[0].strip(),item[1].strip(),item[2].strip(),item[4].strip()])
    return pageStories

\#加载并提取页面的内容，加入到列表中
def loadPage(self):
    \#如果当前未看的页数少于2页，则加载新一页
    if self.enable == True:
        if len(self.stories) < 2:
            \#获取新一页
            pageStories = self.getPageItems(self.pageIndex)
            \#将该页的段子存放到全局list中
            if pageStories:
                self.stories.append(pageStories)
```

```
            \#获取完之后页码索引加一，表示下次读取下一页
            self.pageIndex += 1

\#调用该方法，每次敲回车打印输出一个段子
def getOneStory(self,pageStories,page):
    \#遍历一页的段子
    for story in pageStories:
        \#等待用户输入
        input = raw_input()
        \#每当输入回车一次，判断一下是否要加载新页面
        self.loadPage()
        \#如果输入Q则程序结束
        if input == "Q":
            self.enable = False
            return
        print u"第%d页\t发布人:%s\t发布时间:%s\n%s\n赞:%s\n" %(page,story[0],story[1],story[2],story[3])

\#开始方法
def start(self):
    print u"正在读取糗事百科,按回车查看新段子，Q退出"
    \#使变量为True，程序可以正常运行
    self.enable = True
    \#先加载一页内容
    self.loadPage()
    \#局部变量，控制当前读到了第几页
    nowPage = 0
    while self.enable:
        if len(self.stories)>0:
            \#从全局list中获取一页的段子
            pageStories = self.stories[0]
            \#当前读到的页数加一
            nowPage += 1
            \#将全局list中第一个元素删除，因为已经取出
            del self.stories[0]
            \#输出该页的段子
            self.getOneStory(pageStories,nowPage)
```

spider = QSBK() spider.start()

```

好啦，大家来测试一下吧，点一下回车会输出一个段子，包括发布人，发布时间，段子内容以及点赞数，是不是感觉爽爆了！
```

10

# 爬取百度贴吧帖子

大家好，上次我们实验了爬取了糗事百科的段子，那么这次我们来尝试一下爬取百度贴吧的帖子。与上一篇不同的是，这次我们需要用到文件的相关操作。

**本篇目标**

1. 对百度贴吧的任意帖子进行抓取
2. 指定是否只抓取楼主发帖内容
3. 将抓取到的内容分析并保存到文件

# #

URL 格式的确定

首先，我们先观察一下百度贴吧的任意一个帖子。

比如：http://tieba.baidu.com/p/3138733512?see_lz=1&pn=1，这是一个关于 NBA50 大的盘点，分析一下这个地址。

```
http:// 代表资源传输使用http协议
tieba.baidu.com 是百度的二级域名，指向百度贴吧的服务器。
/p/3138733512 是服务器某个资源，即这个帖子的地址定位符
see_lz和pn是该URL的两个参数，分别代表了只看楼主和帖子页码，等于1表示该条件为真
```

所以我们可以把 URL 分为两部分，一部分为基础部分，一部分为参数部分。

例如，上面的URL我们划分基础部分是 http://tieba.baidu.com/p/3138733512，参数部分是 ?see_lz=1&pn=1

# #

页面的抓取

熟悉了 URL 的格式，那就让我们用 urllib2 库来试着抓取页面内容吧。上一篇糗事百科我们最后改成了面向对象的编码方式，这次我们直接尝试一下，定义一个类名叫 BDTB(百度贴吧)，一个初始化方法，一个获取页面的方法。

其中，有些帖子我们想指定给程序是否要只看楼主，所以我们把只看楼主的参数初始化放在类的初始化上，即 init 方法。另外，获取页面的方法我们需要知道一个参数就是帖子页码，所以这个参数的指定我们放在该方法中。

综上，我们初步构建出基础代码如下：

```
\_\_author\_\_ = 'CQC'
\# −*− coding:utf−8 −*−
import urllib
import urllib2
import re

\#百度贴吧爬虫类
class BDTB:

    \#初始化，传入基地址，是否只看楼主的参数
    def __init__(self,baseUrl,seeLZ):
        self.baseURL = baseUrl
        self.seeLZ = '?see_lz='+str(seeLZ)

    \#传入页码，获取该页帖子的代码
    def getPage(self,pageNum):
        try:
            url = self.baseURL+ self.seeLZ + '&pn=' + str(pageNum)
            request = urllib2.Request(url)
            response = urllib2.urlopen(request)
            print response.read()
            return response
        except urllib2.URLError, e:
            if hasattr(e,"reason"):
                print u"连接百度贴吧失败,错误原因",e.reason
                return None

baseURL = 'http://tieba.baidu.com/p/3138733512'
```

```
bdtb = BDTB(baseURL,1)
bdtb.getPage(1)
```

运行代码，我们可以看到屏幕上打印出了这个帖子第一页楼主发言的所有内容，形式为 HTML 代码。

# #

提取相关信息

## #

提取帖子标题

首先，让我们提取帖子的标题。

在浏览器中审查元素，或者按 F12，查看页面源代码，我们找到标题所在的代码段，可以发现这个标题的 HTML 代码是

```
<h1 class="core_title_txt  " title="纯原创我心中的NBA2014-2015赛季现役50大" style="width: 396px">纯原创我心中的NBA2014
```

所以我们想提取

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下

```
<h1 class="core_title_txt.*?>(.*?)</h1>
```

所以，我们增加一个获取页面标题的方法

```
\#获取帖子标题
def getTitle(self):
page = self.getPage(1)
pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S)
result = re.search(pattern,page)
if result:
#print result.group(1)  #测试输出
return result.group(1).strip()
else:
return None
```

#### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下

```
\#获取帖子一共有多少页
def getPageNum(self):
page = self.getPage(1)
pattern = re.compile('<li class="l_reply_num.*?</span>.*?<span.*?>(.*?)</span>',re.S)
result = re.search(pattern,page)
if result:
#print result.group(1)  #测试输出
return result.group(1).strip()
else:
return None
```

#### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

```
<div id="post_content_.*?>(.*?)</div>
```

相应地，获取页面所有楼层数据的方法可以写成如下方法

```
\#获取每一层楼的内容,传入页面内容
def getContent(self,page):
pattern = re.compile('<div id="post_content_.*?>(.*?)</div>',re.S)
items = re.findall(pattern,page)
for item in items:
print item
```

好，我们运行一下结果看一下 ![](images/10.png) 真是醉了，还有一大片换行符和图片符，好口怕！既然这样，我们就要对这些文本进行处理，把各种各样复杂的标签给它剔除掉，还原精华内容，把文本处理写成一个方法也可以，不过为了实现更好的代码架构和代码重用，我们可以考虑把标签等的处理写作一个类。那我们就叫它 Tool（工具类吧），里面定义了一个方法，叫 replace，是替换各种标签的。在类中定义了几个正则表达式，主要利用了 re.sub 方法对文本进行匹配后然后替换。具体的思路已经写到注释中，大家可以看一下这个类

```
import re

\#处理页面标签类
class Tool:
\#去除img标签,7位长空格
removeImg = re.compile('<img.*?>| {7}|')
\#删除超链接标签
removeAddr = re.compile('<a.*?>|</a>')
\#把换行的标签换为\n
replaceLine = re.compile('<tr>|<div>|</div>|</p>')
\#将表格制表<td>替换为\t
replaceTD= re.compile('<td>')
\#把段落开头换为\n加空两格
replacePara = re.compile('<p.*?>')
\#将换行符或双换行符替换为\n
```

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

```
replaceBR = re.compile('<br><br>|<br>')
\#将其余标签剔除
removeExtraTag = re.compile('<.*?>')
def replace(self,x):
x = re.sub(self.removeImg,"",x)
x = re.sub(self.removeAddr,"",x)
x = re.sub(self.replaceLine,"\n",x)
x = re.sub(self.replaceTD,"\t",x)
x = re.sub(self.replacePara,"\n    ",x)
x = re.sub(self.replaceBR,"\n",x)
x = re.sub(self.removeExtraTag,"",x)
\#strip()将前后多余内容删除
return x.strip()
```

在使用时，我们只需要初始化一下这个类，然后调用 replace 方法即可。 现在整体代码是如下这样子的，现在我的代码是写到这样子的

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-
import urllib
import urllib2
import re

\#处理页面标签类
class Tool:
\#去除img标签,7位长空格
removeImg = re.compile('<img.*?>| {7}|')
\#删除超链接标签
removeAddr = re.compile('<a.*?>|</a>')
```

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

```
\#把换行的标签换为\n
replaceLine = re.compile('<tr>|<div>|</div>|</p>')
\#将表格制表<td>替换为\t
replaceTD= re.compile('<td>')
\#把段落开头换为\n加空两格
replacePara = re.compile('<p.*?>')
\#将换行符或双换行符替换为\n
replaceBR = re.compile('<br><br>|<br>')
\#将其余标签剔除
removeExtraTag = re.compile('<.*?>')
def replace(self,x):
x = re.sub(self.removeImg,"",x)
x = re.sub(self.removeAddr,"",x)
x = re.sub(self.replaceLine,"\n",x)
x = re.sub(self.replaceTD,"\t",x)
x = re.sub(self.replacePara,"\n    ",x)
x = re.sub(self.replaceBR,"\n",x)
x = re.sub(self.removeExtraTag,"",x)
\#strip()将前后多余内容删除
return x.strip()


\#百度贴吧爬虫类
class BDTB:

\#初始化，传入基地址，是否只看楼主的参数
def __init__(self,baseUrl,seeLZ):
```

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

```
self.baseURL = baseUrl
self.seeLZ = '?see_lz='+str(seeLZ)
self.tool = Tool()
\#传入页码，获取该页帖子的代码
def getPage(self,pageNum):
try:
url = self.baseURL+ self.seeLZ + '&pn=' + str(pageNum)
request = urllib2.Request(url)
response = urllib2.urlopen(request)
return response.read().decode('utf-8')
except urllib2.URLError, e:
if hasattr(e,"reason"):
print u"连接百度贴吧失败,错误原因",e.reason
return None

\#获取帖子标题
def getTitle(self):
page = self.getPage(1)
pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S)
result = re.search(pattern,page)
if result:
#print result.group(1)  #测试输出
return result.group(1).strip()
else:
return None

\#获取帖子一共有多少页
```

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

```
def getPageNum(self):
page = self.getPage(1)
pattern = re.compile('<li class="l_reply_num.*?</span>.*?<span.*?>(.*?)</span>',re.S)
result = re.search(pattern,page)
if result:
#print result.group(1)  #测试输出
return result.group(1).strip()
else:
return None

\#获取每一层楼的内容,传入页面内容
def getContent(self,page):
pattern = re.compile('<div id="post_content_.*?>(.*?)</div>',re.S)
items = re.findall(pattern,page)
#for item in items:
#  print item
print self.tool.replace(items[1])


baseURL = 'http://tieba.baidu.com/p/3138733512'
bdtb = BDTB(baseURL,1)
bdtb.getContent(bdtb.getPage(1))
```

我们尝试一下，重新再看一下效果，这下经过处理之后应该就没问题了，是不是感觉好酸爽！ ![](images/11.png) #### 替换楼层 至于这个问题，我感觉直接提取楼层没什么必要呀，因为只看楼主的话，有些楼层的编号是间隔的，所以我们得到的楼层序号是不连续的，这样我们保存下来也没什么用。所以可以尝试下面的方法： 1. 每打印输出一段楼层，写入一行横线来间隔，或者换行符也好。 2. 试着重新编一个楼层，按照顺序，设置一个变量，每打印出一个结果变量加一，打印出这个变量当做楼层。 这里我们尝试一下吧，看看效果怎样 把 getContent 方法修改如下 ```` \#获取每一层楼的内容,传入页面内容 def getContent(self,page): pattern = re.compile('

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

运行一下看看效果



嘿嘿，效果还不错吧，感觉真酸爽！接下来我们完善一下，然后写入文件

###
写入文件

最后便是写入文件的过程，过程很简单，就几句话的代码而已，主要是利用了以下两句

file = open(“tb.txt”,”w”) file.writelines(obj)

这里不再赘述，稍后直接贴上完善之后的代码。

###
完善代码

现在我们对代码进行优化，重构，在一些地方添加必要的打印信息，整理如下

__author__ = 'CQC' \# -*- coding:utf-8 -*- import urllib import urllib2 import re \#处理页面标签类 class Tool: \#去除img标签,7位长空格 removeImg = re.compile('| {7}|') \#删除超链接标签 removeAddr = re.compile('|') \#把换行的标签换为\n replaceLine = re.compile('|

|

') \#将表格制表替换为\t replaceTD= re.compile('') \#把段落开头换为\n加空两格 replacePara = re.compile('') \#将换行符或双换行符替换为\n replaceBR = re.compile('

|

') \#将其余标签剔除 removeExtraTag = re.compile('<.*?>') def replace(self,x): x = re.sub(self.removeImg,"",x)

标签中的内容，同时还要指定这个 class 确定唯一，因为 h1 标签实在太多啦。 正则表达式如下 <h1 class="core_title_txt.*?>(.*?)</h1> 所以，我们增加一个获取页面标题的方法 \#获取帖子标题 def getTitle(self): page = self.getPage(1) pattern = re.compile('<h1 class="core_title_txt.*?>(.*?)</h1>',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取帖子页数 同样地，帖子总页数我们也可以通过分析页面中的共?页来获取。所以我们的获取总页数的方法如下 \#获取帖子一共有多少页 def getPageNum(self): page = self.getPage(1) pattern = re.compile('<li class="l_reply_num.*?.*?<span.*?>(.*?)',re.S) result = re.search(pattern,page) if result: #print result.group(1) #测试输出 return result.group(1).strip() else: return None #### 提取正文内容 审查元素，我们可以看到百度贴吧每一层楼的主要内容都在 标签里面，所以我们可以写如下的正则表达式 <div id="post_content_.*?>(.*?) 相应地，获取页面所有楼层数据的方法可以写成如下方法 \#获取每一层楼的内容,传入页

x = re.sub(self.removeAddr,"",x) x = re.sub(self.replaceLine,"\n",x) x = re.sub(self.replaceTD,"\t",x) x = re.sub(self.replacePara,"\n ",x) x = re.sub(self.replaceBR,"\n",x) x = re.sub(self.removeExtraTag,"",x) \#strip()将前后多余内容删除 return x.strip() \#百度贴吧爬虫类 class BDTB: #初始化，传入基地址，是否只看楼主的参数 def __init__(self,baseUrl,seeLZ,floorTag): \#base链接地址 self.baseURL = baseUrl \#是否只看楼主 self.seeLZ = '?see_lz='+str(seeLZ) \#HTML标签剔除工具类对象 self.tool = Tool() \#全局file变量，文件写入操作对象 self.file = None \#楼层标号，初始为1 self.floor = 1 \#默认的标题，如果没有成功获取到标题的话则会用这个标题 self.defaultTitle = u"百度贴吧" \#是否写入楼分隔符的标记 self.floorTag = floorTag \#传入页码，获取该页帖子的代码 def getPage(self,pageNum): try: \#构建URL url = self.baseURL+ self.seeLZ + '&pn=' + str(pageNum) request = urllib2.Request(url) response = urllib2.urlopen(request) \#返回UTF-8格式编码内容 return response.read().decode('utf-8') \#无法连接，报错 except urllib2.URLError, e: if hasattr(e,"reason"): print u"连接百度贴吧失败,错误原因",e.reason return None \#获取帖子标题 def getTitle(self,page): \#得到标题的正则表达式 pattern = re.compile('

nt u"请输入帖子代号" baseURL = 'http://tieba.baidu.com/p/' + str(raw_input(u'http://tieba.baidu.com/p/')) seeLZ = raw_input("是否只获取楼主发言，是输入1，否输入0\n") floorTag = raw_input("是否写入楼层信息，是输入1，否输入0\n") bdtb = BDTB(baseURL,seeLZ,floorTag) bdtb.start()
```

现在程序演示如下
```

```
请输入帖子代号
http://tieba.baidu.com/p/3513281888
是否只获取楼主发言，是输入1，否输入0
1
是否写入楼层信息，是输入1，否输入0
1
该帖子共有7页
正在写入第1页数据
正在写入第2页数据
正在写入第3页数据
正在写入第4页数据
正在写入第5页数据
正在写入第6页数据
正在写入第7页数据
写入任务完成
```

完成之后，可以查看一下当前目录下多了一个以该帖子命名的 txt 文件，内容便是帖子的所有数据。

抓贴吧，就是这么简单和任性！

# 计算大学本学期绩点

大家好，本次为大家带来的项目是计算大学本学期绩点。首先说明的是，博主来自山东大学，有属于个人的学生成绩管理系统，需要学号密码才可以登录，不过可能广大读者没有这个学号密码，不能实际进行操作，所以最主要的还是获取它的原理。最主要的是了解cookie的相关操作。

**本篇目标**

1. 模拟登录学生成绩管理系统
2. 抓取本学期成绩界面
3. 计算打印本学期成绩

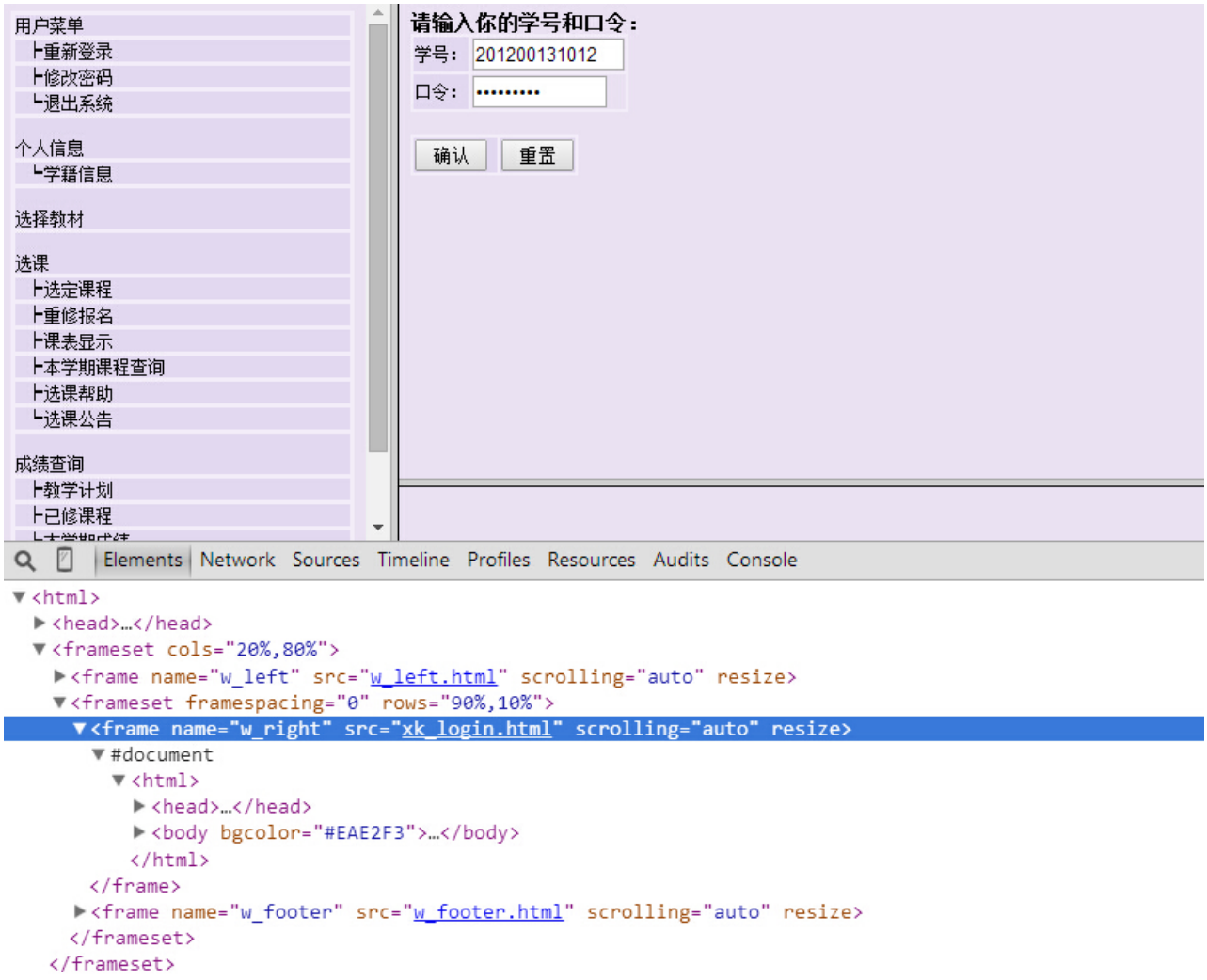
# #

URL 的获取

恩，博主来自山东大学~

先贴一个 URL，让大家知道我们学校学生信息系统的网站构架，主页是 http://jwxt.sdu.edu.cn:7890/zhxt_bks/zhxt_bks.html，山东大学学生个人信息系统，进去之后，Oh 不，他竟然用了frame，一个多么古老的而又任性的写法，真是惊出一身冷汗~

算了，就算他是frame又能拿我怎么样？我们点到登录界面，审查一下元素，先看看登录界面的URL 是怎样的？



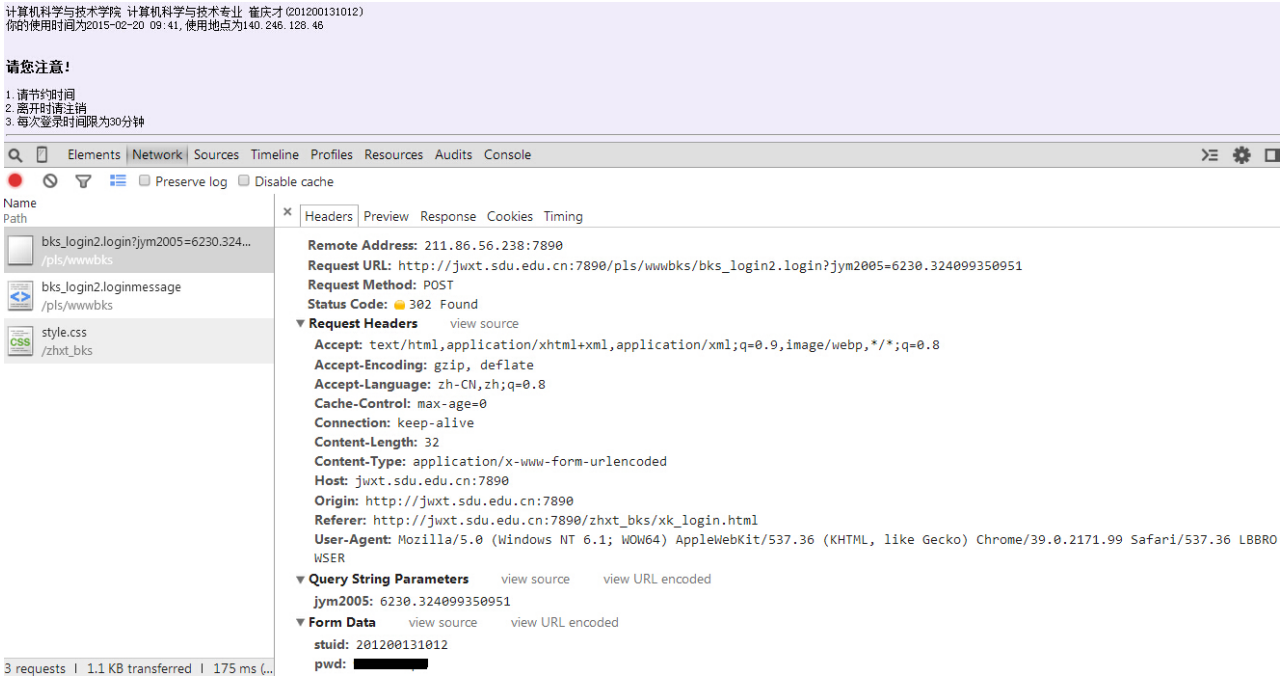

恩，看到了右侧的 frame 名称，src=”xk_login.html”，可以分析出完整的登录界面的网址为 http://jwxt.sdu.edu.cn:7890/zhxt_bks/xk_login.html，点进去看看，真是棒棒哒，他喵的竟然是清华大学选课系统，醉了，你说你抄袭就抄袭吧，改改名字也不错啊~

算了，就不和他计较了。现在，我们登录一下，用浏览器监听网络。

我用的是猎豹浏览器，审查元素时会有一个网络的选项，如果大家用的 Chrome，也有相对应的功能，Firefox 需要装插件 HttpFox，同样可以实现。

这个网络监听功能可以监听表单的传送以及请求头，响应头等等的信息。截个图看一下，恩，我偷偷把密码隐藏了，你看不到~

大家看到的是登录之后出现的信息以及 NetWork 监听，显示了 hearders 的详细信息。




最主要的内容，我们可以发现有一个表单提交的过程，提交方式为 POST，两个参数分别为 stuid 和 pwd。

请求的URL为 http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login，没错，找到表单数据和目标地址就是这么简单。

在这里注意，刚才的 http://jwxt.sdu.edu.cn:7890/zhxt_bks/xk_login.html 只是登录界面的地址，刚刚得到的这个地址才是登录索要提交到的真正的URL。希望大家这里不要混淆。

不知道山大这个系统有没有做headers的检查，我们先不管这么多，先尝试一下模拟登录并保存Cookie。

# #

模拟登录

好，通过以上信息，我们已经找到了登录的目标地址为 http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login

有一个表单提交到这个URL，表单的两个内容分别为 stuid 和 pwd，学号和密码，没有其他的隐藏信息，提交方式为 POST。

好，现在我们首先构造以下代码来完成登录。看看会不会获取到登录之后的提示页面。

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import cookielib
import re

\#山东大学绩点运算
class SDU:

    def __init__(self):
        self.loginUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login'
        self.cookies = cookielib.CookieJar()
        self.postdata = urllib.urlencode({
            'stuid':'201200131012',
            'pwd':'xxxxxx'
         })
        self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookies))

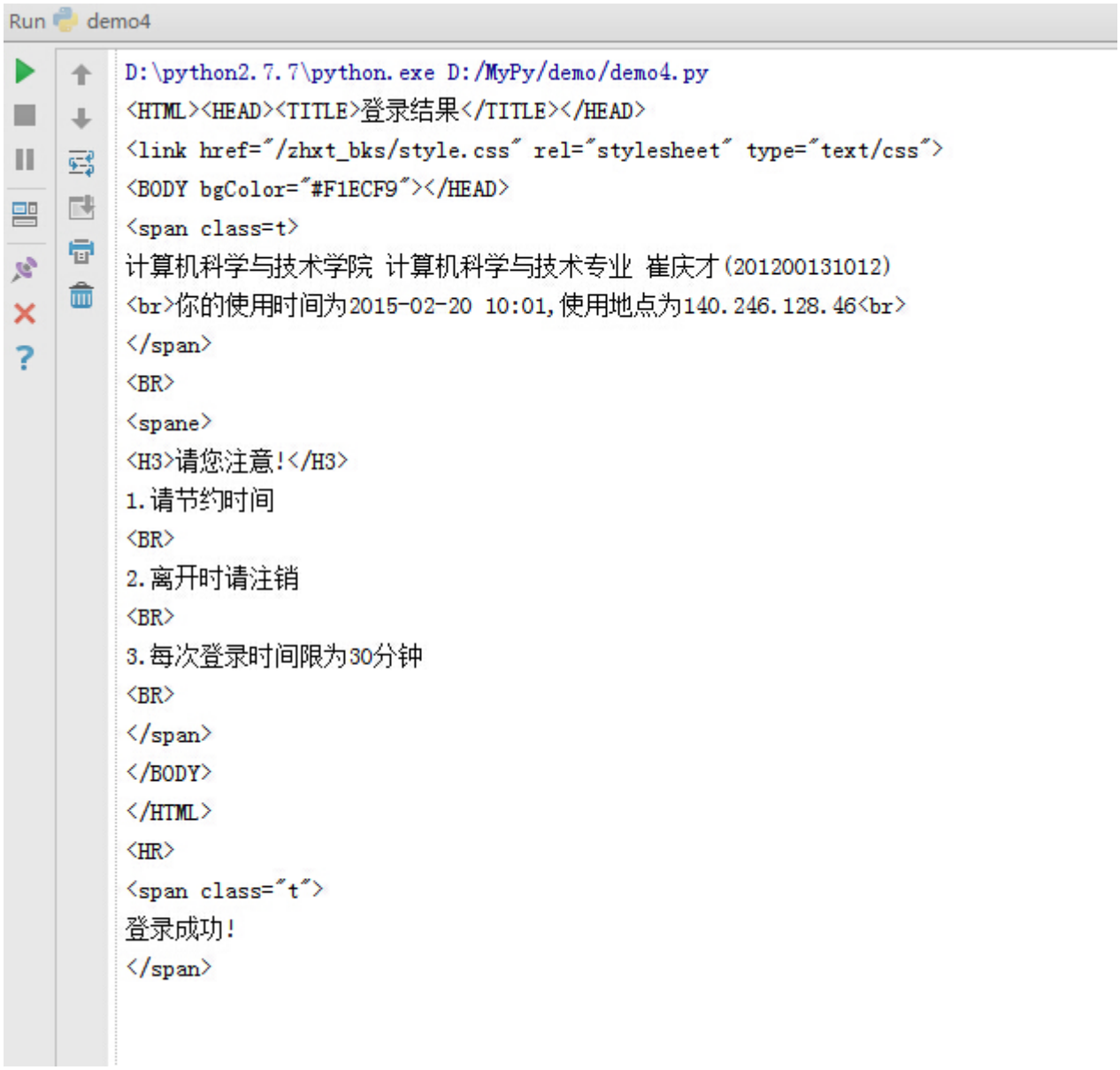
    def getPage(self):
        request  = urllib2.Request(
            url = self.loginUrl,
            data = self.postdata)
        result = self.opener.open(request)
        #打印登录内容
        print result.read().decode('gbk')


sdu = SDU()
sdu.getPage()
```

测试一下，竟然成功了，山大这网竟然没有做 headers 检查，很顺利就登录进去了。

说明一下，在这里我们利用了前面所说的 cookie，用到了 CookieJar 这个对象来保存cookies，另外通过构建 opener，利用 open 方法实现了登录。如果大家觉得这里有疑惑，请看 Python爬虫入门六之Cookie的使用，这篇文章说得比较详细。

好，我们看一下运行结果




酸爽啊，接下来我们只要再获取到本学期成绩界面然后把成绩抓取出来就好了。

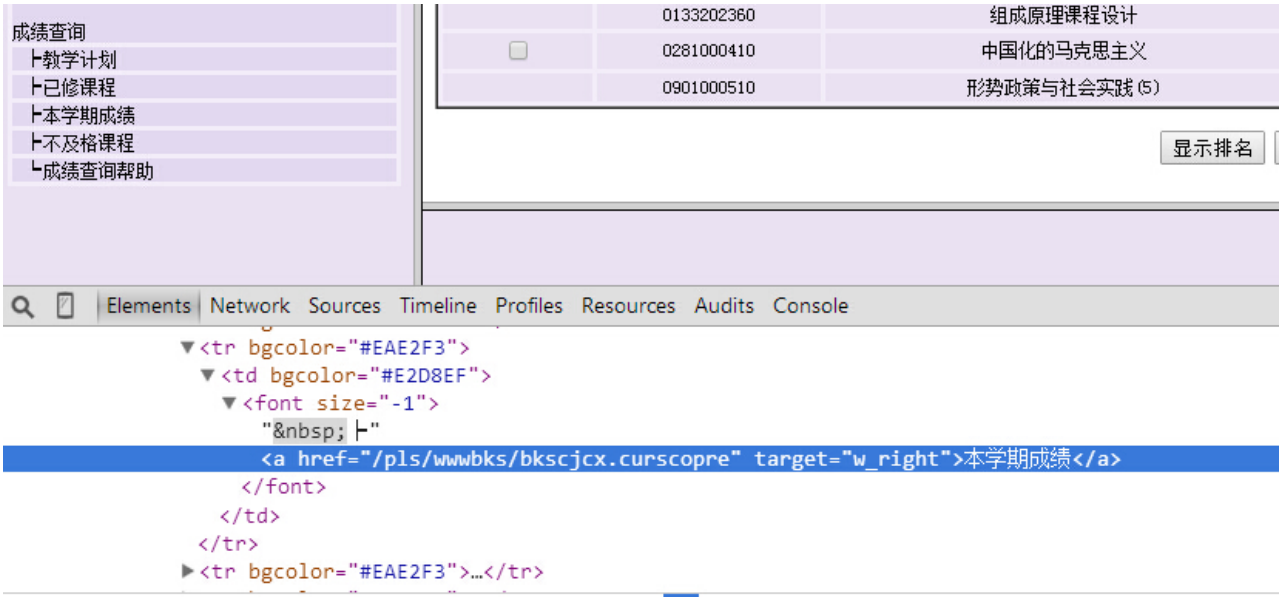
# #

抓取本学期成绩

让我们先在浏览器中找到本学期成绩界面，点击左边的本学期成绩。


重新审查元素，你会发现这个 frame 的 src 还是没有变，仍然是 xk_login.html，引起这个页面变化的原因是在左边的本学期成绩这个超链接设置了一个目标 frame，所以，那个页面就显示在右侧了。

所以，让我们再审查一下本学期成绩这个超链接的内容是什么~



恩，找到它了，本学期成绩。

那么，完整的URL就是 http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bkscjcx.curscopre，好，URL已经找到了，我们继续完善一下代码，获取这个页面。

```
__author__ = 'CQC'
\# −*− coding:utf−8 −*−

import urllib
import urllib2
import cookielib
import re

\#山东大学绩点运算
class SDU:

    def __init__(self):
        \#登录URL
        self.loginUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login'
        \#本学期成绩URL
        self.gradeUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bkscjcx.curscopre'
        self.cookies = cookielib.CookieJar()
        self.postdata = urllib.urlencode({
            'stuid':'201200131012',
            'pwd':'xxxxxx'
         })
        \#构建opener
        self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookies))

    \#获取本学期成绩页面
    def getPage(self):
        request  = urllib2.Request(
            url = self.loginUrl,
            data = self.postdata)
        result = self.opener.open(request)
        result = self.opener.open(self.gradeUrl)
        \#打印登录内容
        print result.read().decode('gbk')


sdu = SDU()
sdu.getPage()
```

上面的代码，我们最主要的是增加了

```
result = self.opener.open(self.gradeUrl)
```

这句代码，用原来的 opener 访问一个本学期成绩的 URL 即可。运行结果如下

```
<td bgcolor="#EAE2F3"><p align="center"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="013320191012015011288    微机原理与接口"></p></td>
<td bgcolor="#EAE2F3"><p align="center">0133201910</p></td>
<td bgcolor="#EAE2F3"><p align="center">微机原理与接口</p></td>
<td bgcolor="#EAE2F3"><p align="center">1</p></td>
<td bgcolor="#EAE2F3"><p align="center">2.5</p></td>
<td bgcolor="#EAE2F3"><p align="center">20150112</p></td>
<td bgcolor="#EAE2F3"><p align="center">88</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">0133202360</p></td>
<td bgcolor="#EAE2F3"><p align="center">组成原理课程设计</p></td>
<td bgcolor="#EAE2F3"><p align="center">1</p></td>
<td bgcolor="#EAE2F3"><p align="center">2</p></td>
<td bgcolor="#EAE2F3"><p align="center">20150112</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"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="028100044352015011288    中国化的马克思主义"></p></td>
<td bgcolor="#EAE2F3"><p align="center">0281000410</p></td>
<td bgcolor="#EAE2F3"><p align="center">中国化的马克思主义</p></td>
<td bgcolor="#EAE2F3"><p align="center">335</p></td>
<td bgcolor="#EAE2F3"><p align="center">3</p></td>
```

恩，本学期成绩的页面已经被我们抓取下来了，接下来用正则表达式提取一下，然后计算学分即可

#

抓取有效信息

接下来我们就把页面内容提取一下，最主要的便是学分以及分数了。

平均绩点 = ∑（每科学分*每科分数）/总学分

所以我们把每科的学分以及分数抓取下来就好了，对于有些课打了良好或者优秀等级的，我们不进行抓取。

我们可以发现每一科都是TR标签，然后是一系列的td标签

```
<TR>
<td bgcolor="#EAE2F3"><p align="center"><INPUT TYPE="checkbox" NAME="p_pm" VALUE="013320131012015011294
<td bgcolor="#EAE2F3"><p align="center">0133201310</p></td>
<td bgcolor="#EAE2F3"><p align="center">面向对象技术</p></td>
<td bgcolor="#EAE2F3"><p align="center">1</p></td>
<td bgcolor="#EAE2F3"><p align="center">2.5</p></td>
<td bgcolor="#EAE2F3"><p align="center">20150112</p></td>
<td bgcolor="#EAE2F3"><p align="center">94</p></td>
<td bgcolor="#EAE2F3"><p align="center">必修</p></td>
</TR>
```

我们用下面的正则表达式进行提取即可，部分代码如下

```
page = self.getPage()
myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?>(.*?)</p>.*?<p.*?<p.*?>(.*?)</p>.*?</TR>',page,re.S)
for item in myItems:
  self.credit.append(item[0].encode('gbk'))
  self.grades.append(item[1].encode('gbk'))
```

主要利用了 findall 方法，这个方法在此就不多介绍了，前面我们已经用过多次了。

得到的学分和分数我们都用列表 list 进行存储，所以用了 append 方法，每获取到一个信息就把它加进去。

# #

整理计算最后绩点

恩，像上面那样把学分绩点都保存到列表 list 中了，所以我们最后用一个公式来计算学分绩点就好了，最后整理后的代码如下：

```
\# −*− coding: utf−8 −*−

import urllib
import urllib2
import cookielib
import re
import string

\#绩点运算
class SDU:

  \#类的初始化
  def __init__(self):
    \#登录URL
    self.loginUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login'
    \#成绩URL
    self.gradeUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bkscjcx.curscopre'
    \#CookieJar对象
    self.cookies = cookielib.CookieJar()
    \#表单数据
    self.postdata = urllib.urlencode({
        'stuid':'201200131012',
        'pwd':'xxxxx'
      })
    \#构建opener
    self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.cookies))
    \#学分list
    self.credit = []
    \#成绩list
    self.grades = []

  def getPage(self):
    req  = urllib2.Request(
        url = self.loginUrl,
        data = self.postdata)
    result = self.opener.open(req)
```

```
    result = self.opener.open(self.gradeUrl)
    \#返回本学期成绩页面
    return result.read().decode('gbk')

  def getGrades(self):
    \#获得本学期成绩页面
    page = self.getPage()
    \#正则匹配
    myItems = re.findall('<TR>.*?<p.*?<p.*?<p.*?<p.*?<p.*?>(.*?)</p>.*?<p.*?<p.*?>(.*?)</p>.*?</TR>',page,re.S)
    for item in myItems:
      self.credit.append(item[0].encode('gbk'))
      self.grades.append(item[1].encode('gbk'))
    self.getGrade()

  def getGrade(self):
    \#计算总绩点
    sum = 0.0
    weight = 0.0
    for i in range(len(self.credit)):
      if(self.grades[i].isdigit()):
        sum += string.atof(self.credit[i])*string.atof(self.grades[i])
        weight += string.atof(self.credit[i])

    print u"本学期绩点为:",sum/weight

sdu = SDU()
sdu.getGrades()
```

好，最后就会打印输出本学期绩点是多少，小伙伴们最主要的了解上面的编程思路就好。

最主要的内容就是 Cookie 的使用，模拟登录的功能。

本文思路参考来源：汪海的爬虫

12

# 抓取淘宝 MM 照片

福利啊福利，本次为大家带来的项目是抓取淘宝 MM 照片并保存起来，大家有没有很激动呢?

**本篇目标**

1. 抓取淘宝 MM 的姓名，头像，年龄
2. 抓取每一个MM的资料简介以及写真图片
3. 把每一个MM的写真图片按照文件夹保存到本地
4. 熟悉文件保存的过程

# #

URL 的格式

在这里我们用到的URL是 http://mm.taobao.com/json/request_top_list.htm?page=1，问号前面是基地址，后面的参数page是代表第几页，可以随意更换地址。点击开之后，会发现有一些淘宝MM的简介，并附有超链接链接到个人详情页面。

我们需要抓取本页面的头像地址，MM 姓名，MM 年龄，MM 居住地，以及 MM 的个人详情页面地址。

#

抓取简要信息

相信大家经过上几次的实战，对抓取和提取页面的地址已经非常熟悉了，这里没有什么难度了，我们首先抓取本页面的MM详情页面地址，姓名，年龄等等的信息打印出来，直接贴代码如下

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import re

class Spider:

    def __init__(self):
        self.siteURL = 'http://mm.taobao.com/json/request_top_list.htm'

    def getPage(self,pageIndex):
        url = self.siteURL + "?page=" + str(pageIndex)
        print url
        request = urllib2.Request(url)
        response = urllib2.urlopen(request)
        return response.read().decode('gbk')

    def getContents(self,pageIndex):
        page = self.getPage(pageIndex)
        pattern = re.compile('<div class="list-item".*?pic-word.*?<a href="(.*?)".*?<img src="(.*?)".*?<a class="lady-name.*?
        items = re.findall(pattern,page)
        for item in items:
            print item[0],item[1],item[2],item[3],item[4]

spider = Spider()
spider.getContents(1)
```

运行结果如下


D:\python2.7.7\python.exe D:/MyPy/mm/spider.py
http://mm.taobao.com/json/request_top_list.htm?page=1
http://mm.taobao.com/687471686.htm http://img07.taobaocdn.com/sns_logo/i7/TB18D3rGVXXXXXvXVXXSutbFXXX.jpg_60x60.jpg 田媛媛 25 广州市
http://mm.taobao.com/405095521.htm http://img05.taobaocdn.com/sns_logo/i5/TB1FM8wGFXXXXb1XXXXSutbFXXX.jpg_60x60.jpg 朱亦颖 25 杭州市
http://mm.taobao.com/631300490.htm http://img08.taobaocdn.com/sns_logo/i8/TB1qi5WGVXXXXXxXFXXSutbFXXX.jpg_60x60.jpg 崔辰辰 26 杭州市
http://mm.taobao.com/414457129.htm http://img01.taobaocdn.com/sns_logo/i1/TB1gxkiGVXXXXXZXpXXSutbFXXX.jpg_60x60.jpg 大猫儿 29 广州市
http://mm.taobao.com/141234233.htm http://img01.taobaocdn.com/sns_logo/i1/TB1P980FVXXXXXQXFXXSutbFXXX.jpg_60x60.jpg 金甜甜 24 广州市
http://mm.taobao.com/96614110.htm http://img04.taobaocdn.com/sns_logo/i4/TB1nxAbGFXXXXa3aXXXSutbFXXX.jpg_60x60.jpg 紫轩 28 杭州市
http://mm.taobao.com/37448401.htm http://img08.taobaocdn.com/sns_logo/i8/TB1bU.UFFXXXXc0XpXXSutbFXXX.jpg_60x60.jpg 谢婷婷 26 杭州市
http://mm.taobao.com/74386764.htm http://img08.taobaocdn.com/sns_logo/i8/TB1HmB7GXXXXXbHXFXXSutbFXXX.jpg_60x60.jpg 夏晨洁 26 杭州市
http://mm.taobao.com/523216808.htm http://img03.taobaocdn.com/sns_logo/i3/TB1CQbiGVXXXXaYXFXXSutbFXXX.jpg_60x60.jpg Cherry 27 广州市
http://mm.taobao.com/46599595.htm http://img07.taobaocdn.com/sns_logo/i7/TB1QPszGXXXXXbSXVXXSutbFXXX.jpg_60x60.jpg 雪倩nika 24 杭州市
Process finished with exit code 0

# #

文件写入简介

在这里，我们有写入图片和写入文本两种方式

## #

写入图片

```
\#传入图片地址，文件名，保存单张图片
def saveImg(self,imageURL,fileName):
    u = urllib.urlopen(imageURL)
    data = u.read()
    f = open(fileName, 'wb')
    f.write(data)
    f.close()
```

## #

写入文本

```
def saveBrief(self,content,name):
    fileName = name + "/" + name + ".txt"
    f = open(fileName,"w+")
    print u"正在偷偷保存她的个人信息为",fileName
    f.write(content.encode('utf-8'))
```

## #

创建新目录

```
\#创建新目录
def mkdir(self,path):
    path = path.strip()
    \# 判断路径是否存在
    \# 存在     True
    \# 不存在   False
    isExists=os.path.exists(path)
```

```
\# 判断结果
if not isExists:
    \# 如果不存在则创建目录
    \# 创建目录操作函数
    os.makedirs(path)
    return True
else:
    \# 如果目录存在则不创建，并提示目录已存在
    return False
```

#

代码完善

主要的知识点已经在前面都涉及到了，如果大家前面的章节都已经看了，完成这个爬虫不在话下，具体的详情在此不再赘述，直接帖代码啦。

spider.py

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import re
import tool
import os

\#抓取MM
class Spider:

    \#页面初始化
    def __init__(self):
        self.siteURL = 'http://mm.taobao.com/json/request_top_list.htm'
        self.tool = tool.Tool()

    \#获取索引页面的内容
    def getPage(self,pageIndex):
        url = self.siteURL + "?page=" + str(pageIndex)
        request = urllib2.Request(url)
        response = urllib2.urlopen(request)
        return response.read().decode('gbk')

    \#获取索引界面所有MM的信息，list格式
    def getContents(self,pageIndex):
        page = self.getPage(pageIndex)
        pattern = re.compile('<div class="list-item".*?pic-word.*?<a href="(.*?)".*?<img src="(.*?)".*?<a class="lady-name.*?
        items = re.findall(pattern,page)
        contents = []
        for item in items:
            contents.append([item[0],item[1],item[2],item[3],item[4]])
        return contents
```

```
\#获取MM个人详情页面
def getDetailPage(self,infoURL):
    response = urllib2.urlopen(infoURL)
    return response.read().decode('gbk')

\#获取个人文字简介
def getBrief(self,page):
    pattern = re.compile('<div class="mm-aixiu-content".*?>(.*?)<!--',re.S)
    result = re.search(pattern,page)
    return self.tool.replace(result.group(1))

\#获取页面所有图片
def getAllImg(self,page):
    pattern = re.compile('<div class="mm-aixiu-content".*?>(.*?)<!--',re.S)
    \#个人信息页面所有代码
    content = re.search(pattern,page)
    \#从代码中提取图片
    patternImg = re.compile('<img.*?src="(.*?)"',re.S)
    images = re.findall(patternImg,content.group(1))
    return images


\#保存多张写真图片
def saveImgs(self,images,name):
    number = 1
    print u"发现",name,u"共有",len(images),u"张照片"
    for imageURL in images:
        splitPath = imageURL.split('.')
        fTail = splitPath.pop()
        if len(fTail) > 3:
            fTail = "jpg"
        fileName = name + "/" + str(number) + "." + fTail
        self.saveImg(imageURL,fileName)
        number += 1

\# 保存头像
def saveIcon(self,iconURL,name):
    splitPath = iconURL.split('.')
    fTail = splitPath.pop()
    fileName = name + "/icon." + fTail
    self.saveImg(iconURL,fileName)

\#保存个人简介
def saveBrief(self,content,name):
    fileName = name + "/" + name + ".txt"
```

```
    f = open(fileName,"w+")
    print u"正在偷偷保存她的个人信息为",fileName
    f.write(content.encode('utf-8'))


\#传入图片地址，文件名，保存单张图片
def saveImg(self,imageURL,fileName):
     u = urllib.urlopen(imageURL)
     data = u.read()
     f = open(fileName, 'wb')
     f.write(data)
     print u"正在悄悄保存她的一张图片为",fileName
     f.close()

\#创建新目录
def mkdir(self,path):
    path = path.strip()
    \# 判断路径是否存在
    \# 存在     True
    \# 不存在   False
    isExists=os.path.exists(path)
    \# 判断结果
    if not isExists:
        \# 如果不存在则创建目录
        print u"偷偷新建了名字叫做",path,u'的文件夹'
        \# 创建目录操作函数
        os.makedirs(path)
        return True
    else:
        \# 如果目录存在则不创建，并提示目录已存在
        print u"名为",path,'的文件夹已经创建成功'
        return False

\#将一页淘宝MM的信息保存起来
def savePageInfo(self,pageIndex):
    \#获取第一页淘宝MM列表
    contents = self.getContents(pageIndex)
    for item in contents:
        #item[0]个人详情URL,item[1]头像URL,item[2]姓名,item[3]年龄,item[4]居住地
        print u"发现一位模特,名字叫",item[2],u"芳龄",item[3],u",她在",item[4]
        print u"正在偷偷地保存",item[2],"的信息"
        print u"又意外地发现她的个人地址是",item[0]
        \#个人详情页面的URL
        detailURL = item[0]
        \#得到个人详情页面代码
```

```
            detailPage = self.getDetailPage(detailURL)
            \#获取个人简介
            brief = self.getBrief(detailPage)
            \#获取所有图片列表
            images = self.getAllImg(detailPage)
            self.mkdir(item[2])
            \#保存个人简介
            self.saveBrief(brief,item[2])
            \#保存头像
            self.saveIcon(item[1],item[2])
            \#保存图片
            self.saveImgs(images,item[2])

    \#传入起止页码，获取MM图片
    def savePagesInfo(self,start,end):
        for i in range(start,end+1):
            print u"正在偷偷寻找第",i,u"个地方，看看MM们在不在"
            self.savePageInfo(i)


\#传入起止页码即可，在此传入了2,10,表示抓取第2到10页的MM
spider = Spider()
spider.savePagesInfo(2,10)
```

tool.py

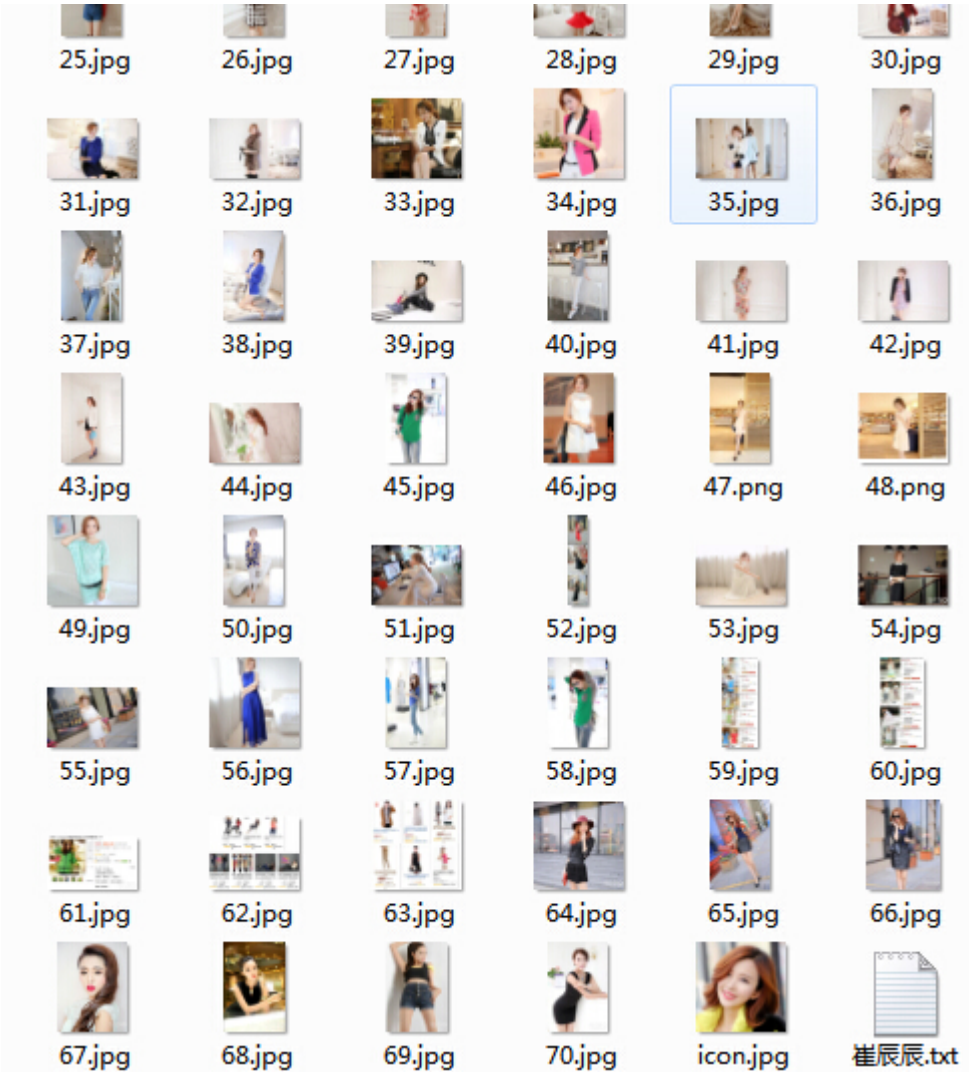
```
__author__ = 'CQC'
\#-*- coding:utf-8 -*-
import re

\#处理页面标签类
class Tool:
    \#去除img标签,1-7位空格, 
    removeImg = re.compile('<img.*?>| {1,7}| ')
    \#删除超链接标签
    removeAddr = re.compile('<a.*?>|</a>')
    \#把换行的标签换为\n
    replaceLine = re.compile('<tr>|<div>|</div>|</p>')
    \#将表格制表<td>替换为\t
    replaceTD= re.compile('<td>')
    \#将换行符或双换行符替换为\n
    replaceBR = re.compile('<br><br>|<br>')
    \#将其余标签剔除
    removeExtraTag = re.compile('<.*?>')
    \#将多行空行删除
    removeNoneLine = re.compile('\n+')
```

```
    def replace(self,x):
        x = re.sub(self.removeImg,"",x)
        x = re.sub(self.removeAddr,"",x)
        x = re.sub(self.replaceLine,"\n",x)
        x = re.sub(self.replaceTD,"\t",x)
        x = re.sub(self.replaceBR,"\n",x)
        x = re.sub(self.removeExtraTag,"",x)
        x = re.sub(self.removeNoneLine,"\n",x)
        #strip()将前后多余内容删除
        return x.strip()
```

以上两个文件就是所有的代码内容，运行一下试试看，那叫一个酸爽啊

看看文件夹里面有什么变化




不知不觉，海量的MM图片已经进入了你的电脑，还不快快去试试看！！

# 13

# 模拟登录淘宝并获取所有订单

经过多次尝试，模拟登录淘宝终于成功了，实在是不容易，淘宝的登录加密和验证太复杂了，煞费苦心，在此写出来和大家一起分享，希望大家支持。

# #

本篇内容

1. python模拟登录淘宝网页
2. 获取登录用户的所有订单详情
3. 学会应对出现验证码的情况
4. 体会一下复杂的模拟登录机制

# #

探索部分成果

1. 淘宝的密码用了AES加密算法，最终将密码转化为256位，在 POST 时，传输的是256位长度的密码。
2. 淘宝在登录时必须要输入验证码，在经过几次尝试失败后最终获取了验证码图片让用户手动输入来验证。
3. 淘宝另外有复杂且每天在变的 ua 加密算法，在程序中我们需要提前获取某一 ua 码才可进行模拟登录。
4. 在获取最后的登录 st 码时，历经了多次请求和正则表达式提取，且 st 码只可使用一次。

#

整体思路梳理

1. 手动到浏览器获取 ua 码以及 加密后的密码，只获取一次即可，一劳永逸。
2. 向登录界面发送登录请求，POST 一系列参数，包括 ua 码以及密码等等，获得响应，提取验证码图像。
3. 用户输入手动验证码，重新加入验证码数据再次用 POST 方式发出请求，获得响应，提取 J_Htoken。
4. 利用 J、_Htoken 向 alipay 发出请求，获得响应，提取 st 码。
5. 利用 st 码和用户名，重新发出登录请求，获得响应，提取重定向网址，存储 cookie。
6. 利用 cookie 向其他个人页面如订单页面发出请求，获得响应，提取订单详情。

是不是没看懂？没事，下面我将一点点说明自己模拟登录的过程，希望大家可以理解。

#

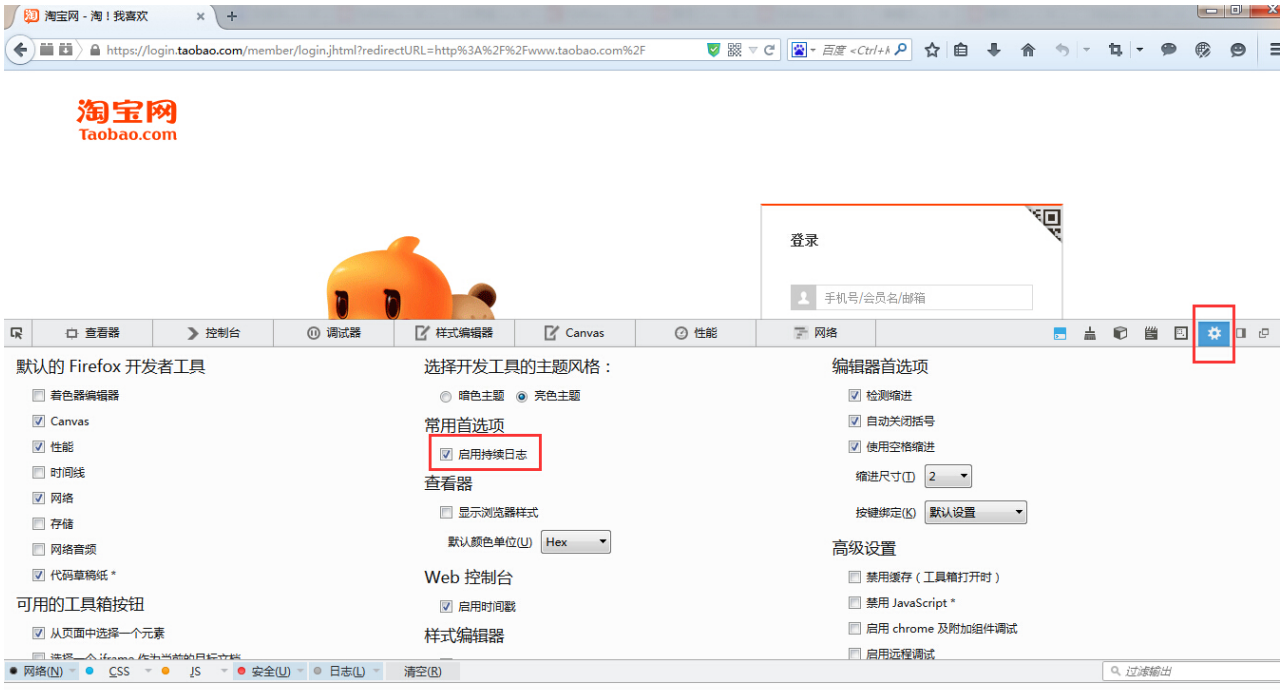
前期准备

由于淘宝的 ua 算法和 aes 密码加密算法太复杂了，ua 算法在淘宝每天都是在变化的，不过，这个内容你获取之后一直用即可，经过测试之后没有问题，一劳永逸。

那么 ua 和 aes 密码怎样获取呢?

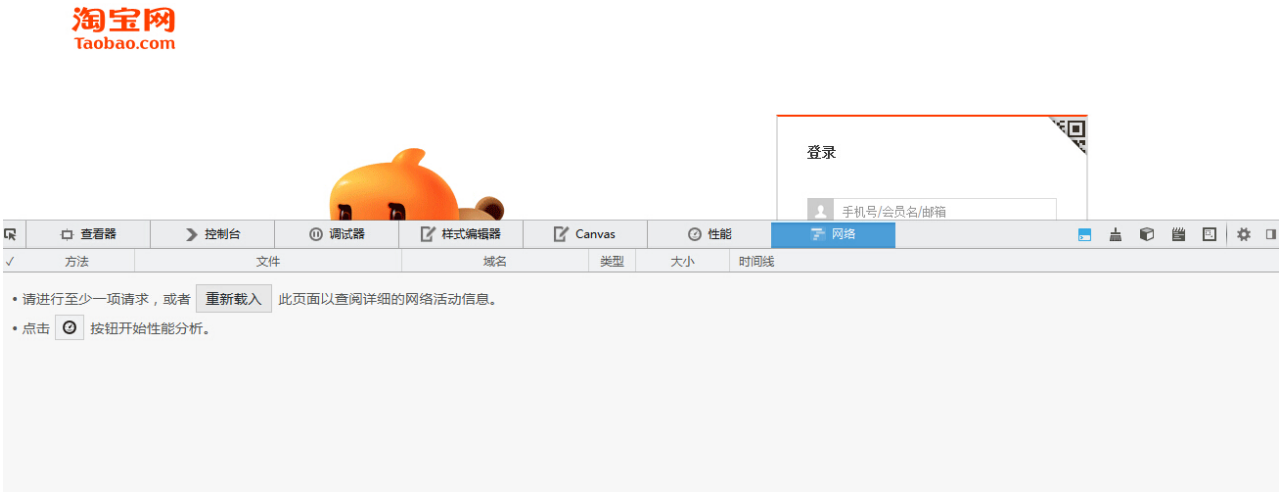
我们就从浏览器里面直接获取吧，打开浏览器，找到淘宝的登录界面，按 F12 或者浏览器右键审查元素。

在这里我用的是火狐浏览器，首先记得在浏览器中设置一下显示持续日志，要不然页面跳转了你就看不到之前抓取的信息了。在这里截图如下:

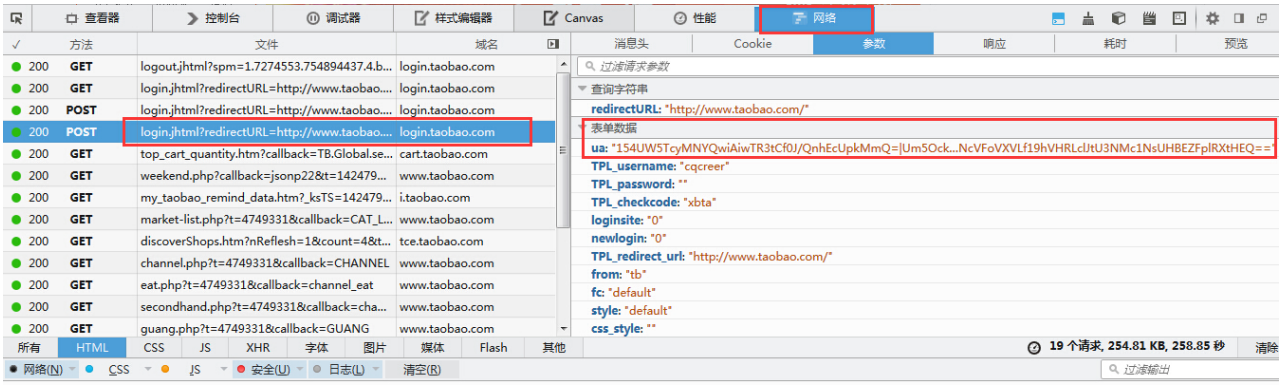



好，那么接下来我们就从浏览器中获取 ua 和 aes 密码

点击网络选项卡，这时都是空的，什么数据也没有截取。这时你就在网页上登录一下试试吧，输入用户名啊，密码啊，有必要时需要输入验证码，点击登录。




等跳转成功后，你就可以看到好多日志记录了，点击图中的那一行 login.taobo.com，然后查看参数，你就会发现表单数据了，其中就包括 ua 还有下面的 password2，把这俩复制下来，我们之后要用到的。这就是我们需要的 ua 还有 aes 加密后的密码。




恩，读到这里，你应该获取到了属于自己的 ua 和 password2 两个内容。

#

输入验证码并获取 J_HToken

经过博主本人亲自验证，有时候，在模拟登录时你并不需要输入验证码，它直接返回的结果就是前面所说的下一步用到的 J_Token，而有时候你则会需要输入验证码，等你手动输入验证码之后，重新请求登录一次。

博主是边写程序边更新文章的，现在写完了是否有必要输入验证码的检验以及在浏览器中呈现验证码。

代码如下

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import cookielib
import re
import webbrowser

\#模拟登录淘宝类
class Taobao:

    \#初始化方法
    def __init__(self):
        \#登录的URL
        self.loginURL = "https://login.taobao.com/member/login.jhtml"
        \#代理IP地址，防止自己的IP被封禁
        self.proxyURL = 'http://120.193.146.97:843'
        \#登录POST数据时发送的头部信息
        self.loginHeaders = {
            'Host':'login.taobao.com',
            'User-Agent' : 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0',
            'Referer' : 'https://login.taobao.com/member/login.jhtml',
            'Content-Type': 'application/x-www-form-urlencoded',
            'Connection' : 'Keep-Alive'
        }
        \#用户名
        self.username = 'cqcre'
        \#ua字符串，经过淘宝ua算法计算得出，包含了时间戳,浏览器,屏幕分辨率,随机数,鼠标移动,鼠标点击,其实还有键盘输入记录,鼠标
        self.ua = '191UW5TcyMNYQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um5Ockt0TXdPc011TXVKdyE=|U2xMHDJ+H2QJZwl
        \#密码，在这里不能输入真实密码，淘宝对此密码进行了加密处理，256位，此处为加密后的密码
        self.password2 = '7511aa68sx629e45de220d29174f1066537a73420ef6dbb5b46f202396703a2d56b0312df8769d886
```

```
self.post = post = {
    'ua':self.ua,
    'TPL_checkcode':'',
    'CtrlVersion': '1,0,0,7',
    'TPL_password':'',
    'TPL_redirect_url':'http://i.taobao.com/my_taobao.htm?nekot=udm8087E1424147022443',
    'TPL_username':self.username,
    'loginsite':'0',
    'newlogin':'0',
    'from':'tb',
    'fc':'default',
    'style':'default',
    'css_style':'',
    'tid':'XOR_1_000000000000000000000000000000_625C4720470A0A050976770A',
    'support':'000001',
    'loginType':'4',
    'minititle':'',
    'minipara':'',
    'umto':'NaN',
    'pstrong':'3',
    'llnick':'',
    'sign':'',
    'need_sign':'',
    'isIgnore':'',
    'full_redirect':'',
    'popid':'',
    'callback':'',
    'guf':'',
    'not_duplite_str':'',
    'need_user_id':'',
    'poy':'',
    'gvfdcname':'10',
    'gvfdcre':'',
    'from_encoding ':'',
    'sub':'',
    'TPL_password_2':self.password2,
    'loginASR':'1',
    'loginASRSuc':'1',
    'allp':'',
    'oslanguage':'zh-CN',
    'sr':'1366*768',
    'osVer':'windows|6.1',
    'naviVer':'firefox|35'
}
\#将POST的数据进行编码转换
```

```
    self.postData = urllib.urlencode(self.post)
    \#设置代理
    self.proxy = urllib2.ProxyHandler({'http':self.proxyURL})
    \#设置cookie
    self.cookie = cookielib.LWPCookieJar()
    \#设置cookie处理器
    self.cookieHandler = urllib2.HTTPCookieProcessor(self.cookie)
    \#设置登录时用到的opener，它的open方法相当于urllib2.urlopen
    self.opener = urllib2.build_opener(self.cookieHandler,self.proxy,urllib2.HTTPHandler)


\#得到是否需要输入验证码，这次请求的相应有时会不同，有时需要验证有时不需要
def needIdenCode(self):
    \#第一次登录获取验证码尝试，构建request
    request = urllib2.Request(self.loginURL,self.postData,self.loginHeaders)
    \#得到第一次登录尝试的相应
    response = self.opener.open(request)
    \#获取其中的内容
    content = response.read().decode('gbk')
    \#获取状态吗
    status = response.getcode()
    \#状态码为200，获取成功
    if status == 200:
        print u"获取请求成功"
        \#\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801这六个字是请输入验证码的utf-8编码
        pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801',re.S)
        result = re.search(pattern,content)
        \#如果找到该字符，代表需要输入验证码
        if result:
            print u"此次安全验证异常，您需要输入验证码"
            return content
        \#否则不需要
        else:
            print u"此次安全验证通过，您这次不需要输入验证码"
            return False
    else:
        print u"获取请求失败"

\#得到验证码图片
def getIdenCode(self,page):
    \#得到验证码的图片
    pattern = re.compile('<img id="J_StandardCode_m.*?data-src="(.*?)"',re.S)
    \#匹配的结果
    matchResult = re.search(pattern,page)
    \#已经匹配得到内容，并且验证码图片链接不为空
```

```
        if matchResult and matchResult.group(1):
            print matchResult.group(1)
            return matchResult.group(1)
        else:
            print u"没有找到验证码内容"
            return False

    \#程序运行主干
    def main(self):
        \#是否需要验证码，是则得到页面内容，不是则返回False
        needResult = self.needIdenCode()
        if not needResult == False:
            print u"您需要手动输入验证码"
            idenCode = self.getIdenCode(needResult)
            \#得到了验证码的链接
            if not idenCode == False:
                print u"验证码获取成功"
                print u"请在浏览器中输入您看到的验证码"
                webbrowser.open_new_tab(idenCode)
            \#验证码链接为空，无效验证码
            else:
                print u"验证码获取失败，请重试"
        else:
            print u"不需要输入验证码"



taobao = Taobao()
taobao.main()
```

恩，请把里面的 ua 和 password2 还有用户名换成自己的进行尝试，用我的可能会产生错误的。

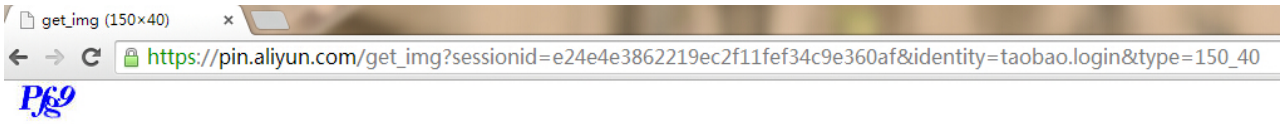
运行结果

```
C:\>python -V
Python 2.7.9

C:\>python taobao_login.py
Traceback (most recent call last):
  File "taobao_login.py", line 153, in <module>
    taobao.main()
  File "taobao_login.py", line 135, in main
    needResult = self.needIdenCode()
  File "taobao_login.py", line 94, in needIdenCode
    response = self.opener.open(request)
  File "c:\Python27\lib\urllib2.py", line 431, in open
    response = self._open(req, data)
  File "c:\Python27\lib\urllib2.py", line 449, in _open
    '_open', req)
  File "c:\Python27\lib\urllib2.py", line 409, in _call_chain
    result = func(*args)
  File "c:\Python27\lib\urllib2.py", line 1240, in https_open
    context=self._context)
  File "c:\Python27\lib\urllib2.py", line 1197, in do_open
    raise URLError(err)
urllib2.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate ve
rify failed (_ssl.c:581)>
```

然后会蹦出浏览器，显示了验证码的内容，这个需要你来手动输入。

在这里有小伙伴向我反映有这么个错误




经过查证，竟然是版本问题，博主本人用的是 2.7.7，而小伙伴用的是 2.7.9。后来换成 2.7.7 就好了…，我也是醉了，希望有相同错误的小伙伴，可以尝试换一下版本…

好啦，运行时会弹出浏览器，如图


那么，我们现在需要手动输入验证码，重新向登录界面发出登录请求，之前的 post 数据内容加入验证码这一项，重新请求一次，如果请求成功，则会返回下一步我们需要的 J_HToken，如果验证码输入错误，则会返回验证码输入错误的选项。好，下面，我已经写到了获取 J_HToken 的进度，代码如下，现在运行程序，会蹦出浏览器，然后提示你输入验证码，用户手动输入之后，则会返回一个页面，我们提取出 J_Htoken 即可。

注意，到现在为止，你还没有登录成功，只是获取到了 J_HToken 的值。

目前写到的代码如下

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import cookielib
import re
import webbrowser

\#模拟登录淘宝类
class Taobao:

    \#初始化方法
    def __init__(self):
        \#登录的URL
        self.loginURL = "https://login.taobao.com/member/login.jhtml"
        \#代理IP地址，防止自己的IP被封禁
        self.proxyURL = 'http://120.193.146.97:843'
```

```
\#登录POST数据时发送的头部信息
self.loginHeaders = {
    'Host':'login.taobao.com',
    'User-Agent' : 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0',
    'Referer' : 'https://login.taobao.com/member/login.jhtml',
    'Content-Type': 'application/x-www-form-urlencoded',
    'Connection' : 'Keep-Alive'
}
\#用户名
self.username = 'cqcre'
\#ua字符串，经过淘宝ua算法计算得出，包含了时间戳,浏览器,屏幕分辨率,随机数,鼠标移动,鼠标点击,其实还有键盘输入记录,鼠标
self.ua = '191UW5TcyMNYQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um5Ockt0TXdPc011TXVKdyE=|U2xMHDJ+H2QJZwl
\#密码，在这里不能输入真实密码，淘宝对此密码进行了加密处理，256位，此处为加密后的密码
self.password2 = '7511aa6854629e45de220d29174f1066537a73420ef6dbb5b46f202396703a2d56b0312df8769d886
self.post = post = {
    'ua':self.ua,
    'TPL_checkcode':'',
    'CtrlVersion': '1,0,0,7',
    'TPL_password':'',
    'TPL_redirect_url':'http://i.taobao.com/my_taobao.htm?nekot=udm8087E1424147022443',
    'TPL_username':self.username,
    'loginsite':'0',
    'newlogin':'0',
    'from':'tb',
    'fc':'default',
    'style':'default',
    'css_style':'',
    'tid':'XOR_1_000000000000000000000000000000_625C4720470A0A050976770A',
    'support':'000001',
    'loginType':'4',
    'minititle':'',
    'minipara':'',
    'umto':'NaN',
    'pstrong':'3',
    'llnick':'',
    'sign':'',
    'need_sign':'',
    'isIgnore':'',
    'full_redirect':'',
    'popid':'',
    'callback':'',
    'guf':'',
    'not_duplite_str':'',
    'need_user_id':'',
    'poy':'',
```

```
        'gvfdcname':'10',
        'gvfdcre':'',
        'from_encoding ':'',
        'sub':'',
        'TPL_password_2':self.password2,
        'loginASR':'1',
        'loginASRSuc':'1',
        'allp':'',
        'oslanguage':'zh-CN',
        'sr':'1366*768',
        'osVer':'windows|6.1',
        'naviVer':'firefox|35'
    }
    \#将POST的数据进行编码转换
    self.postData = urllib.urlencode(self.post)
    \#设置代理
    self.proxy = urllib2.ProxyHandler({'http':self.proxyURL})
    \#设置cookie
    self.cookie = cookielib.LWPCookieJar()
    \#设置cookie处理器
    self.cookieHandler = urllib2.HTTPCookieProcessor(self.cookie)
    \#设置登录时用到的opener，它的open方法相当于urllib2.urlopen
    self.opener = urllib2.build_opener(self.cookieHandler,self.proxy,urllib2.HTTPHandler)


\#得到是否需要输入验证码，这次请求的相应有时会不同，有时需要验证有时不需要
def needCheckCode(self):
    \#第一次登录获取验证码尝试，构建request
    request = urllib2.Request(self.loginURL,self.postData,self.loginHeaders)
    \#得到第一次登录尝试的相应
    response = self.opener.open(request)
    \#获取其中的内容
    content = response.read().decode('gbk')
    \#获取状态吗
    status = response.getcode()
    \#状态码为200，获取成功
    if status == 200:
        print u"获取请求成功"
        \#\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801这六个字是请输入验证码的utf-8编码
        pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801',re.S)
        result = re.search(pattern,content)
        print content
        \#如果找到该字符，代表需要输入验证码
        if result:
            print u"此次安全验证异常，您需要输入验证码"
```

```
            return content
        \#否则不需要
        else:
            \#返回结果直接带有J_HToken字样，表明直接验证通过
            tokenPattern = re.compile('id="J_HToken"')
            tokenMatch = re.search(tokenPattern,content)
            if tokenMatch:
                print u"此次安全验证通过，您这次不需要输入验证码"
                return False
    else:
        print u"获取请求失败"
        return None

\#得到验证码图片
def getCheckCode(self,page):
    \#得到验证码的图片
    pattern = re.compile('<img id="J_StandardCode_m.*?data-src="(.*?)"',re.S)
    \#匹配的结果
    matchResult = re.search(pattern,page)
    \#已经匹配得到内容，并且验证码图片链接不为空
    if matchResult and matchResult.group(1):
        print matchResult.group(1)
        return matchResult.group(1)
    else:
        print u"没有找到验证码内容"
        return False


\#输入验证码，重新请求，如果验证成功，则返回J_HToken
def loginWithCheckCode(self):
    \#提示用户输入验证码
    checkcode = raw_input('请输入验证码:')
    \#将验证码重新添加到post的数据中
    self.post['TPL_checkcode'] = checkcode
    \#对post数据重新进行编码
    self.postData = urllib.urlencode(self.post)
    try:
        \#再次构建请求，加入验证码之后的第二次登录尝试
        request = urllib2.Request(self.loginURL,self.postData,self.loginHeaders)
        \#得到第一次登录尝试的相应
        response = self.opener.open(request)
        \#获取其中的内容
        content = response.read().decode('gbk')
        \#检测验证码错误的正则表达式，\u9a8c\u8bc1\u7801\u9519\u8bef 是验证码错误五个字的编码
        pattern = re.compile(u'\u9a8c\u8bc1\u7801\u9519\u8bef',re.S)
```

```
        result = re.search(pattern,content)
        \#如果返回页面包括了，验证码错误五个字
        if result:
            print u"验证码输入错误"
            return False
        else:
            \#返回结果直接带有J_HToken字样，说明验证码输入成功，成功跳转到了获取HToken的界面
            tokenPattern = re.compile('id="J_HToken" value="(.*?)"')
            tokenMatch = re.search(tokenPattern,content)
            \#如果匹配成功，找到了J_HToken
            if tokenMatch:
                print u"验证码输入正确"
                print tokenMatch.group(1)
                return tokenMatch.group(1)
            else:
                \#匹配失败，J_Token获取失败
                print u"J_Token获取失败"
                return False
    except urllib2.HTTPError, e:
        print u"连接服务器出错，错误原因",e.reason
        return False

\#程序运行主干
def main(self):
    \#是否需要验证码，是则得到页面内容，不是则返回False
    needResult = self.needCheckCode()
    \#请求获取失败，得到的结果是None
    if not needResult ==None:
        if not needResult == False:
            print u"您需要手动输入验证码"
            idenCode = self.getCheckCode(needResult)
            \#得到了验证码的链接
            if not idenCode == False:
                print u"验证码获取成功"
                print u"请在浏览器中输入您看到的验证码"
                webbrowser.open_new_tab(idenCode)
                J_HToken = self.loginWithCheckCode()
                print "J_HToken",J_HToken
            \#验证码链接为空，无效验证码
            else:
                print u"验证码获取失败，请重试"
        else:
            print u"不需要输入验证码"
    else:
        print u"请求登录页面失败，无法确认是否需要验证码"
```

```
taobao = Taobao()
taobao.main()
```

现在的运行结果是这样的，我们已经可以得到 J_HToken 了，离成功又迈进了一步。


好，到现在为止，我们应该可以获取到 J_HToken 的值啦。

# #

利用 J_HToken 获取 st

st 也是一个经计算得到的 code，可以这么理解，st 是淘宝后台利用 J_HToken 以及其他数据经过计算之后得到的，可以利用 st 和用户名直接用 get 方式登录，所以 st 可以理解为一个秘钥。这个 st 值只会使用一次，如果第二次用 get 方式登录则会失效。所以它是一次性使用的。

下面 J_HToken 计算 st 的方法如下

```
\#通过token获得st
def getSTbyToken(self,token):
    tokenURL = 'https://passport.alipay.com/mini_apply_st.js?site=0&token=%s&callback=stCallback6' % token
    request = urllib2.Request(tokenURL)
    response = urllib2.urlopen(request)
    \#处理st，获得用户淘宝主页的登录地址
    pattern = re.compile('{"st":"(.*?)"}',re.S)
    result = re.search(pattern,response.read())
    \#如果成功匹配
    if result:
        print u"成功获取st码"
        \#获取st的值
        st = result.group(1)
        return st
    else:
        print u"未匹配到st"
        return False
```

# #

直接利用 st 登录

得到 st 之后，基本上就大功告成啦，一段辛苦终于没有白费，你可以直接构建 get 方式请求的URL，直接访问这个URL便可以实现登录。

```
stURL = 'https://login.taobao.com/member/vst.htm?st=%s&TPL_username=%s' % (st,username)
```

比如

```
https://login.taobao.com/member/vst.htm?st=1uynJELa4hKfsfWU3OjPJCw&TPL_username=cqcre
```

直接访问该链接即可实现登录，不过我这个应该已经失效了吧~

代码在这先不贴了，剩下的一起贴了~

# #

获取已买到的宝贝页面

已买到的宝贝的页面地址是

```
http://buyer.trade.taobao.com/trade/itemlist/list_bought_items.htm
```

另外还有页码的参数。

重新构建一个带有 cookie的opener，将上面的带有 st 的 URL 打开，保存它的 cookie，然后再利用这个 opener 打开已买到的宝贝的页面，你就会得到已买到的宝贝页面详情了。

```
\#获得已买到的宝贝页面
def getGoodsPage(self,pageIndex):
    goodsURL = 'http://buyer.trade.taobao.com/trade/itemlist/listBoughtItems.htm?action=itemlist/QueryAction&event_sub
    response = self.newOpener.open(goodsURL)
    page =  response.read().decode('gbk')
    return page
```

正则表达式提取信息

这是我的已买到的宝贝界面，审查元素可以看到，每一个宝贝都是 tbody 标签包围着。

```
D:\python2.7.7\Lib>python --version
Python 2.7.7
```

我们现在想获取订单时间，订单号，卖家店铺名称，宝贝名称，原价，购买数量，最后付款多少，交易状态这几个量，具体就不再分析啦，正则表达式还不熟悉的同学请参考前面所说的正则表达式的用法，在这里，正则表达式匹配的代码是

```
\#u'\u8ba2\u5355\u53f7'是订单号的编码
pattern = re.compile(u'dealtime.*?>(.*?)</span>.*?\u8ba2\u5355\u53f7.*?<em>(.*?)</em>.*?shopname.*?title="(.*?)".*?b
                u'price.*?title="(.*?)".*?quantity.*?title="(.*?)".*?amount.*?em.*?>(.*?)</em>.*?trade-status.*?<a.*?>(.*?)</a>
result = re.findall(pattern,page)
for item in result:
    print '----------------------------------------------------------------'
    print "购买日期:",item[0].strip(), '订单号:',item[1].strip(),'卖家店铺:',item[2].strip()
    print '宝贝名称:',item[3].strip()
    print '原价:',item[4].strip(),'购买数量:',item[5].strip(),'实际支付:',item[6].strip(),'交易状态',item[7].strip()
```

# #

最终代码整理

恩，你懂得，最重要的东西来了，经过博主2天多的奋战，代码基本就构建完成。写了两个类，其中提取页面信息的方法我单独放到了一个类中，叫 tool.py，类名为 Tool。

先看一下运行结果吧~

```
>>> import win32com
>>> _
```

最终代码如下

tool.py

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import re

\#处理获得的宝贝页面
class Tool:

    \#初始化
    def __init__(self):
        pass


    \#获得页码数
    def getPageNum(self,page):
        pattern = re.compile(u'<div class="total">.*?\u5171(.*?)\u9875',re.S)
        result = re.search(pattern,page)
        if result:
            print "找到了共多少页"
            pageNum = result.group(1).strip()
            print '共',pageNum,'页'
            return pageNum

    def getGoodsInfo(self,page):
        \#u'\u8ba2\u5355\u53f7'是订单号的编码
        pattern = re.compile(u'dealtime.*?>(.*?)</span>.*?\u8ba2\u5355\u53f7.*?<em>(.*?)</em>.*?shopname.*?title="(.*?)"
                             u'price.*?title="(.*?)".*?quantity.*?title="(.*?)".*?amount.*?em.*?>(.*?)</em>.*?trade-status.*?<a.*?>(.*?)
        result = re.findall(pattern,page)
```

```
        for item in result:
            print '----------------------------------------------------------------'
            print "购买日期:",item[0].strip(), '订单号:',item[1].strip(),'卖家店铺:',item[2].strip()
            print '宝贝名称:',item[3].strip()
            print '原价:',item[4].strip(),'购买数量:',item[5].strip(),'实际支付:',item[6].strip(),'交易状态',item[7].strip()
```

taobao.py

```
__author__ = 'CQC'
\# -*- coding:utf-8 -*-

import urllib
import urllib2
import cookielib
import re
import webbrowser
import tool

\#模拟登录淘宝类
class Taobao:

    \#初始化方法
    def __init__(self):
        \#登录的URL
        self.loginURL = "https://login.taobao.com/member/login.jhtml"
        \#代理IP地址，防止自己的IP被封禁
        self.proxyURL = 'http://120.193.146.97:843'
        \#登录POST数据时发送的头部信息
        self.loginHeaders = {
            'Host':'login.taobao.com',
            'User-Agent' : 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0',
            'Referer' : 'https://login.taobao.com/member/login.jhtml',
            'Content-Type': 'application/x-www-form-urlencoded',
            'Connection' : 'Keep-Alive'
        }
        \#用户名
        self.username = 'cqcre'
        \#ua字符串，经过淘宝ua算法计算得出，包含了时间戳,浏览器,屏幕分辨率,随机数,鼠标移动,鼠标点击,其实还有键盘输入记录,鼠标
        self.ua = '191UW5TcyMNYQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um5Ockt0TXdPc011TXVKdyE=|U2xMHDJ+H2QJZwl
        #密码，在这里不能输入真实密码，淘宝对此密码进行了加密处理，256位，此处为加密后的密码
        self.password2 = '7511aa6854629e45de220d29174f1066537a73420ef6dbb5b46f202396703a2d56b0312df8769d886
        self.post = post = {
            'ua':self.ua,
            'TPL_checkcode':'',
            'CtrlVersion': '1,0,0,7',
            'TPL_password':'',
```

```
    'TPL_redirect_url':'http://i.taobao.com/my_taobao.htm?nekot=udm8087E1424147022443',
    'TPL_username':self.username,
    'loginsite':'0',
    'newlogin':'0',
    'from':'tb',
    'fc':'default',
    'style':'default',
    'css_style':'',
    'tid':'XOR_1_000000000000000000000000000000_625C4720470A0A050976770A',
    'support':'000001',
    'loginType':'4',
    'minititle':'',
    'minipara':'',
    'umto':'NaN',
    'pstrong':'3',
    'llnick':'',
    'sign':'',
    'need_sign':'',
    'islgnore':'',
    'full_redirect':'',
    'popid':'',
    'callback':'',
    'guf':'',
    'not_duplite_str':'',
    'need_user_id':'',
    'poy':'',
    'gvfdcname':'10',
    'gvfdcre':'',
    'from_encoding ':'',
    'sub':'',
    'TPL_password_2':self.password2,
    'loginASR':'1',
    'loginASRSuc':'1',
    'allp':'',
    'oslanguage':'zh-CN',
    'sr':'1366*768',
    'osVer':'windows|6.1',
    'naviVer':'firefox|35'
}
\#将POST的数据进行编码转换
self.postData = urllib.urlencode(self.post)
\#设置代理
self.proxy = urllib2.ProxyHandler({'http':self.proxyURL})
\#设置cookie
self.cookie = cookielib.LWPCookieJar()
```

```
    \#设置cookie处理器
    self.cookieHandler = urllib2.HTTPCookieProcessor(self.cookie)
    \#设置登录时用到的opener，它的open方法相当于urllib2.urlopen
    self.opener = urllib2.build_opener(self.cookieHandler,self.proxy,urllib2.HTTPHandler)
    \#赋值J_HToken
    self.J_HToken = ''
    \#登录成功时，需要的Cookie
    self.newCookie = cookielib.CookieJar()
    \#登陆成功时，需要的一个新的opener
    self.newOpener = urllib2.build_opener(urllib2.HTTPCookieProcessor(self.newCookie))
    \#引入工具类
    self.tool = tool.Tool()


\#得到是否需要输入验证码，这次请求的相应有时会不同，有时需要验证有时不需要
def needCheckCode(self):
    \#第一次登录获取验证码尝试，构建request
    request = urllib2.Request(self.loginURL,self.postData,self.loginHeaders)
    \#得到第一次登录尝试的相应
    response = self.opener.open(request)
    \#获取其中的内容
    content = response.read().decode('gbk')
    \#获取状态吗
    status = response.getcode()
    \#状态码为200，获取成功
    if status == 200:
        print u"获取请求成功"
        \#\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801这六个字是请输入验证码的utf-8编码
        pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801',re.S)
        result = re.search(pattern,content)
        \#如果找到该字符，代表需要输入验证码
        if result:
            print u"此次安全验证异常，您需要输入验证码"
            return content
        \#否则不需要
        else:
            \#返回结果直接带有J_HToken字样，表明直接验证通过
            tokenPattern = re.compile('id="J_HToken" value="(.*?)"')
            tokenMatch = re.search(tokenPattern,content)
            if tokenMatch:
                self.J_HToken = tokenMatch.group(1)
                print u"此次安全验证通过，您这次不需要输入验证码"
                return False
    else:
        print u"获取请求失败"
```

```
        return None

\#得到验证码图片
def getCheckCode(self,page):
    #得到验证码的图片
    pattern = re.compile('<img id="J_StandardCode_m.*?data-src="(.*?)"',re.S)
    \#匹配的结果
    matchResult = re.search(pattern,page)
    \#已经匹配得到内容，并且验证码图片链接不为空
    if matchResult and matchResult.group(1):
        return matchResult.group(1)
    else:
        print u"没有找到验证码内容"
        return False


\#输入验证码，重新请求，如果验证成功，则返回J_HToken
def loginWithCheckCode(self):
    \#提示用户输入验证码
    checkcode = raw_input('请输入验证码:')
    \#将验证码重新添加到post的数据中
    self.post['TPL_checkcode'] = checkcode
    \#对post数据重新进行编码
    self.postData = urllib.urlencode(self.post)
    try:
        \#再次构建请求，加入验证码之后的第二次登录尝试
        request = urllib2.Request(self.loginURL,self.postData,self.loginHeaders)
        \#得到第一次登录尝试的相应
        response = self.opener.open(request)
        \#获取其中的内容
        content = response.read().decode('gbk')
        \#检测验证码错误的正则表达式，\u9a8c\u8bc1\u7801\u9519\u8bef 是验证码错误五个字的编码
        pattern = re.compile(u'\u9a8c\u8bc1\u7801\u9519\u8bef',re.S)
        result = re.search(pattern,content)
        \#如果返回页面包括了，验证码错误五个字
        if result:
            print u"验证码输入错误"
            return False
        else:
            \#返回结果直接带有J_HToken字样，说明验证码输入成功，成功跳转到了获取HToken的界面
            tokenPattern = re.compile('id="J_HToken" value="(.*?)"')
            tokenMatch = re.search(tokenPattern,content)
            \#如果匹配成功，找到了J_HToken
            if tokenMatch:
                print u"验证码输入正确"
```

```
            self.J_HToken = tokenMatch.group(1)
            return tokenMatch.group(1)
        else:
            \#匹配失败，J_Token获取失败
            print u"J_Token获取失败"
            return False
    except urllib2.HTTPError, e:
        print u"连接服务器出错，错误原因",e.reason
        return False


\#通过token获得st
def getSTbyToken(self,token):
    tokenURL = 'https://passport.alipay.com/mini_apply_st.js?site=0&token=%s&callback=stCallback6' % token
    request = urllib2.Request(tokenURL)
    response = urllib2.urlopen(request)
    \#处理st，获得用户淘宝主页的登录地址
    pattern = re.compile('{"st":"(.*?)"}',re.S)
    result = re.search(pattern,response.read())
    \#如果成功匹配
    if result:
        print u"成功获取st码"
        \#获取st的值
        st = result.group(1)
        return st
    else:
        print u"未匹配到st"
        return False

\#利用st码进行登录,获取重定向网址
def loginByST(self,st,username):
    stURL = 'https://login.taobao.com/member/vst.htm?st=%s&TPL_username=%s' % (st,username)
    headers = {
        'User-Agent' : 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0',
        'Host':'login.taobao.com',
        'Connection' : 'Keep-Alive'
    }
    request = urllib2.Request(stURL,headers = headers)
    response = self.newOpener.open(request)
    content =  response.read().decode('gbk')
    \#检测结果，看是否登录成功
    pattern = re.compile('top.location = "(.*?)"',re.S)
    match = re.search(pattern,content)
    if match:
        print u"登录网址成功"
```

```
            location = match.group(1)
            return True
        else:
            print "登录失败"
            return False


    \#获得已买到的宝贝页面
    def getGoodsPage(self,pageIndex):
        goodsURL = 'http://buyer.trade.taobao.com/trade/itemlist/listBoughtItems.htm?action=itemlist/QueryAction&event_s
        response = self.newOpener.open(goodsURL)
        page =  response.read().decode('gbk')
        return page

    \#获取所有已买到的宝贝信息
    def getAllGoods(self,pageNum):
        print u"获取到的商品列表如下"
        for x in range(1,int(pageNum)+1):
            page = self.getGoodsPage(x)
            self.tool.getGoodsInfo(page)



    \#程序运行主干
    def main(self):
        \#是否需要验证码，是则得到页面内容，不是则返回False
        needResult = self.needCheckCode()
        \#请求获取失败，得到的结果是None
        if not needResult ==None:
            if not needResult == False:
                print u"您需要手动输入验证码"
                checkCode = self.getCheckCode(needResult)
                \#得到了验证码的链接
                if not checkCode == False:
                    print u"验证码获取成功"
                    print u"请在浏览器中输入您看到的验证码"
                    webbrowser.open_new_tab(checkCode)
                    self.loginWithCheckCode()
                \#验证码链接为空，无效验证码
                else:
                    print u"验证码获取失败，请重试"
            else:
                print u"不需要输入验证码"
        else:
            print u"请求登录页面失败，无法确认是否需要验证码"
```

```
        \#判断token是否正常获取到
        if not self.J_HToken:
            print "获取Token失败，请重试"
            return
        \#获取st码
        st = self.getSTbyToken(self.J_HToken)
        \#利用st进行登录
        result = self.loginByST(st,self.username)
        if result:
            #获得所有宝贝的页面
            page = self.getGoodsPage(1)
            pageNum = self.tool.getPageNum(page)
            self.getAllGoods(pageNum)
        else:
            print u"登录失败"



taobao = Taobao()
taobao.main()
```

好啦，运行结果就是上面贴的图片，可以成功获取到自己的商品列表，前提是把你们的 用户名，ua，password 2 这三个设置好。

14

# 爬虫框架 Scrapy 安装配置

初级的爬虫我们利用 urllib 和 urllib2 库以及正则表达式就可以完成了，不过还有更加强大的工具，爬虫框架 Scrapy，这安装过程也是煞费苦心哪，在此整理如下。

# #

Windows 平台

我的系统是 Win7，首先，你要有 Python，我用的是2.7.7版本，Python3 相仿，只是一些源文件不同。

官网文档：http://doc.scrapy.org/en/latest/intro/install.html，最权威哒，下面是我的亲身体验过程。

## #

安装 Python

安装过程我就不多说啦，我的电脑中已经安装了 Python 2.7.7 版本啦，安装完之后记得配置环境变量，比如我的安装在 D 盘，D:\python2.7.7，就把以下两个路径添加到 Path 变量中

```
D:\python2.7.7;D:\python2.7.7\Scripts
```

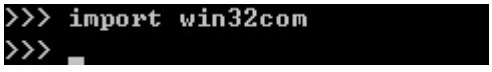
配置好了之后，在命令行中输入 python –version，如果没有提示错误，则安装成功


## #

安装 pywin32

在 windows下，必须安装 pywin32，安装地址：http://sourceforge.net/projects/pywin32/

下载对应版本的 pywin32，直接双击安装即可，安装完毕之后验证：




在 python 命令行下输入

import win32com

如果没有提示错误，则证明安装成功

## #

安装 pip

pip 是用来安装其他必要包的**工具**，首先下载 get-pip.py

下载好之后，选中该文件所在路径，执行下面的命令

```
python get-pip.py
```

执行命令后便会安装好 pip，并且同时，它帮你安装了 setuptools

安装完了之后在命令行中执行

```
pip --version
```

如果提示如下，说明就安装成功了，如果提示不是内部或外部命令，那么就检查一下**环境变量**有没有配置好吧，有两个路径。


## #

安装 pyOPENSSL

在 Windows下，是没有预装 pyOPENSSL 的，而在 Linux 下是已经安装好的。

安装地址：https://launchpad.net/pyopenssl

## #

安装 xml

lxml 的详细介绍 点我 ，是一种使用 Python 编写的库，可以迅速、灵活地处理 XML

直接执行如下命令

```
pip install lxml
```

就可完成安装，如果提示 Microsoft Visual C++ 库没安装，则 点我 下载支持的库。

## #

安装 Scrapy

最后就是激动人心的时刻啦，上面的铺垫做好了，我们终于可以享受到胜利的果实啦！

执行如下命令

```
pip install Scrapy
```

```
D:\python2.7.7\Lib>pip install Scrapy
Collecting Scrapy
  Using cached Scrapy-0.24.4-py2-none-any.whl
Collecting cssselect>=0.9 (from Scrapy)
```

pip 会另外下载其他依赖的包，这些就不要我们手动安装啦，等待一会，大功告成！

## #

验证安装

输入 Scrapy

如果提示如下命令，就证明安装成功啦，如果失败了，请检查上述步骤有何疏漏。

```
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
```

# #

Linux Ubuntu 平台： ##

Linux 下安装非常简单，只需要执行几条命令几个

## #

安装 Python

```
sudo apt-get install python2.7 python2.7-dev
```

## #

安装 pip

首先下载 get-pip.py

下载好之后，选中该文件所在路径，执行下面的命令

```
sudo python get-pip.py
```

## #

直接安装 Scrapy

由于 Linux下已经预装了 lxml 和 OPENSSL

如果想验证 lxml ，可以分别输入

```
sudo pip install lxml
```

出现下面的提示这证明已经安装成功

```
Requirement already satisfied (use --upgrade to upgrade): lxml in /usr/lib/python2.7/dist-packages
```

如果想验证 openssl，则直接输入 openssl 即可，如果跳转到 OPENSSL 命令行，则安装成功。

接下来直接安装 Scrapy 即可

```
sudo pip install Scrapy
```

安装完毕之后，输入 scrapy

注意，这里linux下不要输入 Scrapy，linux 依然严格区分大小写的，感谢 kamen 童鞋提醒。

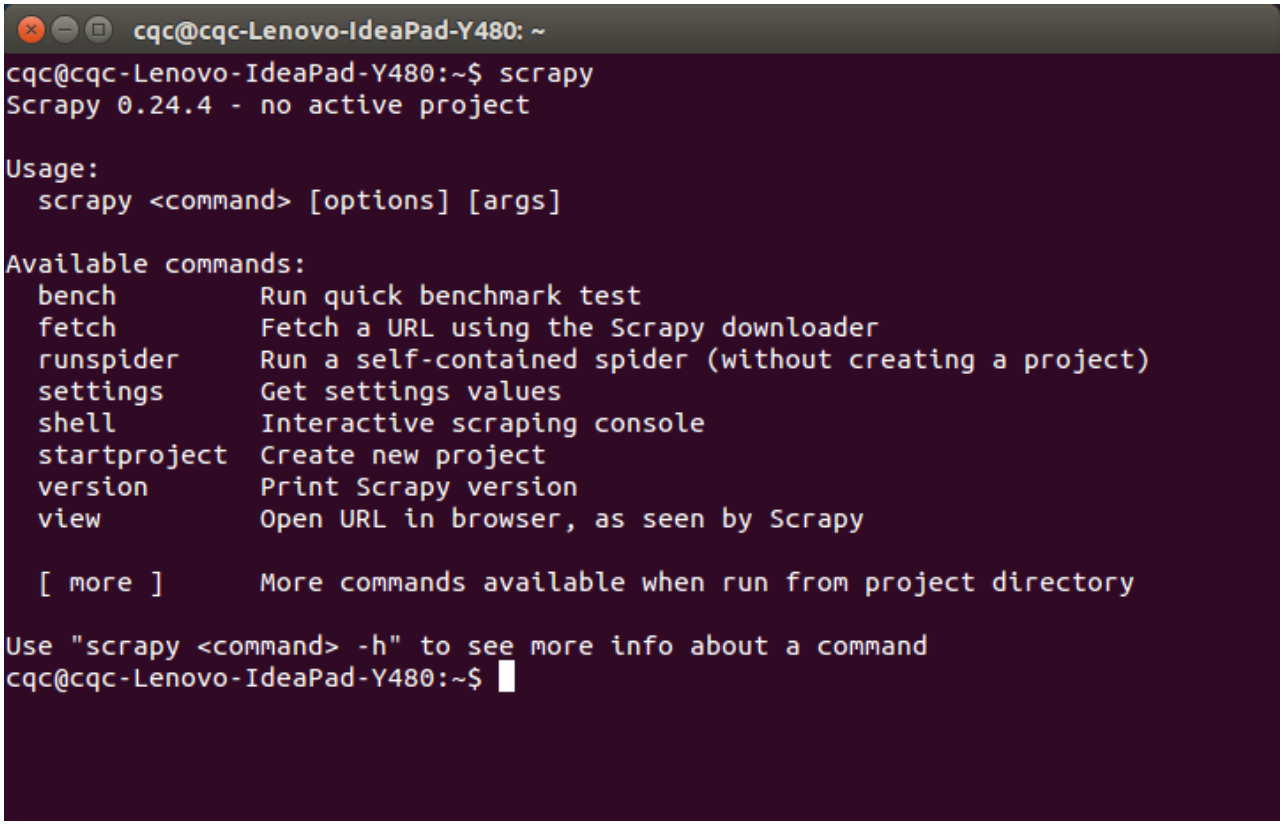
如果出现如下提示，这证明安装成功

```
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
```

截图如下



如有问题，欢迎留言！祝各位小伙伴顺利安装！

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-guide/