

Table of Contents

Python爬虫学习系列教程	1.1
一、爬虫入门	1.2
1. Python爬虫入门一之综述	1.2.1
2. Python爬虫入门二之爬虫基础了解	1.2.2
3. Python爬虫入门三之Urllib库的基本使用	1.2.3
4. Python爬虫入门四之Urllib库的高级用法	1.2.4
5. Python爬虫入门五之URLError异常处理	1.2.5
6. Python爬虫入门六之Cookie的使用	1.2.6
7. Python爬虫入门七之正则表达式	1.2.7
二、爬虫实战	1.3
1. Python爬虫实战一之爬取糗事百科段子	1.3.1
2. Python爬虫实战二之爬取百度贴吧帖子	1.3.2
3. Python爬虫实战三之实现山东大学无线网络掉线自动重连	
4. Python爬虫实战四之抓取淘宝MM照片	1.3.4
5. Python爬虫实战五之模拟登录淘宝并获取所有订单	1.3.5
6. Python爬虫实战六之抓取爱问知识人问题并保存至数据库	
7. Python爬虫实战七之计算大学本学期绩点	1.3.7
8. Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺	1.3.6
三、爬虫利器	1.3.8
1. Python爬虫利器一之Requests库的用法	1.4.1
2. Python爬虫利器二之Beautiful Soup的用法	1.4.2
3. Python爬虫利器三之Xpath语法与lxml库的用法	1.4.3
4. Python爬虫利器四之PhantomJS的用法	1.4.4
5. Python爬虫利器五之Selenium的用法	1.4.5
6. Python爬虫利器六之PyQuery的用法	1.4.6
四、爬虫进阶	1.5
1. Python爬虫进阶一之爬虫框架概述	1.5.1
2. Python爬虫进阶二之PySpider框架安装配置	1.5.2
3. Python爬虫进阶三之爬虫框架Scrapy安装配置	1.5.3
4. Python爬虫进阶四之PySpider的用法	1.5.4
5. Python爬虫进阶五之多线程的用法	1.5.5
6. Python爬虫进阶六之多进程的用法	1.5.6
7. Python爬虫进阶七之设置ADSL拨号服务器代理	1.5.7

Python爬虫学习系列教程

以下为Python2爬虫系列教程：

大家好哈，我呢最近在学习Python爬虫，感觉非常有意思，真的让生活可以方便很多。学习过程中我把一些学习的笔记总结下来，还记录了一些自己实际写的一些小爬虫，在这里跟大家一同分享，希望对Python爬虫感兴趣的童鞋有帮助，如果有机会期待与大家的交流。

Python版本： 2.7

**年度重磅大放送！博主录制的Python3
爬虫视频教程出炉啦！！！欢迎大家支
持！！！详情请看：**

[Python3爬虫视频学习教程](#)

[自己动手，丰衣足食！Python3网络爬虫实战案例](#)

以下为Python2爬虫系列教程：

大家好哈，我呢最近在学习Python爬虫，感觉非常有意思，真的让生活可以方便很多。学习过程中我把一些学习的笔记总结下来，还记录了一些自己实际写的一些小爬虫，在这里跟大家一同分享，希望对Python爬虫感兴趣的童鞋有帮助，如果有机会期待与大家的交流。

Python版本：2.7

一、爬虫入门

1. [Python爬虫入门一之综述](#)
2. [Python爬虫入门二之爬虫基础了解](#)
3. [Python爬虫入门三之Urllib库的基本使用](#)
4. [Python爬虫入门四之Urllib库的高级用法](#)
5. [Python爬虫入门五之URLError异常处理](#)
6. [Python爬虫入门六之Cookie的使用](#)
7. [Python爬虫入门七之正则表达式](#)

二、爬虫实战

1. [Python爬虫实战一之爬取糗事百科段子](#)
2. [Python爬虫实战二之爬取百度贴吧帖子](#)
3. [Python爬虫实战三之实现山东大学无线网络掉线自动重连](#)
4. [Python爬虫实战四之抓取淘宝MM照片](#)
5. [Python爬虫实战五之模拟登录淘宝并获取所有订单](#)
6. [Python爬虫实战六之抓取爱问知识人问题并保存至数据库](#)
7. [Python爬虫实战七之计算大学本学期绩点](#)

[8. Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺](#)

三、爬虫利器

- [1. Python爬虫利器一之Requests库的用法](#)
- [2. Python爬虫利器二之Beautiful Soup的用法](#)
- [3. Python爬虫利器三之Xpath语法与lxml库的用法](#)
- [4. Python爬虫利器四之PhantomJS的用法](#)
- [5. Python爬虫利器五之Selenium的用法](#)
- [6. Python爬虫利器六之PyQuery的用法](#)

四、爬虫进阶

- [1. Python爬虫进阶一之爬虫框架概述](#)
- [2. Python爬虫进阶二之PySpider框架安装配置](#)
- [3. Python爬虫进阶三之爬虫框架Scrapy安装配置](#)
- [4. Python爬虫进阶四之PySpider的用法](#)
- [5. Python爬虫进阶五之多线程的用法](#)
- [6. Python爬虫进阶六之多进程的用法](#)
- [7. Python爬虫进阶七之设置ADSL拨号服务器代理](#)

目前暂时是这些文章，随着学习的进行，会不断更新哒，敬请期待~

希望对大家有所帮助，谢谢！

转载请注明：[静觅](#) » [Python爬虫学习系列教程](#)

Python爬虫入门一之综述

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

1. 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。

1) 慕课网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：[慕课网Python教程](#)

2) 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](#)

3) 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](#)

4) 汪海的实验室

这是我的本科实验室学长，入门的时候参考的他的文章，自己重新做了总结，后来这些系列文章又在他的基础上增加了一些内容。

学习网址：[汪海的实验室](#)

2. 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。

3. 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。

4. 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 为您提供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!

扯了这么多, 好像没多少有用的东西额, 那就不扯啦!

下面开始我们正式进入爬虫之旅吧!

Python爬虫入门二之爬虫基础了解

1.什么是爬虫

爬虫，即网络爬虫，大家可以理解为在网络上爬行的一只蜘蛛，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，在这个网中它发现了一条道路，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。

2.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。

3.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。

4. 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

下一节，我们就正式步入 Python 爬虫学习的殿堂了，小伙伴准备好了嘛？

Python爬虫入门三之Urllib库的基本使用

那么接下来，小伙伴们就一起和我真正迈向我们的爬虫之路吧。

1. 分分钟扒一个网页下来

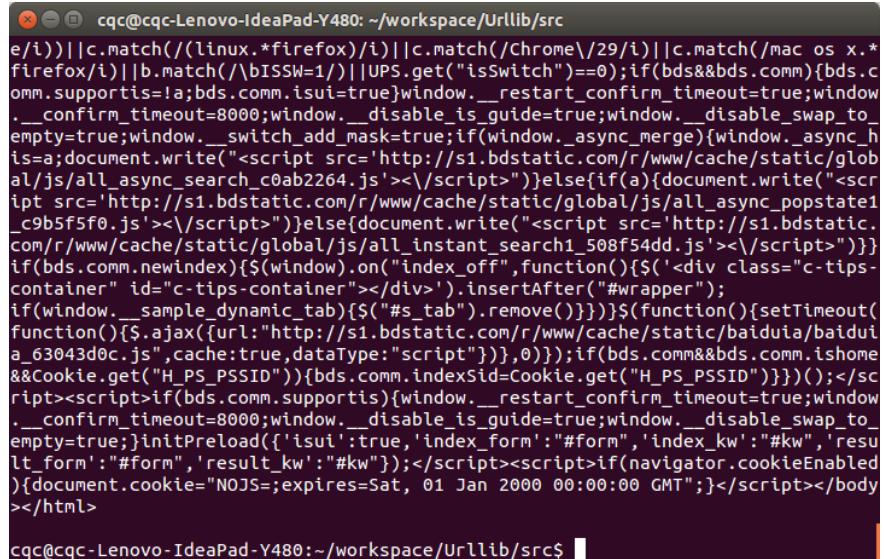
怎样扒网页呢？其实就是根据URL来获取它的网页信息，虽然我们在浏览器中看到的是一幅幅优美的画面，但是其实是由浏览器解释才呈现出来的，实质它是一段HTML代码，加JS、CSS，如果把网页比作一个人，那么HTML便是他的骨架，JS便是他的肌肉，CSS便是它的衣服。所以最重要的部分是存在于HTML中的，下面我们就写个例子来扒一个网页下来。

```
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
```



```
cqc@cqc-Lenovo-IdeaPad-Y480: ~/workspace/Urllib/src
<html><head></head><body><script>...</script></body></html>
```

看，这个网页的源码已经被我们扒下来了，是不是很酸爽？

2. 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 ob
```

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！

3.构造Request

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。

4. 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.cs
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**CSDN**还有个流水号的字段，没有设置全，比较复杂在这里就不写上去了，在此只是说明登录的原理。一般的登录网站一般是这种写法。

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，返回的便是**POST**后呈现的页面内容。

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"
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"
geturl = url + "?" + data
request = urllib2.Request(geturl)
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
```

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。

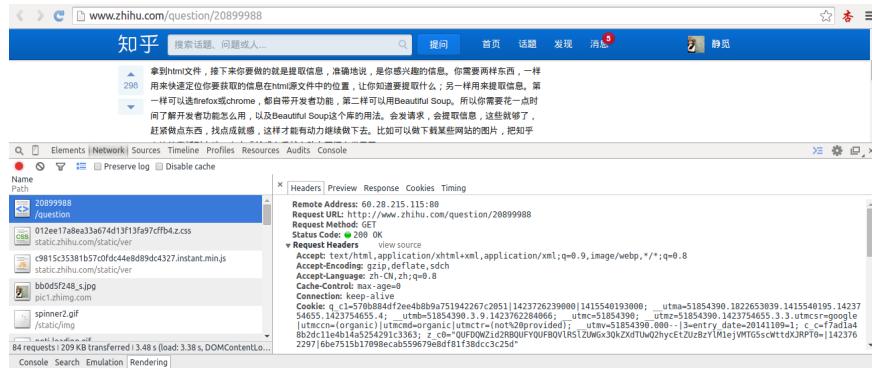
本节讲解了一些基本使用，可以抓取到一些基本的网页信息，小伙伴们加油！

Python爬虫入门四之Urllib库的高级用法

1. 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“防盗链”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; W
'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。

2. 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'})
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)
```

3.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)
```

4. 使用 HTTP 的 PUT 和 DELETE 方法

http协议有六种请求方法，get,head,put,delete,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 **httpplib** 库。虽然如此，我们还是能通过下面的方式，使 **urllib2** 能够发出 **PUT** 或 **DELETE** 的请求，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)
```

5. 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！

Python爬虫入门五之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。

1.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

request = urllib2.Request('http://www.xxxxxx.com')
try:
    urllib2.urlopen(request)
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

2.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好处的时候才应该采取类似措施。
- **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：若程序中能够处理，则进行进一步处理，如果程序中不能处理，则丢弃
- **301:** 请求到的资源都会分配一个永久的URL，这样就可以在将来通过该URL来访问此资源 处理方式：重定向到分配的URL
- **302:** 请求到的资源在一个不同的URL处临时保存 处理方式：重定向到临时的URL
- **304:** 请求的资源未更新 处理方式：丢弃
- **400:** 非法请求 处理方式：丢弃
- **401:** 未授权 处理方式：丢弃
- **403:** 禁止 处理方式：丢弃
- **404:** 没有找到 处理方式：丢弃
- **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的源代码出现错误时出现。
- **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**URLLError**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**URLLError**，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**URLLError**异常。如果发生的不是**HTTPError**，则会去捕获**URLLError**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,"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！

Python爬虫入门六之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。

1.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。

2.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 —> FileCookieJar —> MozillaCookieJar和LWPCookieJar

1) 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
```

2) 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)

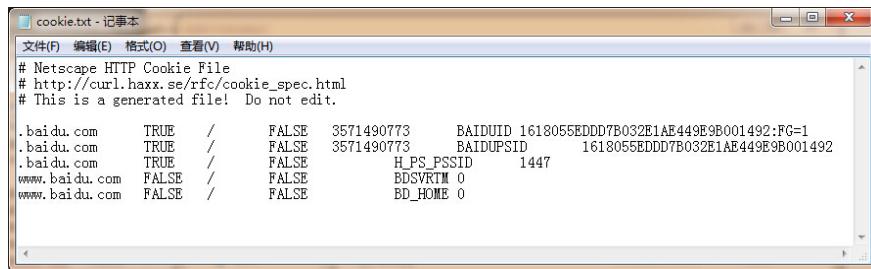
```

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



3) 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。

4) 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！

注意：密码我改了啊，别偷偷登录本宫的选课系统 o(╯□╰)o

```
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.lo
#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.cursc
#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！

Python爬虫入门七之正则表达式

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！

1.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同样也不例外，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。

2.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。在字符串集中如果要使用^、-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
\$	匹配字符串未尾。 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	仅匹配字符串未尾。	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}	abcabc a(123 456)c a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcabc 1abc1 5abc5
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造(不作为分组)			
(?:...)	(...)的不分组版本，用于使用' '或后接数量词。	(?:abc){2}	abcabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后面的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容不需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\\d)a	前面不是数字的a
(?(id/name)yes-pattern no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 no-pattern可以省略。	(\d)abc(?:1)\d abc	1abc2 abcabc

3. 正则表达式相关注解

(1) 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。

(2) 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。

4. 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
- **re.U**(全拼: **UNICODE**): 使预定字符类 \w \W \b \B \s \S \d \D 取决于当前区域
- **re.X**(全拼: **VERBOSE**): 详细模式。这个模式下正则表达式可以是多行, 忽略空白行

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。

(1) **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, 'heloo CQC!')
result3 = re.match(pattern, 'heoo 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heloo 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hele 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？下面我们就说一下关于match对象的属性和方法 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代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
- 2.groups([default]): 以元组形式返回全部分组截获的字符串。相当于调用group(1,2,...last)。default表示没有截获字符串的组以这个值替代，默认为None。
- 3.groupdict([default]): 返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
- 4.start([group]): 返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
- 5.end([group]): 返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。
- 6.span([group]): 返回(start(group), end(group))。
- 7.expand(template): 将匹配到的分组代入template中然后返回。template中可以使用\id或\g、\g引用分组，但不能使用编号0。\\id与\\g是等价的；但\\10将被认为是第10个分组，如果你想表达\\1之后是字符'0'，只能使用\\g0。

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<sign>.*')', '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!

```

(2) **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**是否是在**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
```

(3) **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', '']
```

(4) **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']
```

(5) **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
```

(6) **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\1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!
```

(7) **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)

```

5. 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[, fla
search(string[, pos[, endpos]]) | re.search(pattern, string[, f
split(string[, maxsplit]) | re.split(pattern, string[, maxsplit
findall(string[, pos[, endpos]]) | re.findall(pattern, string[, 
finditer(string[, pos[, endpos]]) | re.finditer(pattern, string
sub(repl, string[, count]) | re.sub(pattern, repl, string[, cou
subn(repl, string[, count]) |re.sub(pattern, repl, string[, cou

```

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](#)

**年度重磅大放送！博主录制的Python3
爬虫视频教程出炉啦！！！欢迎大家支
持！！！详情请看：**

[Python3爬虫视频学习教程](#)

[自己动手，丰衣足食！Python3网络爬虫实战案例](#)

以下为Python2爬虫系列教程：

大家好哈，我呢最近在学习Python爬虫，感觉非常有意思，真的让生活可以方便很多。学习过程中我把一些学习的笔记总结下来，还记录了一些自己实际写的一些小爬虫，在这里跟大家一同分享，希望对Python爬虫感兴趣的童鞋有帮助，如果有机会期待与大家的交流。

Python版本：2.7

一、爬虫入门

1. [Python爬虫入门一之综述](#)
2. [Python爬虫入门二之爬虫基础了解](#)
3. [Python爬虫入门三之Urllib库的基本使用](#)
4. [Python爬虫入门四之Urllib库的高级用法](#)
5. [Python爬虫入门五之URLError异常处理](#)
6. [Python爬虫入门六之Cookie的使用](#)
7. [Python爬虫入门七之正则表达式](#)

二、爬虫实战

1. [Python爬虫实战一之爬取糗事百科段子](#)
2. [Python爬虫实战二之爬取百度贴吧帖子](#)
3. [Python爬虫实战三之实现山东大学无线网络掉线自动重连](#)
4. [Python爬虫实战四之抓取淘宝MM照片](#)
5. [Python爬虫实战五之模拟登录淘宝并获取所有订单](#)
6. [Python爬虫实战六之抓取爱问知识人问题并保存至数据库](#)
7. [Python爬虫实战七之计算大学本学期绩点](#)

[8. Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺](#)

三、爬虫利器

- [1. Python爬虫利器一之Requests库的用法](#)
- [2. Python爬虫利器二之Beautiful Soup的用法](#)
- [3. Python爬虫利器三之Xpath语法与lxml库的用法](#)
- [4. Python爬虫利器四之PhantomJS的用法](#)
- [5. Python爬虫利器五之Selenium的用法](#)
- [6. Python爬虫利器六之PyQuery的用法](#)

四、爬虫进阶

- [1. Python爬虫进阶一之爬虫框架概述](#)
- [2. Python爬虫进阶二之PySpider框架安装配置](#)
- [3. Python爬虫进阶三之爬虫框架Scrapy安装配置](#)
- [4. Python爬虫进阶四之PySpider的用法](#)
- [5. Python爬虫进阶五之多线程的用法](#)
- [6. Python爬虫进阶六之多进程的用法](#)
- [7. Python爬虫进阶七之设置ADSL拨号服务器代理](#)

目前暂时是这些文章，随着学习的进行，会不断更新哒，敬请期待~

希望对大家有所帮助，谢谢！

转载请注明：[静觅](#) » [Python爬虫学习系列教程](#)

Python爬虫实战一之爬取糗事百科段子

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。

友情提示

糗事百科在前一段时间进行了改版，导致之前的代码没法用了，会导致无法输出和CPU占用过高的情况，是因为正则表达式没有匹配到的缘故。现在，博主已经对程序进行了重新修改，代码亲测可用，包括截图和说明，之前一直在忙所以没有及时更新，望大家海涵！更新时间：2015/8/2

糗事百科又又又改版了，博主已经没心再去一次次匹配它了，如果大家遇到长时间运行不出结果也不报错的情况，请大家参考最新的评论，热心小伙伴提供的正则来修改下吧～更新时间：

2016/3/27

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，每按一次回车我们显示一个段子。

1.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。

2. 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

```
<div class="single-clear"></div>
<div class="article-block untagged mb15" id="qiushi_tag_112073719">
  <div class="author">
    <a href="/user/15941497" target="_blank">
      
    <span>小胖墩~</span>
  </a>
</div>
<div class="content">
  <p>和闺蜜一起开车出去玩，路过一个加油站写着加油送礼，心里想着加油还能有小礼物，待加油完毕，问工作人员送什么小礼物，工作人员一听立马站直了给我敬了一个礼，哎呦卧槽，都别拦着我，让我开车撞死这货。。</p>
  <br/>
  <div>2015-08-02 08:46:56--></div>
<div class="stats">
  <span class="stats-vote">
    <i class="number">3268</i>
    <span>搞笑</span>
  </span>
  <span class="stats-comments"></span>
  <span class="stats-tag"></span>
</div>
<div id="qiushi_counts_112073719" class="stats-buttons-bar clearfix"></div>

```

我们可以看到，每一个段子都是 `<div class="article-block untagged mb15" id="...">...</div>` 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.*?author".*?<a.*?<img.*?>(.*?)</a>.*?
  <content>(.*?)<!--(.*?)-->.*?</div>(.*?
items = re.findall(pattern,content)
for item in items:
  print item[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，以后我们还会大量用到 `.*?` 的搭配。

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，`item[1]`就代表第二个 `(.*?)` 所指代的内容，以此类推。

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。

这样我们就获取了发布人，发布时间，发布内容，附加图片以及点赞数。

在这里注意一下，我们要获取的内容如果是带有图片，直接输出出来比较繁琐，所以这里我们只获取不带图片的段子就好了。

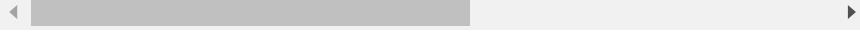
所以，在这里我们就需要对带图片的段子进行过滤。

我们可以发现，带有图片的段子会带有类似下面的代码，而不带图片的则没有，所以，我们的正则表达式的`item[3]`就是获取了下面的内容，如果不带图片，`item[3]`获取的内容便是空。

```
<div class="thumb">

<a href="/article/112061287?list=hot&s=4794990" target="_bla
.*?<a.*?<img.*?>(.*)</' +
                         'content">(.*)<!--(.*)-->.*?</div>(.*' +
                         items = re.findall(pattern,content)
    for item in items:
        haveImg = re.search("img",item[3])
        if not haveImg:
            print item[0],item[1],item[2],item[4]
except urllib2.URLError, e:
    if hasattr(e,"code"):
        print e.code
    if hasattr(e,"reason"):
        print e.reason

```



运行一下看下效果

蛊毒情调

跟爷爷聊天
爷爷您有记恨的人吗？
“没有”
您老心胸真宽广
“不是，因为他们都死了”
2015-08-02 05:10:12 9558

A禁小菜

刚考完驾照，老爸对我说，咱们去看车吧。顿时心里那个激动啊。你还别说，天桥上风还挺大的！
2015-08-02 07:22:52 6044

小胖墩~

和闺蜜一起开车出去玩，路过一个加油站写着加油送礼，心里想着加油还能有小礼物，待加油完毕，问工作人员送什么小礼物，工作人员一听立马站直了给我敬了一个
2015-08-02 08:46:56 5899

不期而遇、

第一次发视频，这娃太萌了。
2015-08-01 10:51:09 4005

ljjqhm

恩，带有图片的段子已经被剔除啦。是不是很开森？

3.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 5.0)'
        #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.*?author".*?<a.*?<img.*?>(.*)<content">(.*)<!--(.*)-->.*/></div>(.*)')
        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:
        replaceBR = re.compile('<br/>')
        text = re.sub(replaceBR,"\\n",item[1])
        #item[0]是一个段子的发布者，item[1]是内容，item[2]
        pageStories.append([item[0].strip(),text.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\t赞:%s\n%s" %(p

#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！

我们第一个爬虫实战项目介绍到这里，欢迎大家继续关注，小伙伴们加油！

Python爬虫实战二之爬取百度贴吧帖子

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。

前言

亲爱的们，教程比较旧了，百度贴吧页面可能改版，可能代码不好使，八成是正则表达式那儿匹配不到了，请更改一下正则，当然最主要的还是帮助大家理解思路。

2016/12/2

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

1.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

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

2.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。

3. 提取相关信息

1) 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
```

所以我们想提取 `<h1>` 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>')
    result = re.search(pattern,page)
    if result:
        #print result.group(1) #测试输出
        return result.group(1).strip()
    else:
        return None
```

2) 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>.*?<sp')
    result = re.search(pattern,page)
    if result:
        #print result.group(1) #测试输出
        return result.group(1).strip()
    else:
        return None
```

3) 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_.*?>(.*)</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>')
    items = re.findall(pattern,page)
    for item in items:
        print item
```

好，我们运行一下结果看一下

```
很多媒体都在每赛季之前给球员排个名，我也有这个癖好…………，我会尽量理性的分析球队地位，个人能力kit=gbk&ttn=SE\_hlfp00990\_u6vqbx10 cl
![BDE_Image](http://imagesrc.baidu.com/forum/yk30580/icon=cb6ab1f8708b4710c+e2ffdc4f3cc3b2/0581f30d924b89948+ca30e16c081d950b7bf671.jpg)

纯手打，更新数据，大家慢慢看

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/ae0a19d8bc3eb135cf7e70162a41ea8d9fcf44c3.jpg)

今天再更新一个就回家啦

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/613a20a46c30097f138012cb700aefc3d6cfaed0d86.jpg)

搞定今天更新完毕，准备下班了



晚上偷懒顶一个![BDE_Smiley](http://tb2.bdstatic.com/tb/editor/images/face/s_f26.png?i=20140602)开更今天的了！

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/20f41hd5ad0edd4c89ea8e213bhb6fd45396335e.jpg)
![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/b202d158ccbfb6c01e290a32ba3eb13532f4a0c6.jpg)
![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/c0379e0f6997b8c31088723429cf8db1ca137018.jpg)
![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/e7177f3e8708c93d40420248d3df8dc0005481.jpg)

顶一顶，难道写着成单机了？



吃饭去了 下午来吧



下午回来继续更新了

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/012eb09389b804fc2c3ddc5df7dde71191ef6d6d4.jpg)

上午的所再回来更新下一个

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/e7177f3e8708c93d40420248d3df8dc0005481.jpg)

再顶一下，写得好辛苦，别又变成单机了



话说现在除了一个人都没有来说离了，做了呢，好奇怪

![BDE_Image](http://imagesrc.baidu.com/forum/pic/item/15c79f3dt8dc4100fb179efb708b4710b8122f64.jpg)
```

真是醉了，还有一大片换行符和图片符，好可怕！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
    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>')
    #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):
        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(pageNu
            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.*?>(.*)')
        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
    def getPageNum(self):
        page = self.getPage(1)
        pattern = re.compile('<li class="l_reply_num.*?</span>.*')
        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_.*?>(.*)</d')
        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！

```

50 惊喜新人王 迈卡威
上赛季数据
篮板 6.2 助攻 6.3 抢断 1.9 盖帽 0.6 失误 3.5 犯规 3 得分 16.7

新赛季第50场，我台上赛季的新秀迈卡威。上赛季迈卡威在彻底重建的76人中迅速掌握了球队，一开始三双奠定了热火赢得了万千眼球。后来也是屡有经验的表现，新秀赛季就拿过三双的球员不多，作为上赛季领队的老大，迈卡威做出了不错的数据，但人们静下心来着他，还是发现他有很多问题。首先，投篮命中率刚刚40%的命中率和惨淡的20%三分命中率肯定不是合格的！加之身体瘦弱，个子高说不完缺点。来说说优点：作为后卫篮板球非常突出，高大的身性能较好的影响对方的出手，也能发现己方的空位球员。突破虽然速度一般，但节奏感不错，大局观也在平均水准之上。挑衅豪迈高大，不会长就球队地位而言，迈卡威现在是绝对的老大，球你想怎么玩就怎么玩，数据你想怎么刷就怎么刷！去年的潜力新人诺尔是盖领，其他人都可以清道夫，奥拉迪波还受伤不能打，76人队的战绩怎么样，就看你们了！

Process finished with exit code 0

```

4) 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('<div id="post_content_.*?>(.*)</div>')
    items = re.findall(pattern,page)

    floor = 1
    for item in items:
        print floor,u"楼-----"
        print self.tool.replace(item)
        floor += 1

```

运行一下看看效果

```

21 楼-----
41 麦迪35秒13分？我有28秒11分的 米尔斯普
上赛季数据
篮板 8.5 助攻 9.1 抢断 1.7 盖帽 1.1 失误 2.5 犯规 2.8 得分 17.9

大米群上赛季继续着自己低调而发光发热的表现，老鹰用他取代史密斯是太聪明的决定了，米尔斯除了快攻的时候跑得没史密斯快，几乎是完爆史密斯，入选全明星是对他的最大肯定！
米尔斯的进攻手段很丰富，虽然跑得不快，跳投一般，臂展也不行，但他的运动力强！米尔斯力量很好，打球速率很快！（这里解释一下，就是是变幻动作的速度，米尔斯的第一步转身的那一秒非常迅速，可以防守端，米尔斯受制于身体条件，对高举高打的打法没有办法。但是米尔斯作为内线场均约1.8的抢断显示着防守端下三路很优秀。防守端不如前任史密斯，但也不算太坏（起码比布鲁克好多了）。虽然是个攻击缺点嘛，米尔斯天赋是硬伤，作为一个纯末尾的球员，打成现在这个样子已经是他的上限了，对比易建联，他靠着自身的努力和教练打出了全明星，但是可惜，下赛季霍福德归来依然是老鹰老大，米尔斯普
22 楼-----
再顶一页，写得好辛苦，别又变成单机了
23 楼-----
话你现在除了一个人都没人来说高了，低了呢，好奇怪
24 楼-----
蓝下终结机 小乔丹
上赛季数据
篮板 12.6 助攻 0.9 抢断 1 盖帽 2.5 失误 1.5 犯规 3.2 得分 10.4
在里佛斯倒了快船后，小乔丹获得了新生，从一个无脸逆袭人+暴扣男，变成了一个防守覆盖整个分区区域，进攻终结能力极强的真男人。他现在是名符其实的快船第三巨头。
上赛季小乔丹内线数据完胜全联盟，1场MVP 13.6的篮板球，2是他的11.6的命中率，你说投得少命中率高就不说了，但这家伙本赛季得分创下了生涯新高，第一次上双，命中率还越来越高了（庄神哭晕在防守端的小乔丹就更恐怖了，光看数据就知道，篮板王+盖帽第三，还有些数据无法体现的，比如脚步灵活高臂展，防守效果很好。速度很快，协防潜力很大，经常飞人的冒，加之篮球智商的进步还差说缺点，首先那可怜的罚球命中率也不提了，进攻端还是完全没有自主进攻的能力，作为高大中锋拍下后场篮板后第一时间一传能力也不强（好吧我要求过高了）。防守端太喜欢协防，有时候会犯总来说，小乔丹用自己的表现来回报快船给他的大合同，快船三叉戟有望在未来1-2年内打出自己最佳的表现。
25 楼-----

```

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

4. 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。

5. 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('<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
    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,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(pageNu
            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('<h1 class="core_title_txt.*?>(.*)')
        result = re.search(pattern,page)
        if result:
            #如果存在，则返回标题
            return result.group(1).strip()
        else:
            return None

        #获取帖子一共有多少页
    def getPageNum(self,page):
        #获取帖子页数的正则表达式
        pattern = re.compile('<li class="l_reply_num.*?</span>.*')
        result = re.search(pattern,page)
        if result:
            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):
        #匹配所有楼层的内容
        pattern = re.compile('<div id="post_content_.*?>(.*)</d
        items = re.findall(pattern,page)
        contents = []

```

```

        for item in items:
            #将文本进行去除标签处理，同时在前后加入换行符
            content = "\n"+self.tool.replace(item)+"\n"
            contents.append(content.encode('utf-8'))
        return contents

    def setFileTitle(self,title):
        #如果标题不是为None，即成功获取到标题
        if title is not None:
            self.file = open(title + ".txt","w+")
        else:
            self.file = open(self.defaultTitle + ".txt","w+")

    def writeData(self,contents):
        #向文件写入每一楼的信息
        for item in contents:
            if self.floorTag == '1':
                #楼之间的分隔符
                floorLine = "\n" + str(self.floor) + u"-----"
                self.file.write(floorLine)
            self.file.write(item)
            self.floor += 1

    def start(self):
        indexPage = self.getPage(1)
        pageNum = self.getPageNum(indexPage)
        title = self.getTitle(indexPage)
        self.setFileTitle(title)
        if pageNum == None:
            print "URL已失效，请重试"
            return
        try:
            print "该帖子共有" + str(pageNum) + "页"
            for i in range(1,int(pageNum)+1):
                print "正在写入第" + str(i) + "页数据"
                page = self.getPage(i)
                contents = self.getContent(page)
                self.writeData(contents)
            #出现写入异常
        except IOError,e:
            print "写入异常，原因" + e.message
        finally:
            print "写入任务完成"

print u"请输入帖子代号"
baseURL = 'http://tieba.baidu.com/p/' + str(raw_input(u'http://t

```

```
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！

Python爬虫实战三之实现山东大学无线网络掉线自动重连

综述

最近山大软件园校区QLSC_STU无线网掉线掉的厉害，连上之后平均十分钟左右掉线一次，很是让人心烦，还能不能愉快地上自习了？能忍吗？反正我是不能忍了，嗯，自己动手，丰衣足食！写个程序解决掉它！

假若你不能连这个无线，那就照照思路啦～

决战前夕

首先我们看一下那个验证页面是咋样滴，上个图先

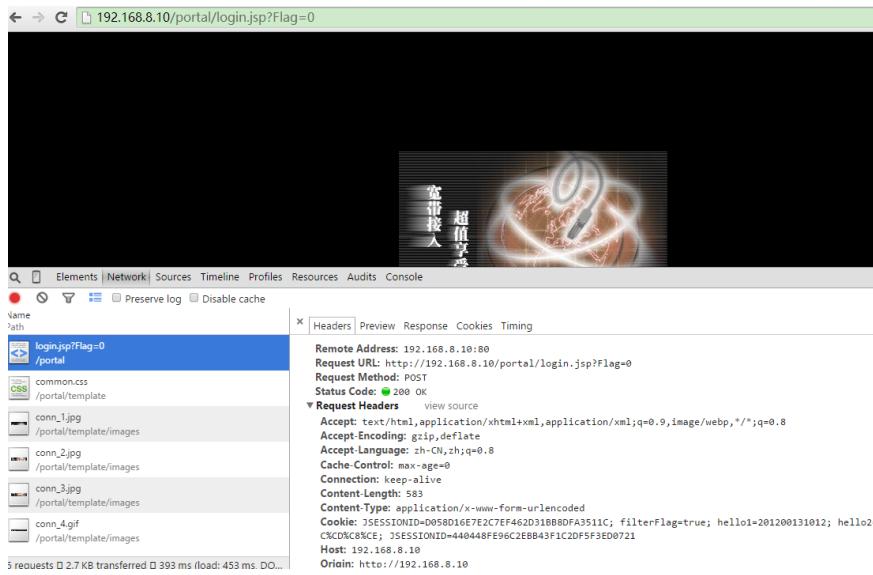


嘿，这界面还算可以把，需要我们输入的东西就是俩，一个就是学号，另一个是身份证号后六位，然后就可以登录，享受免费的无线网啦。

不过不知道谁设置了个登录时长，一段时间后就会掉线了，于是，自动模拟登陆系统就要应运而生啦。

来，我们先点击一下连接，看一下浏览器怎么工作的。

按下F12，监听网络，我们点击第一个响应，也就是login.jsp，看一下。



我们具体看一下headers，里面form提交了什么东西，真的是茫茫多的数据啊。

```

x Headers Preview Response Cookies Timing
  referer: http://192.168.8.10/portal/index_default.jsp?li
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.121 Safari/537.36
  ▼ Query String Parameters      view source      view URL encoded
    Flag: 0
  ▼ Form Data      view source      view URL encoded
    username: 201200131012
    password: 293271
    serverType:
    isSavePass: on
    Submit1: (unable to decode value)
    Language: Chinese
    ClientIP: 211.87.237.15
    sessionID: 5764998259339241327
    timeoutvalue: 45
    heartbeat: 240
    fastwebornot: false
    StartTime: 1442748550298
    shkOvertime: 720
    strOSName:
    iAdptIndex:
    strAdptName:
    strAdptStdName:
    strFileEncoding:
    PhysAddr:

```

嗯，一目了然POST的数据和提交的地址。让我们来分析几个数据吧：

- **ClientIP:** 当前客户端的IP地址，在山大软件园校区这个地址是211.87开头的
- **timeoutvalue:** 连接等待时间，也就是俗话说的timeout
- **StartTime:** 登录时间，也就是在你登录的那一刻的时间戳，这个时间戳是13位的，精确到了毫秒，不过一般是10位的，我们加3个0就好了
- **shkOvertime:** 登录持续时间，这个数据默认是720，也就是12分钟之后，登录就失效了，自动掉线，我们可以手动更改
- **username:** 学号
- **password:** 密码，也就是我们身份证号后六位

我们需要在登录的时候把form表单中的所有信息都POST一下，然后就可以完成登录啦。万事俱备，只欠东风，来来来，程序写起来！

一触即发

说走咱就走啊，天上的星星参北斗啊！

登陆地址：Request URL:<http://192.168.8.10/portal/login.jsp?Flag=0>

首先，我们需要验证一下IP地址，先写一个获取IP地址的函数，首先判断当前IP是不是211.87开头的，如果是的话，证明连接的IP是有效的。

首先我们写一个获取本机IP的方法：

```
self.ip_pre = "211.87"
#获取本机无线IP
def getIP(self):
    local_ip = socket.gethostname()
    if self.ip_pre in str(local_ip):
        return str(local_ip)
    ip_lists = socket.gethostbyname_ex(socket.gethostname())

    for ip_list in ip_lists:
        if isinstance(ip_list, list):
            for i in ip_list:
                if self.ip_pre in str(i):
                    return str(i)
        elif type(ip_list) is types.StringType:
            if self.ip_pre in ip_list:
                return ip_list
```

这个方法利用了gethostname和gethostbyname_ex方法，获取了各个网卡的IP地址，遍历一下，找到那个211.87开头的IP，返回接下来，获取到IP之后，我们便可以构建form，然后进行模拟登陆了。

```

#模拟登录
def login(self):
    print self.getCurrentTime(), u"正在尝试认证QLSC_STU无线网"
    ip = self.getIP()
    data = {
        "username": self.username,
        "password": self.password,
        "serverType": "",
        "isSavePass": "on",
        "Submit1": "",
        "Language": "Chinese",
        "ClientIP": self.getIP(),
        "timeoutvalue": 45,
        "heartbeat": 240,
        "fastwebornot": False,
        "StartTime": self.getNowTime(),
        #持续时间，超过这个时间自动掉线，可进行设置
        "shkOvertime": self.overtime,
        "strOSName": "",
        "iAdptIndex": "",
        "strAdptName": "",
        "strAdptStdName": "",
        "strFileEncoding": "",
        "PhysAddr": "",
        "bDHCPEnabled": "",
        "strIPAddrArray": "",
        "strMaskArray": "",
        "strMask": "",
        "iDHCPDelayTime": "",
        "iDHCPTryTimes": "",
        "strOldPrivateIP": self.getIP(),
        "strOldPublicIP": self.getIP(),
        "strPrivateIP": self.getIP(),
        "PublicIP": self.getIP(),
        "iIPCONFIG":0,
        "sHttpPrefix": "http://192.168.8.10",
        "title": "CAMS Portal"
    }
    #消息头
    headers = {
        'User-Agent' : 'Mozilla/5.0 (Windows NT 6.3; WOW64)',
        'Host': '192.168.8.10',
        'Origin': 'http://192.168.8.10',
        'Referer': 'http://192.168.8.10/portal/index_default'
    }
    post_data = urllib.urlencode(data)
    login_url = "http://192.168.8.10/portal/login.jsp?Flag=0"
    request = urllib2.Request(login_url, post_data, headers)

```

```
response = urllib2.urlopen(request)
result = response.read().decode('gbk')
```

比较多的内容就在于**form**表单的数据内容以及请求头，后来利用**urllib2**的**urlopen**方法实现模拟登陆。

如果大家对此不熟悉，可以参见

Urllib的基本使用

这样，登录后的结果就会保存在**result**变量中，我们只需要从**result**中提取出我们需要的数据就可以了。

乘胜追击

接下来，我们就分析一下数据啦，结果有这么几种：

- 1.登录成功
- 2.已经登录
- 3.用户不存在
- 4.密码错误
- 5.未知错误

好，利用**result**分析一下结果

```
#打印登录结果
def getLoginResult(self, result):
    if u"用户上线成功" in result:
        print self.getCurrentTime(),u"用户上线成功,在线时长为"
    elif u"您已经建立了连接" in result:
        print self.getCurrentTime(),u"您已经建立了连接,无需重新登录"
    elif u"用户不存在" in result:
        print self.getCurrentTime(),u"用户不存在,请检查学号是否正确"
    elif u"用户密码错误" in result:
        pattern = re.compile('<td class="tWhite">.*?2553:(.*')
        res = re.search(pattern, result)
        if res:
            print self.getCurrentTime(),res.group(1),u"请重新输入用户名或密码"
    else:
        print self.getCurrentTime(),u"未知错误,请检查学号密码是否正确"
```

通过字符串匹配和正则表达式，我们分辨并提取出了上述五种情况。

增加循环检测 既然是检测网络是否断开，那么我们只需要每隔一段时间检测一下就好了，那就10秒吧。

因为这个10秒是可配置的，为了方便配置，统一配置到**init**方法里面。

```
#检测间隔时间，单位为秒
self.every = 10
```

然后，我们写一个循环来检测一下

```
while True:
    nowIP = self.getIP()
    if not nowIP:
        print self.getCurrentTime(), u"请检查是否正常连接"
    else:
        print self.getCurrentTime(), u"成功连接了QLSC_STU"
        self.login()
        while True:
            can_connect = self.canConnect()
            if not can_connect:
                nowIP = self.getIP()
                if not nowIP:
                    print self.getCurrentTime(), u"当前已断开"
                else:
                    print self.getCurrentTime(), u"当前已连接"
                    self.login()
            else:
                print self.getCurrentTime(), u"当前网络连接正常"
                time.sleep(self.every)
                time.sleep(self.every)
```

其中我们用到了**canConnect**方法，这个就是检测网络是否已经断开的方法，我们可以利用**ping**百度的方法来检测一下。

方法实现如下

```
#判断当前是否可以联网
def canConnect(self):
    fnull = open(os.devnull, 'w')
    result = subprocess.call('ping www.baidu.com', shell = True)
    fnull.close()
    if result:
        return False
    else:
        return True
```

好啦，所有的要点我们已经逐一击破，等着凯旋吧

收拾战场

好了，所有的代码要点已经被我们攻破了，接下来就整理一下，让他们组合起来，变成一个应用程序吧。

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

import urllib
import urllib2
import socket
import types
import time
import re
import os
import subprocess

class Login:

    #初始化
    def __init__(self):
        #学号密码
        self.username = '201200131012'
        self.password = 'XXXXXX'
        #山大无线STU的IP网段
        self.ip_pre = '211.87'
        #登录时长
        self.overtime = 720
        #检测间隔时间，单位为秒
        self.every = 10

    #模拟登录
    def login(self):
        print self.getCurrentTime(), u"正在尝试认证QLSC_STU无线网络"
        ip = self.getIP()
        data = {
            "username": self.username,
            "password": self.password,
            "serverType": "",
            "isSavePass": "on",
            "Submit1": "",
            "Language": "Chinese",
            "ClientIP": self.getIP(),
            "timeoutvalue": 45,
            "heartbeat": 240,
            "fastwebornot": False,
            "StartTime": self.getNowTime(),
            #持续时间，超过这个时间自动掉线，可进行设置
            "shkOvertime": self.overtime,
            "strOSName": "",
            "iAdptIndex": "",
            "strAdptName": "",
            "strAdptStdName": ""}
```

```

        "strFileEncoding": "",  

        "PhysAddr": "",  

        "bDHCPEnabled": "",  

        "strIPAddrArray": "",  

        "strMaskArray": "",  

        "strMask": "",  

        "iDHCPDelayTime": "",  

        "iDHCPTryTimes": "",  

        "strOldPrivateIP": self.getIP(),  

        "strOldPublicIP": self.getIP(),  

        "strPrivateIP": self.getIP(),  

        "PublicIP": self.getIP(),  

        "iIPCONFIG":0,  

        "sHttpPrefix": "http://192.168.8.10",  

        "title": "CAMS Portal"  

    }  

#消息头  

headers = {  

    'User-Agent' : 'Mozilla/5.0 (Windows NT 6.3; WOW64)  

    'Host': '192.168.8.10',  

    'Origin': 'http://192.168.8.10',  

    'Referer': 'http://192.168.8.10/portal/index_default  

}  

post_data = urllib.urlencode(data)  

login_url = "http://192.168.8.10/portal/login.jsp?Flag=0  

request = urllib2.Request(login_url, post_data, headers)  

response = urllib2.urlopen(request)  

result = response.read().decode('gbk')  

self.getLoginResult(result)

#打印登录结果
def getLoginResult(self, result):
    if u"用户上线成功" in result:
        print self.getCurrentTime(),u"用户上线成功,在线时长为"
    elif u"您已经建立了连接" in result:
        print self.getCurrentTime(),u"您已经建立了连接,无需重复"
    elif u"用户不存在" in result:
        print self.getCurrentTime(),u"用户不存在,请检查学号是否正确"
    elif u"用户密码错误" in result:
        pattern = re.compile('<td class="tWhite">.*?2553:(.*')
        res = re.search(pattern, result)
        if res:
            print self.getCurrentTime(),res.group(1),u"请重新输入用户名或密码"
    else:
        print self.getCurrentTime(),u"未知错误,请检查学号或密码"

#获取当前时间戳, 13位

```

```

def getNowTime(self):
    return str(int(time.time()))+"000"

#获取本机无线IP
def getIP(self):
    local_ip = socket.gethostbyname(socket.gethostname())
    if self.ip_pre in str(local_ip):
        return str(local_ip)
    ip_lists = socket.gethostbyname_ex(socket.gethostname())

    for ip_list in ip_lists:
        if isinstance(ip_list, list):
            for i in ip_list:
                if self.ip_pre in str(i):
                    return str(i)
        elif type(ip_list) is types.StringType:
            if self.ip_pre in ip_list:
                return ip_list

#判断当前是否可以联网
def canConnect(self):
    fnull = open(os.devnull, 'w')
    result = subprocess.call('ping www.baidu.com', shell = True)
    fnull.close()
    if result:
        return False
    else:
        return True

#获取当前时间
def getCurrentTime(self):
    return time.strftime('[%Y-%m-%d %H:%M:%S]',time.localtime())

#主函数
def main(self):
    print self.getCurrentTime(), u"您好，欢迎使用模拟登陆系统"
    while True:
        nowIP = self.getIP()
        if not nowIP:
            print self.getCurrentTime(), u"请检查是否正常连接"
        else:
            print self.getCurrentTime(),u"成功连接了QLSC_STU"
            self.login()
            while True:
                can_connect = self.canConnect()
                if not can_connect:
                    nowIP = self.getIP()
                    if not nowIP:

```

```
        print self.getCurrentTime(), u"当前已
    else:
        print self.getCurrentTime(), u"当前已
        self.login()
    else:
        print self.getCurrentTime(), u"当前网络连
time.sleep(self.every)
time.sleep(self.every)

login = Login()
login.main()
```

来，我们来运行一下，看下效果吧！执行

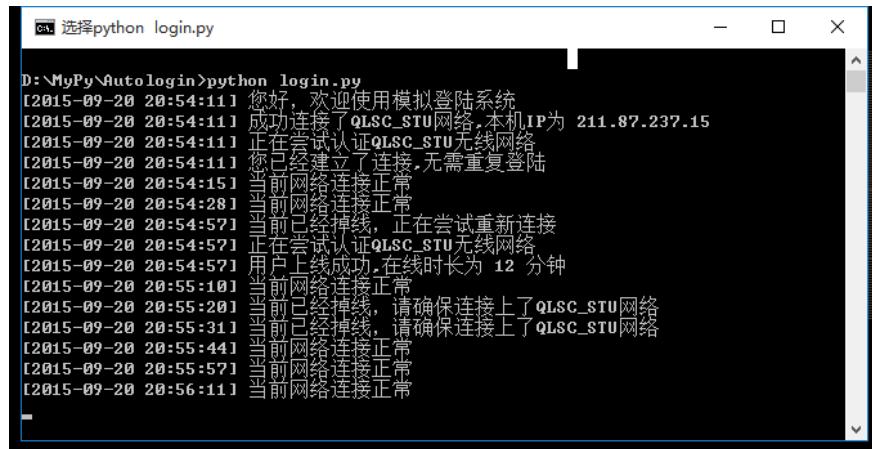
```
python login.py
```

当前是可以联网的，我分别在网页上操作执行了断开，操作，程序自动检测到掉线，自动重新连接。

接下来我直接断开了QLSC_STU网络的链接，程序同样检测到QLSC_STU这个热点没有连接上，提示用户链接。

接下来我重新连接上了这个热点，由于刚才已经登录上线，且持续时间较短，网络自动恢复正常。

下面是运行结果：



```
D:\MyPy\Autologin>python login.py
[2015-09-20 20:54:11] 您好，欢迎使用模拟登陆系统
[2015-09-20 20:54:11] 成功连接了QLSC_STU网络,本机IP为 211.87.237.15
[2015-09-20 20:54:11] 正在尝试认证QLSC_STU无线网络
[2015-09-20 20:54:11] 您已经建立了连接,无需重复登陆
[2015-09-20 20:54:15] 当前网络连接正常
[2015-09-20 20:54:28] 当前网络连接正常
[2015-09-20 20:54:57] 当前已经掉线,正在尝试重新连接
[2015-09-20 20:54:57] 正在尝试认证QLSC_STU无线网络
[2015-09-20 20:54:57] 用户上线成功,在线时长为 12 分钟
[2015-09-20 20:55:10] 当前网络连接正常
[2015-09-20 20:55:20] 当前已经掉线,请确保连接上了QLSC_STU网络
[2015-09-20 20:55:31] 当前已经掉线,请确保连接上了QLSC_STU网络
[2015-09-20 20:55:44] 当前网络连接正常
[2015-09-20 20:55:57] 当前网络连接正常
[2015-09-20 20:56:11] 当前网络连接正常
```

嗯，这样我们就是实现了自动掉线的检测和模拟登录。

凯旋而归

咿呀伊尔哟，想约妹子上自习吗？那就赶紧来试试吧！一网在手，天下我有！追男神女神都不再是梦想！

如果有问题，欢迎留言讨论，代码肯定有不完善的地方，仅供参考。

Python爬虫实战四之抓取淘宝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？

最新动态

更新时间：2015/8/2 最近好多读者反映代码已经不能用了，原因是淘宝索引页的MM链接改了。网站改版了，URL的索引已经和之前的不一样了，之前可以直接跳转到每个MM的个性域名，现在中间加了一个跳转页，本以为可以通过这个页面然后跳转到原来的个性域名，而经过一番折腾发现，这个跳转页中的内容是JS动态生成的，所以不能用Urllib库来直接抓取了，本篇就只提供学习思路，代码不能继续用了。之后博主会利用其它方法来尝试解决，如果解决，第一时间更新！谢谢大家！

更新时间：2016/3/26 如上问题已解决，利用 PhantomJS的动态解析即可完成。因为 PySpider 同样支持 PhantomJS，所以我直接利用了 PySpider 来完成，解决方案如下 [解决方案](#) 另外如果不使用框架，可以直接利用 Selenium + PhantomJS 来解析，同样方便，解决方案可以参考 [动态解析解决方案](#)

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

1.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。

2. 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?page=1'

    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.*?>.*?(<img.*?src="(.*)".*?(</div>')
        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://img07.taobacd... http://img07.taobacd... 田师傅 25 广州市
http://img05.taobacd... http://img05.taobacd... 朱力霸 25 杭州市
http://img15.taobacd... http://img15.taobacd... 崔辰辰 26 杭州市
http://img01.taobacd... http://img01.taobacd... 大猫儿 29 广州市
http://img01.taobacd... http://img01.taobacd... 金甜甜 24 广州市
http://img04.taobacd... http://img04.taobacd... 紫轩 28 杭州市
http://img08.taobacd... http://img08.taobacd... 谢婷婷 26 杭州市
http://img08.taobacd... http://img08.taobacd... 夏晨洁 26 杭州市
http://img03.taobacd... http://img03.taobacd... Cherry 27 广州市
http://img07.taobacd... http://img07.taobacd... 雪倩nike 24 杭州市

Process finished with exit code 0
```

2. 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

1) 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()
```

2) 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'))
```

3) 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
```

3.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.json'
        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.*?>')
        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".*?>')
        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".*?>')

```

```

#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]
        print u"发现一位模特，名字叫",item[2],u"芳龄",item[3],u
        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

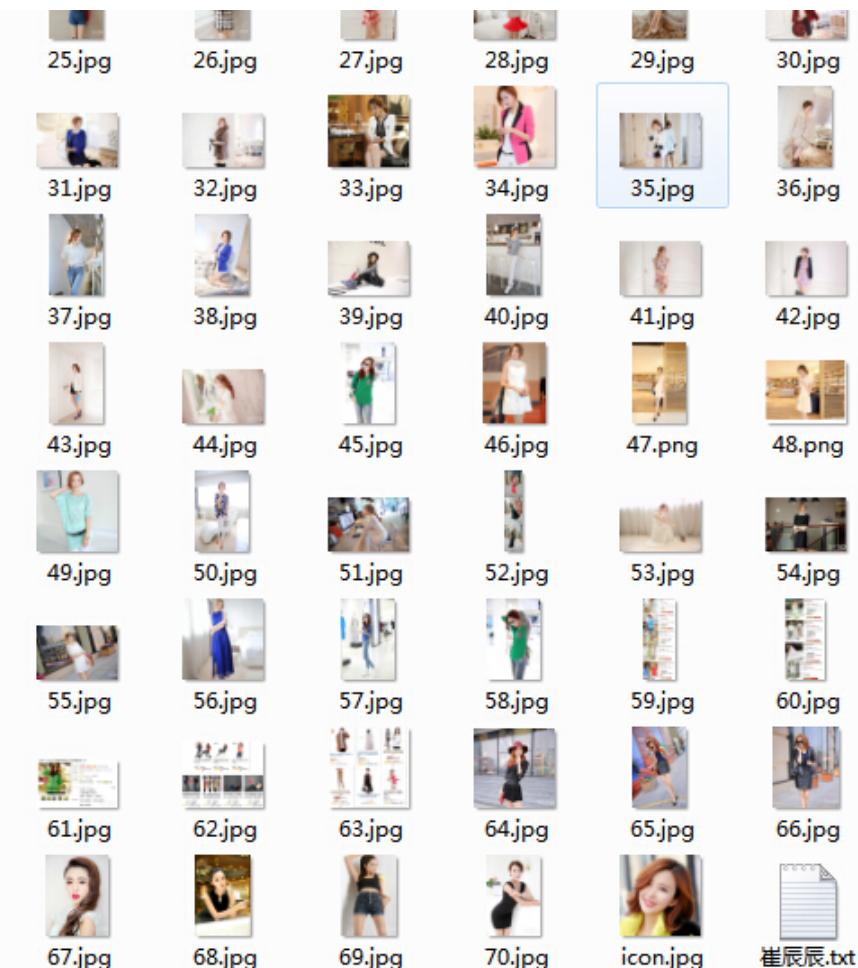
#处理页面标签类
class Tool:
    #去除img标签,1-7位空格,&nbsp;
    removeImg = re.compile('<img.*?>| {1,7}|&nbsp;')
    #删除超链接标签
    removeAddr = re.compile('<a.*?>|</a>')
    #把换行的标签换为\n
    replaceLine = re.compile('<tr>|<div>|</div>|</p>')
    #将表格制表<td>替换为\t
    replaceTD= re.compile('<td>')
    #将换行符或双换行符替换为\n
    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

正在悄悄保存她的一张图片为 戴誉利/99.jpg
正在悄悄保存她的一张图片为 戴誉利/100.jpg
正在悄悄保存她的一张图片为 戴誉利/101.jpg
正在悄悄保存她的一张图片为 戴誉利/102.jpg
正在悄悄保存她的一张图片为 戴誉利/103.jpg
正在偷偷寻找第 3 个地方，看看MM们在不在
发现一位模特，名字叫 滕雨佳 芳龄 23，她在 杭州市
正在偷偷地保存 滕雨佳 的信息
又意外地发现她的个人地址是 <http://mm.taobao.com/539549300.htm>
名为 滕雨佳 的文件夹已经创建成功
正在偷偷保存她的个人信息为 滕雨佳/滕雨佳.txt
正在悄悄保存她的一张图片为 滕雨佳/icon.jpg
发现 滕雨佳 共有 83 张照片
正在悄悄保存她的一张图片为 滕雨佳/1.jpg
正在悄悄保存她的一张图片为 滕雨佳/2.jpg
正在悄悄保存她的一张图片为 滕雨佳/3.jpg
正在悄悄保存她的一张图片为 滕雨佳/4.jpg
正在悄悄保存她的一张图片为 滕雨佳/5.jpg
正在悄悄保存她的一张图片为 滕雨佳/6.jpg
正在悄悄保存她的一张图片为 滕雨佳/7.jpg
正在悄悄保存她的一张图片为 滕雨佳/8.jpg
正在悄悄保存她的一张图片为 滕雨佳/9.jpg

看看文件夹里面有什么变化

CC涵梵	2015/2/21 1:56	文件夹
Cherry	2015/2/21 1:36	文件夹
CILY	2015/2/21 1:56	文件夹
rikki	2015/2/21 1:58	文件夹
阿朵拉	2015/2/21 1:56	文件夹
崔辰辰	2015/2/21 1:22	文件夹
大猫儿	2015/2/21 1:34	文件夹
戴普利	2015/2/21 1:57	文件夹
姬嘉琳	2015/2/21 1:55	文件夹
金甜甜	2015/2/21 1:34	文件夹
李喵喵	2015/2/21 1:36	文件夹
喵小咪	2015/2/21 1:58	文件夹
沁源	2015/2/21 1:54	文件夹
滕雨佳	2015/2/21 1:54	文件夹
田媛媛	2015/2/21 1:18	文件夹
夏晨洁	2015/2/21 1:35	文件夹
夏欢欢	2015/2/21 1:49	文件夹
晓莉莉	2015/2/21 1:57	文件夹
谢婷婷	2015/2/21 1:35	文件夹
熊仔欣	2015/2/21 1:56	文件夹
雪倩nika	2015/2/21 1:36	文件夹
杨羽凡	2015/2/21 1:59	文件夹
悦小舞	2015/2/21 1:49	文件夹



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！！

代码均为本人所敲，写的不好，大神勿喷，写来方便自己，同时分享给大家参考！希望大家支持！

Python爬虫实战五之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。

温馨提示

更新时间，2016-02-01，现在淘宝换成了滑块验证了，比较难解决这个问题，以下的代码没法用了，仅作学习参考研究之用吧。

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。

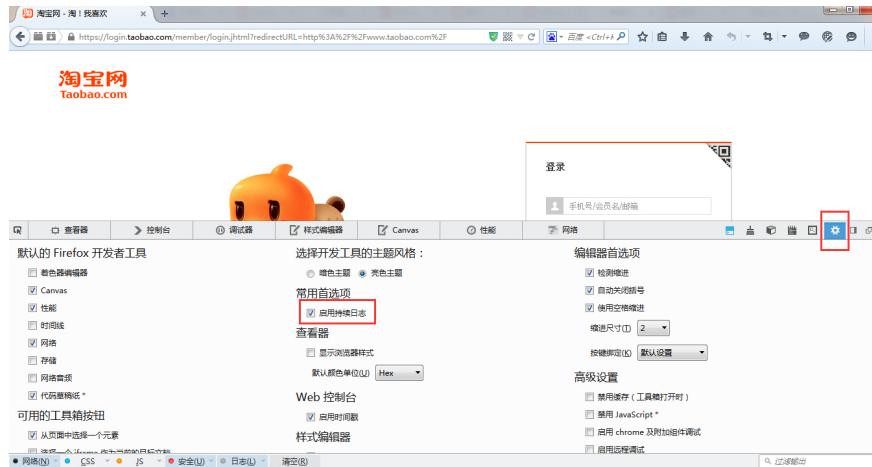
前期准备

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：

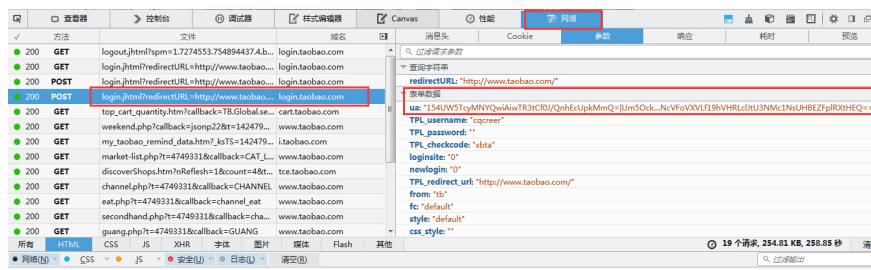


好，那么接下来我们就从浏览器中获取 ua 和 aes 密码

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.taobao.com`，然后查看参数，你就会发现表单数据了，其中包括 `ua` 还有下面的 `password2`，把这两复制下来，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.j
        #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;
            'Referer' : 'https://login.taobao.com/member/login.j
            '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
        self.ua = '191UW5TcyMNQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um50c
        #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
        self.password2 = '7511aa68sx629e45de220d29174f1066537a73
        self.post = post = {
            'ua':self.ua,
            'TPL_checkcode':'',
            'CtrlVersion': '1,0,0,7',
            'TPL_password':'',
            'TPL_redirect_url':'http://i.taobao.com/my_taobao.ht
            'TPL_username':self.username,
            'loginsite':'0',
            'newlogin':'0',
            'from':'tb',
            'fc':'default',
            'style':'default',
            'css_style':'',
            'tid':'XOR_1_000000000000000000000000000000000000000000000000000000000000000_625C4720
            'support':'00001',
            'loginType':'4',
        }

```

```

'minititle':'',
'minipara':'',
'umto':'NaN',
'pstrong':'3',
'lnick':'',
'sign':'',
'need_sign':'',
'isIgnore':'',
'full_redirect':'',
'popid':'',
'callback':'',
'guf':'',
'not_duplete_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.co
#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,se

#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,se
    #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!
    pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1')
    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-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



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

```

C:\>python -U
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 verify failed <_ssl.c:581>>

```

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，会跳出浏览器，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.j
        #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;
            'Referer' : 'https://login.taobao.com/member/login.j
            '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
        self.ua = '191UW5TcyMNYQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um50c
        #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
        self.password2 = '7511aa6854629e45de220d29174f1066537a73
        self.post = post = {
            'ua':self.ua,
            'TPL_checkcode':'',
            'CtrlVersion': '1,0,0,7',
            'TPL_password':'',
            'TPL_redirect_url':'http://i.taobao.com/my_taobao.ht
            'TPL_username':self.username,
            'loginsite':'0',
            'newlogin':'0',
            'from':'tb',
            'fc':'default',
            'style':'default',
            'css_style':'',
            'tid':'XOR_1_00000000000000000000000000000000000_625C4720
            'support':'00001',
            'loginType':'4',
        }
```

```

'minititle':'',
'minipara':'',
'umto':'NaN',
'pstrong':'3',
'lnick':'',
'sign':'',
'need_sign':'',
'isIgnore':'',
'full_redirect':'',
'popid':'',
'callback':'',
'guf':'',
'not_duplete_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.co
#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,se

#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,se
    #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!
    pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1\u7801')
    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-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)
    #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\u8be
    pattern = re.compile(u'\u9a8c\u8bc1\u7801\u9519\u8be')
    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，后
        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。

```
此次安全验证异常，您需要输入验证码
您需要手动输入验证码
https://pin.aliyun.com/get\_img?sessionid=797ad63763410ea7afcf16d45426f2ca&identity=taobao.login&ttype=150\_40
验证码获取成功
请在浏览器中输入您看到的验证码
请输入验证码:wvm4
验证码输入正确
1_TspmvWZBCnW_0Hd3qc-9w
J_HToken 1_TspmvWZBCnW_0Hd3qc-9w
```

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?sit'
    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_ usern
```

比如

```
https://login.taobao.com/member/vst.htm?st=1uynJELa4hKfsfWU30jP
```

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.h
```

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/list'
    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。



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
                     u'price.*?title="(.*?)" .*?quantity.*?title=
result = re.findall(pattern,page)
for item in result:
    print '-----'
    print "购买日期:",item[0].strip(), '订单号:',item[1].strip(),
    print '宝贝名称:',item[3].strip()
    print '原价:',item[4].strip(),'购买数量:',item[5].strip(),'实|
```

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~

```
■ | ↓ | 获取请求成功  
|| | 此次安全验证异常，您需要输入验证码  
| | | 您需要手动输入验证码  
| | | 验证码获取成功  
| | | 请在浏览器中输入您看到的验证码  
| | | 请输入验证码: 5psk  
| | | 验证码输入正确  
| | | 成功获取st码  
| | | 登录网址成功  
| | | 找到了共多少页  
| | | 共 7 页  
| | | 获取到的商品列表如下  
| | |  
| | | 购买日期: 2015-02-20 订单号: 982310262226149 卖家店铺: 小米官方旗舰店  
| | | 宝贝名称: 小米旗舰店MIUI小米 小米盒子增强版1G高清网络电视机顶盒播放器  
| | | 原价: 299.01 购买数量: 1 实际支付: 277.93 交易状态: 快件已揽收  
| | |  
| | | 购买日期: 2015-01-29 订单号: 959538696226149 卖家店铺: 五颗星时尚男装店  
| | | 宝贝名称: 新款冬季男士保暖衬衫加绒加厚大码衬衣男款修身韩版免烫潮流男衬衫  
| | | 原价: 398.00 购买数量: 1 实际支付: 96.00 交易状态: 交易成功  
| | |  
| | | 购买日期: 2015-01-25 订单号: 952957475766149 卖家店铺: Hers无线特卖店  
| | | 宝贝名称: 山东wlan一天卡【潍坊专用】非3天非7天到次日8点 自动发货  
| | | 原价: 1.48 购买数量: 1 实际支付: 1.48 交易状态: 交易成功  
| | |  
| | | 购买日期: 2015-01-23 订单号: 951139653306149 卖家店铺: 颜妍堂旗舰店
```

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(.*)')
        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
                             u'price.*?title="(.*?)".*?quantity.
        result = re.findall(pattern,page)
        for item in result:
            print '-----'
            print "购买日期:",item[0].strip(), '订单号:',item[1].strip()
            print '宝贝名称:',item[3].strip()
            print '原价:',item[4].strip(), '购买数量:',item[5].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.j
        #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;
            'Referer' : 'https://login.taobao.com/member/login.j
            '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
        self.ua = '191UW5TcyMNYQwiAiwTR3tCf0J/QnhEcUpkMmQ=|Um50c
        #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, 25位
        self.password2 = '7511aa6854629e45de220d29174f1066537a73
        self.post = post = {
            'ua':self.ua,
            'TPL_checkcode':'',
            'CtrlVersion': '1,0,0,7',
            'TPL_password':'',
            'TPL_redirect_url':'http://i.taobao.com/my_taobao.ht
            'TPL_username':self.username,
            'loginsite':'0',
            'newlogin':'0',
            'from':'tb',
            'fc':'default',
            'style':'default',
            'css_style':'',
            'tid':'XOR_1_000000000000000000000000000000000000000000000000000000000000000_625C4720
            '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,se
    #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.HTTPCookie
    #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
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,se
    #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
        pattern = re.compile(u'\u8bf7\u8f93\u5165\u9a8c\u8bc1')
        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-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)
        #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\u8be3
        pattern = re.compile(u'\u9a8c\u8bc1\u7801\u9519\u8be3')
        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，后
            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_HToken获取失败
                print u"J_HToken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'
    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&T'
    headers = {
        'User-Agent' : 'Mozilla/5.0 (Windows NT 6.1; WOW64;',
        '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
    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，password2这三个设置好。

以上均为博主亲身所敲，代码写的不好，谨在此贴出和大家一起分享经验~

小伙伴们试一下吧，希望对大家有帮助~

Python爬虫实战六之抓取爱问知识人问题并保存至数据库

大家好，本次为大家带来的是抓取爱问知识人的问题并将问题和答案保存到数据库的方法，涉及的内容包括：

- `Urllib`的用法及异常处理
- `Beautiful Soup`的简单应用
- `MySQLdb`的基础用法
- 正则表达式的简单应用

环境配置

在这之前，我们需要先配置一下环境，我的Python的版本为2.7，需要额外安装的库有两个，一个是`Beautiful Soup`，一个是`MySQLdb`，在这里附上两个库的下载地址，

[Beautiful Soup](#)

[MySQLdb](#)

大家可以下载之后通过如下命令安装

```
python setup.py install
```

环境配置好之后，我们便可以开心地撸爬虫了

框架思路

首先我们随便找一个分类地址，[外语学习 – 爱问知识人](#)，打开之后可以看到一系列的问题列表。

我们在这个页面需要获取的东西有：

总的页码数，每一页的所有问题链接。

接下来我们需要遍历所有的问题，来抓取每一个详情页面，提取问题，问题内容，回答者，回答时间，回答内容。

最后，我们需要把这些内容存储到数据库中。

要点简析

其实大部分内容相信大家会了前面的内容，这里的爬虫思路已经融汇贯通了，这里就说一下一些扩展的功能

1. 日志输出

日志输出，我们要输出时间和爬取的状态，比如像下面这样：

- [2015-08-10 03:05:20] 113011 号问题存在其他答案 我个人认为应该是樱桃沟很美的
- [2015-08-10 03:05:20] 保存到数据库,此问题的ID为 113011
- [2015-08-10 03:05:20] 当前爬取第 2 的内容,发现一个问题 百度有一个地方，花儿带着芳香，水儿流淌奔腾是什么意思 多多帮忙哦 回答数量 1
- [2015-08-10 03:05:19] 保存到数据库,此问题的ID为 113010

所以，我们需要引入时间函数，然后写一个获取当前时间的函数

```
import time

#获取当前时间
def getCurrentTime(self):
    return time.strftime('[%Y-%m-%d %H:%M:%S]',time.localtime(ti

#获取当前时间
def getCurrentDate(self):
    return time.strftime('%Y-%m-%d',time.localtime(time.time()))
```

以上分别是获取带具体时间和获取日期的函数，在输出时，我们可以在输出语句的前面调用这函数即可。

然后我们需要将缓冲区设置输出到log中，在程序的最前面加上这两句即可

```
f_handler=open('out.log', 'w')
sys.stdout=f_handler
```

这样，所有的print语句输出的内容就会保存到out.log文件中了。

2. 页码保存

爬虫爬取过程中可能出现各种各样的错误，这样会导致爬虫的中断，如果我们重新运行爬虫，那么就会导致爬虫从头开始运行了，这样显然是不合理的。所以，我们需要把当前爬取的页面保存下来，比如可以保存到文本中，假如爬虫中断了，重新运行爬虫，读取文本文件的内容，接着爬取即可。

大家可以稍微参考一下函数的实现：

```
#主函数
def main(self):
    f_handler=open('out.log', 'w')
    sys.stdout=f_handler
    page = open('page.txt', 'r')
    content = page.readline()
    start_page = int(content.strip()) - 1
    page.close()
    print self.getCurrentTime(),"开始页码",start_page
    print self.getCurrentTime(),"爬虫正在启动,开始爬取爱问知识人问题是"
    self.total_num = self.getTotalPageNum()
    print self.getCurrentTime(),"获取到目录页面个数",self.total_num
    if not start_page:
        start_page = self.total_num
    for x in range(1,start_page):
        print self.getCurrentTime(),"正在抓取第",start_page-x+1,""
        try:
            self.getQuestions(start_page-x+1)
        except urllib2.URLError, e:
            if hasattr(e, "reason"):
                print self.getCurrentTime(),"某总页面内抓取或提取失败"
        except Exception,e:
            print self.getCurrentTime(),"某总页面内抓取或提取失败,"
        if start_page-x+1 < start_page:
            f=open('page.txt','w')
            f.write(str(start_page-x+1))
            print self.getCurrentTime(),"写入新页码",start_page-x
            f.close()
```

这样，不管我们爬虫中途遇到什么错误，妈妈也不会担心了

3. 页面处理

页面处理过程中，我们可能遇到各种各样奇葩的HTML代码，和上一节一样，我们沿用一个页面处理类即可。

```
import re

#处理页面标签类
class Tool:

    #将超链接广告剔除
    removeADLink = re.compile('<div class="link_layer.*?</div>')
    #去除img标签,1-7位空格,&nbsp;
    removeImg = re.compile('<img.*?>| {1,7}|&nbsp;')
    #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.removeADLink,"",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()
```

我们可以用一段含有HTML代码的文字，经过调用replace方法之后，各种冗余的HTML代码就会处理好了。

比如我们这么一段代码：

```
<article class="article-content">
<h2>前言</h2>
<p>最近发现MySQL服务隔三差五就会挂掉，导致我的网站和爬虫都无法正常运作</p>
<p>好了，闲言碎语不多讲，开始我们的配置之旅。</p>
<p>运行环境：<strong>Ubuntu Linux 14.04</strong></p>
<h2>编写Shell脚本</h2>
<p>首先，我们要编写一个shell脚本，脚本主要执行的逻辑如下：</p>
<p>显示mysqld进程状态，如果判断进程未在运行，那么输出日志到文件，然后启动</p>
<p>可能大家对于shell脚本比较陌生，在这里推荐官方的shell脚本文档来参考</p>
<p><a href="http://wiki.ubuntu.org.cn/Shell%E7%BC%96%E7%A8%8B%E5%BC%80%E6%95%B0">Ubuntu Shell编程基础</a></p>
<p>shell脚本的后缀为sh，在任何位置新建一个脚本文件，我选择在 /etc/mysql/scripts/ 目录下</p>
<p>执行如下命令：</p>
```

经过处理后便会变成如下的样子：

```
前言
最近发现MySQL服务隔三差五就会挂掉，导致我的网站和爬虫都无法正常运作。好了，闲言碎语不多讲，开始我们的配置之旅。
运行环境：UbuntuLinux14.04
编写Shell脚本
首先，我们要编写一个shell脚本，脚本主要执行的逻辑如下：
显示mysqld进程状态，如果判断进程未在运行，那么输出日志到文件，然后启动
可能大家对于shell脚本比较陌生，在这里推荐官方的shell脚本文档来参考一下
UbuntuShell编程基础
shell脚本的后缀为sh，在任何位置新建一个脚本文件，我选择在/etc/mysql目录下
执行如下命令：
```

经过上面的处理，所有乱乱的代码都会被处理好了。

4.保存到数据库

在这里，我们想实现一个通用的方法，就是把存储的一个个内容变成字典的形式，然后执行插入语句的时候，自动构建对应的sql语句，插入数据。

比如我们构造如下的字典：

```
#构造最佳答案的字典
good_ans_dict = {
    "text": good_ans[0],
    "answerer": good_ans[1],
    "date": good_ans[2],
    "is_good": str(good_ans[3]),
    "question_id": str(insert_id)
}
```

构造sql语句并插入到数据库的方法如下：

```
#插入数据
def insertData(self, table, my_dict):
    try:
        self.db.set_character_set('utf8')
        cols = ', '.join(my_dict.keys())
        values = ''',' '.join(my_dict.values())
        sql = "INSERT INTO %s (%s) VALUES (%s)" % (table, c
        try:
            result = self.cur.execute(sql)
            insert_id = self.db.insert_id()
            self.db.commit()
            #判断是否执行成功
            if result:
                return insert_id
            else:
                return 0
        except MySQLdb.Error,e:
            #发生错误时回滚
            self.db.rollback()
            #主键唯一，无法插入
            if "key 'PRIMARY'" in e.args[1]:
                print self.getCurrentTime(),"数据已存在，未插
            else:
                print self.getCurrentTime(),"插入数据失败，原
        except MySQLdb.Error,e:
            print self.getCurrentTime(),"数据库错误，原因%d: %s"
```

这里我们只需要传入那个字典，便会构建出对应字典键值和键名的sql语句，完成插入。

5.PHP读取日志

我们将运行结果输出到了日志里，那么怎么查看日志呢？很简单，在这里提供两种方法

方法一：

PHP倒序输出所有日志内容

```
<html>
<head>
<meta charset="utf-8">
<meta http-equiv="refresh" content = "5">
</head>
<body>
<?php
$fp = file("out.log");
if ($fp) {
    for($i = count($fp) - 1;$i >= 0; $i --)
        echo $fp[$i]."<br>";
}
?>
</body>
</html>
```

此方法可以看到所有的输入日志，但是如果日志太大了，那么就会报耗费内存太大，无法输出。为此我们有了第二种方法，利用linux命令，输出后十行内容。

方法二：

```
<html>
<head>
<meta charset="utf-8">
<meta http-equiv="refresh" content = "5">
</head>
<body>
<?php
$ph = popen('tail -n 100 out.log','r');
while($r = fgets($ph)){
    echo $r."<br>";
}
pclose($ph);
?
</body>
</html>
```

上面两种方法都是5秒刷新一次网页来查看最新的日志。

源代码放送

好了，闲言碎语不多讲，直接上源码了

spider.py

```

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

import urllib
import urllib2
import re
import time
import types
import page
import mysql
import sys
from bs4 import BeautifulSoup

class Spider:

    #初始化
    def __init__(self):
        self.page_num = 1
        self.total_num = None
        self.page_spider = page.Page()
        self.mysql = mysql.Mysql()

    #获取当前时间
    def getCurrentTime(self):
        return time.strftime('[%Y-%m-%d %H:%M:%S]',time.localtime)

    #获取当前时间
    def getCurrentDate(self):
        return time.strftime('%Y-%m-%d',time.localtime(time.time))

    #通过网页的页码数来构建网页的URL
    def getPageURLByNum(self, page_num):
        page_url = "http://iask.sina.com.cn/c/978-all-" + str(page_num)
        return page_url

    #通过传入网页页码来获取网页的HTML
    def getPageByNum(self, page_num):
        request = urllib2.Request(self.getPageURLByNum(page_num))
        try:
            response = urllib2.urlopen(request)
        except urllib2.URLError, e:
            if hasattr(e, "code"):
                print self.getCurrentTime(),"获取页面失败,错误代号",e.code
                return None
            if hasattr(e, "reason"):
                print self.getCurrentTime(),"获取页面失败,原因",e.reason
                return None
        else:

```

```

page = response.read().decode("utf-8")
return page

#获取所有的页码数
def getTotalPageNum(self):
    print self.getCurrentTime(),"正在获取目录页面个数,请稍候"
    page = self.getPageByNum(1)
    #匹配所有的页码数,\u4e0b\u4e00\u9875是下一页的UTF8编码
    pattern = re.compile(u'<span class="more".*?>.*?<span.*?
match = re.search(pattern, page)
if match:
    return match.group(1)
else:
    print self.getCurrentTime(),"获取总页码失败"

#分析问题的代码,得到问题的提问者,问题内容,回答个数,提问时间
def getQuestionInfo(self, question):
    if not type(question) is types.StringType:
        question = str(question)
    #print question
    pattern = re.compile(u'<span.*?question-face.*?>.*?<img.
    match = re.search(pattern, question)
    if match:
        #获得提问者
        author = match.group(1)
        #问题链接
        href = match.group(2)
        #问题详情
        text = match.group(3)
        #回答个数
        ans_num = match.group(4)
        #回答时间
        time = match.group(5)
        time_pattern = re.compile('\d{4}\-\d{2}\-\d{2}', re.
        time_match = re.search(time_pattern, time)
        if not time_match:
            time = self.getCurrentDate()
        return [author, href, text, ans_num, time]
    else:
        return None

#获取全部问题
def getQuestions(self, page_num):
    #获得目录页面的HTML
    page = self.getPageByNum(page_num)
    soup = BeautifulSoup(page)
    #分析获得所有问题
    questions = soup.select("div.question_list ul li")

```

```

#遍历每一个问题
for question in questions:
    #获得问题的详情
    info = self.getQuestionInfo(question)
    if info:
        #得到问题的URL
        url = "http://iask.sina.com.cn/" + info[1]
        #通过URL来获取问题的最佳答案和其他答案
        ans = self.page_spider.getAnswer(url)
        print self.getCurrentTime(),"当前爬取第",page_num
        #构造问题的字典,插入问题
        ques_dict = {
            "text": info[2],
            "questioner": info[0],
            "date": info[4],
            "ans_num": info[3],
            "url": url
        }
        #获得插入的问题的自增ID
        insert_id = self.mysql.insertData("iask_question")
        #得到最佳答案
        good_ans = ans[0]
        print self.getCurrentTime(),"保存到数据库,此问题的"
        #如果存在最佳答案,那么就插入
        if good_ans:
            print self.getCurrentTime(),insert_id,"号问题"
            #构造最佳答案的字典
            good_ans_dict = {
                "text": good_ans[0],
                "answerer": good_ans[1],
                "date": good_ans[2],
                "is_good": str(good_ans[3]),
                "question_id": str(insert_id)
            }
            #插入最佳答案
            if self.mysql.insertData("iask_answers",good
                print self.getCurrentTime(),"保存最佳答案"
            else:
                print self.getCurrentTime(),"保存最佳答案"
            #获得其他答案
            other_anse = ans[1]
            #遍历每一个其他答案
            for other_ans in other_anse:
                #如果答案存在
                if other_ans:
                    print self.getCurrentTime(),insert_id,""
                    #构造其他答案的字典
                    other_ans_dict = {

```

```

        "text": other_ans[0],
        "answerer": other_ans[1],
        "date": other_ans[2],
        "is_good": str(other_ans[3]),
        "question_id": str(insert_id)
    }
#插入这个答案
if self.mysql.insertData("iask_answers",
    print self.getCurrentTime(),"保存其他
else:
    print self.getCurrentTime(),"保存其他

#主函数
def main(self):
    f_handler=open('out.log', 'w')
    sys.stdout=f_handler
    page = open('page.txt', 'r')
    content = page.readline()
    start_page = int(content.strip()) - 1
    page.close()
    print self.getCurrentTime(),"开始页码",start_page
    print self.getCurrentTime(),"爬虫正在启动,开始爬取爱问知识,
    self.total_num = self.getTotalPageNum()
    print self.getCurrentTime(),"获取到目录页面个数",self.tot
    if not start_page:
        start_page = self.total_num
    for x in range(1,start_page):
        print self.getCurrentTime(),"正在抓取第",start_page-x
        try:
            self.getQuestions(start_page-x+1)
        except urllib2.URLError, e:
            if hasattr(e, "reason"):
                print self.getCurrentTime(),"某总页面内抓取或提取失败",e
        except Exception,e:
            print self.getCurrentTime(),"某总页面内抓取或提取失败",e
        if start_page-x+1 < start_page:
            f=open('page.txt','w')
            f.write(str(start_page-x+1))
            print self.getCurrentTime(),"写入新页码",start_pa
            f.close()

spider = Spider()
spider.main()

```

page.py

```

# -*- coding:utf-8 -*-
import urllib
import urllib2
import re
import time
import types
import tool
from bs4 import BeautifulSoup

#抓取分析某一问题和答案
class Page:

    def __init__(self):
        self.tool = tool.Tool()

    #获取当前时间
    def getCurrentDate(self):
        return time.strftime('%Y-%m-%d',time.localtime(time.time))

    #获取当前时间
    def getCurrentTime(self):
        return time.strftime('[%Y-%m-%d %H:%M:%S]',time.localtime(time.time))

    #通过页面的URL来获取页面的代码
    def getPageByURL(self, url):
        try:
            request = urllib2.Request(url)
            response = urllib2.urlopen(request)
            return response.read().decode("utf-8")
        except urllib2.URLError, e:
            if hasattr(e, "code"):
                print self.getCurrentTime(),"获取问题页面失败,错误码:",e.code
                return None
            if hasattr(e, "reason"):
                print self.getCurrentTime(),"获取问题页面失败,原因:",e.reason
                return None

    #传入一个List,返回它的标签里的内容,如果为空返回None
    def getText(self, html):
        if not type(html) is types.StringType:
            html = str(html)
        #提取出<pre>标签里的内容
        pattern = re.compile('<pre.*?>(.*)</pre>', re.S)
        match = re.search(pattern, html)
        #如果匹配成功
        if match:

```

```

        return match.group(1)
    else:
        return None

#传入最佳答案的HTML,分析出回答者和回答时间
def getGoodAnswerInfo(self, html):
    pattern = re.compile('"answer_tip.*?<a.*?>(.*)</a>.*?<s')
    match = re.search(pattern, html)
    #如果匹配,返回回答者和回答时间
    if match:
        time = match.group(2)
        time_pattern = re.compile('\d{2}\-\d{2}\-\d{2}', re.
        time_match = re.search(time_pattern, time)
        if not time_match:
            time = self.getCurrentDate()
        else:
            time = "20"+time
        return [match.group(1),time]
    else:
        return [None,None]

#获得最佳答案
def getGoodAnswer(self, page):
    soup = BeautifulSoup(page)
    text = soup.select("div.good_point div.answer_text pre")
    if len(text) > 0:
        #获得最佳答案的内容
        ansText = self.getText(str(text[0]))
        ansText = self.tool.replace(ansText)
        #获得最佳答案的回答者信息
        info = soup.select("div.good_point div.answer_tip")
        ansInfo = self.getGoodAnswerInfo(str(info[0]))
        #将三者组合成一个List
        answer = [ansText, ansInfo[0], ansInfo[1],1]
        return answer
    else:
        #如果不存在最佳答案,那么就返回空
        return None

#传入回答者HTML,分析出回答者,回答时间
def getOtherAnswerInfo(self, html):
    if not type(html) is types.StringType:
        html = str(html)
    pattern = re.compile('"author_name.*?>(.*)</a>.*?answer')
    match = re.search(pattern, html)
    #获得每一个回答的回答者信息和回答时间
    if match:
        time = match.group(2)

```

```

        time_pattern = re.compile('\d{2}\-\d{2}\-\d{2}', re.
        time_match = re.search(time_pattern, time)
        if not time_match:
            time = self.getCurrentDate()
        else:
            time = "20"+time
        return [match.group(1),time]
    else:
        return [None,None]

#获得其他答案
def getOtherAnswers(self, page):
    soup = BeautifulSoup(page)
    results = soup.select("div.question_box li.clearfix .an")
    #所有答案,包含好多个List,每个List包含了回答内容,回答者,回答时间
    answers = []
    for result in results:
        #获得回答内容
        ansSoup = BeautifulSoup(str(result))
        text = ansSoup.select(".answer_txt span pre")
        ansText = self.getText(str(text[0]))
        ansText = self.tool.replace(ansText)
        #获得回答者和回答时间
        info = ansSoup.select(".answer_tj")
        ansInfo = self.getOtherAnswerInfo(info[0])
        #将三者组合成一个List
        answer = [ansText, ansInfo[0], ansInfo[1], 0]
        #加入到answers
        answers.append(answer)
    return answers

#主函数
def getAnswer(self, url):
    if not url:
        url = "http://iask.sina.com.cn/b/gQiuSNCMV.html"
    page = self.getPageByURL(url)
    good_ans = self.getGoodAnswer(page)
    other_ans = self.getOtherAnswers(page)
    return [good_ans,other_ans]

page = Page()
page.getAnswer(None)

```

tool.py

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

#处理页面标签类
class Tool:

    #将超链接广告剔除
    removeADLink = re.compile('<div class="link_layer.*?</div>')
    #去除img标签,1-7位空格,&nbsp;
    removeImg = re.compile('<img.*?>| {1,7}&nbsp;')
    #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('<.*?>')
```

mysql.py

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

import MySQLdb
import time

class Mysql:

    #获取当前时间
    def getCurrentTime(self):
        return time.strftime('[%Y-%m-%d %H:%M:%S]',time.localtime())

    #数据库初始化
    def __init__(self):
        try:
            self.db = MySQLdb.connect('ip','username','password')
            self.cur = self.db.cursor()
        except MySQLdb.Error,e:
            print self.getCurrentTime(),"连接数据库错误, 原因%d:"


    #插入数据
    def insertData(self, table, my_dict):
        try:
            self.db.set_character_set('utf8')
            cols = ', '.join(my_dict.keys())
            values = '"," '.join(my_dict.values())
            sql = "INSERT INTO %s (%s) VALUES (%s)" % (table, c
            try:
                result = self.cur.execute(sql)
                insert_id = self.db.insert_id()
                self.db.commit()
                #判断是否执行成功
                if result:
                    return insert_id
                else:
                    return 0
            except MySQLdb.Error,e:
                #发生错误时回滚
                self.db.rollback()
                #主键唯一, 无法插入
                if "key 'PRIMARY'" in e.args[1]:
                    print self.getCurrentTime(),"数据已存在, 未插入"
                else:
                    print self.getCurrentTime(),"插入数据失败, 原因%d:"

        except MySQLdb.Error,e:
            print self.getCurrentTime(),"数据库错误, 原因%d: %s"
```

数据库建表SQL如下：

```
CREATE TABLE IF NOT EXISTS `iask_answers` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '自增ID',
  `text` text NOT NULL COMMENT '回答内容',
  `question_id` int(18) NOT NULL COMMENT '问题ID',
  `answerer` varchar(255) NOT NULL COMMENT '回答者',
  `date` varchar(255) NOT NULL COMMENT '回答时间',
  `is_good` int(11) NOT NULL COMMENT '是否是最佳答案',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `iask_questions` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '问题ID',
  `text` text NOT NULL COMMENT '问题内容',
  `questioner` varchar(255) NOT NULL COMMENT '提问者',
  `date` date NOT NULL COMMENT '提问时间',
  `ans_num` int(11) NOT NULL COMMENT '回答数量',
  `url` varchar(255) NOT NULL COMMENT '问题链接',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

运行的时候执行如下命令即可

```
nohup python spider.py &
```

代码写的不好，仅供大家学习参考使用，如有问题，欢迎留言交流。

运行结果查看

我们把PHP文件和log文件放在同一目录下，运行PHP文件，便可以看到如下的内容：

```
[2015-08-10 03:05:16] 保存其他答案成功
[2015-08-10 03:05:16] 113007 问号悬存在其他答案 言值。言语的价值。
[2015-08-10 03:05:16] 保存到数据库,此问题的ID为 113007
[2015-08-10 03:05:16] 当前爬取第 2 的内容,发现一个问题 谁知道言值是什么鬼东西啊? 回答数量 1
[2015-08-10 03:05:16] 保存其他答案成功
[2015-08-10 03:05:15] 113006 问号悬存在其他答案 言值指的是言语的价值，也就是说话的分量；颜值指的是容貌的价值，也就是好看不好看、漂亮不漂亮。
[2015-08-10 03:05:15] 保存到数据库,此问题的ID为 113006
[2015-08-10 03:05:15] 保存到数据库,此问题的ID为 113006
[2015-08-10 03:05:15] 当前爬取第 2 的内容,发现一个问题 谁知道言值和颜值的区别呢? 回答数量 1
[2015-08-10 03:05:13] 保存其他答案成功
[2015-08-10 03:05:13] 113005 问号悬存在其他答案 意思是, 我就是一个人, 天地都背弃了。
[2015-08-10 03:05:13] 保存到数据库,此问题的ID为 113005
[2015-08-10 03:05:13] 保存到数据库,此问题的ID为 113005
[2015-08-10 03:05:12] 我只是一个人, 天地都背弃。什么意思? 这一整句话什么意思?  
回答数量 1
[2015-08-10 03:05:12] 保存到数据库,此问题的ID为 113005
[2015-08-10 03:05:12] 当前爬取第 2 的内容,发现一个问题 谁问我暮晚倾江秋, 薄旅清欢情难收, 谁念丹青抚做弦, 塞阳红粉独自愁”这句话的含义 回答数量 0
[2015-08-10 03:05:10] 保存到数据库,此问题的ID为 113003
[2015-08-10 03:05:10] 当前爬取第 2 的内容,发现一个问题 给一些欣赏不累的句子, 如题, 不要古诗, 是句子 回答数量 0
[2015-08-10 03:05:09] 保存到数据库,此问题的ID为 113002
[2015-08-10 03:05:09] 当前爬取第 2 的内容,发现一个问题 Chris_Kim 怎么读 汉语读什么<br> 回答数量 0
[2015-08-10 03:05:07] 保存其他答案成功
[2015-08-10 03:05:07] 113001 问号悬存在其他答案 这个乐器是埙, 读音xun一串。
[2015-08-10 03:05:07] 保存到数据库,此问题的ID为 113001
[2015-08-10 03:05:07] 当前爬取第 2 的内容,发现一个问题 一种乐器我不造怎么读 就是土字旁, 右面是成员的员, 怎么读! 忽焉答案! 回答数量 1
[2015-08-10 03:05:06] 保存到数据库,此问题的ID为 113000
[2015-08-10 03:05:06] 保存到数据库,此问题的ID为 113000
[2015-08-10 03:05:04] 保存到数据库,此问题的ID为 112999
[2015-08-10 03:05:04] 保存到数据库,此问题的ID为 112998
[2015-08-10 03:05:01] 保存其他答案成功
[2015-08-10 03:05:01] 112997 问号悬存在其他答案 买个点读机, 多听一阵子声母韵母发音, 跟着读然后自己拼吧, 就当自己从头来, 先从发音开始纠正
```

小伙伴们赶快试一下吧。

Python爬虫实战七之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

1.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？

```

<html>
  <head>...</head>
  <frameset cols="20%,80%">
    <frame name="w_left" src="w_left.html" scrolling="auto" resize>
    <frameset framespacing="0" rows="90%,10%">
      <frame name="w_right" src="xk_login.html" scrolling="auto" resize>
        #document
          <html>
            <head>...</head>
            <body bgcolor="#EAE2F3">...</body>
          </html>
        </frame>
      <frame name="w_footer" src="w_footer.html" scrolling="auto" resize>
    </frameset>
  </frameset>

```

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递以及请求头，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headers的详细信息。

Name	Value
Path	/bks_login2/login?jym2005=6230324.../jwxt_bks
	/bks_login2/loginmessage
	style.css
	CSRF
	/jwxt_bks

Request Headers:

- Remote Address: 211.86.56.238:7890
- Request URL: http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bks_login2.login?jym2005=6230324099350951
- Request Method: POST
- Status Code: 302 Found
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: zh-CN,zh;q=0.8
- Cache-Control: max-age=0
- Connection: keep-alive
- Content-Length: 32
- Content-Type: application/x-www-form-urlencoded
- Host: jwxt.sdu.edu.cn:7890
- Origin: http://jwxt.sdu.edu.cn:7890
- Referer: http://jwxt.sdu.edu.cn:7890/zhxt_bks/xk_login.html
- User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.99 Safari/537.36 LBBRO
- USER

Query String Parameters:

- jym2005: 6230324099350951

Form Data:

- stuid: 201200131012
- pwd: [REDACTED]

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。

2. 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/'
        self.cookies = cookielib.CookieJar()
        self.postdata = urllib.urlencode({
            'stuid':'201200131012',
            'pwd':'xxxxxx'
        })
        self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor)

    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

```

Run demo4
D:\python2.7\python.exe D:/MyPy/demo/demo4.py
<HTML><HEAD><TITLE>登录结果</TITLE></HEAD>
<link href="/zhxt_bks/style.css" rel="stylesheet" type="text/css">
<BODY bgColor="#F1ECF9">
<span class=t>
计算机科学与技术学院 计算机科学与技术专业 崔庆才(201200131012)
<br>你的使用时间为2015-02-20 10:01, 使用地点为140.246.128.46<br>
</span>
<BR>
<span>
<H3>请您注意!</H3>
1.请节约时间
<BR>
2.离开时请注销
<BR>
3.每次登录时间限为30分钟
<BR>
</span>
</BODY>
</HTML>
<HR>
<span class="t">
登录成功!
</span>

```

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。

3. 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。

本学期课程成绩表						
排名序号	课程号	课程名	课程序号	学分	考试时间	成绩
1	0132000310	编译原理与技术	1	2.5	20150112	83
1	0132000461	操作系统设计(双语)	1	2	20150112	85
1	0132000610	汇编语言	1	2.5	20150112	91
1	013201160	计算机网络课程设计	1	2	20150112	90
1	013201310	面向对象技术	1	2.5	20150112	94
1	013201910	微机原理与接口	1	2.5	20150112	88
1	013202360	组织原理课程设计	1	2	20150112	85
335	0281000410	中国近现代史纲要	3	3	20150112	88
336	0901000510	形势政策与社会实践(S)			20150112	85

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~

0133202360	组成原理课程设计
0281000410	中国化的马克思主义
0901000510	形势政策与社会实践(5)

显示排名

```
<tr>
<td>
<font size="-1">
    &nbsp;|
    <a href="/pls/wwwbks/bkscjcx.curscope" target="w_right">本学期成绩</a>
</font>
</td>
</tr>
<tr>
```

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.curscope>，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/'
        #本学期成绩URL
        self.gradeUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/'
        self.cookies = cookielib.CookieJar()
        self.postdata = urllib.urlencode({
            'stuid':'201200131012',
            'pwd':'xxxxxx'
        })
        #构建opener
        self.opener = urllib2.build_opener(urllib2.HTTPCookieProcessor)

    #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">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">01332020360</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">02810004410</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

4. 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 = \sum (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="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>'
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。

5. 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/bk'
        #成绩URL
        self.gradeUrl = 'http://jwxt.sdu.edu.cn:7890/pls/wwwbks/bk'
        #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)
        #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.*?>(.*)')
        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.
                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](#)

希望小伙伴们加油，加深一下理解。

Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺

更新

其实本文的初衷是为了获取淘宝的非匿名旺旺，在淘宝详情页的最下方有相关评论，含有非匿名旺旺号，快一年了淘宝都没有修复这个。

可就在今天，淘宝把所有的账号设置成了匿名显示，SO，获取非匿名旺旺号已经不可能了。那本节就带大家抓取匿名旺旺号熟悉一下 Selenium吧。

2016/7/1

前言

嗯，淘宝，它一直是个难搞的家伙。

而且买家在买宝贝的时候大多数都是匿名评论的，大家都知道非匿名评论是非常有用的，比如对于大数据分析，分析某个宝贝的购买用户星级状况等等。

现在已经不能获取非匿名了，此句已没有意义了。

对于抓淘宝，相信尝试过的童鞋都能体会到抓取它到艰辛，最简单的方法莫过于模拟浏览器了，本节我们就讲解一下利用 Selenium 抓取淘宝评论的方法。

项目提供了如下功能：

- 输入淘宝关键字采集淘宝链接并写入到文件
- 从文件读取链接，执行评论采集
- 将评论和旺旺号保存到Excel中
- 记录当前采集链接索引，保存进度

准备工作

在开始本节之前

你需要了解一些基础知识，我们需要用到 Selenium 这个东西，详情请看

[Selenium用法](#)

我们首先讲解一下你需要做怎样的配置。

首先你需要安装 Python, 版本是2.7

然后需要安装的 Python 类库。

```
pip install pyquery selenium twisted requests xlrd xlwt xlutils
```

安装浏览器 Chrome, 安装浏览器 Chrome, 安装浏览器Chrome。

然后下载ChromeDriver, ChromeDriver是驱动浏览器的工具, 需要把它配置到环境变量里。

有的童鞋说, 为什么不用 PhantomJS, 因为为了防止淘宝禁掉我们, 需要登录淘宝账号, 登录过程可能会出现奇奇怪怪得验证码, 滚动条, 手机验证, 如果用 PhantomJS 的话不方便操作, 所以在这里我们就使用 Chrome 了。

[ChromeDriver](#)

上面是 ChromeDriver 的下载地址, 谷歌都上得了, 这个不在话下吧, 这是最官方的版本, 其他链接请自行搜索。

找到对应平台的 ChromeDriver, 解压后将可执行文件配置到环境变量里, 配置到环境变量里, 配置到环境变量里! 重要的话说三遍。

流程简述

首先我们拿一个例子来演示一下全过程。

随意打开天猫一个链接

[示例链接](#)

我们首先观察一下评论, 可以发现所有的评论都是匿名的。即使这个用户不是匿名评论的, 那也会显示匿名, 淘宝这保密做的挺好。

全部 追评 (633) 图片 (702) 有内容

一开始我没能弄准自己的尺寸导致宝贝尺码大了不正好在售后服务很好很快就帮我调好了很满意的一次购物，很期待下次合作
 今天

质量很好，做工不错，没有色差，物流快，穿上身很显瘦，服务很贴心，姐妹们都说好看

 今天

衣服料子很好，颜色也很好看，很喜欢，客服也很好说话很热情，很不错，好评
 今天

初次评价：衣服质量很好，做工不错，面料很柔和，没有色差，物流很快，很百搭
 今天

收货当天追加：穿著很显身材，不起球，不褪色，值得拥有

衣服收到了，质量不错，又很时尚，真心的喜欢，下次还会继续光顾！
 今天

初次评价：第一次给差评，服务不好，说好的换衣服，衣服都寄回去十多天了，现在联系客服倒不回信息了，很不满意
 今天

收货6天后追加：刚刚老板打来电话，態度很好，解釋為什麼沒到，竟是我誤會了，不好意思了老板，因為質量很好，所以一直在等這條裙子

初次评价：衣服质量很好，穿上去很合身，朋友看了很喜欢，值得购买的衣服，谢谢卖家！
 今天

收货3天后追加：衣服非常棒，洗完也不褪色，姐妹们很喜欢，非常满意的一次购物！

衣服很漂亮很喜欢朋友见了让我给她定一件这个價位不算貴挺好的
 今天

初次评价：没有色差的，照片拍得较灰，质量款式都不错，身材好的话会和图片一样，我105斤，158厘米，中號的有些偏大，不過小號的話可能就小了，非常喜欢


收货当天追加：姐妹看到了说非常可爱，不过我不给她链接的，因为她的身材比我好，不能忍受，谁讓誰尴尬啊

刚收到衣服迫不及待的打开试穿了一下，颜色很正，料子手感不错，穿着特别合身，太满意

心机的淘宝啊，那我们如果想获取一些旺旺号该咋办？

接下来我们返回宝贝详情页面，然后一直下拉下拉，拉到最最后，可以看到有个“看了又看”板块。

看了又看



有没有！！发现了新大陆，这是什么？这是此宝贝相关宝贝以及它的一些评论。

看到了有非匿名用户了，哈哈哈，淘宝加密了评论，推荐部分却没有加密。

嗯，就从这里，我们把它们的旺旺号都抓下来，顺便把评论和购买的宝贝抓下来。

现在已经全部改成了匿名，上述话已经无意义了。

那么抓取完之后，保存到哪里呢？为了便于管理和统计，在这里保存到 Excel 中，那么就需要用到 xlrd, xlwt, xlutils 等库。

嗯，动机就是这样。

实战爬取

抓取过程

首先我们观察这个链接，在最初的时候，其实网页并没有加载最下方的“看了又看”内容的，慢慢往下滑动网页，滑到最下方之后，才发现看了又看页面才慢慢加载出来。

很明显，这个地方使用了Ajax，由于我们用的是Selenium，所以这里我们不能直接来模拟Ajax的Request，需要我们来模拟真实的用户操作。

所以我们要模拟的就是，在网页部分加载出来之后，模拟浏览器滑动到下方，使“看了又看”内容显示出来，然后获取网页源代码，解析之即可。

那么在这里就出现了两个至关重要的点，一个是判断网页框架大体加载出来，另一个是模拟滑动直到最下方的内容加载出来。

首先，我们解决第一个问题，怎样判断网页框架大体加载出来。我们可以用网页中的某个元素的出现与否来判断。

比如



这一部分是否加载出来。

审查一下代码，ID叫做J_TabBarBox，好，那就用它来作为网页初步加载成功的标志。

在Selenium中，我们用显式等待的方法来判断该元素是否已经加载成功。

```
try:  
    driver.get(url)  
    WebDriverWait(driver, timeout).until(  
        EC.presence_of_element_located((By.ID, "J_TabBarBox"))  
    )  
except TimeoutException:  
    return False  
if is_recommends_appear(driver, max_scroll_time):  
    print u'已经成功加载出下方橱窗推荐宝贝信息'  
    return driver.page_source
```

接下来我们需要模拟下拉浏览器，不妨直接下拉到底部，再从底部向上拉，可能需要下拉多次，所以在这里定义了一个下拉次数，那么判断“看了又看”正文内容是否出现依然可以用显式等待的方法。

浏览器审查元素发现它的选择器是 #J_TjWaterfall li

```

s Network Sources Timeline Profiles Resources Audits Security Console AdBlock


</div>


</div>


</div>


</div>


</div>


<div class="hd">看了又看</div>
<div id="J_Tjwrap">
<ul id="J_TjWaterfall" class="clearfix alid-03040" style="height: 2920px;">
-


```

那么可以用如下方法来判断是否加载成功

```

try:
    driver.find_element_by_css_selector('#J_TjWaterfall li')
except NoSuchElementException:
    return False
return True

```

下拉过程可以用执行 JavaScript 的方法实现。

```

js = "window.scrollTo(0,document.body.scrollHeight-" + str(count)
driver.execute_script(js)

```

其中 count 是下拉的次数，经过测试之后，每次拉动距离和 count 是平方关系比较科学，具体不再描述，当然你可以改成自己想要的数值。

嗯，加载出来之后，就可以用

```
driver.page_source
```

来获取网页源代码了

用 pyquery 解析即可。

```
doc = pq(html)
items = doc('#J_TjWaterfall > li')
print u'分析得到下方宝贝中的用户评论：'
for item in items.items():
    url = item.find('a').attr('href')
    if not url.startswith('http'):
        url = 'https:' + url
    comments_info = []
    comments = item.find('p').items()
    for comment in comments:
        comment_user = comment.find('b').remove().text()
        comment_content = comment.text()
        anonymous_str = config.ANONYMOUS_STR
        <del>if not anonymous_str in comment_user:</del> #此句
        comments_info.append((comment_content, comment_user))
    info.append({'url': url, 'comments_info': comments_info})
return info
```

然后保存到 Excel 中。

运行结果截图

The terminal output shows the following steps:

- Script path: /System/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 /private/var/py/TaobaoUser/from_file.py
- Success message: 共有 1019 个链接
- Process status: 上次爬取到第 5 个链接，继续爬取。输入 2 重新爬取；输入 1 继续爬取，输入 2 重新爬取；
- Progress: 正在爬取第 5 个网页，共 1019 个
- Start of analysis loop:
 - 分析得到下方宝贝中的用户评论：comment_user 11037452759
 - 正在检测 11037452759 是否存在于文件中
 - 用户名 excel 中不存在
 - 准备将 11037452759 写入文件
 - 正在写入文件中 11037452759
 - 正在检测 11037452759 是否存在于文件中
 - 用户名在excel中不存在
 - 准备成功输出下方宝贝推荐宝贝信息
 - 分析得到下方宝贝中的用户评论：
- Comment processing loop:
 - comment_user 11037452759
 - 正在检测 11037452759 是否存在于文件中
 - 用户名 excel 中不存在
 - 准备将 11037452759 写入文件
 - 正在写入文件中 11037452759
 - 正在检测 11037452759 是否存在于文件中
 - 用户名在excel中不存在
 - 准备成功输出下方宝贝推荐宝贝信息
 - 分析得到下方宝贝中的用户评论：
- File saving loop:
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 已成功写入到文件 file/result.xls 第 87 行
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 已成功写入到文件 file/result.xls 第 87 行
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 已成功写入到文件 file/result.xls 第 87 行
 - comment_user 小码仔
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 准备将 小码仔 写入文件
 - 正在写入文件中 小码仔
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 已成功写入到文件 file/result.xls 第 88 行
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 已成功写入到文件 file/result.xls 第 88 行
 - comment_user 小码仔
 - 正在检测 /file/result.xls 是否存在于文件中
 - 用户名在excel中不存在
 - 准备将 恋爱418 写入文件
 - 正在写入文件中 恋爱418

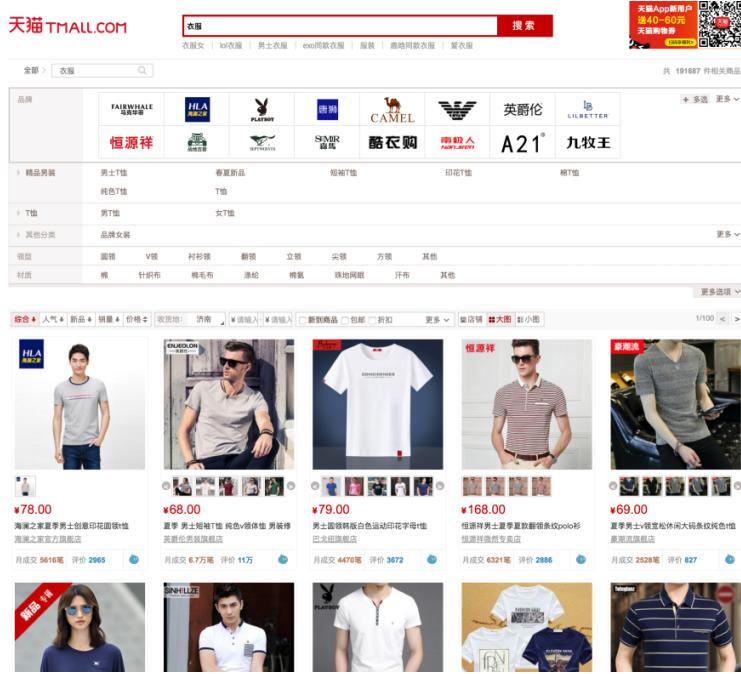
可以发现，另外提供了先登陆后爬取的功能，然后保存了爬取进度。

采集链接

刚才我们测试的链接是哪里来的？我们不能一个个去找吧？所以，在这里又提供了一个采集链接的过程，将采集的链接保存到文本，然后抓取的时候从文本读取一个个链接即可。

所以在这里我们模拟搜索的过程，关键字让用户输入，将搜索的链接采集下来。

在此 Selenium 模拟了输入文字，点击按钮和翻页的功能。



核心代码如下

下面的方法模拟了加载出搜索框之后输入文字点击回车的过程，将网页的结果返回。

```
def get_results(keyword):
    driver = config.DRIVER
    link = config.SEARCH_LINK
    driver.get(link)
    try:
        WebDriverWait(driver, config.TIMEOUT).until(
            EC.presence_of_element_located((By.ID, "mq"))
        )
    except TimeoutException:
        print u'加载页面失败'
    try:
        element = driver.find_element_by_css_selector('#mq')
        print u'成功找到了搜索框'
        keyword = keyword.decode('utf-8', 'ignore')
        print keyword
        print u'输入关键字', keyword
        for word in keyword:
            print word
            element.send_keys(word)
        element.send_keys(Keys.ENTER)
    except NoSuchElementException:
        print u'没有找到搜索框'
    print u'正在查询该关键字'
    try:
        WebDriverWait(driver, config.TIMEOUT).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "#J"))
        )
    except TimeoutException:
        print u'查询失败'
    html = driver.page_source
    return html
```

下面的方法模拟了翻页的过程，到指定的翻页数目为止

```
def get_more_link():
    print u'正在采集下一页的宝贝链接'
    driver = config.DRIVER
    try:
        js = "window.scrollTo(0,document.body.scrollHeight)"
        driver.execute_script(js)
    except WebDriverException:
        print u'页面下拉失败'
    try:
        next = driver.find_element_by_css_selector('#content b.u')
        next.click()
    except NoSuchElementException:
        print u'找到了翻页按钮'
    driver.implicitly_wait(5)
    try:
        WebDriverWait(driver, config.TIMEOUT).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "#J"))
        )
    except TimeoutException:
        print u'查询失败'
    html = driver.page_source
    parse_html(html)
```



运行结果截图

正在清空爬取链接，等待重新爬取新链接
成功找到了搜索框
衣服
输入关键字 衣服
衣
服
正在查询该关键字
当前已采集 1 个链接
当前已采集 2 个链接
当前已采集 3 个链接
当前已采集 4 个链接
当前已采集 5 个链接
当前已采集 6 个链接
当前已采集 7 个链接
当前已采集 8 个链接
当前已采集 9 个链接
当前已采集 10 个链接
当前已采集 11 个链接
当前已采集 12 个链接
当前已采集 13 个链接
当前已采集 14 个链接
当前已采集 15 个链接
当前已采集 16 个链接
当前已采集 17 个链接
当前已采集 18 个链接
当前已采集 19 个链接
当前已采集 20 个链接
当前已采集 21 个链接
当前已采集 22 个链接
当前已采集 23 个链接
当前已采集 24 个链接
当前已采集 25 个链接

采集到的内容保存到 urls.txt 中

```

//detail.tmall.com/item.htm?id=529722757519&skuId=3129541850881&areaId=370700&cat_id=26&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=439874725685&skuId=81640478015&areaId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=444669818985&skuId=46166662694376102&readId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=529065556355&skuId=46116662743954283&readId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5291168794885&skuId=3166787077373&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=522129854625&skuId=3128332840794&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=4498581286655&skuId=86301086766&areaId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=4498581286655&skuId=86301086766&areaId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=3880751437265&skuId=3158496448855&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5302945258955&skuId=3138127781811&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5273358354155&skuId=3138127781811&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=52712480104655&skuId=3136333518193&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=4394469395265&skuId=46116662743954283&readId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=44299118931&skuId=3135166871577&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=996574937165&skuId=318097195153&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5282406013645&skuId=31555008345926&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=4456448369155&skuId=816351381266&areaId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=529799449012755&skuId=31423448316866&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=52966124709955&skuId=31701631543143&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5277579733355&skuId=3148537203509&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=528600718711555&skuId=3147233612090&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5270536478775&skuId=31502162226676&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=4452957607575&skuId=97738566716&areaId=370700&cat_id=2&n=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5277579733355&skuId=3148537203509&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=528600718711555&skuId=3147233612090&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=5271162435555&skuId=3136912467951&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=528257408035&skuId=3140918678505&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=52589210704655&skuId=31306770185676&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed
//detail.tmall.com/item.htm?id=52589210704655&skuId=31306770185676&areaId=370700&cat_id=2&rn=323cf53c6799bf37a452ed

```

嗯，这下采集链接和爬取链接都有了。

代码放送

扯了这么多，许多童鞋已经蠢蠢欲动了，大声告诉我你们想要的是什么？

哦没错！代码！

嗯在这呢！

[代码](#)

附加扯淡

嗯想说一句，在这里还提供了一些可配置项，比如翻页最大次数，超时时间，下拉次数，登录链接等等。

都可以在 config.py 中配置。

- URLs_FILE 保存链接单的文件
- OUT_FILE 输出文本EXCEL路径
- COUNT_TXT 计数文件
- DRIVER 浏览器驱动
- TIMEOUT 采集超时时间
- MAX_SCROLL_TIME 下拉滚动条最大次数
- NOW_URL_COUNT 当前采集到第几个链接
- LOGIN_URL 登录淘宝的链接
- SEARCH_LINK 采集淘宝链接搜索页面
- CONTENT 采集链接临时变量
- PAGE 采集淘宝链接翻页数目
- FILTER_SHOP 是否过滤相同店铺
- ANONYMOUS_STR 匿名用户标志，已失效

哦，对了，程序怎么用啊？看 README！

**年度重磅大放送！博主录制的Python3
爬虫视频教程出炉啦！！！欢迎大家支
持！！！详情请看：**

[Python3爬虫视频学习教程](#)

[自己动手，丰衣足食！Python3网络爬虫实战案例](#)

以下为Python2爬虫系列教程：

大家好哈，我呢最近在学习Python爬虫，感觉非常有意思，真的让生活可以方便很多。学习过程中我把一些学习的笔记总结下来，还记录了一些自己实际写的一些小爬虫，在这里跟大家一同分享，希望对Python爬虫感兴趣的童鞋有帮助，如果有机会期待与大家的交流。

Python版本：2.7

一、爬虫入门

1. [Python爬虫入门一之综述](#)
2. [Python爬虫入门二之爬虫基础了解](#)
3. [Python爬虫入门三之Urllib库的基本使用](#)
4. [Python爬虫入门四之Urllib库的高级用法](#)
5. [Python爬虫入门五之URLError异常处理](#)
6. [Python爬虫入门六之Cookie的使用](#)
7. [Python爬虫入门七之正则表达式](#)

二、爬虫实战

1. [Python爬虫实战一之爬取糗事百科段子](#)
2. [Python爬虫实战二之爬取百度贴吧帖子](#)
3. [Python爬虫实战三之实现山东大学无线网络掉线自动重连](#)
4. [Python爬虫实战四之抓取淘宝MM照片](#)
5. [Python爬虫实战五之模拟登录淘宝并获取所有订单](#)
6. [Python爬虫实战六之抓取爱问知识人问题并保存至数据库](#)
7. [Python爬虫实战七之计算大学本学期绩点](#)

[8. Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺](#)

三、爬虫利器

- [1. Python爬虫利器一之Requests库的用法](#)
- [2. Python爬虫利器二之Beautiful Soup的用法](#)
- [3. Python爬虫利器三之Xpath语法与lxml库的用法](#)
- [4. Python爬虫利器四之PhantomJS的用法](#)
- [5. Python爬虫利器五之Selenium的用法](#)
- [6. Python爬虫利器六之PyQuery的用法](#)

四、爬虫进阶

- [1. Python爬虫进阶一之爬虫框架概述](#)
- [2. Python爬虫进阶二之PySpider框架安装配置](#)
- [3. Python爬虫进阶三之爬虫框架Scrapy安装配置](#)
- [4. Python爬虫进阶四之PySpider的用法](#)
- [5. Python爬虫进阶五之多线程的用法](#)
- [6. Python爬虫进阶六之多进程的用法](#)
- [7. Python爬虫进阶七之设置ADSL拨号服务器代理](#)

目前暂时是这些文章，随着学习的进行，会不断更新哒，敬请期待~

希望对大家有所帮助，谢谢！

转载请注明：[静觅](#) » [Python爬虫学习系列教程](#)

Python爬虫利器一之**Requests**库的用法

前言

之前我们用了 `urllib` 库，这个作为入门的工具还是不错的，对了解一些爬虫的基本理念，掌握爬虫爬取的流程有所帮助。入门之后，我们就需要学习一些更加高级的内容和工具来方便我们的爬取。那么这一节来简单介绍一下 `requests` 库的基本用法。

注：Python 版本依然基于 2.7

官方文档

以下内容大多来自于官方文档，本文进行了一些修改和总结。要了解更多可以参考[官方文档](#)

安装

利用 `pip` 安装

```
$ pip install requests
```

或者利用 `easy_install`

```
$ easy_install requests
```

通过以上两种方法均可以完成安装。

引入

首先我们引入一个小例子来感受一下

```
import requests

r = requests.get('http://cuiqingcai.com')
print type(r)
print r.status_code
print r.encoding
#print r.text
print r.cookies
```

以上代码我们请求了本站点的网址，然后打印出了返回结果的类型，状态码，编码方式，Cookies等內容。

运行结果如下

```
<class 'requests.models.Response'>
200
UTF-8
<RequestsCookieJar[]>
```

怎样，是不是很方便。别急，更方便的在后面呢。

基本请求

requests库提供了http所有的基本请求方式。例如

```
r = requests.post("http://httpbin.org/post")
r = requests.put("http://httpbin.org/put")
r = requests.delete("http://httpbin.org/delete")
r = requests.head("http://httpbin.org/get")
r = requests.options("http://httpbin.org/get")
```

嗯，一句话搞定。

基本GET请求

最基本的GET请求可以直接用get方法

```
r = requests.get("http://httpbin.org/get")
```

如果想要加参数，可以利用 params 参数

```
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get("http://httpbin.org/get", params=payload)
print r.url
```

运行结果

```
http://httpbin.org/get?key2=value2&key1=value1
```

如果想请求JSON文件，可以利用 json() 方法解析

例如自己写一个JSON文件命名为a.json，内容如下

```
["foo", "bar", {
    "foo": "bar"
}]
```

利用如下程序请求并解析

```
import requests

r = requests.get("a.json")
print r.text
print r.json()
```

运行结果如下，其中一个是直接输出内容，另外一个方法是利用 json() 方法解析，感受下它们的不同

```
["foo", "bar", {
    "foo": "bar"
}]
[u'foo', u'bar', {u'foo': u'bar'}]
```

如果想获取来自服务器的原始套接字响应，可以取得 r.raw。不过需要在初始请求中设置 stream=True。

```
r = requests.get('https://github.com/timeline.json', stream=True
r.raw
<requests.packages.urllib3.response.HTTPResponse object at 0x101
r.raw.read(10)
'\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x00\x03'
```

这样就获取了网页原始套接字内容。

如果想添加 headers，可以传 headers 参数

```
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
headers = {'content-type': 'application/json'}
r = requests.get("http://httpbin.org/get", params=payload, headers=headers)
print r.url
```

通过headers参数可以增加请求头中的headers信息

基本POST请求

对于 POST 请求来说，我们一般需要为它增加一些参数。那么最基本的传参方法可以利用 `data` 这个参数。

```
import requests

payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.post("http://httpbin.org/post", data=payload)
print r.text
```

运行结果

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key1": "value1",
    "key2": "value2"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Content-Length": "23",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.9.1"
  },
  "json": null,
  "url": "http://httpbin.org/post"
}
```

可以看到参数传成功了，然后服务器返回了我们传的数据。

有时候我们需要传送的信息不是表单形式的，需要我们传JSON格式的数据过去，所以我们可以用 `json.dumps()` 方法把表单数据序列化。

```
import json
import requests

url = 'http://httpbin.org/post'
payload = {'some': 'data'}
r = requests.post(url, data=json.dumps(payload))
print r.text
```

运行结果

```
{  
    "args": {},  
    "data": "{\"some\": \"data\"}",  
    "files": {},  
    "form": {},  
    "headers": {  
        "Accept": "*/*",  
        "Accept-Encoding": "gzip, deflate",  
        "Content-Length": "16",  
        "Host": "httpbin.org",  
        "User-Agent": "python-requests/2.9.1"  
    },  
    "json": {  
        "some": "data"  
    },  
    "url": "http://httpbin.org/post"  
}
```

通过上述方法，我们可以POST JSON格式的数据

如果想要上传文件，那么直接用 `file` 参数即可

新建一个 `a.txt` 的文件，内容写上 Hello World!

```
import requests  
  
url = 'http://httpbin.org/post'  
files = {'file': open('test.txt', 'rb')}  
r = requests.post(url, files=files)  
print r.text
```

可以看到运行结果如下

```
{  
    "args": {},  
    "data": "",  
    "files": {  
        "file": "Hello World!"  
    },  
    "form": {},  
    "headers": {  
        "Accept": "*/*",  
        "Accept-Encoding": "gzip, deflate",  
        "Content-Length": "156",  
        "Content-Type": "multipart/form-data; boundary=7d8eb5ff99a04",  
        "Host": "httpbin.org",  
        "User-Agent": "python-requests/2.9.1"  
    },  
    "json": null,  
    "url": "http://httpbin.org/post"  
}
```

这样我们便成功完成了一个文件的上传。

`requests` 是支持流式上传的，这允许你发送大的数据流或文件而无需先把它们读入内存。要使用流式上传，仅需为你的请求体提供一个类文件对象即可

```
with open('massive-body') as f:  
    requests.post('http://some.url/streamed', data=f)
```

这是一个非常实用方便的功能。

Cookies

如果一个响应中包含了`cookie`，那么我们可以利用 `cookies` 变量来拿到

```
import requests  
  
url = 'http://example.com'  
r = requests.get(url)  
print r.cookies  
print r.cookies['example_cookie_name']
```

以上程序仅是样例，可以用 `cookies` 变量来得到站点的 `cookies`

另外可以利用 `cookies` 变量来向服务器发送 `cookies` 信息

```
import requests

url = 'http://httpbin.org/cookies'
cookies = dict(cookies_are='working')
r = requests.get(url, cookies=cookies)
print r.text
```

运行结果

```
'{"cookies": {"cookies_are": "working"}}'
```

可以已经成功向服务器发送了 cookies

超时配置

可以利用 `timeout` 变量来配置最大请求时间

```
requests.get('http://github.com', timeout=0.001)
```

注：`timeout` 仅对连接过程有效，与响应体的下载无关。

也就是说，这个时间只限制请求的时间。即使返回的 `response` 包含很大内容，下载需要一定时间，然而这并没有什么卵用。

会话对象

在以上的请求中，每次请求其实都相当于发起了一个新的请求。也就是相当于我们每个请求都用了不同的浏览器单独打开的效果。也就是它并不是指的一个会话，即使请求的是同一个网址。比如

```
import requests

requests.get('http://httpbin.org/cookies/set/sessioncookie/12345'
r = requests.get("http://httpbin.org/cookies")
print(r.text)
```

结果是

```
{
  "cookies": {}
}
```

很明显，这不在一个会话中，无法获取 **cookies**，那么在一些站点中，我们需要保持一个持久的会话怎么办呢？就像用一个浏览器逛淘宝一样，在不同的选项卡之间跳转，这样其实就是一个长会话。

解决方案如下

```
import requests

s = requests.Session()
s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')
r = s.get("http://httpbin.org/cookies")
print(r.text)
```

在这里我们请求了两次，一次是设置 **cookies**，一次是获得 **cookies**

运行结果

```
{
  "cookies": {
    "sessioncookie": "123456789"
  }
}
```

发现可以成功获取到 **cookies** 了，这就是建立一个会话的作用。体会一下。

那么既然会话是一个全局的变量，那么我们肯定可以用来全局的配置了。

```
import requests

s = requests.Session()
s.headers.update({'x-test': 'true'})
r = s.get('http://httpbin.org/headers', headers={'x-test2': 'true'})
print r.text
```

通过 **s.headers.update** 方法设置了 **headers** 的变量。然后我们又在请求中设置了一个 **headers**，那么会出现什么结果？

很简单，两个变量都传送过去了。

运行结果

```
{  
    "headers": {  
        "Accept": "*/*",  
        "Accept-Encoding": "gzip, deflate",  
        "Host": "httpbin.org",  
        "User-Agent": "python-requests/2.9.1",  
        "X-Test": "true",  
        "X-Test2": "true"  
    }  
}
```

如果get方法传的headers 同样也是 x-test 呢？

```
r = s.get('http://httpbin.org/headers', headers={'x-test': 'true'})
```

嗯， 它会覆盖掉全局的配置

```
{  
    "headers": {  
        "Accept": "*/*",  
        "Accept-Encoding": "gzip, deflate",  
        "Host": "httpbin.org",  
        "User-Agent": "python-requests/2.9.1",  
        "X-Test": "true"  
    }  
}
```

那如果不想要全局配置中的一个变量了呢？很简单，设置为 `None` 即可

```
r = s.get('http://httpbin.org/headers', headers={'x-test': None})
```

运行结果

```
{  
    "headers": {  
        "Accept": "*/*",  
        "Accept-Encoding": "gzip, deflate",  
        "Host": "httpbin.org",  
        "User-Agent": "python-requests/2.9.1"  
    }  
}
```

嗯， 以上就是 `session` 会话的基本用法

SSL证书验证

现在随处可见 https 开头的网站，Requests可以为HTTPS请求验证SSL证书，就像web浏览器一样。要想检查某个主机的SSL证书，你可以使用 verify 参数

现在 12306 证书不是无效的嘛，来测试一下

```
import requests

r = requests.get('https://kyfw.12306.cn/otn/', verify=True)
print r.text
```

结果

```
requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] c
```

果真如此

来试下 github 的

```
import requests

r = requests.get('https://github.com', verify=True)
print r.text
```

嗯，正常请求，内容我就不输出了。

如果我们想跳过刚才 12306 的证书验证，把 verify 设置为 False 即可

```
import requests

r = requests.get('https://kyfw.12306.cn/otn/', verify=False)
print r.text
```

发现就可以正常请求了。在默认情况下 verify 是 True，所以如果需要的话，需要手动设置下这个变量。

代理

如果需要使用代理，你可以通过为任意请求方法提供 proxies 参数来配置单个请求

```
import requests

proxies = {
    "https": "http://41.118.132.69:4433"
}
r = requests.post("http://httpbin.org/post", proxies=proxies)
print r.text
```

也可以通过环境变量 `HTTP_PROXY` 和 `HTTPS_PROXY` 来配置代理

```
export HTTP_PROXY="http://10.10.1.10:3128"
export HTTPS_PROXY="http://10.10.1.10:1080"
```

通过以上方式，可以方便地设置代理。

API

以上讲解了 `requests` 中最常用的参数，如果需要用到更多，请参考官方文档 [API](#)

结语

以上总结了一下 `requests` 的基本用法，如果你对爬虫有了一定的基础，那么肯定可以很快上手，在此就不多赘述了。

练习才是王道，大家尽快投注于实践中吧。

Python爬虫利器二之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。

1. 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、html5lib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~

2. 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。

可以利用 pip 或者 easy_install 来安装，以下两种方法均可

```
easy_install beautifulsoup4
```

```
pip install beautifulsoup4
```

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
```

然后需要安装 **lxml**

```
easy_install lxml
```

```
pip install lxml
```

另一个可供选择的解析器是纯Python实现的 **html5lib**，**html5lib**的解析方式与浏览器相同,可以选择下列方法来安装**html5lib**:

```
easy_install html5lib
```

```
pip install html5lib
```

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。

解析器	使用方法	优势	劣势
Python 标准库	<code>BeautifulSoup(markup, "html.parser")</code>	Python 的内置标准库 执行速度适中 文档容错能力强	Python 2.7.3 or 3.2.2)前的版本中 文档容错能力差
lxml HTML 解析器	<code>BeautifulSoup(markup, "lxml")</code>	速度快 文档容错能力强	需要安装 C 语言库
lxml XML 解析器	<code>BeautifulSoup(markup, ["lxml", "xml"])BeautifulSoup(markup, "xml")</code>	速度快 唯一支持 XML 的解析器	需要安装 C 语言库
html5lib	<code>BeautifulSoup(markup, "html5lib")</code>	最好的容错性 以浏览器的方式解析 文档生成 HTML5 格式的 文档	速度慢 不依赖外部扩展

3. 开启Beautiful Soup 之旅

在这里先分享[官方文档](#)，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。

4. 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 sister
<a href="http://example.com/elsie" class="sister" id="link1"><!--
<a href="http://example.com/lacie" class="sister" id="link2">Lac
<a href="http://example.com/tillie" class="sister" id="link3">Ti
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。

5. 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

(1) 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">Els
```

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
#<head><title>The Dormouse's story</title></head>

print soup.a
#<a class="sister" href="http://example.com/elsie" id="link1"><!
```



```
print soup.p
#<p class="title" name="dromouse"><b>The Dormouse's story</b></p
```

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的第一个符合要求的标签，如果要查询所有的标签，我们在后面进行介绍。

我们可以验证一下这些对象的类型

```
print type(soup.a)
#<class 'bs4.element.Tag'>
```

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
#<p class="newClass" name="dromouse"><b>The Dormouse's story</b>
```

还可以对这个属性进行删除，例如

```
del soup.p['class']
print soup.p
#<p name="dromouse"><b>The Dormouse's story</b></p>
```

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，如果有需要，请查看前面提供的官方文档

(2) 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 可以遍历的字符串，不过我们最好还是称它英文名字吧。

来检查一下它的类型

```
print type(soup.p.string)  
#<class 'bs4.element.NavigableString'>
```

(3) 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型，名称，以及属性来感受一下

```
print type(soup.name)  
#<type 'unicode'>  
print soup.name  
# [document]  
print soup.attrs  
#{ } 空字典
```

(4) 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，可能会对我们的文本处理造成意想不到的麻烦。

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

```
<a class="sister" href="http://example.com/elsie" id="link1"><!--  
Elsie  
<class 'bs4.element.Comment'>
```

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来不必要的麻烦。

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。

6. 遍历文档树

(1) 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  
#[<title>The Dormouse's story</title>]
```

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

`.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  
#<listiterator object at 0x7f71457f5710>
```

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
```

```
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>  
  
<p class="story">Once upon a time there were three little sister  
<a class="sister" href="http://example.com/elsie" id="link1"><!--  
<a class="sister" href="http://example.com/lacie" id="link2">Lac
```

(2) 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，我们也需要遍历获取其中的内容。

```
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。

```
<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 sister  
<a class="sister" href="http://example.com/elsie" id="link1"><!--  
<a class="sister" href="http://example.com/lacie" id="link2">Lac
```

(3) 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tag也可以使用 .string 方法,输出结果与当前唯一子节点的 .string 结果相同。

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 .string 也会返回最里面的内容。例如

```
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
```

(4) 多个内容

知识点: .strings .stripped_strings 属性 .strings

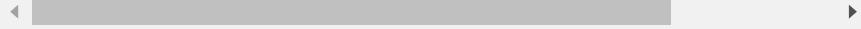
获取多个内容, 不过需要遍历获取, 比如下面的例子

```
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 th  
    # 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 th  
    # u'Elsie'  
    # u','  
    # u'Lacie'  
    # u'and'  
    # u'Tillie'  
    # u';\nand they lived at the bottom of a well.'  
    # u'...'
```



(5) 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
```

(6) 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]
```

(7) 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, 如果节点不存在, 则返回 `None`

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, 所以得到的结果可能是空白或者换行

```
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
#<p class="story">Once upon a time there were three little sist
#<a class="sister" href="http://example.com/elsie" id="link1"><!
#<a class="sister" href="http://example.com/lacie" id="link2">La
#<a class="sister" href="http://example.com/tillie" id="link3">T
#and they lived at the bottom of a well.</p>
#下一个节点的下一个兄弟节点是我们可以看到的节点
```

(8) 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'
    # <a class="sister" href="http://example.com/lacie" id="link
    # u' and\n'
    # <a class="sister" href="http://example.com/tillie" id="lin
    # u'; and they lived at the bottom of a well.'
    # None
```

(9) 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

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

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

(10) 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 bottom of a well.'  
# u'\n\n'  
# <p class="story">...</p>  
# u'...'  
# u'\n'  
# None
```

以上是遍历文档树的基本用法。

7. 搜索文档树

(1) `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

1) `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

A.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中所有的 `` 标签

```
soup.find_all('b')
# [<b>The Dormouse's story</b>]

print soup.find_all('a')
#[<a class="sister" href="http://example.com/elsie" id="link1"><

```

B.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 `<body>` 和 `` 标签都应该被找到

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

C.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 `` 标签

```
soup.find_all(["a", "b"])
# [<b>The Dormouse's story</b>,
#   <a class="sister" href="http://example.com/elsie" id="link1">
#     <a class="sister" href="http://example.com/lacie" id="link2">
#       <a class="sister" href="http://example.com/tillie" id="link3">
```

D.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
```

E. 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,如果不是则返回 `False`

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)
# [<p class="title"><b>The Dormouse's story</b></p>,
#   <p class="story">Once upon a time there were...</p>,
#   <p class="story">...</p>]
```

2) 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` 的参数,Beautiful Soup会搜索每个tag的"id"属性

```
soup.find_all(id='link2')
# [<a class="sister" href="http://example.com/lacie" id="link2">
```

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

◀ ▶

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')
# [<a class="sister" href="http://example.com/elsie" id="link1">
```

◀ ▶

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")
# [<a class="sister" href="http://example.com/elsie" id="link1">
#   <a class="sister" href="http://example.com/lacie" id="link2">
#   <a class="sister" href="http://example.com/tillie" id="link3"
```

◀ ▶

有些tag属性在搜索不能使用,比如HTML5中的 data-* 属性

```
data_soup = BeautifulSoup('<div data-foo="value">foo!</div>')
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"})
# [<div data-foo="value">foo!</div>]
```

3) 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"]
```

4) 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.效果与 SQL 中的 `limit` 关键字类似,当搜索到的结果数量达到 `limit` 的限制时,就停止搜索返回结果.

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)
# [<a class="sister" href="http://example.com/elsie" id="link1">
#   <a class="sister" href="http://example.com/lacie" id="link2">
```

5) 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 `recursive=False`.

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")
# [<title>The Dormouse's story</title>]

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

(2) **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

(3) **find_parents() find_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,搜索方法与普通tag的搜索方法相同,搜索文档搜索文档包含的内容

(4) **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后面的兄弟节点,`find_next_sibling()` 只返回符合条件的后面的第一个`tag`节点

(5) **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()` 方法返回所有符合条件的前面的兄弟节点, `find_previous_sibling()` 方法返回第一个符合条件的前面的兄弟节点

(6) **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, `find_next()` 方法返回第一个符合条件的节点

(7) **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点, `find_previous()`方法返回第一个符合条件的节点

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。

8.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 `soup.select()`，返回类型是 `list`

（1）通过标签名查找

```
print soup.select('title')
#[<title>The Dormouse's story</title>]

print soup.select('a')
#[<a class="sister" href="http://example.com/elsie" id="link1"><

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

（2）通过类名查找

```
print soup.select('.sister')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

（3）通过 `id` 名查找

```
print soup.select('#link1')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

（4）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，二者需要用空格分开

```
print soup.select('p #link1')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

直接子标签查找

```
print soup.select("head > title")
#[<title>The Dormouse's story</title>]
```

(5) 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"]')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

```
print soup.select('a[href="http://example.com/elsie"]')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

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"]')
#[<a class="sister" href="http://example.com/elsie" id="link1"><
```

以上的 `select` 方法返回的结果都是列表形式，可以遍历形式输出，然后用 `get_text()` 方法来获取它的内容。

```
soup = BeautifulSoup(html, 'lxml')
print type(soup.select('title'))
print soup.select('title')[0].get_text()

for title in soup.select('title'):
    print title.get_text()
```

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！

Python爬虫利器三之Xpath语法与lxml库的用法

前言

前面我们介绍了 BeautifulSoup 的用法，这个已经是非常强大的库了，不过还有一些比较流行的解析库，例如 lxml，使用的是 Xpath 语法，同样是效率比较高的解析方法。如果大家对 BeautifulSoup 使用不太习惯的话，可以尝试下 Xpath。

参考来源

[lxml用法](#)源自 [lxml python](#) 官方文档，更多内容请直接参阅官方文档，本文对其进行翻译与整理。

XPath语法参考 [w3school](#)

安装

```
pip install lxml
```

利用 pip 安装即可

XPath语法

XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 都构建于 XPath 表达之上。

节点关系

(1) 父 (Parent)

每个元素以及属性都有一个父。

在下面的例子中，book 元素是 title、author、year 以及 price 元素的父：

```
<book>
  <title>Harry Potter</title>
  <author>J. K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

(2) 子 (Children)

元素节点可有零个、一个或多个子。

在下面的例子中，`title`、`author`、`year` 以及 `price` 元素都是 `book` 元素的子：

```
<book>
  <title>Harry Potter</title>
  <author>J. K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

(3) 同胞 (Sibling)

拥有相同的父的节点

在下面的例子中，`title`、`author`、`year` 以及 `price` 元素都是同胞：

```
<book>
  <title>Harry Potter</title>
  <author>J. K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

(4) 先辈 (Ancestor)

某节点的父、父的父，等等。

在下面的例子中，`title` 元素的先辈是 `book` 元素和 `bookstore` 元素：

```
<bookstore>

<book>
    <title>Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
</book>

</bookstore>
```

(5) 后代 (Descendant)

某个节点的子，子的子，等等。

在下面的例子中，`bookstore` 的后代是 `book`、`title`、`author`、`year` 以及 `price` 元素：

```
<bookstore>

<book>
    <title>Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
</book>

</bookstore>
```

选取节点

XPath 使用路径表达式在 XML 文档中选取节点。节点是通过沿着路径或者 **step** 来选取的。

下面列出了最有用的路径表达式：

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

实例

在下面的表格中，我们已列出了一些路径表达式以及表达式的结果：

路径表达式	结果
bookstore	选取 bookstore 元素的所有子节点。
/bookstore	选取根元素 bookstore。注释：假如路径起始于正斜杠(/)，则此路径始终代表到某元素的绝对路径！
bookstore/book	选取属于 bookstore 的子元素的所有 book 元素。
//book	选取所有 book 子元素，而不管它们在文档中的位置。
bookstore//book	选择属于 bookstore 元素的后代的所有 book 元素，而不管它们位于 bookstore 之下的什么位置。
//@lang	选取名为 lang 的所有属性。

谓语（Predicates）

谓语用来查找某个特定的节点或者包含某个指定的值的节点。

谓语被嵌在方括号中。

实例

在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式	结果
/bookstore/book[1]	选取属于 bookstore 子元素的第一个 book 元素。
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素。
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素。
/bookstore/book[position()<3]	选取最前面的两个属于 bookstore 元素的子元素的 book 元素。
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素。
//title[@lang='eng']	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性。
/bookstore/book[price>35.00]	选取 bookstore 元素的所有 book 元素，且其中的 price 元素的值须大于 35.00。
/bookstore/book[price>35.00]/title	选取 bookstore 元素中的 book 元素的所有 title 元素，且其中的 price 元素的值须大于 35.00。

选取未知节点

XPath 通配符可用来选取未知的 XML 元素。

通配符	描述
*	匹配任何元素节点。
@*	匹配任何属性节点。
node()	匹配任何类型的节点。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结果：

路径表达式	结果
/bookstore/*	选取 bookstore 元素的所有子元素。
//*	选取文档中的所有元素。
//title[@*]	选取所有带有属性的 title 元素。

选取若干路径

通过在路径表达式中使用“|”运算符，您可以选取若干个路径。

实例

在下面的表格中，我们列出了一些路径表达式，以及这些表达式的结
果：

路径表达式	结果
//book/title //book/price	选取 book 元素的所有 title 和 price 元素。
//title //price	选取文档中的所有 title 和 price 元素。
/bookstore/book/title //price	选取属于 bookstore 元素的 book 元素的所 有 title 元素，以及文档中所有的 price 元 素。

XPath 运算符

下面列出了可用在 XPath 表达式中的运算符：

运算符	描述	实例	返回值
	计算两个节点集	//book //cd	返回所有拥有 book 和 cd 元素的节点集
+	加法	6 + 4	10
-	减法	6 - 4	2
*	乘法	6 * 4	24
div	除法	8 div 4	2
=	等于	price=9.80	如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。
!=	不等于	price!=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
<	小于	price<9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
<=	小于或等于	price<=9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
>	大于	price>9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
>=	大于或等于	price>=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。
or	或	price=9.80 or price=9.70	如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。
and	与	price>9.00 and price<9.90	如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。
mod	计算除法的余数	5 mod 2	1

Ixml用法

初步使用

首先我们利用它来解析 HTML 代码，先来一个小例子来感受一下它的基本用法。

```
from lxml import etree
text = """
<div>
    <ul>
        <li class="item-0"><a href="link1.html">first item</a><
        <li class="item-1"><a href="link2.html">second item</a>
        <li class="item-inactive"><a href="link3.html">third it
        <li class="item-1"><a href="link4.html">fourth item</a>
        <li class="item-0"><a href="link5.html">fifth item</a>
    </ul>
</div>
"""

html = etree.HTML(text)
result = etree.tostring(html)
print(result)
```

首先我们使用 `lxml` 的 `etree` 库，然后利用 `etree.HTML` 初始化，然后我们将其打印出来。

其中，这里体现了 `lxml` 的一个非常实用的功能就是自动修正 `html` 代码，大家应该注意到了，最后一个 `li` 标签，其实我把尾标签删掉了，是不闭合的。不过，`lxml` 因为继承了 `libxml2` 的特性，具有自动修正 `HTML` 代码的功能。

所以输出结果是这样的

```
<html><body>
<div>
    <ul>
        <li class="item-0"><a href="link1.html">first item</a><
        <li class="item-1"><a href="link2.html">second item</a>
        <li class="item-inactive"><a href="link3.html">third it
        <li class="item-1"><a href="link4.html">fourth item</a>
        <li class="item-0"><a href="link5.html">fifth item</a><
    </ul>
</div>

</body></html>
```

不仅补全了 `li` 标签，还添加了 `body`, `html` 标签。

文件读取

除了直接读取字符串，还支持从文件读取内容。比如我们新建一个文件叫做 `hello.html`，内容为

```
<div>
  <ul>
    <li class="item-0"><a href="link1.html">first item</a><
      <li class="item-1"><a href="link2.html">second item</a>
        <li class="item-inactive"><a href="link3.html"><span cl
          <li class="item-1"><a href="link4.html">fourth item</a>
            <li class="item-0"><a href="link5.html">fifth item</a><
              </ul>
            </div>
```

利用 `parse` 方法来读取文件。

```
from lxml import etree
html = etree.parse('hello.html')
result = etree.tostring(html, pretty_print=True)
print(result)
```

同样可以得到相同的结果。

XPath实例测试

依然以上一段程序为例

(1) 获取所有的 `` 标签

```
from lxml import etree
html = etree.parse('hello.html')
print type(html)
result = html.xpath('//li')
print result
print len(result)
print type(result)
print type(result[0])
```

运行结果

```
<type 'lxml.etree._ElementTree'>
[<Element li at 0x1014e0e18>, <Element li at 0x1014e0ef0>, <Element li at 0x1014e0f08>, <Element li at 0x1014e0f20>, <Element li at 0x1014e0f38>]
```

可见，`etree.parse` 的类型是 `ElementTree`，通过调用 `xpath` 以后，得到了一个列表，包含了 5 个

- 元素，每个元素都是 `Element` 类型

(2) 获取 `` 标签的所有 `class`

```
result = html.xpath('//li/@class')
print result
```

运行结果

```
['item-0', 'item-1', 'item-inactive', 'item-1', 'item-0']
```

(3) 获取 `` 标签下 `href` 为 `link1.html` 的 `<a>` 标签

```
result = html.xpath('//li/a[@href="link1.html"]')
print result
```

运行结果

```
[<Element a at 0x10ffaae18>]
```

(4) 获取 `` 标签下的所有 `` 标签

注意这么写是不对的

```
result = html.xpath('//li/span')
```

因为 / 是用来获取子元素的，而 `` 并不是 `` 的子元素，所以，要用双斜杠

```
result = html.xpath('//li//span')
print result
```

运行结果

```
[<Element span at 0x10d698e18>]
```

(5) 获取 `` 标签下的所有 `class`, 不包括 ``

```
result = html.xpath('//li/a/@class')
print result
```

运行结果

```
['bold']
```

(6) 获取最后一个 `` 的 `<a>` 的 `href`

```
result = html.xpath('//li[last()]/a/@href')
print result
```

运行结果

```
['link5.html']
```

(7) 获取倒数第二个元素的内容

```
result = html.xpath('//li[last()-1]/a')
print result[0].text
```

运行结果

```
fourth item
```

(8) 获取 `class` 为 `bold` 的标签名

```
result = html.xpath('//*[@class="bold"]')
print result[0].tag
```

运行结果

```
span
```

通过以上实例的练习，相信大家对 **XPath** 的基本用法有了基本的了解。也可以利用 `text` 方法来获取元素的内容。

大家多加练习！

结语

XPath 是一个非常好用的解析方法，同时也作为爬虫学习的基础，在后面的 **selenium** 以及 **scrapy** 框架中都会涉及到这部分知识，希望大家可以吧它的语法掌握清楚，为后面的深入研究做好铺垫。

Python爬虫利器四之PhantomJS的用法

前言

大家有没有发现之前我们写的爬虫都有一个共性，就是只能爬取单纯的html代码，如果页面是JS渲染的该怎么办呢？如果我们单纯去分析一个个后台的请求，手动去摸索JS渲染的到的一些结果，那简直没天理了。所以，我们需要有一些好用的工具来帮助我们像浏览器一样渲染JS处理的页面。

其中有一个比较常用的工具，那就是[PhantomJS](#)

Full web stack No browser required PhantomJS is a headless WebKit scriptable with a JavaScript API. It has fast and native support for various web standards: DOM handling, CSS selector, JSON, Canvas, and SVG.

PhantomJS是一个无界面的,可脚本编程的WebKit浏览器引擎。它原生支持多种web 标准：DOM 操作，CSS选择器，JSON，Canvas 以及SVG。

好，接下来我们就一起来了解一下这个神奇好用的库的用法吧。

安装

PhantomJS安装方法有两种，一种是下载源码之后自己来编译，另一种是直接下载编译好的二进制文件。然而自己编译需要的时间太长，而且需要挺多的磁盘空间。官方推荐直接下载二进制文件然后安装。

大家可以依照自己的开发平台选择不同的包进行[下载](#)

当然如果你不嫌麻烦，可以选择[下载源码](#)

然后自己编译。

目前（2016/3/21）最新发行版本是 v2.1，

安装完成之后命令行输入

```
phantomjs -v
```

如果正常显示版本号，那么证明安装成功了。如果提示错误，那么请重新安装。

本文介绍大部分内容来自于官方文档，博主对其进行了整理，学习更多请参考[官方文档](#)

快速开始

第一个程序

第一个程序当然是Hello World，新建一个 js 文件。命名为 helloworld.js

```
console.log('Hello, world!');  
phantom.exit();
```

命令行输入

```
phantomjs helloworld.js
```

程序输出了 Hello, world！程序第二句话终止了 phantom 的执行。

注意：phantom.exit();这句话非常重要，否则程序将永远不会终止。

页面加载

可以利用 phantom 来实现页面的加载，下面的例子实现了页面的加载并将页面保存为一张图片。

```
var page = require('webpage').create();  
page.open('http://cuiqingcai.com', function (status) {  
    console.log("Status: " + status);  
    if (status === "success") {  
        page.render('example.png');  
    }  
    phantom.exit();  
});
```

首先创建了一个webpage对象，然后加载本站点主页，判断响应状态，如果成功，那么保存截图为 example.png

以上代码命名为 pageload.js，命令行

```
phantomjs pageload.js
```

发现执行成功，然后目录下多了一张图片，example.png



热门排行

-
- 1 强势回归博客，顺便祝大家小年 评论(9) ❤10喜欢
 - 2 弃用多说，改用畅言 评论(9) ❤6喜欢
 - 3 干货！IT小伙伴们实用的网站及 评论(7) ❤45喜欢
-

Python ➤ Python爬虫利器—之Requests库的用法



前言 之前我们用了 `urllib` 库，这个作为入门的工具还是不错的，对了解一些爬虫的基本理念，掌握爬虫爬取的流程有所帮助。入门之

JavaScript ➤ jQuery易忽略的知识点总结



前言 之前在用jQuery，不过有时候用着用着一些用法发现并没有用到过，比较陌生，现在重新梳理一下，把易忽略的知识点总结一下，长

Other ➤ PhpStorm使用File Watchers自动编译less



综述 PhpStorm可以使用File Watchers自动编译Less，有了这个IDE，妈妈再也不用担心我

因为这个 `render` 方法，`phantom` 经常会用到网页截图的功能。

测试页面加载速度

下面这个例子计算了一个页面的加载速度，同时还用到了命令行传参的特性。新建文件保存为 `loadspeed.js`

```
var page = require('webpage').create(),
    system = require('system'),
    t, address;

if (system.args.length === 1) {
    console.log('Usage: loadspeed.js <some URL>');
    phantom.exit();
}

t = Date.now();
address = system.args[1];
page.open(address, function(status) {
    if (status !== 'success') {
        console.log('FAIL to load the address');
    } else {
        t = Date.now() - t;
        console.log('Loading ' + system.args[1]);
        console.log('Loading time ' + t + ' msec');
    }
    phantom.exit();
});
```

程序判断了参数的多少，如果参数不够，那么终止运行。然后记录了打开页面的时间，请求页面之后，再纪录当前时间，二者之差就是页面加载速度。

```
phantomjs loadspeed.js http://cuiqingcai.com
```

运行结果

```
Loading http://cuiqingcai.com
Loading time 11678 msec
```

这个时间包括JS渲染的时间，当然和网速也有关。

代码评估

To evaluate JavaScript code in the context of the web page, use `evaluate()` function. The execution is “sandboxed”, there is no way for the code to access any JavaScript objects and variables outside its own page context. An object can be returned from `evaluate()`, however it is limited to simple objects and can't contain functions or closures.

利用 `evaluate` 方法我们可以获取网页的源代码。这个执行是“沙盒式”的，它不会去执行网页外的 JavaScript 代码。`evaluate` 方法可以返回一个对象，然而返回值仅限于对象，不能包含函数（或闭包）

```
var url = 'http://www.baidu.com';
var page = require('webpage').create();
page.open(url, function(status) {
    var title = page.evaluate(function() {
        return document.title;
    });
    console.log('Page title is ' + title);
    phantom.exit();
});
```

以上代码获取了百度的网站标题。

```
Page title is 百度一下，你就知道
```

任何来自于网页并且包括来自 `evaluate()` 内部代码的控制台信息，默认不会显示。

需要重写这个行为，使用 `onConsoleMessage` 回调函数，示例可以改写成

```
var url = 'http://www.baidu.com';
var page = require('webpage').create();
page.onConsoleMessage = function (msg) {
    console.log(msg);
};
page.open(url, function (status) {
    page.evaluate(function () {
        console.log(document.title);
    });
    phantom.exit();
});
```

这样的话，如果你用浏览器打开百度首页，打开调试工具的 `console`，可以看到控制台输出信息。

重写了 `onConsoleMessage` 方法之后，可以发现控制台输出的结果和我们需要输出的标题都打印出来了。

一张网页，要经历怎样的过程，才能抵达用户面前？
一位新人，要经历怎样的成长，才能站在技术之巅？
探寻这里的秘密；
体验这里的挑战；
成为这里的主人；
加入百度，加入网页搜索，你，可以影响世界。

请将简历发送至 %c ps_recruiter@baidu.com (邮件标题请以“姓名-应聘XX”
职位介绍: <http://dwz.cn/hr2013>
百度一下，你就知道

啊，我没有在为百度打广告！

屏幕捕获

Since PhantomJS is using WebKit, a real layout and rendering engine, it can capture a web page as a screenshot. Because PhantomJS can render anything on the web page, it can be used to convert contents not only in HTML and CSS, but also SVG and Canvas.

因为 PhantomJS 使用了 WebKit内核，是一个真正的布局和渲染引擎，它可以像屏幕截图一样捕获一个web界面。因为它可以渲染网页中的人和元素，所以它不仅用到HTML，CSS的内容转化，还用在SVG，Canvas。可见其功能是相当强大的。

下面的例子就捕获了github网页的截图。上文有类似内容，不再演示。

```
var page = require('webpage').create();
page.open('http://github.com/', function() {
    page.render('github.png');
    phantom.exit();
});
```

除了 png 格式的转换，PhantomJS还支持 jpg, gif, pdf等格式。[测试样例](#)

其中最重要的方法便是 `viewportSize` 和 `clipRect` 属性。

`viewportSize` 是视区的大小，你可以理解为你打开了一个浏览器，然后把浏览器窗口拖到了多大。

`clipRect` 是裁切矩形的大小，需要四个参数，前两个是基准点，后两个参数是宽高。

通过下面的小例子感受一下。

```

var page = require('webpage').create();
//viewportSize being the actual size of the headless browser
page.viewportSize = { width: 1024, height: 768 };
//the clipRect is the portion of the page you are taking a screen shot of
page.clipRect = { top: 0, left: 0, width: 1024, height: 768 };
//the rest of the code is the same as the previous example
page.open('http://cuiqingcai.com/', function() {
    page.render('germy.png');
    phantom.exit();
});

```

运行结果



就相当于把浏览器窗口拖到了 1024×768 大小，然后从左上角裁切出了 1024×768 的页面。

网络监听

Because PhantomJS permits the inspection of network traffic, it is suitable to build various analysis on the network behavior and performance.

因为 PhantomJS 有网络通信的检查功能，它也很适合用来做网络行为的分析。

When a page requests a resource from a remote server, both the request and the response can be tracked via onResourceRequested and onResourceReceived callback.

当接受到请求时，可以通过改写**onResourceRequested**和**onResourceReceived**回调函数来实现接收到资源请求和资源接受完毕的监听。例如

```
var url = 'http://www.cuiqingcai.com';
var page = require('webpage').create();
page.onResourceRequested = function(request) {
    console.log('Request ' + JSON.stringify(request, undefined, 4));
};
page.onResourceReceived = function(response) {
    console.log('Receive ' + JSON.stringify(response, undefined, 4));
};
page.open(url);
```

运行结果会打印出所有资源的请求和接收状态，以JSON格式输出。

页面自动化处理

Because PhantomJS can load and manipulate a web page, it is perfect to carry out various page automations.

因为 PhantomJS 可以加载和操作一个web页面，所用来自动化处理也是非常适合的。

DOM操作

Since the script is executed as if it is running on a web browser, standard DOM scripting and CSS selectors work just fine.

脚本都是像在浏览器中运行的，所以标准的 JavaScript 的 DOM 操作和 CSS 选择器也是生效的。

例如下面的例子就修改了 User-Agent，然后还返回了页面中某元素的内容。

```
var page = require('webpage').create();
console.log('The default user agent is ' + page.settings.userAgent);
page.settings.userAgent = 'SpecialAgent';
page.open('http://www.httpuseragent.org', function(status) {
    if (status !== 'success') {
        console.log('Unable to access network');
    } else {
        var ua = page.evaluate(function() {
            return document.getElementById('myagent').textContent;
        });
        console.log(ua);
    }
    phantom.exit();
});
```

运行结果

```
The default user agent is Mozilla/5.0 (Macintosh; Intel Mac OS X
Your Http User Agent string is: SpecialAgent
```

首先打印出了默认的 **User-Agent**, 然后通过修改它, 请求验证 **User-Agent** 的一个站点, 通过选择器得到了修改后的 **User-Agent**。

使用附加库

在1.6版本之后允许添加外部的JS库, 比如下面的例子添加了jQuery, 然后执行了jQuery代码。

```
var page = require('webpage').create();
page.open('http://www.sample.com', function() {
    page.includeJs("http://ajax.googleapis.com/ajax/libs/jquery/1.
        page.evaluate(function() {
            $("button").click();
        });
        phantom.exit()
    });
});
```

引用了 jQuery 之后, 我们便可以在下面写一些 jQuery 代码了。

Webpage对象

在前面我们介绍了 `webpage` 对象的几个方法和属性，其实它本身还有其它很多的属性。具体的内容可以参考[Webpage](#)和[Webpage用例](#)

里面介绍了 `webpage` 的所有属性，方法，回调。

命令行

Command-line Options

PhantomJS提供的命令行选项有：

- `-help` or `-h` lists all possible command-line options. Halts immediately, will not run a script passed as argument. [帮助列表]
- `-version` or `-v` prints out the version of PhantomJS. Halts immediately, will not run a script passed as argument. [查看版本]
- `-cookies-file=/path/to/cookies.txt` specifies the file name to store the persistent Cookies. [指定存放cookies的路径]
- `-disk-cache=[true|false]` enables disk cache (at desktop services cache storage location, default is false). Also accepted: [yes|no]. [硬盘缓存开关, 默认为关]
- `-ignore-ssl-errors=[true|false]` ignores SSL errors, such as expired or self-signed certificate errors (default is false). Also accepted: [yes|no]. [忽略ssl错误, 默认不忽略]
- `-load-images=[true|false]` load all inlined images (default is true). Also accepted: [yes|no]. [加载图片, 默认为加载]
- `-local-storage-path=/some/path` path to save LocalStorage content and WebSQL content. [本地存储路径, 如本地文件和SQL文件等]
- `-local-storage-quota=number` maximum size to allow for data. [本地文件最大大小]
- `-local-to-remote-url-access=[true|false]` allows local content to access remote URL (default is false). Also accepted: [yes|no]. [是否允许远程加载文件, 默认不允许]
- `-max-disk-cache-size=size` limits the size of disk cache (in KB). [最大缓存空间]
- `-output-encoding=encoding` sets the encoding used for terminal output (default is utf8). [默认输出编码, 默认utf8]
- `-remote-debugger-port` starts the script in a debug harness and listens on the specified port [远程调试端口]
- `-remote-debugger-autorun` runs the script in the debugger immediately: 'yes' or 'no' (default) [在调试环境下是否立即执行脚本, 默认否]
- `-proxy=address:port` specifies the proxy server to use (e.g. `-proxy=192.168.1.42:8080`). [代理]
- `-proxy-type=[http|socks5|none]` specifies the type of the proxy server (default is http). [代理类型, 默认http]
- `-proxy-auth` specifies the authentication information for the proxy, e.g. `-proxy-auth=username:password`. [代理认证]
- `-script-encoding=encoding` sets the encoding used for the starting script (default is utf8). [脚本编码, 默认utf8]
- `-ssl-protocol=[sslv3|sslv2|tlsv1|any]` sets the SSL protocol for secure connections (default is SSLv3). [SSL协议, 默认SSLv3]

- `--ssl-certificates-path`= Sets the location for custom CA certificates (if none set, uses system default). [SSL证书路径, 默认系统默认路径]
- `--web-security=[true|false]` enables web security and forbids cross-domain XHR (default is true). Also accepted: [yes|no]. [是否开启安全保护和禁止异站Ajax, 默认开启保护]
- `--webdriver` starts in ‘Remote WebDriver mode’ (embedded GhostDriver): ‘[[:]’ (default ‘127.0.0.1:8910’) [以远程 WebDriver模式启动]
- `--webdriver-selenium-grid-hub` URL to the Selenium Grid HUB: ‘URLTOHUB’ (default ‘none’) (NOTE: works only together with `--webdriver`) [Selenium接口]
- `--config=/path/to/config.json` can utilize a JavaScript Object Notation (JSON) configuration file instead of passing in multiple command-line options [所有的命令行配置从 config.json 中读取]

注： JSON文件配置格式

```
{
  /* Same as: --ignore-ssl-errors=true */
  "ignoreSslErrors": true,

  /* Same as: --max-disk-cache-size=1000 */
  "maxDiskCacheSize": 1000,

  /* Same as: --output-encoding=utf8 */
  "outputEncoding": "utf8"

  /* etc. */
}
```

There are some keys that do not translate directly:

```
* --disk-cache => diskCacheEnabled
* --load-images => autoLoadImages
* --local-storage-path => offlineStoragePath
* --local-storage-quota => offlineStorageDefaultQuota
* --local-to-remote-url-access => localToRemoteUrlAccessEnabled
* --web-security => webSecurityEnabled
```

以上是命令行的基本配置

实例

在此提供[官方文档实例](#)，多对照实例练习，使用起来会更得心应手。

结语

以上是博主对 **PhantomJS** 官方文档的基本总结和翻译，如有差错，希望大家可以指正。另外可能有的小伙伴觉得这个工具和 **Python** 有什么关系？不要急，后面会有 **Python** 和 **PhantomJS** 的综合使用的。

Python爬虫利器五之Selenium的用法

前言

在上一节我们学习了 PhantomJS 的基本用法，归根结底它是一个没有界面的浏览器，而且运行的是 JavaScript 脚本，然而这就能写爬虫了吗？这又和 Python 有什么关系？说好的 Python 爬虫呢？库都学完了你给我看这个？客官别急，接下来我们介绍的这个工具，统统解决掉你的疑惑。

简介

Selenium 是什么？一句话，自动化测试工具。它支持各种浏览器，包括 Chrome, Safari, Firefox 等主流界面式浏览器，如果你在这些浏览器里面安装一个 Selenium 的插件，那么便可以方便地实现 Web 界面的测试。换句话说叫 Selenium 支持这些浏览器驱动。话说回来，PhantomJS 不也是一个浏览器吗，那么 Selenium 支持不？答案是肯定的，这样二者便可以实现无缝对接了。

然后又有什么好消息呢？Selenium 支持多种语言开发，比如 Java, C, Ruby 等等，有 Python 吗？那是必须的！哦这可真是天大的好消息啊。

嗯，所以呢？安装一下 Python 的 Selenium 库，再安装好 PhantomJS，不就可以实现 Python + Selenium + PhantomJS 的无缝对接了嘛！PhantomJS 用来渲染解析 JS，Selenium 用来驱动以及与 Python 的对接，Python 进行后期的处理，完美的三剑客！

有人问，为什么不直接用浏览器而用一个没界面的 PhantomJS 呢？答案是：效率高！

Selenium 有两个版本，目前最新版本是 2.53.1 (2016/3/22)

Selenium 2，又名 WebDriver，它的主要新功能是集成了 Selenium 1.0 以及 WebDriver（WebDriver 曾经是 Selenium 的竞争对手）。也就是说 Selenium 2 是 Selenium 和 WebDriver 两个项目的合并，即 Selenium 2 兼容 Selenium，它既支持 Selenium API 也支持 WebDriver API。

更多详情可以查看 [Webdriver](#) 的简介。

嗯，通过以上描述，我们应该对 Selenium 有了大概的认识，接下来就让我们开始进入动态爬取的新世界吧。

本文参考内容来自 [Selenium 官网](#) 和 [Selenium Python 文档](#)

安装

首先安装 Selenium

```
pip install selenium
```

或者[下载源码](#)

然后解压后运行下面的命令进行安装

```
python setup.py install
```

安装好了之后我们便开始探索抓取方法了。

快速开始

初步体验

我们先来一个小例子感受一下 Selenium，这里我们用 Chrome 浏览器来测试，方便查看效果，到真正爬取的时候换回 PhantomJS 即可。

```
from selenium import webdriver  
  
browser = webdriver.Chrome()  
browser.get('http://www.baidu.com/')
```

运行这段代码，会自动打开浏览器，然后访问百度。

如果程序执行错误，浏览器没有打开，那么应该是没有装 Chrome 浏览器或者 Chrome 驱动没有配置在环境变量里。[下载驱动](#)，然后将驱动文件路径配置在环境变量即可。

比如我的是 Mac OS，就把下载好的文件放在 /usr/bin 目录下就可以了。

模拟提交

下面的代码实现了模拟提交搜索的功能，首先等页面加载完成，然后输入到搜索框文本，点击提交。

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
print driver.page_source
```

同样是在 Chrome 里面测试，感受一下。

The `driver.get` method will navigate to a page given by the URL. WebDriver will wait until the page has fully loaded (that is, the “`onload`” event has fired) before returning control to your test or script. It’s worth noting that if your page uses a lot of AJAX on load then WebDriver may not know when it has completely loaded.

其中 `driver.get` 方法会打开请求的URL，WebDriver 会等待页面完全加载完成之后才会返回，即程序会等待页面的所有内容加载完成，JS渲染完毕之后才继续往下执行。注意：如果这里用到了特别多的 Ajax 的话，程序可能不知道是否已经完全加载完毕。

WebDriver offers a number of ways to find elements using one of the `find_element_by*` methods. For example, the input text element can be located by its name attribute using `find_element_by_name` method

WebDriver 提供了许多寻找网页元素的方法，譬如 `find_element_by*` 的方法。例如一个输入框可以通过 `find_element_by_name` 方法寻找 `name` 属性来确定。

Next we are sending keys, this is similar to entering keys using your keyboard. Special keys can be send using `Keys` class imported from `selenium.webdriver.common.keys`

然后我们输入来文本然后模拟点击了回车，就像我们敲击键盘一样。我们可以利用 `Keys` 这个类来模拟键盘输入。

最后最重要的一点

获取网页渲染后的源代码。

输出 `page_source` 属性即可。

这样，我们就可以做到网页的动态爬取了。

测试用例

有了以上特性，我们当然可以用来写测试样例了。

```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

class PythonOrgSearch(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()

    def test_search_in_python_org(self):
        driver = self.driver
        driver.get("http://www.python.org")
        self.assertIn("Python", driver.title)
        elem = driver.find_element_by_name("q")
        elem.send_keys("pycon")
        elem.send_keys(Keys.RETURN)
        assert "No results found." not in driver.page_source

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

运行程序，同样的功能，我们将其封装为测试标准类的形式。

The test case class is inherited from `unittest.TestCase`. Inheriting from `TestCase` class is the way to tell `unittest` module that this is a test case. The `setUp` is part of initialization, this method will get called before every test function which you are going to write in this test case class. The test case method should always start with characters `test`. The `tearDown` method will get called after every test method. This is a place to do all cleanup actions. You can also call `quit` method instead of `close`. The `quit` will exit the entire browser, whereas `close` will close a tab, but if it is the only tab opened, by default most browser will exit entirely.

测试用例是继承了 `unittest.TestCase` 类，继承这个类表明这是一个测试类。`setUp`方法是初始化的方法，这个方法会在每个测试类中自动调用。每一个测试方法命名都有规范，必须以 `test` 开头，会自动执行。最后的 `tearDown` 方法会在每一个测试方法结束之后调用。这相当于最后的析构方法。在这个方法里写的是 `close` 方法，你还可以写 `quit` 方法。

不过 `close` 方法相当于关闭了这个 TAB 选项卡，然而 `quit` 是退出了整个浏览器。当你只开启了一个 TAB 选项卡的时候，关闭的时候也会将整个浏览器关闭。

页面操作

页面交互

仅仅抓取页面没有多大卵用，我们真正要做的是做到和页面交互，比如点击，输入等等。那么前提就是要找到页面中的元素。`WebDriver`提供了各种方法来寻找元素。例如下面有一个表单输入框。

```
<input type="text" name="passwd" id="passwd-id" />
```

我们可以这样获取它

```
element = driver.find_element_by_id("passwd-id")
element = driver.find_element_by_name("passwd")
element = driver.find_elements_by_tag_name("input")
element = driver.find_element_by_xpath("//input[@id='passwd-id']")
```

你还可以通过它的文本链接来获取，但是要小心，文本必须完全匹配才可以，所以这并不是一个很好的匹配方式。

而且你在用 `xpath` 的时候还需要注意的是，如果有多个元素匹配了 `xpath`，它只会返回第一个匹配的元素。如果没有找到，那么会抛出 `NoSuchElementException` 的异常。

获取了元素之后，下一步当然就是向文本输入内容了，可以利用下面的方法

```
element.send_keys("some text")
```

同样你还可以利用 `Keys` 这个类来模拟点击某个按键。

```
element.send_keys("and some", Keys.ARROW_DOWN)
```

你可以对任何获取到到元素使用 `send_keys` 方法，就像你在 GMail 里面点击发送键一样。不过这样会导致的结果就是输入的文本不会自动清除。所以输入的文本都会在原来的基础上继续输入。你可以用下面的方法来清除输入文本的内容。

```
element.clear()
```

这样输入的文本会被清除。

填充表单

我们已经知道了怎样向文本框中输入文字，但是其它的表单元素呢？例如下拉选项卡的处理可以如下

```
element = driver.find_element_by_xpath("//select[@name='name']")
all_options = element.find_elements_by_tag_name("option")
for option in all_options:
    print("Value is: %s" % option.get_attribute("value"))
    option.click()
```

首先获取了第一个 `select` 元素，也就是下拉选项卡。然后轮流设置了 `select` 选项卡中的每一个 `option` 选项。你可以看到，这并不是一个非常有效的方法。

其实 WebDriver 中提供了一个叫 `Select` 的方法，可以帮助我们完成这些事情。

```
from selenium.webdriver.support.ui import Select
select = Select(driver.find_element_by_name('name'))
select.select_by_index(index)
select.select_by_visible_text("text")
select.select_by_value(value)
```

如你所见，它可以根据索引来选择，可以根据值来选择，可以根据文字来选择。是十分方便的。

全部取消选择怎么办呢？很简单

```
select = Select(driver.find_element_by_id('id'))
select.deselect_all()
```

这样便可以取消所有的选择。

另外我们还可以通过下面的方法获取所有的已选选项。

```
select = Select(driver.find_element_by_xpath("xpath"))
all_selected_options = select.all_selected_options
```

获取所有可选选项是

```
options = select.options
```

如果你把表单都填好了，最后肯定要提交表单对吧。怎么提交呢？很简单

```
driver.find_element_by_id("submit").click()
```

这样就相当于模拟点击了 `submit` 按钮，做到表单提交。

当然你也可以单独提交某个元素

```
element.submit()
```

方法，`WebDriver`会在表单中寻找它所在的表单，如果发现这个元素并没有被表单所包围，那么程序会抛出 `NoSuchElementException` 的异常。

元素拖拽

要完成元素的拖拽，首先你需要指定被拖动的元素和拖动目标元素，然后利用 `ActionChains` 类来实现。

```
element = driver.find_element_by_name("source")
target = driver.find_element_by_name("target")

from selenium.webdriver import ActionChains
action_chains = ActionChains(driver)
action_chains.drag_and_drop(element, target).perform()
```

这样就实现了元素从 `source` 拖动到 `target` 的操作。

页面切换

一个浏览器肯定会有很多窗口，所以我们肯定要有方法来实现窗口的切换。切换窗口的方法如下

```
driver.switch_to_window("windowName")
```

另外你可以使用 `window_handles` 方法来获取每个窗口的操作对象。例如

```
for handle in driver.window_handles:  
    driver.switch_to_window(handle)
```

另外切换 **frame** 的方法如下

```
driver.switch_to_frame("frameName.0.child")
```

这样焦点会切换到一个 **name** 为 **child** 的 **frame** 上。

弹窗处理

当你出发了某个事件之后，页面出现了弹窗提示，那么你怎样来处理这个提示或者获取提示信息呢？

```
alert = driver.switch_to_alert()
```

通过上述方法可以获取弹窗对象。

历史记录

那么怎样来操作页面的前进和后退功能呢？

```
driver.forward()  
driver.back()
```

嗯，简洁明了。

Cookies 处理

为页面添加 Cookies，用法如下

```
# Go to the correct domain  
driver.get("http://www.example.com")  
  
# Now set the cookie. This one's valid for the entire domain  
cookie = {'name' : 'foo', 'value' : 'bar'}  
driver.add_cookie(cookie)
```

获取页面 Cookies，用法如下

```
# Go to the correct domain
driver.get("http://www.example.com")

# And now output all the available cookies for the current URL
driver.get_cookies()
```

以上便是 Cookies 的处理，同样是非常简单的。

元素选取

关于元素的选取，有如下的API

单个元素选取

- find_element_by_id
- find_element_by_name
- find_element_by_xpath
- find_element_by_link_text
- find_element_by_partial_link_text
- find_element_by_tag_name
- find_element_by_class_name
- find_element_by_css_selector

多个元素选取

- find_elements_by_name
- find_elements_by_xpath
- find_elements_by_link_text
- find_elements_by_partial_link_text
- find_elements_by_tag_name
- find_elements_by_class_name
- find_elements_by_css_selector

另外还可以利用 By 类来确定哪种选择方式

```
from selenium.webdriver.common.by import By

driver.find_element(By.XPATH, '//button[text()="Some text"]')
driver.find_elements(By.XPATH, '//button')
```

By 类的一些属性如下

```
ID = "id"
XPATH = "xpath"
LINK_TEXT = "link text"
PARTIAL_LINK_TEXT = "partial link text"
NAME = "name"
TAG_NAME = "tag name"
CLASS_NAME = "class name"
CSS_SELECTOR = "css selector"
```

更详细的元素选择方法参见[官方文档](#)

页面等待

这是非常重要的一部分，现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。这会让元素定位困难而且会提高产生 `ElementNotVisibleException` 的概率。

所以 Selenium 提供了两种等待方式，一种是隐式等待，一种是显式等待。

隐式等待是等待特定的时间，显式等待是指定某一条件直到这个条件成立时继续执行。

显式等待

显式等待指定某个条件，然后设置最长等待时间。如果在这个时间还没有找到元素，那么便会抛出异常了。

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("http://somedomain/url_that_delays_loading")
try:
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "myDynamicElement"))
)
finally:
    driver.quit()
```

程序默认会 500ms 调用一次来查看元素是否已经生成，如果本来元素就是存在的，那么会立即返回。

下面是一些内置的等待条件，你可以直接调用这些条件，而不用自己写某些等待条件了。

- title_is
- title_contains
- presence_of_element_located
- visibility_of_element_located
- visibility_of
- presence_of_all_elements_located
- text_to_be_present_in_element
- text_to_be_present_in_element_value
- frame_to_be_available_and_switch_to_it
- invisibility_of_element_located
- element_to_be_clickable – it is Displayed and Enabled.
- staleness_of
- element_to_be_selected
- element_located_to_be_selected
- element_selection_state_to_be
- element_located_selection_state_to_be
- alert_is_present

```
from selenium.webdriver.support import expected_conditions as EC  
  
wait = WebDriverWait(driver, 10)  
element = wait.until(EC.element_to_be_clickable((By.ID,'someid'))
```

隐式等待

隐式等待比较简单，就是简单地设置一个等待时间，单位为秒。

```
from selenium import webdriver  
  
driver = webdriver.Chrome()  
driver.implicitly_wait(10) # seconds  
driver.get("http://somedomain/url_that_delays_loading")  
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

当然如果不设置，默认等待时间为0。

程序框架

对于页面测试和分析，官方提供了一个比较明晰的代码结构，可以参考[页面测试架构](#)

API

到最后，肯定是放松最全最重要的API了，比较多，希望大家可以多加练习。

结语

以上就是 Selenium 的基本用法，我们讲解了页面交互，页面渲染之后的源代码的获取。这样，即使页面是 JS 渲染而成的，我们也可以手到擒来了。就是这么溜！

Python爬虫利器六之PyQuery的用法

前言

你是否觉得 XPath 的用法多少有点晦涩难记呢？

你是否觉得 BeautifulSoup 的语法多少有些悭吝难懂呢？

你是否甚至还在苦苦研究正则表达式却因为少些了一个点而抓狂呢？

你是否已经有了一些前端基础了解选择器却与另外一些奇怪的选择器语法混淆了呢？

嗯，那么，前端大大们的福音来了，PyQuery 来了，乍听名字，你一定联想到了 jQuery，如果你对 jQuery 熟悉，那么 PyQuery 来解析文档就是不二之选！包括我在内！

PyQuery 是 Python 仿照 jQuery 的严格实现。语法与 jQuery 几乎完全相同，所以不用再去费心去记一些奇怪的方法了。

天下竟然有这等好事？我都等不及了！

安装

有这等神器还不赶紧安装了！来！

```
pip install pyquery
```

还是原来的配方，还是熟悉的味道。

参考来源

本文内容参考[官方文档](#)，更多内容，大家可以去官方文档学习，毕竟那里才是最原汁原味的。

目前版本 1.2.4 (2016/3/24)

简介

pyquery allows you to make jquery queries on xml documents. The API is as much as possible the similar to jquery. pyquery uses lxml for fast xml and html manipulation.

This is not (or at least not yet) a library to produce or interact with javascript code. I just liked the jquery API and I missed it in python so I told myself “Hey let’s make jquery in python”. This is the result.

It can be used for many purposes, one idea that I might try in the future is to use it for templating with pure http templates that you modify using pyquery. I can also be used for web scrapping or for theming applications with Deliverance.

pyquery 可让你用 jQuery 的语法来对 xml 进行操作。这和 jQuery 十分类似。如果利用 lxml, pyquery 对 xml 和 html 的处理将更快。

这个库不是（至少还不是）一个可以和 JavaScript 交互的代码库，它只是非常像 jQuery API 而已。

初始化

在这里介绍四种初始化方式。

(1) 直接字符串

```
from pyquery import PyQuery as pq
doc = pq("<html></html>")
```

pq 参数可以直接传入 HTML 代码, doc 现在就相当于 jQuery 里面的 \$ 符号了。

(2) lxml.etree

```
from lxml import etree
doc = pq(etree.fromstring("<html></html>"))
```

可以首先用 lxml 的 etree 处理一下代码, 这样如果你的 HTML 代码出现一些不完整或者疏漏, 都会自动转化为完整清晰结构的 HTML 代码。

(3) 直接传URL

```
from pyquery import PyQuery as pq
doc = pq('http://www.baidu.com')
```

这里就像直接请求了一个网页一样, 类似用 urllib2 来直接请求这个链接, 得到 HTML 代码。

(4) 传文件

```
from pyquery import PyQuery as pq
doc = pq(filename='hello.html')
```

可以直接传某个路径的文件名。

快速体验

现在我们以本地文件为例，传入一个名字为 `hello.html` 的文件，文件内容为

```
<div>
  <ul>
    <li class="item-0">first item</li>
    <li class="item-1"><a href="link2.html">second item</a>
    <li class="item-0 active"><a href="link3.html"><span cl
    <li class="item-1 active"><a href="link4.html">fourth i
    <li class="item-0"><a href="link5.html">fifth item</a>
  </ul>
</div>
```

编写如下程序

```
from pyquery import PyQuery as pq
doc = pq(filename='hello.html')
print doc.html()
print type(doc)
li = doc('li')
print type(li)
print li.text()
```

运行结果

```
<ul>
    <li class="item-0">first item</li>
    <li class="item-1"><a href="link2.html">second item</a>
        <li class="item-0 active"><a href="link3.html"><span cl
        <li class="item-1 active"><a href="link4.html">fourth i
        <li class="item-0"><a href="link5.html">fifth item</a>
    </ul>

<class 'pyquery.pyquery.PyQuery'>
<class 'pyquery.pyquery.PyQuery'>
first item second item third item fourth item fifth item
```

看，回忆一下 `jQuery` 的语法，是不是运行结果都是一样的呢？

在这里我们注意到了一点，`PyQuery` 初始化之后，返回类型是 `PyQuery`，利用了选择器筛选一次之后，返回结果的类型依然还是 `PyQuery`，这简直和 `jQuery` 如出一辙，不能更赞！然而想一下 `BeautifulSoup` 和 `XPath` 返回的是什么？列表！一种不能再进行二次筛选（在这里指依然利用 `BeautifulSoup` 或者 `XPath` 语法）的对象！

然而比比 `PyQuery`，哦我简直太爱它了！

属性操作

你可以完全按照 `jQuery` 的语法来进行 `PyQuery` 的操作。

```
from pyquery import PyQuery as pq

p = pq('<p id="hello" class="hello"></p>)('p')
print p.attr("id")
print p.attr("id", "plop")
print p.attr("id", "hello")
```

运行结果

```
hello
<p id="plop" class="hello"/>
<p id="hello" class="hello"/>
```

再来一发

```
from pyquery import PyQuery as pq

p = pq('<p id="hello" class="hello"></p>')('p')
print p.addClass('beauty')
print p.removeClass('hello')
print p.css('font-size', '16px')
print p.css({'background-color': 'yellow'})
```

运行结果

```
<p id="hello" class="hello beauty"/>
<p id="hello" class="beauty"/>
<p id="hello" class="beauty" style="font-size: 16px"/>
<p id="hello" class="beauty" style="font-size: 16px; background-
```

依旧是那么优雅与自信！

在这里我们发现了，这是一连串的操作，而 `p` 是一直在原来的结果上变化的。

因此执行上述操作之后，`p` 本身也发生了变化。

DOM操作

同样的原汁原味的 jQuery 语法

```
from pyquery import PyQuery as pq

p = pq('<p id="hello" class="hello"></p>')('p')
print p.append(' check out <a href="http://reddit.com/r/python">
print p.prepend('Oh yes!')
d = pq('<div class="wrap"><div id="test"><a href="http://cuiqing
p.prependTo(d('#test'))
print p
print d
d.empty()
print d
```

运行结果

```
<p id="hello" class="hello"> check out <a href="http://reddit.co
<p id="hello" class="hello">Oh yes! check out <a href="http://re
<p id="hello" class="hello">Oh yes! check out <a href="http://re
<div class="wrap"><div id="test"><p id="hello" class="hello">Oh
<div class="wrap"/>
```

这不需要多解释了吧。

DOM 操作也是与 `jQuery` 如出一辙。

遍历

遍历用到 `items` 方法返回对象列表，或者用 `lambda`

```
from pyquery import PyQuery as pq
doc = pq(filename='hello.html')
lis = doc('li')
for li in lis.items():
    print li.html()

print lis.each(lambda e: e)
```

运行结果

```
first item
<a href="link2.html">second item</a>
<a href="link3.html"><span class="bold">third item</span></a>
<a href="link4.html">fourth item</a>
<a href="link5.html">fifth item</a>
<li class="item-0">first item</li>
<li class="item-1"><a href="link2.html">second item</a></li>
<li class="item-0 active"><a href="link3.html"><span class="bol
<li class="item-1 active"><a href="link4.html">fourth item</a><
<li class="item-0"><a href="link5.html">fifth item</a></li>
```

不过最常用的还是 `items` 方法

网页请求

`PyQuery` 本身还有网页请求功能，而且会把请求下来的网页代码转为 `PyQuery` 对象。

```
from pyquery import PyQuery as pq
print pq('http://cuiqingcai.com/', headers={'user-agent': 'pyque
print pq('http://httpbin.org/post', {'foo': 'bar'}, method='post
```

感受一下，GET，POST，样样通。

Ajax

PyQuery 同样支持 Ajax 操作，带有 `get` 和 `post` 方法，不过不常用，一般我们不会用 PyQuery 来做网络请求，仅仅是用来解析。

[PyQueryAjax](#)

API

最后少不了的，[API](#)大放送。

原汁原味最全的API，都在里面了！如果你对 jQuery 语法不熟，强烈建议先学习下 jQuery，再回来看 PyQuery，你会感到异常亲切！

结语

用完了 PyQuery，我已经深深爱上了他！

你呢？

**年度重磅大放送！博主录制的Python3
爬虫视频教程出炉啦！！！欢迎大家支
持！！！详情请看：**

[Python3爬虫视频学习教程](#)

[自己动手，丰衣足食！Python3网络爬虫实战案例](#)

以下为Python2爬虫系列教程：

大家好哈，我呢最近在学习Python爬虫，感觉非常有意思，真的让生活可以方便很多。学习过程中我把一些学习的笔记总结下来，还记录了一些自己实际写的一些小爬虫，在这里跟大家一同分享，希望对Python爬虫感兴趣的童鞋有帮助，如果有机会期待与大家的交流。

Python版本：2.7

一、爬虫入门

1. [Python爬虫入门一之综述](#)
2. [Python爬虫入门二之爬虫基础了解](#)
3. [Python爬虫入门三之Urllib库的基本使用](#)
4. [Python爬虫入门四之Urllib库的高级用法](#)
5. [Python爬虫入门五之URLError异常处理](#)
6. [Python爬虫入门六之Cookie的使用](#)
7. [Python爬虫入门七之正则表达式](#)

二、爬虫实战

1. [Python爬虫实战一之爬取糗事百科段子](#)
2. [Python爬虫实战二之爬取百度贴吧帖子](#)
3. [Python爬虫实战三之实现山东大学无线网络掉线自动重连](#)
4. [Python爬虫实战四之抓取淘宝MM照片](#)
5. [Python爬虫实战五之模拟登录淘宝并获取所有订单](#)
6. [Python爬虫实战六之抓取爱问知识人问题并保存至数据库](#)
7. [Python爬虫实战七之计算大学本学期绩点](#)

[8. Python爬虫实战八之利用Selenium抓取淘宝匿名旺旺](#)

三、爬虫利器

- [1. Python爬虫利器一之Requests库的用法](#)
- [2. Python爬虫利器二之Beautiful Soup的用法](#)
- [3. Python爬虫利器三之Xpath语法与lxml库的用法](#)
- [4. Python爬虫利器四之PhantomJS的用法](#)
- [5. Python爬虫利器五之Selenium的用法](#)
- [6. Python爬虫利器六之PyQuery的用法](#)

四、爬虫进阶

- [1. Python爬虫进阶一之爬虫框架概述](#)
- [2. Python爬虫进阶二之PySpider框架安装配置](#)
- [3. Python爬虫进阶三之爬虫框架Scrapy安装配置](#)
- [4. Python爬虫进阶四之PySpider的用法](#)
- [5. Python爬虫进阶五之多线程的用法](#)
- [6. Python爬虫进阶六之多进程的用法](#)
- [7. Python爬虫进阶七之设置ADSL拨号服务器代理](#)

目前暂时是这些文章，随着学习的进行，会不断更新哒，敬请期待~

希望对大家有所帮助，谢谢！

转载请注明：[静觅](#) » [Python爬虫学习系列教程](#)

Python爬虫进阶一之爬虫框架概述

综述

爬虫入门之后，我们有两条路可以走。

一个是继续深入学习，以及关于设计模式的一些知识，强化Python相关知识，自己动手造轮子，继续为自己的爬虫增加分布式，多线程等功能扩展。另一条路便是学习一些优秀的框架，先把这些框架用熟，可以确保能够应付一些基本的爬虫任务，也就是所谓的解决温饱问题，然后再深入学习它的源码等知识，进一步强化。

就个人而言，前一种方法其实就是自己动手造轮子，前人其实已经有了些比较好的框架，可以直接拿来用，但是为了自己能够研究得更加深入和对爬虫有更全面的了解，自己动手去多做。后一种方法就是直接拿来前人已经写好的比较优秀的框架，拿来用好，首先确保可以完成你想要完成的任务，然后自己再深入研究学习。第一种而言，自己探索的多，对爬虫的知识掌握会比较透彻。第二种，拿别人的来用，自己方便了，可是可能就会没有了深入研究框架的心情，还有可能思路被束缚。

不过个人而言，我自己偏向后者。造轮子是不错，但是就算你造轮子，你这不也是在基础类库上造轮子么？能拿来用的就拿来用，学了框架的作用是确保自己可以满足一些爬虫需求，这是最基本的温饱问题。倘若你一直在造轮子，到最后都没造出什么来，别人找你写个爬虫研究了这么长时间了都写不出来，岂不是有点得不偿失？所以，进阶爬虫我还是建议学习一下框架，作为自己的几把武器。至少，我们可以做到了，就像你拿了把枪上战场了，至少，你是可以打击敌人的，比你一直在磨刀好的多吧？

框架概述

博主接触了几个爬虫框架，其中比较好用的是 **Scrapy** 和 **PySpider**。就个人而言，**pyspider**上手更简单，操作更加简便，因为它增加了 WEB 界面，写爬虫迅速，集成了 **phantomjs**，可以用来抓取js渲染的页面。**Scrapy**自定义程度高，比 **PySpider**更底层一些，适合学习研究，需要学习的相关知识多，不过自己拿来研究分布式和多线程等等是非常合适的。

在这里博主会一一把自己的学习经验写出来与大家分享，希望大家可以喜欢，也希望可以给大家一些帮助。

PySpider

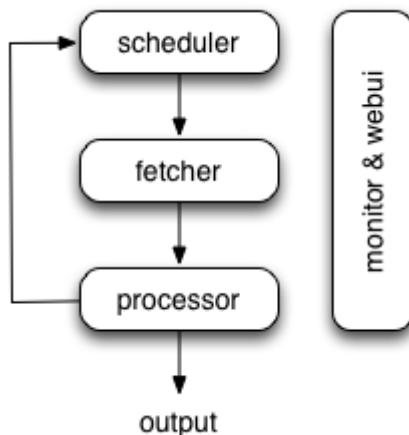
PySpider是binux做的一个爬虫架构的开源化实现。主要的功能需求是：

- 抓取、更新调度多站点的特定的页面
- 需要对页面进行结构化信息提取
- 灵活可扩展，稳定可监控

而这也是绝大多数**python**爬虫的需求——定向抓取，结构化化解析。但是面对结构迥异的各种网站，单一的抓取模式并不一定能满足，灵活的抓取控制是必须的。为了达到这个目的，单纯的配置文件往往不够灵活，于是，通过脚本去控制抓取是最后的选择。而去重调度，队列，抓取，异常处理，监控等功能作为框架，提供给抓取脚本，并保证灵活性。最后加上**web**的编辑调试环境，以及**web**任务监控，即成为了这套框架。

pyspider的设计基础是：以**python**脚本驱动的抓取环模型爬虫

- 通过**python**脚本进行结构化信息的提取，**follow**链接调度抓取控制，实现最大的灵活性
- 通过**web**化的脚本编写、调试环境。**web**展现调度状态
- 抓取环模型成熟稳定，模块间相互独立，通过消息队列连接，从单进程到多机分布式灵活拓展



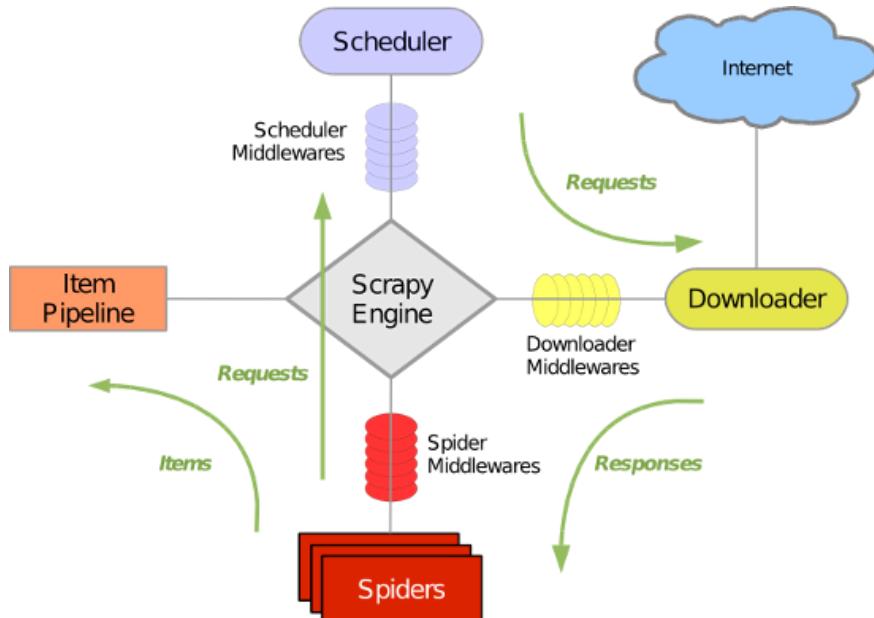
pyspider的架构主要分为 **scheduler**（调度器），**fetcher**（抓取器），**processor**（脚本执行）：

- 各个组件间使用消息队列连接，除了**scheduler**是单点的，**fetcher**和**processor**都是可以多实例分布式部署的。**scheduler**负责整体的调度控制
- 任务由**scheduler**发起调度，**fetcher**抓取网页内容，**processor**执行预先编写的**python**脚本，输出结果或产生新的提链任务（发往**scheduler**），形成闭环。
- 每个脚本可以灵活使用各种**python**库对页面进行解析，使用框架API控制下一步抓取动作，通过设置回调控制解析动作。

Scrapy

Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架。可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。其最初是为了页面抓取(更确切来说，网络抓取)所设计的，也可以应用在获取API所返回的数据(例如 Amazon Associates Web Services)或者通用的网络爬虫。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。

Scrapy 使用了 Twisted 异步网络库来处理网络通讯。整体架构大致如下



Scrapy主要包括了以下组件：

- 引擎(**Scrapy**): 用来处理整个系统的数据流处理, 触发事务(框架核心)
- 调度器(**Scheduler**): 用来接受引擎发过来的请求, 压入队列中, 并在引擎再次请求的时候返回. 可以想像成一个URL (抓取网页的网址或者说是链接) 的优先队列, 由它来决定下一个要抓取的网址是什么, 同时去除重复的网址
- 下载器(**Downloader**): 用于下载网页内容, 并将网页内容返回给蜘蛛 (**Scrapy**下载器是建立在**twisted**这个高效的异步模型上的)
- 爬虫(**Spiders**): 爬虫是主要干活的, 用于从特定的网页中提取自己需要的信息, 即所谓的实体(**Item**)。用户也可以从中提取出链接, 让 **Scrapy**继续抓取下一个页面
- 项目管道(**Pipeline**): 负责处理爬虫从网页中抽取的实体, 主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。当页面被爬虫解析后, 将被发送到项目管道, 并经过几个特定的次序处理数据。
- 下载器中间件(**Downloader Middlewares**): 位于**Scrapy**引擎和下载器之间的框架, 主要是处理**Scrapy**引擎与下载器之间的请求及响应。

- 爬虫中间件(**Spider Middlewares**): 介于Scrapy引擎和爬虫之间的框架，主要工作是处理蜘蛛的响应输入和请求输出。
- 调度中间件(**Scheduler Middewares**): 介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。

Scrapy运行流程大概如下：

- 首先，引擎从调度器中取出一个链接(URL)用于接下来的抓取
- 引擎把URL封装成一个请求(Request)传给下载器，下载器把资源下载下来，并封装成应答包(Response)
- 然后，爬虫解析Response
- 若是解析出实体(Item)，则交给实体管道进行进一步的处理。
- 若是解析出的是链接(URL)，则把URL交给Scheduler等待抓取

结语

对这两个框架进行基本的介绍之后，接下来我会介绍这两个框架的安装以及框架的使用方法，希望对大家有帮助。

Python爬虫进阶二之PySpider框架安装配置

关于

首先，在此附上[PySpider项目地址](#)，以及[官方文档](#)

安装

1. pip

首先确保你已经安装了pip，若没有安装，请参照[pip安装](#)

2. phantomjs

PhantomJS 是一个基于 WebKit 的服务器端 JavaScript API。它全面支持 web 而不需浏览器支持，其快速、原生支持各种 Web 标准：DOM 处理、CSS 选择器、JSON、Canvas 和 SVG。PhantomJS 可以用于页面自动化、网络监测、网页截屏以及无界面测试等。[安装](#)

以上附有官方安装方式，如果你是 Ubuntu 或 Mac OS X 用户，可以直接用命令来安装

Ubuntu:

```
sudo apt-get install phantomjs
```

Mac OS X:

```
brew install phantomjs
```

3. pyspider

直接利用 pip 安装即可

```
pip install pyspider
```

如果你是 Ubuntu 用户，请提前安装好以下支持类库

```
sudo apt-get install python python-dev python-distribute python-
```

测试

如果安装过程没有提示任何错误，那就证明一些OK。

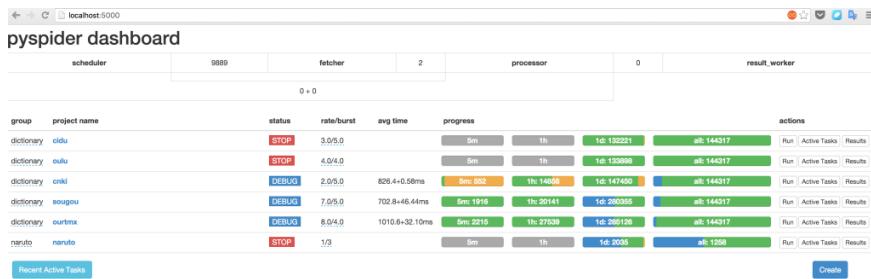
命令行输入

```
pyspider all
```

然后浏览器访问 <http://localhost:5000>

观察一下效果，如果可以正常出现 PySpider 的页面，那证明一切OK

在此附图一张，这是我写了几个爬虫之后的界面。



好，接下来我会进一步介绍这个框架的使用。

常见错误

我曾遇到过的一个错误：

[PySpider HTTP 599: SSL certificate problem错误的解决方法](#)，后来在作者那发了issue得到了答案，其他的暂时没什么问题。

不过发现有的小伙伴提了各种各样的问题啊，不过我确实都没遇到过，我再Win10, Linux Ubuntu, Linux CentOS, Mac OS X都成功运行。不过确实有些奇怪的问题，跑着跑着崩了，一点就崩了我也就比较纳闷了。

如果大家有问题，可以看看作者项目里面有没有类似的issue，另外也推荐大家直接到作者的GitHub上发issue。

毕竟，这个框架不是我写的。

在此附上[PySpider Issue地址](#)

Python爬虫进阶三之Scrapy框架安装配置

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。

1.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

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

2.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：

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

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

3.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。

```
D:\python2.7.7\Lib>pip --version
pip 6.0.8 from D:\python2.7.7\lib\site-packages <python 2.7>
```

4. 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>

5. 安装lxml

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。

6. 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！

7.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

1.安装Python

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

2.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
```

3.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 i
```



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

```
cqc@cqc-Lenovo-IdeaPad-Y480: ~
cqc@cqc-Lenovo-IdeaPad-Y480:~$ scrapy
Scrapy 0.24.4 - no active project

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

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

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
cqc@cqc-Lenovo-IdeaPad-Y480:~$
```

如有问题，欢迎留言！祝各位小伙伴顺利安装！

Python爬虫进阶四之PySpider的用法

审时度势

PySpider 是一个我个人认为非常方便并且功能强大的爬虫框架，支持多线程爬取、JS动态解析，提供了可操作界面、出错重试、定时爬取等等的功能，使用非常人性化。

本篇内容通过跟我做一个好玩的 PySpider 项目，来理解 PySpider 的运行流程。

招兵买马

具体的安装过程请查看本节讲述

嗯，安装好了之后就与我大干一番吧。

鸿鹄之志

我之前写过的一篇文章[抓取淘宝MM照片](#)

由于网页改版，爬取过程中需要的 URL 需要 JS 动态解析生成，所以之前用的 urllib2 不能继续使用了，在这里我们利用 PySpider 重新实现一下。

所以现在我们需要做的是抓取淘宝MM的个人信息和图片存储到本地。

审时度势

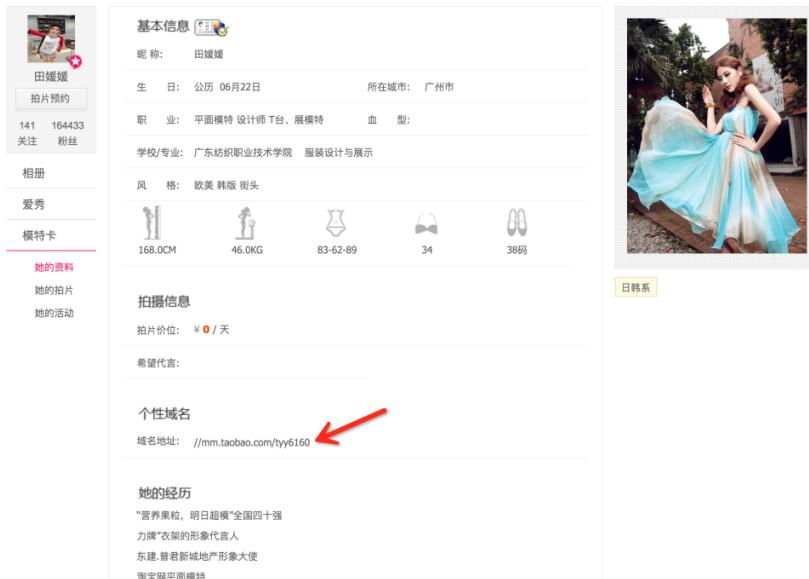
爬取目标网站：https://mm.taobao.com/json/request_top_list.htm?page=1，大家打开之后可以看到许多淘宝MM的列表。

列表有多少？

https://mm.taobao.com/json/request_top_list.htm?page=10000，第 10000 页都有，看你想要多少。我什么也不知道。

随机点击一位 MM 的姓名，可以看到她的基本资料。

淘女郎  首页 / 潮品汇 / 搭配购 / 美人库 / 活动 / 个人中心 



基本信息

昵称: 田媛媛
生日: 公历 06月22日 所在城市: 广州市
职业: 平面模特 设计师 T台、展模特 血型:
学校/专业: 广东纺织职业技术学院 服装设计与展示
风格: 欧美 韩版 街头
168.0CM 46.0KG 83-62-89 34 38码

拍摄信息

拍片价位: ¥ 0 / 天
希望代言:

个性域名

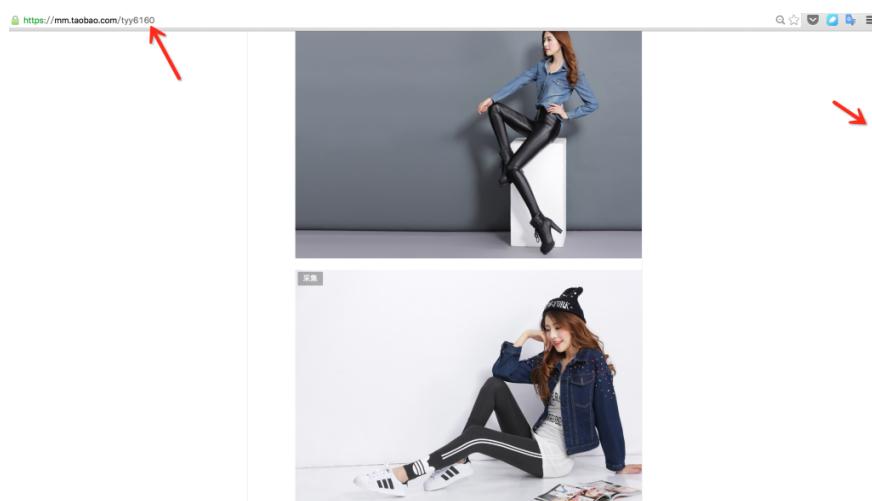
域名地址: <https://mm.taobao.com/tyy6160>

她的经历

"营养果粒,明日超模"全国四十强
力牌"衣服的形象代言人
东建·普君新城地产形象大使
淘宝网平面模特

可以看到图中有一个个性域名，我们复制到浏览器打开。

mm.taobao.com/tyy6160



嗯，往下拖，海量的 MM 图片都在这里了，怎么办你懂得，我们要把她们的照片和个人信息都存下来。

P.S. 注意图中进度条！你猜有多少图片～

利剑出鞘

安装成功之后，跟我一步步地完成一个网站的抓取，你就会明白 PySpider 的基本用法了。

命令行下执行

```
pyspider all
```

这句命令的意思是，运行 `pyspider` 并启动它的所有组件。

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

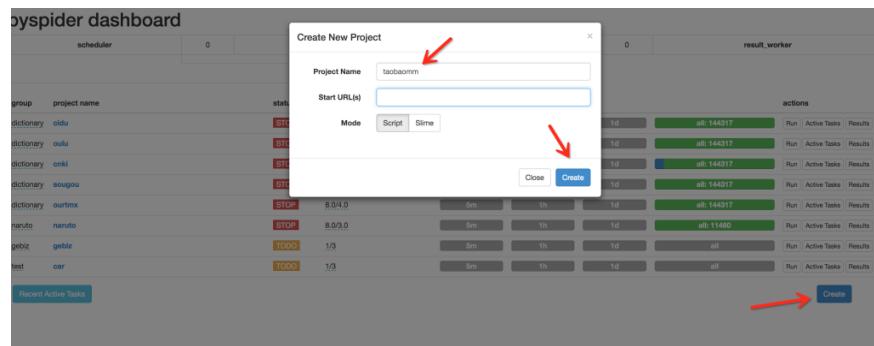
[ more ]        More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
[CQC-MAC:~ GERMY$ pyspider all
Web server running on port 25555
[I 160325 00:33:18 result_worker:49] result_worker starting...
[I 160325 00:33:19 scheduler:453] loading projects
[I 160325 00:33:19 processor:208] processor starting...
[I 160325 00:33:19 tornado_fetcher:429] fetcher starting...
[I 160325 00:33:19 scheduler:394] in 5m: new:0,success:0,retry:0,failed:0
[I 160325 00:33:19 app:74] webui running on http://0.0.0.0:5000/
```

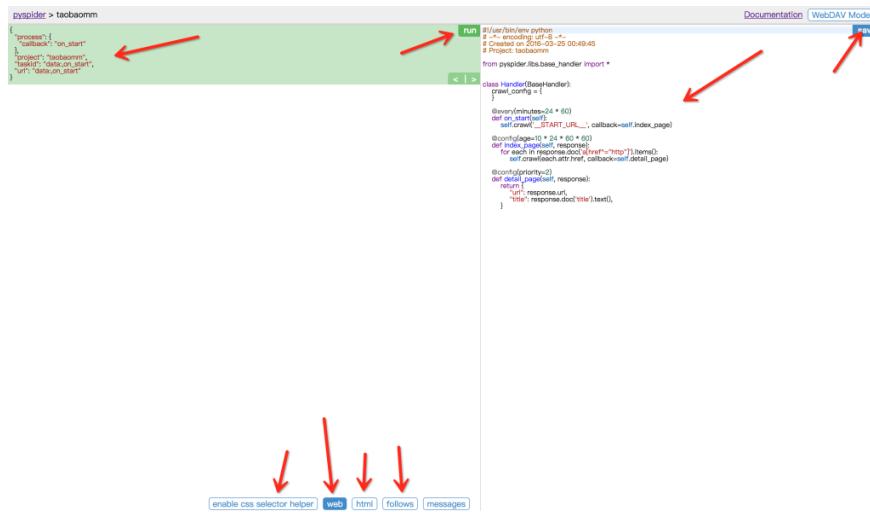
可以发现程序已经正常启动，并在 5000 这个端口运行。

一触即发

接下来在浏览器中输入 <http://localhost:5000>，可以看到 PySpider 的主界面，点击右下角的 **Create**，命名为 `taobaomm`，当然名称你可以随意取，继续点击 **Create**。



这样我们会进入到一个爬取操作的页面。



整个页面分为两栏，左边是爬取页面预览区域，右边是代码编写区域。
下面对区块进行说明：

左侧绿色区域：这个请求对应的 JSON 变量，在 PySpider 中，其实每个请求都有与之对应的 JSON 变量，包括回调函数，方法名，请求链接，请求数据等等。

绿色区域右上角Run：点击右上角的 run 按钮，就会执行这个请求，可以在左边的白色区域出现请求的结果。

左侧 enable css selector helper：抓取页面之后，点击此按钮，可以方便地获取页面中某个元素的 CSS 选择器。

左侧 web：即抓取的页面的实时预览图。

左侧 html：抓取页面的 HTML 代码。

左侧 follows：如果当前抓取方法中又新建了爬取请求，那么接下来的请求就会出现在 follows 里。

左侧 messages：爬取过程中输出的一些信息。

右侧代码区域：你可以在右侧区域书写代码，并点击右上角的 Save 按钮保存。

右侧 WebDAV Mode：打开调试模式，左侧最大化，便于观察调试。

乘胜追击

依然是上一节的那个网址，

https://mm.taobao.com/json/request_top_list.htm?page=1，其中 page 参数代表页码。所以我们暂时抓取前 30 页。页码到最后可以随意调整。

首先我们定义基地址，然后定义爬取的页码和总页码。

```
from pyspider.libs.base_handler import *

class Handler(BaseHandler):
    crawl_config = {
    }

    def __init__(self):
        self.base_url = 'https://mm.taobao.com/json/request_top_'
        self.page_num = 1
        self.total_num = 30

    @every(minutes=24 * 60)
    def on_start(self):
        while self.page_num <= self.total_num:
            url = self.base_url + str(self.page_num)
            print url
            self.crawl(url, callback=self.index_page)
            self.page_num += 1

    @config(age=10 * 24 * 60 * 60)
    def index_page(self, response):
        for each in response.doc('a[href^="http"]').items():
            self.crawl(each.attr.href, callback=self.detail_page)

    @config(priority=2)
    def detail_page(self, response):
        return {
            "url": response.url,
            "title": response.doc('title').text(),
        }
```

点击 save 保存代码，然后点击左边的 run，运行代码。

```

pyspider > taobaojm
{
    "process": {
        "callback": "on_start"
    },
    "project": "taobaojm",
    "request": "https://mm.taobao.com/json/request_top_list.htm?page=1",
    "url": "data_on_start"
}

```

```

run 21 user@0:~/env/pyspider
# -.- exposing port 8080 --#
# Created on 2016-03-25 00:59:45
# A project named 'taobaojm' was created.

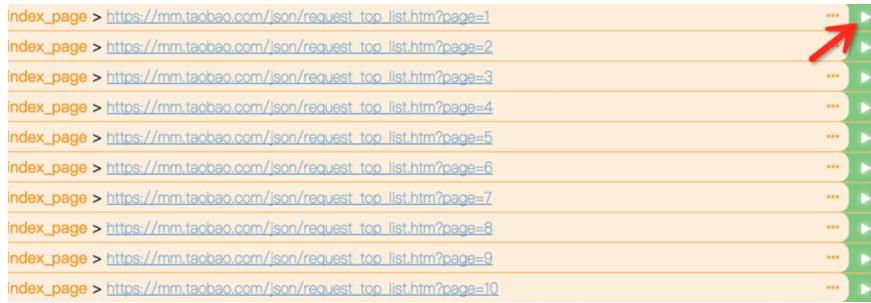
from pyspider.libs.base_handler import *

<|> class Handler(BaseHandler):
    config = {
        'url': 'http://www.taobao.com',
        'base_url': 'https://mm.taobao.com/json/request_top_list.htm?page='
    }
    total_page_num = 30
    total_page_index = 1
    page_num = 1
    @every(50 * minutes * 24 * 60)
    def on_start(self):
        self.page_num += self.total_page_num
        if self.page_num <= self.total_page_num:
            print url
            print self.base_url + str(self.page_num)
            self.page_index_callback(self.page_index)
            self.page_num += 1
    @config(pages=10, 24 * 60 * 60)
    def on_response(self, response):
        for each in response.doc.select('a[href^="http"]'):
            each.attr['callback'] = 'data_on_start'
            each.attr['url'] = each.attr['href']
            each.attr['callback'] = 'data_on_start'
    @config(follow=2)
    def on_detail_page(self, response):
        url = response.doc.url
        if url:
            self.follow(url)

```

运行后我们会发现 `follows` 出现了 30 这个数字，说明我们接下来有 30 个新请求，点击可查看所有爬取列表。另外控制台也有输出，将所有要爬取的 URL 打印了出来。

然后我们点击左侧任意一个绿色箭头，可以继续爬取这个页面。例如点击第一个 URL，来爬取这个 URL



点击之后，再查看下方的 `web` 页面，可以预览实时页面，这个页面被我们爬取了下来，并且回调到 `index_page` 函数来处理，目前 `index_page` 函数我们还没有处理，所以是继续构件了所有的链接请求。

田媛媛 26岁 广州市 加关注
平面模特 设计师 T台、展模特 164433粉丝

1 总积分: 60742

- 新增积分: 529
- 好评率: 90.0 %
- 导购照片: 887 张
- 签约数量: 406 次

2016春装新款韩版外搭亮丝针织衫女开衫短外套短款薄毛衣

喜欢 0

v悦悦 26岁 杭州市 加关注
平面模特 演员 148672粉丝

2 总积分: 59978

- 新增积分: 586
- 好评率: 100.0 %
- 导购照片: 3034 张
- 签约数量: 144 次

冬季通勤加厚保暖中长款羊毛外套

enable css selector helper web html follow 30 messages

好，接下来我们怎么办？当然是进入到 MM 到个人页面去爬取了。

如火如荼

爬取到了 MM 的列表，接下来就要进入到 MM 详情页了，修改 `index_page` 方法。

```
def index_page(self, response):
    for each in response.doc('.lady-name').items():
        self.crawl(each.attr.href, callback=self.detail_page)
```

其中 `response` 就是刚才爬取的列表页，`response` 其实就相当于列表页的 `html` 代码，利用 `doc` 函数，其实是调用了 `PyQuery`，用 CSS 选择器得到每一个MM的链接，然后重新发起新的请求。

比如，我们这里拿到的 `each.attr.href` 可能是 `mm.taobao.com/self/model_card.htm?user_id=687471686`，在这里继续调用了 `crawl` 方法，代表继续抓取这个链接的详情。

```
self.crawl(each.attr.href, callback=self.detail_page)
```

然后回调函数就是 `detail_page`, 爬取的结果会作为 `response` 变量传过去。`detail_page` 接到这个变量继续下面的分析。



```
pyspider > taobaomm
{
  "fetch": 0,
  "process": {
    "callback": "index_page"
  },
  "project": "taobaomm",
  "schedule": {
    "age": 864000
  },
  "taskid": "0a10868bf7990a78c3874d68118f25c1c",
  "url": "https://mm.taobao.com/json/request_top_list.htm?page=1"
}
```

田媛媛 26岁 广州市 加关注

平面模特 设计师 T台、展模特 164433粉丝

1
总积分: 60742

- 新增积分: 529
- 好评率: 90.0 %
- 导购照片: 887 张
- 签约数量: 406 次

好, 我们继续点击 `run` 按钮, 开始下一个页面的爬取。得到的结果是这样的。



```
pyspider > taobaomm
{
  "fetch": 0,
  "process": {
    "callback": "detail_page"
  },
  "project": "taobaomm",
  "schedule": {
    "priority": 2
  },
  "taskid": "51ba21b146d78da35982204a646ac051",
  "url": "https://mm.taobao.com/self/model_card.htm?user_id=687471686"
}

{'title': '海女郎 – 田媛媛',  
'url': 'https://mm.taobao.com/self/model_info.htm?user_id=687471686&is_coment=false'}
```

淘女郎 

首页 / 潮品汇 / 搭配购 / 美人库 / 活动 / 个人中心



相册

爱秀

模特卡

她的资料

她的拍片

她的活动

enable css selector helper web html follows messages

哦, 有些页面没有加载出来, 这是为什么?

在之前的文章说过，这个页面比较特殊，右边的页面使用 JS 渲染生成的，而普通的抓取是不能得到 JS 渲染后的页面的，这可麻烦了。

然而，幸运的是，PySpider 提供了动态解析 JS 的机制。

友情提示：可能有的小伙伴不知道 PhantomJS，可以参考[爬虫JS动态解析](#)

因为我们在前面装好了 PhantomJS，所以，这时候就轮到它来出场了。在最开始运行 PySpider 的时候，使用了 `pyspider all` 命令，这个命令是把 PySpider 所有的组件启动起来，其中也包括 PhantomJS。

所以我们代码怎么改呢？很简单。

```
def index_page(self, response):
    for each in response.doc('.lady-name').items():
        self.crawl(each.attr.href, callback=self.detail_page, fe
```

只是简单地加了一个 `fetch_type='js'`，点击绿色的返回箭头，重新运行一下。

可以发现，页面已经被我们成功加载出来了，简直不能更帅！

pyspider > taobaomm

```
{"schedule": {
    "priority": 2
},
"taskid": "51ba21b146d78da35982204a646ac051",
"url": "https://mm.taobao.com/self/model_card.htm?user_id=687471686"
}

{'title': u'淘女郎 - 田媛媛',
'url': u'https://mm.taobao.com/self/model_info.htm?user_id=687471686&is_coment=false'}
```

淘宝网 亲，请登录 免费注册 联系客服 网站导航

淘女郎 首页 / 潮品汇 / 搭配购 / 美人库 / 活动 / 个人中心

基本信息

昵称：田媛媛
生 日：公历 06月22日 所在城市：广州市
职 业：平面模特 设计师 T台、展模特 血 型：
学校/专业：广东纺织职业技术学院 服装设计与展示
风 格：欧美 韩版 街头

168.0CM 46.0KG 83-62-89 34 38码

拍摄信息
拍片价位：¥ 0 / 天
希望代言：

个性域名
域名地址：//mm.taobao.com/tyy6160

看下面的个性域名，所有我们需要的 MM 图片都在那里面了，所以我们需要继续抓取这个页面。

胜利在望

好，继续修改 `detail_page` 方法，然后增加一个 `domain_page` 方法，用来处理每个 MM 的个性域名。

```
def detail_page(self, response):
    domain = 'https:' + response.doc('.mm-p-domain-info li > span').text
    print domain
    self.crawl(domain, callback=self.domain_page)

def domain_page(self, response):
    pass
```

好，继续重新 run，预览一下页面，终于，我们看到了 MM 的所有图片。



嗯，你懂得！

只欠东风

好，照片都有了，那么我们就偷偷地下载下来吧～

完善 `domain_page` 代码，实现保存简介和遍历保存图片的方法。

在这里，PySpider 有一个特点，所有的 `request` 都会保存到一个队列中，并具有去重和自动重试机制。所以，我们最好的解决方法是，把每张图片的请求都写成一个 `request`，然后成功后用文件写入即可，这样会避免图片加载不全的问题。

曾经在之前文章写过图片下载和文件夹创建的过程，在这里就不多赘述原理了，直接上写好的工具类，后面会有完整代码。

```
import os

class Deal:
    def __init__(self):
        self.path = DIR_PATH
        if not self.path.endswith('/'):
            self.path = self.path + '/'
        if not os.path.exists(self.path):
            os.makedirs(self.path)

    def mkdir(self, path):
        path = path.strip()
        dir_path = self.path + path
        exists = os.path.exists(dir_path)
        if not exists:
            os.makedirs(dir_path)
            return dir_path
        else:
            return dir_path

    def saveImg(self, content, path):
        f = open(path, 'wb')
        f.write(content)
        f.close()

    def saveBrief(self, content, dir_path, name):
        file_name = dir_path + "/" + name + ".txt"
        f = open(file_name, "w+")
        f.write(content.encode('utf-8'))

    def getExtension(self, url):
        extension = url.split('.')[ -1]
        return extension
```

这里面包含了四个方法。

mkdir: 创建文件夹，用来创建 MM 名字对应的文件夹。
saveBrief: 保存简介，保存 MM 的文字简介。 **saveImg:** 传入图片二进制流以及保存路径，存储图片。 **getExtension:** 获得链接的后缀名，通过图片 URL 获得。然后在 domain_page 中具体实现如下

```

def domain_page(self, response):
    name = response.doc('.mm-p-model-info-left-top dd > a').text
    dir_path = self.deal.mkdir(name)
    brief = response.doc('.mm-aixiu-content').text()
    if dir_path:
        imgs = response.doc('.mm-aixiu-content img').items()
        count = 1
        self.deal.saveBrief(brief, dir_path, name)
        for img in imgs:
            url = img.attr.src
            if url:
                extension = self.deal.getExtension(url)
                file_name = name + str(count) + '.' + extension
                count += 1
                self.crawl(img.attr.src, callback=self.save_img,
                           save={'dir_path': dir_path, 'file_name': file_name})

def save_img(self, response):
    content = response.content
    dir_path = response.save['dir_path']
    file_name = response.save['file_name']
    file_path = dir_path + '/' + file_name
    self.deal.saveImg(content, file_path)

```

以上方法首先获取了页面的所有文字，然后调用了 `saveBrief` 方法存储简介。

然后遍历了 MM 所有的图片，并通过链接获取后缀名，和 MM 的姓名以及自增计数组合成一个新的文件名，调用 `saveImg` 方法保存图片。

炉火纯青

好，基本的东西都写好了。

接下来。继续完善一下代码。第一版本完成。

版本一功能：按照淘宝MM姓名分文件夹，存储MM的 txt 文本简介以及所有美图至本地。

可配置项：

PAGE_START: 列表开始页码	PAGE_END: 列表结束页码
DIR_PATH: 资源保存路径	

```

#!/usr/bin/env python
# -*- encoding: utf-8 -*-
# Created on 2016-03-25 00:59:45
# Project: taobaomm

from pyspider.libs.base_handler import *

PAGE_START = 1
PAGE_END = 30
DIR_PATH = '/var/py/mm'

class Handler(BaseHandler):
    crawl_config = {
    }

    def __init__(self):
        self.base_url = 'https://mm.taobao.com/json/request_top'
        self.page_num = PAGE_START
        self.total_num = PAGE_END
        self.deal = Deal()

    def on_start(self):
        while self.page_num <= self.total_num:
            url = self.base_url + str(self.page_num)
            self.crawl(url, callback=self.index_page)
            self.page_num += 1

    def index_page(self, response):
        for each in response.doc('.lady-name').items():
            self.crawl(each.attr.href, callback=self.detail_page)

    def detail_page(self, response):
        domain = response.doc('.mm-p-domain-info li > span').text()
        if domain:
            page_url = 'https:' + domain
            self.crawl(page_url, callback=self.domain_page)

    def domain_page(self, response):
        name = response.doc('.mm-p-model-info-left-top dd > a').text()
        dir_path = self.deal.mkdir(name)
        brief = response.doc('.mm-aixiu-content').text()
        if dir_path:
            imgs = response.doc('.mm-aixiu-content img').items()
            count = 1
            self.deal.saveBrief(brief, dir_path, name)
            for img in imgs:
                url = img.attr.src

```

```

        if url:
            extension = self.deal.getExtension(url)
            file_name = name + str(count) + '.' + extension
            count += 1
            self.crawl(img.attr.src, callback=self.save_
                        save={'dir_path': dir_path, 'file'

def save_img(self, response):
    content = response.content
    dir_path = response.save['dir_path']
    file_name = response.save['file_name']
    file_path = dir_path + '/' + file_name
    self.deal.saveImg(content, file_path)

import os

class Deal:
    def __init__(self):
        self.path = DIR_PATH
        if not self.path.endswith('/'):
            self.path = self.path + '/'
        if not os.path.exists(self.path):
            os.makedirs(self.path)

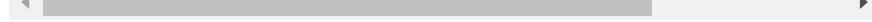
    def mkdir(self, path):
        path = path.strip()
        dir_path = self.path + path
        exists = os.path.exists(dir_path)
        if not exists:
            os.makedirs(dir_path)
            return dir_path
        else:
            return dir_path

    def saveImg(self, content, path):
        f = open(path, 'wb')
        f.write(content)
        f.close()

    def saveBrief(self, content, dir_path, name):
        file_name = dir_path + "/" + name + ".txt"
        f = open(file_name, "w+")
        f.write(content.encode('utf-8'))

    def getExtension(self, url):
        extension = url.split('.')[ -1]
        return extension

```



粘贴到你的 PySpider 中运行吧~

其中有一些知识点，我会在后面作详细的用法总结。大家可以先体会一下代码。

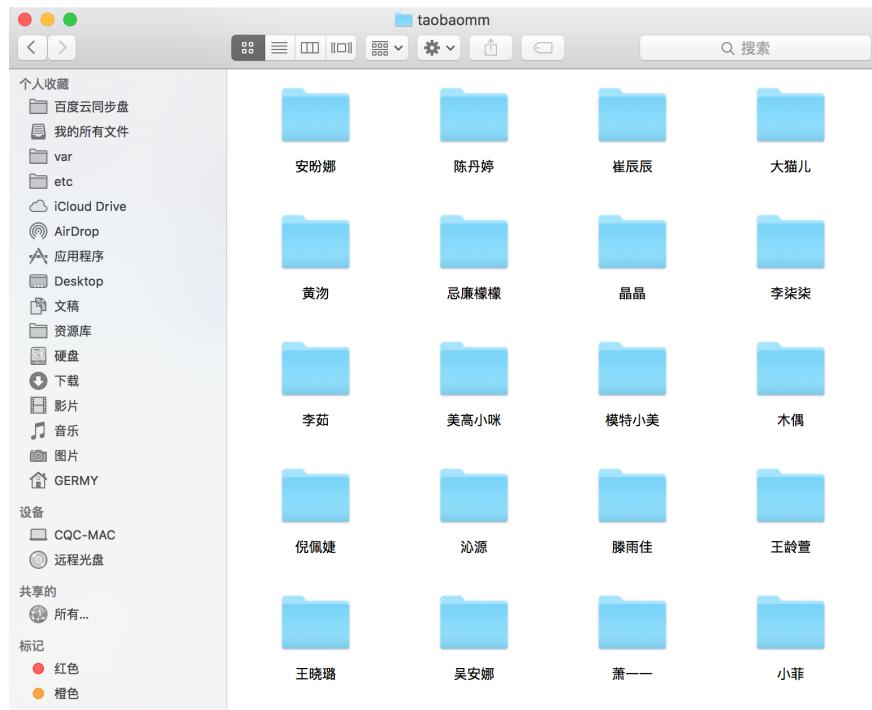
pyspider dashboard

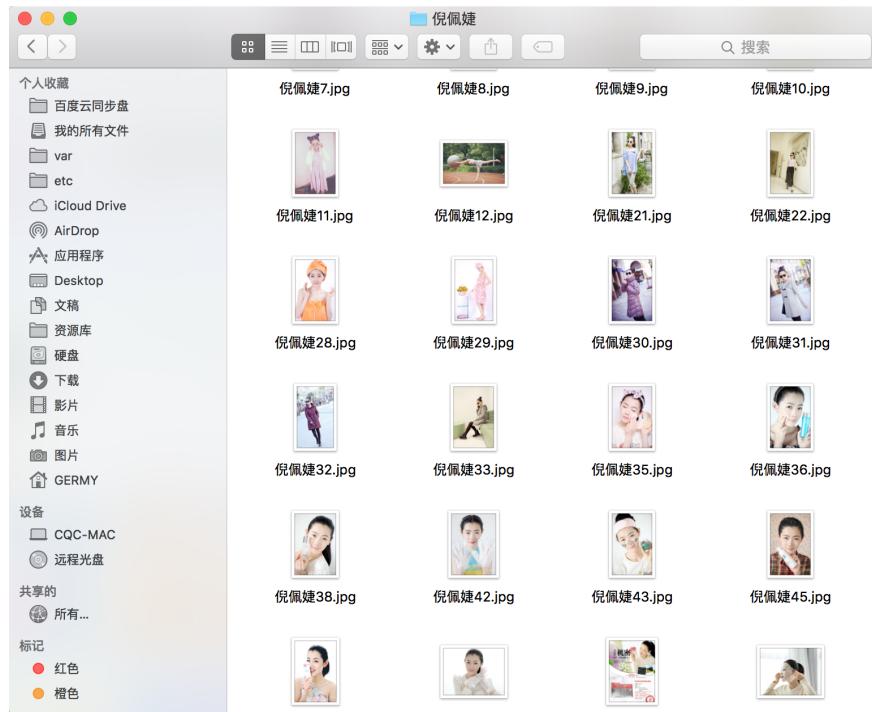
scheduler	0	fetcher	0	processor	0	result_worker	
0 + 0							
group	project name	status	rate/burst	avg time	progress	actions	
dictionary	oudi	STOP	3.0/5.0	5m	1h	1d	all: 144317
dictionary	oulu	STOP	4.0/4.0	5m	1h	1d	all: 144317
dictionary	cnnki	STOP	2.0/5.0	5m	1h	1d	all: 144317
dictionary	sougou	STOP	7.0/5.0	5m	1h	1d	all: 144317
dictionary	ourtmx	STOP	8.0/4.0	5m	1h	1d	all: 144317
naruto	naruto	STOP	8.0/3.0	5m	1h	1d	all: 11460
getit	gebliz	TODD	1/3	5m	1h	1d	all
test	car	TODD	1/3	5m	1h	1d	Run Active Tasks Results
mm	taobaomm	DEBUG	1/3	5m	1h	1d	all Run Active Tasks Results

Recent Active Tasks

Create

保存之后，点击下方的 run，你会发现，海量的 MM 图片已经涌入你的电脑啦~





需要解释？需要我也不解释！

项目代码

[TaobaoMM – GitHub](#)

尚方宝剑

如果想了解 PySpider 的更多内容，可以查看[官方文档](#)。

Python爬虫进阶五之多线程的用法

前言

我们之前写的爬虫都是单个线程的？这怎么够？一旦一个地方卡到不动了，那不就永远等待下去了？为此我们可以使用多线程或者多进程来处理。

首先声明一点！

多线程和多进程是不一样的！一个是 `thread` 库，一个是 `multiprocessing` 库。而多线程 `thread` 在 Python 里面被称作鸡肋的存在！而没错！本节介绍的是就是这个库 `thread`。

不建议你用这个，不过还是介绍了，如果想看可以看看下面，不想浪费时间直接看 [multiprocessing 多进程](#)

鸡肋点

名言：

“Python下多线程是鸡肋，推荐使用多进程！”那当然有同学会问了，为啥？

背景

1、GIL是什么？

GIL的全称是Global Interpreter Lock(全局解释器锁)，来源是python设计之初的考虑，为了数据安全所做的决定。

2、每个CPU在同一时间只能执行一个线程（在单核CPU下的多线程其实都只是并发，不是并行，并发和并行从宏观上来讲都是同时处理多路请求的概念。但并发和并行又有区别，并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔内发生。）

在Python多线程下，每个线程的执行方式：

- 获取GIL
- 执行代码直到sleep或者是python虚拟机将其挂起。
- 释放GIL

可见，某个线程想要执行，必须先拿到GIL，我们可以把GIL看作是“通行证”，并且在一个python进程中，GIL只有一个。拿不到通行证的线程，就不允许进入CPU执行。

在Python2.x里，GIL的释放逻辑是当前线程遇见IO操作或者ticks计数达到100（ticks可以看作是Python自身的一个计数器，专门做用于GIL，每次释放后归零，这个计数可以通过`sys.setcheckinterval`来调整），进行释放。

而每次释放GIL锁，线程进行锁竞争、切换线程，会消耗资源。并且由于GIL锁存在，python里一个进程永远只能同时执行一个线程(拿到GIL的线程才能执行)，这就是为什么在多核CPU上，python的多线程效率并不高。

那么是不是python的多线程就完全没用了呢？

在这里我们进行分类讨论：

1、CPU密集型代码(各种循环处理、计数等等)，在这种情况下，由于计算工作多，ticks计数很快就会达到阈值，然后触发GIL的释放与再竞争（多个线程来回切换当然是需要消耗资源的），所以python下的多线程对CPU密集型代码并不友好。

2、IO密集型代码(文件处理、网络爬虫等)，多线程能够有效提升效率(单线程下有IO操作会进行IO等待，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU的资源，从而能提升程序执行效率)。所以python的多线程对IO密集型代码比较友好。

而在python3.x中，GIL不使用ticks计数，改为使用计时器（执行时间达到阈值后，当前线程释放GIL），这样对CPU密集型程序更加友好，但依然没有解决GIL导致的同一时间只能执行一个线程的问题，所以效率依然不尽如人意。

多核性能

多核多线程比单核多线程更差，原因是单核下多线程，每次释放GIL，唤醒的那个线程都能获取到GIL锁，所以能够无缝执行，但多核下，CPU0释放GIL后，其他CPU上的线程都会进行竞争，但GIL可能会马上又被CPU0拿到，导致其他几个CPU上被唤醒后的线程会醒着等待到切换时间后又进入待调度状态，这样会造成线程颠簸(thrashing)，导致效率更低

多进程为什么不会这样？

每个进程有各自独立的GIL，互不干扰，这样就可以真正意义上的并行执行，所以在python中，多进程的执行效率优于多线程(仅仅针对多核CPU而言)。

所以在这里说结论：多核下，想做并行提升效率，比较通用的方法是使用多进程，能够有效提高执行效率。

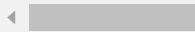
所以，如果不想浪费时间，可以直接看多进程。

直接利用函数创建多线程

Python中使用线程有两种方式：函数或者用类来包装线程对象。

函数式：调用**thread**模块中的**start_new_thread()**函数来产生新线程。语法如下：

```
<span class="s1">thread</span><span class="s2">. </span><span cla
```



参数说明：

- **function** – 线程函数。
- **args** – 传递给线程函数的参数,他必须是个**tuple**类型。
- **kwargs** – 可选参数。

先用一个实例感受一下：

```
# -*- coding: UTF-8 -*-

import thread
import time

# 为线程定义一个函数
def print_time(threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % (threadName, time.ctime(time.time()))

# 创建两个线程
try:
    thread.start_new_thread(print_time, ("Thread-1", 2,))
    thread.start_new_thread(print_time, ("Thread-2", 4,))
except:
    print "Error: unable to start thread"

while 1:
    pass

print "Main Finished"
```

运行结果如下：

```
Thread-1: Thu Nov  3 16:43:01 2016
Thread-2: Thu Nov  3 16:43:03 2016
Thread-1: Thu Nov  3 16:43:03 2016
Thread-1: Thu Nov  3 16:43:05 2016
Thread-2: Thu Nov  3 16:43:07 2016
Thread-1: Thu Nov  3 16:43:07 2016
Thread-1: Thu Nov  3 16:43:09 2016
Thread-2: Thu Nov  3 16:43:11 2016
Thread-2: Thu Nov  3 16:43:15 2016
Thread-2: Thu Nov  3 16:43:19 2016
```

可以发现，两个线程都在执行，睡眠2秒和4秒后打印输出一段话。

注意到，在主线程写了

```
while 1:
    pass
```

这是让主线程一直在等待

如果去掉上面两行，那就直接输出

```
Main Finished
```

程序执行结束。

使用**Threading**模块创建线程

使用**Threading**模块创建线程，直接从**threading.Thread**继承，然后重写**init**方法和**run**方法：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import threading
import time

import thread

exitFlag = 0

class myThread (threading.Thread):      #继承父类threading.Thread
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):                      #把要执行的代码写到run函数里面
        print "Starting " + self.name
        print_time(self.name, self.counter, 5)
        print "Exiting " + self.name

    def print_time(threadName, delay, counter):
        while counter:
            if exitFlag:
                thread.exit()
            time.sleep(delay)
            print "%s: %s" % (threadName, time.ctime(time.time()))
            counter -= 1

# 创建新线程
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# 开启线程
thread1.start()
thread2.start()

print "Exiting Main Thread"
```

运行结果：

```
Starting Thread-1Starting Thread-2

Exiting Main Thread
Thread-1: Thu Nov  3 18:42:19 2016
Thread-2: Thu Nov  3 18:42:20 2016
Thread-1: Thu Nov  3 18:42:20 2016
Thread-1: Thu Nov  3 18:42:21 2016
Thread-2: Thu Nov  3 18:42:22 2016
Thread-1: Thu Nov  3 18:42:22 2016
Thread-1: Thu Nov  3 18:42:23 2016
Exiting Thread-1
Thread-2: Thu Nov  3 18:42:24 2016
Thread-2: Thu Nov  3 18:42:26 2016
Thread-2: Thu Nov  3 18:42:28 2016
Exiting Thread-2
```

有没有发现什么奇怪的地方？打印的输出格式好奇怪。比如第一行之后应该是一个回车的，结果第二个进程就打印出来了。

那是因为什么？因为这几个线程没有设置同步。

线程同步

如果多个线程共同对某个数据修改，则可能出现不可预料的结果，为了保证数据的正确性，需要对多个线程进行同步。

使用**Thread**对象的**Lock**和**Rlock**可以实现简单的线程同步，这两个对象都有**acquire**方法和**release**方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到**acquire**和**release**方法之间。如下：

多线程的优势在于可以同时运行多个任务（至少感觉起来是这样）。但是当线程需要共享数据时，可能存在数据不同步的问题。

考虑这样一种情况：一个列表里所有元素都是0，线程“**set**”从后向前把所有元素改成1，而线程“**print**”负责从前往后读取列表并打印。

那么，可能线程“**set**”开始改的时候，线程“**print**”便来打印列表了，输出就成了一半0一半1，这就是数据的不同步。为了避免这种情况，引入了锁的概念。

锁有两种状态——锁定和未锁定。每当一个线程比如“**set**”要访问共享数据时，必须先获得锁定；如果已经有别的线程比如“**print**”获得锁定了，那么就让线程“**set**”暂停，也就是同步阻塞；等到线程“**print**”访问完毕，释放锁以后，再让线程“**set**”继续。

经过这样的处理，打印列表时要么全部输出0，要么全部输出1，不会再出现一半0一半1的尴尬场面。

看下面的例子：

```

# -*- coding: UTF-8 -*-

import threading
import time

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        print "Starting " + self.name
        # 获得锁，成功获得锁定后返回True
        # 可选的timeout参数不填时将一直阻塞直到获得锁定
        # 否则超时后将返回False
        threadLock.acquire()
        print_time(self.name, self.counter, 3)
        # 释放锁
        threadLock.release()

def print_time(threadName, delay, counter):
    while counter:
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1

threadLock = threading.Lock()
threads = []

# 创建新线程
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# 开启新线程
thread1.start()
thread2.start()

# 添加线程到线程列表
threads.append(thread1)
threads.append(thread2)

# 等待所有线程完成
for t in threads:
    t.join()

print "Exiting Main Thread"

```

在上面的代码中运用了线程锁还有join等待。

运行结果如下：

```
Starting Thread-1
Starting Thread-2
Thread-1: Thu Nov  3 18:56:49 2016
Thread-1: Thu Nov  3 18:56:50 2016
Thread-1: Thu Nov  3 18:56:51 2016
Thread-2: Thu Nov  3 18:56:53 2016
Thread-2: Thu Nov  3 18:56:55 2016
Thread-2: Thu Nov  3 18:56:57 2016
Exiting Main Thread
```

这样一来，你可以发现就不会出现刚才的输出混乱的结果了。

线程优先级队列

Python的**Queue**模块中提供了同步的、线程安全的队列类，包括**FIFO**（先入先出）队列**Queue**, **LIFO**（后入先出）队列**LifoQueue**, 和优先级队列**PriorityQueue**。这些队列都实现了锁原语，能够在多线程中直接使用。可以使用队列来实现线程间的同步。

Queue模块中的常用方法：

- **Queue.qsize()** 返回队列的大小
- **Queue.empty()** 如果队列为空，返回**True**, 反之**False**
- **Queue.full()** 如果队列满了，返回**True**, 反之**False**
- **Queue.full** 与 **maxsize** 大小对应
- **Queue.get([block[, timeout]])** 获取队列，**timeout**等待时间
- **Queue.get_nowait()** 相当**Queue.get(False)**
- **Queue.put(item)** 写入队列，**timeout**等待时间
- **Queue.put_nowait(item)** 相当**Queue.put(item, False)**
- **Queue.task_done()** 在完成一项工作之后，**Queue.task_done()**函数向任务已经完成的队列发送一个信号
- **Queue.join()** 实际上意味着等到队列为空，再执行别的操作

用一个实例感受一下：

```

# -*- coding: UTF-8 -*-

import Queue
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.q = q
    def run(self):
        print "Starting " + self.name
        process_data(self.name, self.q)
        print "Exiting " + self.name

def process_data(threadName, q):
    while not exitFlag:
        queueLock.acquire()
        if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s processing %s" % (threadName, data)
        else:
            queueLock.release()
            time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workQueue = Queue.Queue(10)
threads = []
threadID = 1

# 创建新线程
for tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    threads.append(thread)
    threadID += 1

# 填充队列
queueLock.acquire()
for word in nameList:
    workQueue.put(word)

```

```
queueLock.release()

# 等待队列清空
while not workQueue.empty():
    pass

# 通知线程是时候退出
exitFlag = 1

# 等待所有线程完成
for t in threads:
    t.join()
print "Exiting Main Thread"
```

运行结果：

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-3 processing One
Thread-1 processing Two
Thread-2 processing Three
Thread-3 processing Four
Thread-2 processing Five
Exiting Thread-2
Exiting Thread-3
Exiting Thread-1
Exiting Main Thread
```

上面的例子用了FIFO队列。当然你也可以换成其他类型的队列。

参考文章

1. <http://bbs.51cto.com/thread-1349105-1.html>
2. <http://www.runoob.com/python/python-multithreading.html>

Python爬虫进阶六之多进程的用法

前言

在上一节中介绍了**thread**多线程库。**python**中的多线程其实并不是真正的多线程，并不能做到充分利用多核CPU资源。

如果想要充分利用，在**python**中大部分情况需要使用多进程，那么这个包就叫做 **multiprocessing**。

借助它，可以轻松完成从单进程到并发执行的转换。**multiprocessing**支持子进程、通信和共享数据、执行不同形式的同步，提供了**Process**、**Queue**、**Pipe**、**Lock**等组件。

那么本节要介绍的内容有：

- **Process**
- **Lock**
- **Semaphore**
- **Queue**
- **Pipe**
- **Pool**

Process

基本使用

在**multiprocessing**中，每一个进程都用一个**Process**类来表示。首先看下它的API

```
Process([group [, target [, name [, args [, kwargs]]]]])
```

- **target**表示调用对象，你可以传入方法的名字
- **args**表示被调用对象的位置参数元组，比如**target**是函数**a**，他有两个参数**m, n**，那么**args**就传入(**m, n**)即可
- **kwargs**表示调用对象的字典
- **name**是别名，相当于给这个进程取一个名字
- **group**分组，实际上不使用

我们先用一个实例来感受一下：

```
import multiprocessing

def process(num):
    print 'Process:', num

if __name__ == '__main__':
    for i in range(5):
        p = multiprocessing.Process(target=process, args=(i,))
        p.start()
```

最简单的创建Process的过程如上所示，`target`传入函数名，`args`是函数的参数，是元组的形式，如果只有一个参数，那就是长度为1的元组。

然后调用`start()`方法即可启动多个进程了。

另外你还可以通过`cpu_count()`方法还有`active_children()`方法获取当前机器的CPU核心数量以及得到目前所有的运行的进程。

通过一个实例来感受一下：

```
import multiprocessing
import time

def process(num):
    time.sleep(num)
    print 'Process:', num

if __name__ == '__main__':
    for i in range(5):
        p = multiprocessing.Process(target=process, args=(i,))
        p.start()

    print('CPU number:' + str(multiprocessing.cpu_count()))
    for p in multiprocessing.active_children():
        print('Child process name: ' + p.name + ' id: ' + str(p.

    print('Process Ended')
```

运行结果：

```
Process: 0
CPU number:8
Child process name: Process-2 id: 9641
Child process name: Process-4 id: 9643
Child process name: Process-5 id: 9644
Child process name: Process-3 id: 9642
Process Ended
Process: 1
Process: 2
Process: 3
Process: 4
```

自定义类

另外你还可以继承**Process**类，自定义进程类，实现**run**方法即可。

用一个实例来感受一下：

```
from multiprocessing import Process
import time

class MyProcess(Process):
    def __init__(self, loop):
        Process.__init__(self)
        self.loop = loop

    def run(self):
        for count in range(self.loop):
            time.sleep(1)
            print('Pid: ' + str(self.pid) + ' LoopCount: ' + str(count))

if __name__ == '__main__':
    for i in range(2, 5):
        p = MyProcess(i)
        p.start()
```

在上面的例子中，我们继承了**Process**这个类，然后实现了**run**方法。打印出来了进程号和参数。

运行结果：

```
Pid: 28116 LoopCount: 0
Pid: 28117 LoopCount: 0
Pid: 28118 LoopCount: 0
Pid: 28116 LoopCount: 1
Pid: 28117 LoopCount: 1
Pid: 28118 LoopCount: 1
Pid: 28117 LoopCount: 2
Pid: 28118 LoopCount: 2
Pid: 28118 LoopCount: 3
```

可以看到，三个进程分别打印出了2、3、4条结果。

我们可以把一些方法独立的写在每个类里封装好，等用的时候直接初始化一个类运行即可。

deamon

在这里介绍一个属性，叫做`deamon`。每个线程都可以单独设置它的属性，如果设置为`True`，当父进程结束后，子进程会自动被终止。

用一个实例来感受一下，还是原来的例子，增加了`deamon`属性：

```
from multiprocessing import Process
import time

class MyProcess(Process):
    def __init__(self, loop):
        Process.__init__(self)
        self.loop = loop

    def run(self):
        for count in range(self.loop):
            time.sleep(1)
            print('Pid: ' + str(self.pid) + ' LoopCount: ' + str(count))

if __name__ == '__main__':
    for i in range(2, 5):
        p = MyProcess(i)
        p.daemon = True
        p.start()

    print 'Main process Ended!'
```

在这里，调用的时候增加了设置**daemon**，最后的主进程（即父进程）打印输出了一句话。

运行结果：

```
Main process Ended!
```

结果很简单，因为主进程没有做任何事情，直接输出一句话结束，所以在这些时候也直接终止了子进程的运行。

这样可以有效防止无控制地生成子进程。如果这样写了，你在关闭这个主程序运行时，就无需额外担心子进程有没有被关闭了。

不过这样并不是我们想要达到的效果呀，能不能让所有子进程都执行完了然后再结束呢？那当然是可以的，只需要加入**join()**方法即可。

```
from multiprocessing import Process
import time

class MyProcess(Process):
    def __init__(self, loop):
        Process.__init__(self)
        self.loop = loop

    def run(self):
        for count in range(self.loop):
            time.sleep(1)
            print('Pid: ' + str(self.pid) + ' LoopCount: ' + str(count))

if __name__ == '__main__':
    for i in range(2, 5):
        p = MyProcess(i)
        p.daemon = True
        p.start()
        p.join()

print 'Main process Ended!'
```

在这里，每个子进程都调用了**join()**方法，这样父进程（主进程）就会等待子进程执行完毕。

运行结果：

```
Pid: 29902 LoopCount: 0  
Pid: 29902 LoopCount: 1  
Pid: 29905 LoopCount: 0  
Pid: 29905 LoopCount: 1  
Pid: 29905 LoopCount: 2  
Pid: 29912 LoopCount: 0  
Pid: 29912 LoopCount: 1  
Pid: 29912 LoopCount: 2  
Pid: 29912 LoopCount: 3  
Main process Ended!
```

发现所有子进程都执行完毕之后，父进程最后打印出了结束的结果。

Lock

在上面的一些小实例中，你可能会遇到如下的运行结果：

```
Pid: 44758 LoopCount: 0  
Pid: 44760 LoopCount: 0Pid: 44759 LoopCount: 0  
  
Pid: 44758 LoopCount: 1  
Pid: 44759 LoopCount: 1  
Pid: 44760 LoopCount: 1  
Pid: 44760 LoopCount: 2  
Pid: 44759 LoopCount: 2  
Pid: 44760 LoopCount: 3
```

什么问题？有的输出错位了。这是由于并行导致的，两个进程同时进行了输出，结果第一个进程的换行没有来得及输出，第二个进程就输出了结果。所以导致这种排版的问题。

那这归根结底是因为线程同时资源（输出操作）而导致的。

那怎么来避免这种问题？那自然是在某一时间，只能一个进程输出，其他进程等待。等刚才那个进程输出完毕之后，另一个进程再进行输出。这种现象就叫做“互斥”。

我们可以通过 **Lock** 来实现，在一个进程输出时，加锁，其他进程等待。等此进程执行结束后，释放锁，其他进程可以进行输出。

我们现用一个实例来感受一下：

```
from multiprocessing import Process, Lock
import time

class MyProcess(Process):
    def __init__(self, loop, lock):
        Process.__init__(self)
        self.loop = loop
        self.lock = lock

    def run(self):
        for count in range(self.loop):
            time.sleep(0.1)
            #self.lock.acquire()
            print('Pid: ' + str(self.pid) + ' LoopCount: ' + str(count))
            #self.lock.release()

if __name__ == '__main__':
    lock = Lock()
    for i in range(10, 15):
        p = MyProcess(i, lock)
        p.start()
```

首先看一下不加锁的输出结果：

```
Pid: 45755 LoopCount: 0
Pid: 45756 LoopCount: 0
Pid: 45757 LoopCount: 0
Pid: 45758 LoopCount: 0
Pid: 45759 LoopCount: 0
Pid: 45755 LoopCount: 1
Pid: 45756 LoopCount: 1
Pid: 45757 LoopCount: 1
Pid: 45758 LoopCount: 1
Pid: 45759 LoopCount: 1
Pid: 45755 LoopCount: 2Pid: 45756 LoopCount: 2

Pid: 45757 LoopCount: 2
Pid: 45758 LoopCount: 2
Pid: 45759 LoopCount: 2
Pid: 45756 LoopCount: 3
Pid: 45755 LoopCount: 3
Pid: 45757 LoopCount: 3
Pid: 45758 LoopCount: 3
Pid: 45759 LoopCount: 3
Pid: 45755 LoopCount: 4
Pid: 45756 LoopCount: 4
Pid: 45757 LoopCount: 4
Pid: 45759 LoopCount: 4
Pid: 45758 LoopCount: 4
Pid: 45756 LoopCount: 5
Pid: 45755 LoopCount: 5
Pid: 45757 LoopCount: 5
Pid: 45759 LoopCount: 5
Pid: 45758 LoopCount: 5
Pid: 45756 LoopCount: 6Pid: 45755 LoopCount: 6

Pid: 45757 LoopCount: 6
Pid: 45759 LoopCount: 6
Pid: 45758 LoopCount: 6
Pid: 45755 LoopCount: 7Pid: 45756 LoopCount: 7

Pid: 45757 LoopCount: 7
Pid: 45758 LoopCount: 7
Pid: 45759 LoopCount: 7
Pid: 45756 LoopCount: 8Pid: 45755 LoopCount: 8

Pid: 45757 LoopCount: 8
Pid: 45758 LoopCount: 8Pid: 45759 LoopCount: 8

Pid: 45755 LoopCount: 9
Pid: 45756 LoopCount: 9
Pid: 45757 LoopCount: 9
```

```
Pid: 45758 LoopCount: 9
Pid: 45759 LoopCount: 9
Pid: 45756 LoopCount: 10
Pid: 45757 LoopCount: 10
Pid: 45758 LoopCount: 10
Pid: 45759 LoopCount: 10
Pid: 45757 LoopCount: 11
Pid: 45758 LoopCount: 11
Pid: 45759 LoopCount: 11
Pid: 45758 LoopCount: 12
Pid: 45759 LoopCount: 12
Pid: 45759 LoopCount: 13
```

可以看到有些输出已经造成了影响。

然后我们对其加锁：

```
from multiprocessing import Process, Lock
import time

class MyProcess(Process):
    def __init__(self, loop, lock):
        Process.__init__(self)
        self.loop = loop
        self.lock = lock

    def run(self):
        for count in range(self.loop):
            time.sleep(0.1)
            self.lock.acquire()

            print('Pid: ' + str(self.pid) + ' LoopCount: ' + str(count))
            self.lock.release()

if __name__ == '__main__':
    lock = Lock()
    for i in range(10, 15):
        p = MyProcess(i, lock)
        p.start()
```

我们在**print**方法的前后分别添加了获得锁和释放锁的操作。这样就能保证在同一时间只有一个**print**操作。

看一下运行结果：

```
Pid: 45889 LoopCount: 0
Pid: 45890 LoopCount: 0
Pid: 45891 LoopCount: 0
Pid: 45892 LoopCount: 0
Pid: 45893 LoopCount: 0
Pid: 45889 LoopCount: 1
Pid: 45890 LoopCount: 1
Pid: 45891 LoopCount: 1
Pid: 45892 LoopCount: 1
Pid: 45893 LoopCount: 1
Pid: 45889 LoopCount: 2
Pid: 45890 LoopCount: 2
Pid: 45891 LoopCount: 2
Pid: 45892 LoopCount: 2
Pid: 45893 LoopCount: 2
Pid: 45889 LoopCount: 3
Pid: 45890 LoopCount: 3
Pid: 45891 LoopCount: 3
Pid: 45892 LoopCount: 3
Pid: 45893 LoopCount: 3
Pid: 45889 LoopCount: 4
Pid: 45890 LoopCount: 4
Pid: 45891 LoopCount: 4
Pid: 45892 LoopCount: 4
Pid: 45893 LoopCount: 4
Pid: 45889 LoopCount: 5
Pid: 45890 LoopCount: 5
Pid: 45891 LoopCount: 5
Pid: 45892 LoopCount: 5
Pid: 45893 LoopCount: 5
Pid: 45889 LoopCount: 6
Pid: 45890 LoopCount: 6
Pid: 45891 LoopCount: 6
Pid: 45892 LoopCount: 6
Pid: 45893 LoopCount: 6
Pid: 45892 LoopCount: 6
Pid: 45889 LoopCount: 7
Pid: 45890 LoopCount: 7
Pid: 45891 LoopCount: 7
Pid: 45892 LoopCount: 7
Pid: 45893 LoopCount: 7
Pid: 45889 LoopCount: 8
Pid: 45890 LoopCount: 8
Pid: 45891 LoopCount: 8
Pid: 45892 LoopCount: 8
Pid: 45893 LoopCount: 8
Pid: 45889 LoopCount: 9
Pid: 45890 LoopCount: 9
Pid: 45891 LoopCount: 9
```

```
Pid: 45892 LoopCount: 9  
Pid: 45893 LoopCount: 9  
Pid: 45890 LoopCount: 10  
Pid: 45891 LoopCount: 10  
Pid: 45892 LoopCount: 10  
Pid: 45893 LoopCount: 10  
Pid: 45891 LoopCount: 11  
Pid: 45892 LoopCount: 11  
Pid: 45893 LoopCount: 11  
Pid: 45893 LoopCount: 12  
Pid: 45892 LoopCount: 12  
Pid: 45893 LoopCount: 13  
`
```

嗯，一切都没问题了。

所以在访问临界资源时，使用**Lock**就可以避免进程同时占用资源而导致的一些问题。

Semaphore

信号量，是在进程同步过程中一个比较重要的角色。可以控制临界资源的数量，保证各个进程之间的互斥和同步。

如果你学过操作系统，那么一定对这方面非常了解，如果你还不了解信号量是什么，可以参考[信号量解析](#)

来了解一下它是做什么的。

那么接下来我们就用一个实例来演示一下进程之间利用**Semaphore**做到同步和互斥，以及控制临界资源数量。

```

from multiprocessing import Process, Semaphore, Lock, Queue
import time

buffer = Queue(10)
empty = Semaphore(2)
full = Semaphore(0)
lock = Lock()

class Consumer(Process):

    def run(self):
        global buffer, empty, full, lock
        while True:
            full.acquire()
            lock.acquire()
            buffer.get()
            print('Consumer pop an element')
            time.sleep(1)
            lock.release()
            empty.release()

class Producer(Process):

    def run(self):
        global buffer, empty, full, lock
        while True:
            empty.acquire()
            lock.acquire()
            buffer.put(1)
            print('Producer append an element')
            time.sleep(1)
            lock.release()
            full.release()

if __name__ == '__main__':
    p = Producer()
    c = Consumer()
    p.daemon = c.daemon = True
    p.start()
    c.start()
    p.join()
    c.join()
    print 'Ended!'

```

如上代码实现了注明的生产者和消费者问题，定义了两个进程类，一个是消费者，一个是生产者。

定义了一个共享队列，利用了**Queue**数据结构，然后定义了两个信号量，一个代表缓冲区空余数，一个表示缓冲区占用数。

生产者Producer使用empty.acquire()方法来占用一个缓冲区位置，然后缓冲区空闲区大小减小1，接下来进行加锁，对缓冲区进行操作。然后释放锁，然后让代表占用的缓冲区位置数量+1，消费者则相反。

运行结果如下：

```
Producer append an element
Producer append an element
Consumer pop an element
Consumer pop an element
Producer append an element
Producer append an element
Consumer pop an element
Consumer pop an element
Producer append an element
Producer append an element
Consumer pop an element
Consumer pop an element
Producer append an element
Producer append an element
```

可以发现两个进程在交替运行，生产者先放入缓冲区物品，然后消费者取出，不停地进行循环。

通过上面的例子来体会一下信号量的用法。

Queue

在上面的例子中我们使用了**Queue**，可以作为进程通信的共享队列使用。

在上面的程序中，如果你把**Queue**换成普通的list，是完全起不到效果的。即使在一个进程中改变了这个list，在另一个进程也不能获取到它的状态。

因此进程间的通信，队列需要用**Queue**。当然这里的队列指的是**multiprocessing.Queue**

依然是用上面那个例子，我们一个进程向队列中放入数据，然后另一个进程取出数据。

```
from multiprocessing import Process, Semaphore, Lock, Queue
import time
from random import random

buffer = Queue(10)
empty = Semaphore(2)
full = Semaphore(0)
lock = Lock()

class Consumer(Process):

    def run(self):
        global buffer, empty, full, lock
        while True:
            full.acquire()
            lock.acquire()
            print 'Consumer get', buffer.get()
            time.sleep(1)
            lock.release()
            empty.release()

class Producer(Process):

    def run(self):
        global buffer, empty, full, lock
        while True:
            empty.acquire()
            lock.acquire()
            num = random()
            print 'Producer put ', num
            buffer.put(num)
            time.sleep(1)
            lock.release()
            full.release()

if __name__ == '__main__':
    p = Producer()
    c = Consumer()
    p.daemon = c.daemon = True
    p.start()
    c.start()
    p.join()
    c.join()
    print 'Ended!'
```

运行结果：

```
Producer put 0.719213647437
Producer put 0.44287326683
Consumer get 0.719213647437
Consumer get 0.44287326683
Producer put 0.722859424381
Producer put 0.525321338921
Consumer get 0.722859424381
Consumer get 0.525321338921
```

可以看到生产者放入队列中数据，然后消费者将数据取出来。

`get`方法有两个参数，`blocked`和`timeout`，意思为阻塞和超时时间。默认`blocked`是`true`，即阻塞式。

当一个队列为空的时候如果再用`get`取则会阻塞，所以这时候就需要吧`blocked`设置为`false`，即非阻塞式，实际上它就会调用`get_nowait()`方法，此时还需要设置一个超时时间，在这么长的时间内还没有取到队列元素，那就抛出`Queue.Empty`异常。

当一个队列为满的时候如果再用`put`放则会阻塞，所以这时候就需要吧`blocked`设置为`false`，即非阻塞式，实际上它就会调用`put_nowait()`方法，此时还需要设置一个超时时间，在这么长的时间内还没有放进去元素，那就抛出`Queue.Full`异常。

另外队列中常用的方法

`Queue.qsize()` 返回队列的大小，不过在 Mac OS 上没法运行。

原因：

```
def qsize(self): # Raises NotImplementedError on Mac OSX
    because of broken sem_getvalue() return self._maxsize -
    self._sem._semlock._get_value()
```

`Queue.empty()` 如果队列为空，返回`True`，反之`False`

`Queue.full()` 如果队列满了，返回`True`，反之`False`

`Queue.get([block[, timeout]])` 获取队列，`timeout`等待时间

`Queue.get_nowait()` 相当`Queue.get(False)`

`Queue.put(item)` 阻塞式写入队列，`timeout`等待时间

`Queue.put_nowait(item)` 相当`Queue.put(item, False)`

Pipe

管道，顾名思义，一端发一端收。

Pipe可以是单向(half-duplex)，也可以是双向(duplex)。我们通过multiprocessing.Pipe(duplex=False)创建单向管道(默认为双向)。一个进程从PIPE一端输入对象，然后被PIPE另一端的进程接收，单向管道只允许管道一端的进程输入，而双向管道则允许从两端输入。

用一个实例来感受一下：

```
from multiprocessing import Process, Pipe

class Consumer(Process):
    def __init__(self, pipe):
        Process.__init__(self)
        self.pipe = pipe

    def run(self):
        self.pipe.send('Consumer Words')
        print 'Consumer Received:', self.pipe.recv()

class Producer(Process):
    def __init__(self, pipe):
        Process.__init__(self)
        self.pipe = pipe

    def run(self):
        print 'Producer Received:', self.pipe.recv()
        self.pipe.send('Producer Words')

if __name__ == '__main__':
    pipe = Pipe()
    p = Producer(pipe[0])
    c = Consumer(pipe[1])
    p.daemon = c.daemon = True
    p.start()
    c.start()
    p.join()
    c.join()
    print 'Ended!'
```

在这里声明了一个默认为双向的管道，然后将管道的两端分别传给两个进程。两个进程互相收发。观察一下结果：

```
Producer Received: Consumer Words
Consumer Received: Producer Words
Ended!
```

以上是对pipe的简单介绍。

Pool

在利用Python进行系统管理的时候，特别是同时操作多个文件目录，或者远程控制多台主机，并行操作可以节约大量的时间。当被操作对象数目不大时，可以直接利用multiprocessing中的Process动态生成多个进程，十几个还好，但如果是上百个，上千个目标，手动的去限制进程数量却又太过繁琐，此时可以发挥进程池的功效。Pool可以提供指定数量的进程，供用户调用，当有新的请求提交到pool中时，如果池还没有满，那么就会创建一个新的进程用来执行该请求；但如果池中的进程数已经达到规定最大值，那么该请求就会等待，直到池中有进程结束，才会创建新的进程来它。

在这里需要了解阻塞和非阻塞的概念。

阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态。

阻塞即要等到回调结果出来，在有结果之前，当前进程会被挂起。

Pool的用法有阻塞和非阻塞两种方式。非阻塞即为添加进程后，不一定非要等到改进程执行完就添加其他进程运行，阻塞则相反。

现用一个实例感受一下非阻塞的用法：

```
from multiprocessing import Lock, Pool
import time

def function(index):
    print 'Start process: ', index
    time.sleep(3)
    print 'End process', index

if __name__ == '__main__':
    pool = Pool(processes=3)
    for i in xrange(4):
        pool.apply_async(function, (i,))

    print "Started processes"
    pool.close()
    pool.join()
    print "Subprocess done."
```

在这里利用了apply_async方法，即非阻塞。

运行结果：

```
Started processes
Start process: Start process: 0
1
Start process: 2
End processEnd process 0
1
Start process: 3
End process 2
End process 3
Subprocess done.
```

可以发现在这里添加三个进程进去后，立马就开始执行，不用非要等到某个进程结束后再添加新的进程进去。

下面再看看阻塞的用法：

```
from multiprocessing import Lock, Pool
import time

def function(index):
    print 'Start process: ', index
    time.sleep(3)
    print 'End process', index

if __name__ == '__main__':
    pool = Pool(processes=3)
    for i in xrange(4):
        pool.apply(function, (i,))

    print "Started processes"
    pool.close()
    pool.join()
    print "Subprocess done."
```

在这里只需要把apply_async改成apply即可。

运行结果如下：

```
Start process: 0
End process 0
Start process: 1
End process 1
Start process: 2
End process 2
Start process: 3
End process 3
Started processes
Subprocess done.
```

这样以来就好理解了吧？

下面对函数进行解释：

`apply_async(func[, args[, kwds[, callback]]])` 它是非阻塞，`apply(func[, args[, kwds]])`是阻塞的。

`close()` 关闭pool，使其不在接受新的任务。

`terminate()` 结束工作进程，不在处理未完成的任务。

`join()` 主进程阻塞，等待子进程的退出，`join`方法要在`close`或`terminate`之后使用。

当然每个进程可以在各自的方法返回一个结果。`apply`或`apply_async`方法可以拿到这个结果并进一步进行处理。

```
from multiprocessing import Lock, Pool
import time

def function(index):
    print 'Start process: ', index
    time.sleep(3)
    print 'End process', index
    return index

if __name__ == '__main__':
    pool = Pool(processes=3)
    for i in xrange(4):
        result = pool.apply_async(function, (i,))
        print result.get()
    print "Started processes"
    pool.close()
    pool.join()
    print "Subprocess done."
```

运行结果：

```
Start process: 0
End process 0
0
Start process: 1
End process 1
1
Start process: 2
End process 2
2
Start process: 3
End process 3
3
Started processes
Subprocess done.
```

另外还有一个非常好用的**map**方法。

如果你现在有一堆数据要处理，每一项都需要经过一个方法来处理，那么**map**非常适合。

比如现在你有一个数组，包含了所有的URL，而现在已经有了一个方法用来抓取每个URL内容并解析，那么可以直接在**map**的第一个参数传入方法名，第二个参数传入URL数组。

现在我们用一个实例来感受一下：

```

from multiprocessing import Pool
import requests
from requests.exceptions import ConnectionError

def scrape(url):
    try:
        print requests.get(url)
    except ConnectionError:
        print 'Error Occured ', url
    finally:
        print 'URL ', url, ' Scraped'

if __name__ == '__main__':
    pool = Pool(processes=3)
    urls = [
        'https://www.baidu.com',
        'http://www.meituan.com/',
        'http://blog.csdn.net/',
        'http://xxxxxxx.net'
    ]
    pool.map(scrape, urls)

```

在这里初始化一个Pool，指定进程数为3，如果不指定，那么会自动根据CPU内核来分配进程数。

然后有一个链接列表，map函数可以遍历每个URL，然后对其分别执行scrape方法。

运行结果：

```

<Response [403]>
URL http://blog.csdn.net/ Scraped
<Response [200]>
URL https://www.baidu.com Scraped
Error Occured http://xxxxxxx.net
URL http://xxxxxxx.net Scraped
<Response [200]>
URL http://www.meituan.com/ Scraped

```

可以看到遍历就这么轻松地实现了。

结语

多进程multiprocessing相比多线程功能强大太多，而且使用范围更广，希望本文对大家有帮助！

本文参考

<https://docs.python.org/2/library/multiprocessing.html>

<http://www.cnblogs.com/vamei/archive/2012/10/12/2721484.html>

<http://www.cnblogs.com/kaituorensheng/p/4445418.html>

<https://my.oschina.net/yangyanxing/blog/296052>

Python爬虫进阶七之设置ADSL拨号服务器代理

那夜

那是一个寂静的深夜，科比还没起床练球，虽然他真的可能不练了。

我废了好大劲，爬虫终于写好了！BUG也全部调通了！心想，终于可以坐享其成了！

泡杯茶，安静地坐在椅子上看着屏幕上一行行文字在控制台跳出，一条条数据嗖嗖进入我的数据库，一张张图片悄悄存入我的硬盘。人生没有几个比这更惬意的事情了。

我端起茶杯，抿了一口，静静地回味着茶香。

这时，什么情况！屏幕爆红了！爆红了！一口茶的功夫啊喂！

怎么回事！咋爬不动了，不动了！我用浏览器点开那一个个报错的链接，浏览器显示

您的请求过于频繁，IP已经被暂时封禁，请稍后再试！

沃日，我IP被封了？此时此刻，空气凝固了，茶也不再香了，请给我一个爱的抱抱啊。

时候不早了，还是洗洗睡吧。

次日

那一晚，辗转反侧难以入睡。

怎么办？怎么办？如果是你你该怎么办？

手动换个IP？得了吧，一会又要封了，还能不能安心睡觉啊？

找免费代理？可行，不过我之前测过不少免费代理IP，一大半都不好用，而且慢。不过可以一直维护一个代理池，定时更新。

买代理？可以可以，不过优质的代理服务商价格可是不菲的，我买过一些廉价的，比如几块钱套餐一次提取几百IP的，算了还是不说了都是泪。

然而最行之有效的方法是什么？那当然是ADSL拨号！

这是个啥？且听我慢慢道来。

什么是ADSL

ADSL（Asymmetric Digital Subscriber Line，非对称数字用户环路）是一种新的数据传输方式。它因为上行和下行带宽不对称，因此称为非对称数字用户线环路。它采用频分复用技术把普通的电话线分成了电话、上行和下行三个相对独立的信道，从而避免了相互之间的干扰。

他有个独有的特点，每拨一次号，就获取一个新的IP。也就是它的IP是不固定的，不过既然是拨号上网嘛，速度也是有保障的，用它搭建一个代理，那既能保证可用，又能自由控制拨号切换。

如果你是用的ADSL上网方式，那就不用过多设置了，直接自己电脑调用一个拨号命令就好了，自动换IP，分分钟解决封IP的事。

然而，你可能说？我家宽带啊，我连得公司无线啊，我蹭的网上的啊！那咋办？

这时，你就需要一台VPS拨号主机。

购买服务器

某度广告做的那么好是吧？一搜一片，这点谷歌可是远远比不上啊。

于是乎，我就搜了搜，键入：拨号服务器，有什么骑士互联啊、无极网络啊、挂机宝啊等等的。我选了个价钱还凑合的，选了个无极网络（这里不是在打广告），80一个月的配置，一天两块钱多点。

2核、512M内存，10M带宽。

云立方

大家觉得有更便宜的更好用请告诉我呀！

接下来开始装操作系统，进入后台，有一个自助装系统的页面。



我装的CentOS的，在后面设置代理啊，定时任务啊，远程SSH管理啊之类的比较方便。如果你想用Windows，能配置好代理那也没问题。

有的小伙伴可能会问了，既然它的IP是拨号变化的，你咋用SSH连？其实服务商提供了一个域名，做了动态解析和端口映射，映射到这台主机的22端口就好了，所以不用担心IP变化导致SSH断开的问题。

好了装好了服务器之后，服务商提供了一个ADSL的拨号操作过程，用pppoe命令都可以完成，如果你的是Linux的主机一般都是用这个。然后服务商还会给你一个拨号账号和密码。

那么接下来就是试下拨号了。

服务商会提供详细的拨号流程说明。

比如无极的是这样的：[拨号流程](#)

设置好了之后，就有几个关键命令：

```
pppoe-start 拨号  
pppoe-stop 断开拨号  
pppoe-status 拨号连接状态
```

如果想重新拨号，那就执行stop、start就可以了。

反复执行，然后查看下ip地址，你会发现拨号一次换一个IP，是不是爽翻了！

好，那接下来就设置代理吧。

设置代理服务器

之前总是用别人的代理，没自己设置过吧？那么接下来我们就来亲自搭建HTTP代理。

Linux下搭建HTTP代理，推荐Squid和TinyProxy。都非常好配置，你想用哪个都行，且听我慢慢道来。

我的系统是CentOS，以它为例进行说明。

Squid

首先利用yum安装squid

```
yum -y install squid
```

设置开机启动

```
chkconfig --level 35 squid on
```

修改配置文件

```
vi /etc/squid/squid.conf
```

修改如下几个部分：

```
http_access allow !Safe_ports #deny改成allow  
http_access allow CONNECT !SSL_ports #deny改成allow  
http_access allow all #deny改成allow
```

其他的不需要过多配置。

启动squid

```
sudo service squid start
```

如此一来配置就完成了。

代理使用的端口是3128

TinyProxy

首先添加一下镜像源，然后安装

```
rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-releas  
yum update  
yum install tinyproxy
```

修改配置

```
vi /etc/tinyproxy/tinyos.conf
```

可以修改端口和允许的IP，如果想任意主机都连接那就把Allow这一行注释掉。

```
Port 8888 #预设是8888 Port,你可以更改  
Allow 127.0.0.1 #将127.0.0.1改成你自己的IP  
#例如你的IP 是1.2.3.4,你改成Allow 1.2.3.4,那只有你才可以连上这个Proxy  
#若你想任何IP都可以脸到Proxy在Allow前面打#注释
```

启动TinyProxy

```
service tinyproxy start
```

好了，两个代理都配置好了。

你想用那个都可以！

不过你以为这样就完了吗？太天真了，我被困扰了好几天，怎么都连不上，我还在怀疑是不是我哪里设置得不对？各种搜，一直以为是哪里配置有遗漏，后来发现是**iptables**的锅，万恶的防火墙。踩过的的坑，那就不要让大家踩了，用下面的命令设置下**iptables**，放行3128和8888端口就好了。

```
service iptables save
systemctl stop firewalld
systemctl disable firewalld
systemctl start iptables
systemctl status iptables
systemctl enable iptables
```

修改**iptables**配置

```
vi /etc/sysconfig/iptables
```

在

```
-A IN_public_allow -p tcp -m tcp --dport 22 -m conntrack --ctsta
```

的下面添加两条规则

```
-A IN_public_allow -p tcp -m tcp --dport 3128 -m conntrack --cts
-A IN_public_allow -p tcp -m tcp --dport 8888 -m conntrack --cts
```

如图所示

```

root@localhost:~ — ssh root@xiangyang.huanip.cn -p 2119 — 80x24
-A INPUT_ZONES -g IN_public
-A IN_public -j IN_public_log
-A IN_public -j IN_public_deny
-A IN_public -j IN_public_allow
-A IN_public_allow -p tcp -m tcp --dport 22 -m conntrack --ctstate NEW -j ACCEPT
-A IN_public_allow -p tcp -m tcp --dport 3128 -m conntrack --ctstate NEW -j ACCEPT
-A IN_public_allow -p tcp -m tcp --dport 8888 -m conntrack --ctstate NEW -j ACCEPT
COMMIT
# Completed on Sat Nov 19 00:01:32 2016
# Generated by iptables-save v1.4.21 on Sat Nov 19 00:01:32 2016
*raw
:PREROUTING ACCEPT [1302:185781]
:OUTPUT ACCEPT [21:2488]
:OUTPUT_direct - [0:0]
:PREROUTING_direct - [0:0]
-A PREROUTING -j PREROUTING_direct
-A OUTPUT -j OUTPUT_direct
COMMIT
# Completed on Sat Nov 19 00:01:32 2016
# Generated by iptables-save v1.4.21 on Sat Nov 19 00:01:32 2016
*security

```

保存，然后重启iptables

```
sudo service iptables restart
```

输入

ifconfig得到IP地址，在其他的主机上输入

```
curl -x IP:8888 www.baidu.com
```

测试一下，如果能出现结果，那就说明没问题。

```

CQC — CQC@CQC-MAC — ~ — zsh — 80x24
curl -x 116.208.102.30:8888 www.baidu.com
<!DOCTYPE html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html; charset=utf-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.min.css><title>百度一下，你就知道</title></head> <body link=#0000cc> <div id=wrapper> <div id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id=lg> <img hidefocus=true src=/www.baidu.com/img/bd_logo1.png width=270 height=129> </div> <form id=form name=f action=/www.baidu.com/s class=fm> <input type=hidden name=bdorz_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <input type=hidden name=tn value=baidu> <span class="bg_s_ipt_wr"><input id=kw name=wd class=_ipt value maxlength=255 autocomplete=off autofocus></span> <span class="bg_s_btn_wr"><input type=submit id=su value=百度一下 class="bg_s_btn"></span> </form> </div> </div> <div id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=http://www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=tj_trmap class=mnav>地图</a> <a href=http://v.baidu.com name=tj_trvideo class=mnav>视频</a> <a href=http://tieba.baidu.com name=tj_trtieba class=mnav>贴吧</a> <noscript> <a href=http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=http%3A%2F%2Fwww.baidu.com%2F%3fbdorz_come%3d1 name=tj_login class=lb>登录</a> </noscript> <script>document.write('<a href="http://www.baidu.com/bdorz/login.gif?login&tpl=mn&u=' + encodeURIComponent(window.location.href+ (window.location.search === "" ? "?" : "&")+ "bdorz_come=1")+'"' name=
```

如果怎么配都连不上，那干脆关了你的防火墙吧。虽然不推荐。

连接代理

接下来才是重头戏，你咋知道你的服务器IP现在到底是多少啊？拨一次号IP就换一次，那这还了得？

如果服务商提供了端口映射！那一切都解决了！直接用端口映射过去就好了。然而，我的并没有。

自力更生，艰苦创业！

首先我研究了一下DDNS服务，也就是动态域名解析。即使你的IP在变化，那也可以通过一个域名来映射过来。

原理简单而统一：当前拨号主机定时向一个固定的服务器发请求，服务器获取remote_addr就好了，可以做到定时更新和解析。

那么我找了一下，国内做的比较好的就是花生壳了，然后又找到了DNSPOD的接口解析。

下面简单说下我的折腾过程，大家可以先不用试，后面有更有效的方法。

花生壳

现在花生壳出到3.0版本了，有免费版和付费版之分，我就试用了一下免费版的。这里是花生壳的一些配置和下载：[花生壳配置](#)

下载花生壳客户端之后，会生成SN码，用这个在花生壳的官网登录后，会分配给你一个免费的域名。

接下来这个域名就能解析到你的主机了。

DNSPOD

DNSPOD原理也是一样，不过好处是你可以配置自己的域名。

在GitHub上有[脚本](#)可以使用。

具体的细节我就不说了，实际上就是定时请求，利用remote_addr更新DNSPOD记录，做到动态解析。[解析接口](#)

不过！这两个有个通病！慢！

什么慢？解析慢！但这不是他们的锅，因为DNS修改后完全生效就是需要一定的时间，这一秒你拨号了，然后更新了IP，但是域名可能还是解析着原来的IP，需要过几分钟才能变过来。这能忍吗？

我可是在跑爬虫啊，这还能忍？

自力更生

嗯，V2EX果然是个好地方，逛了一下，收获不小。[链接在此](#)

参考了 abelyao 的思路，自己写了脚本来获取IP，保证秒级更新！

此时，你还需要另一台固定IP的主机或者某个云服务器，只要是地址固定的就好。在这里我用了另一台有固定IP的阿里云主机，当然你如果有什么新浪云啊之类的也可以。

那么现在的思路就是，拨号VPS定时拨号换IP，然后请求阿里云主机，阿里云主机获取VPS的IP地址即可。

拨号VPS做的事情：

定时拨号，定时请求服务器。使用bash脚本，然后crontab定时执行。

远程服务器：

接收请求，获取remote_addr，保存起来。使用Flask搭建服务器，接收请求。

废话少说，上代码[AutoProxy](#)

功能

由于DDNS生效时间过长，对于爬虫等一些时间要求比较紧迫的项目就不太适用，为此本项目根据DDNS基本原理来实现实时获取ADSL拨号主机IP。

基本原理

client文件夹由ADSL拨号客户机运行。它会定时执行拨号操作，然后请求某个固定地址的服务器，以便让服务器获取ADSL拨号客户机的IP，主要是定时bash脚本运行。

server文件夹是服务器端运行，利用Python的Flask搭建服务器，然后接收ADSL拨号客户机的请求，得到remote_addr，获取客户机拨号后的IP。

项目结构

server

- config.py 配置文件。
- ip 客户端请求后获取的客户端IP，文本保存。
- main.py Flask主程序，提供两个接口，一个是接收客户端请求，然后将IP保存，另外一个是获取当前保存的IP。

client

- crontab 定时任务命令示例。

- `pppoe.sh` 拨号脚本，主要是实现重新拨号的几个命令。
- `request.sh` 请求服务器的脚本，主要是实现拨号后请求服务器的操作。
- `request.conf` 配置文件。

使用

服务器

服务器提供两个功能，`record`方法是客户机定时请求，然后获取客户机IP并保存。`proxy`方法是供我们自己用，返回保存的客户机IP，提取代理。

克隆项目

```
git clone https://github.com/Germey/AutoProxy.git
```

修改配置

修改`config.py`文件

- `KEY` 是客户端请求服务器时的凭证，在`client`的`request.conf`也有相同的配置，二者保持一致即可。
- `NEED_AUTH` 在获取当前保存的IP（即代理的IP）的时候，为防止自己的主机代理被滥用，在获取IP的时候，需要加权限验证。
- `AUTH_USER`和`AUTH_PASSWORD`分别是认证用户名密码。
- `PORT`默认端口，返回保存的结果中会自动添加这个端口，组成一个IP:PORT的代理形式。

运行

```
cd server  
nohup python main.py
```

ADSL客户机

克隆项目

```
git clone https://github.com/Germey/AutoProxy.git
```

修改配置

修改`request.conf`文件

- KEY 是客户端请求服务器时的凭证，在server的config.py也有相同的配置，二者保持一致即可。
- SERVER是服务器项目运行后的地址，一般为http://<服务器IP>:<服务器端口>/record。如<http://120.27.14.24:5000/record>。

修改pppoe.sh文件

这里面写上重新拨号的几条命令，记得在前两行配置一下环境变量，配置上拨号命令所在的目录，以防出现脚本无法运行的问题。

运行

设置定时任务

```
crontab -e
```

输入crontab的实例命令

```
*/5 * * * * /var/py/AutoProxy/client/request.sh /var/py/AutoProx
```

注意修改路径，你的项目在哪里，都统一修改成自己项目的路径。

最前面的*/5是5分钟执行一次。

好了，保存之后，定时任务就会开启。

验证结果

这样一来，访问服务器地址，就可以得到ADSL拨号客户机的IP了。

```
import requests

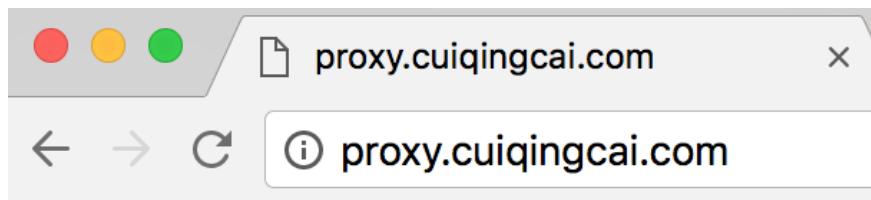
url = 'http://120.27.14.24:5000'
proxy = requests.get(url, auth=('admin', '123')).text
print(proxy)
```

实例结果：

```
116.208.97.22:8888
```

扩展

如果你有域名，可以自己解析一个域名，这样就可以直接请求自己的域名，拿到实时好用的代理了，而且定时更新。



116.208.97.2:8888

代理设置

urllib2

```
import urllib2
proxy_handler = urllib2.ProxyHandler({"http": 'http://'+ proxy})
opener = urllib2.build_opener(proxy_handler)
urllib2.install_opener(opener)
response = urllib2.urlopen('http://httpbin.org/get')
print response.read()
```

requests

```
import requests
proxies = {
    'http': 'http://'+ proxy,
}
r = requests.get('http://httpbin.org/get', proxies=proxies)
print(r.text)
```

以上便秒级解决了动态IP解析，自己实现了一遍DDNS，爽！

那这样以来，以后就可以直接请求你的主机获取一个最新可用的代理IP了，稳定可用，定时变化！

以上便是ADSL拨号服务器配置的全过程，希望对大家有帮助！