Python爬虫：一些常用的爬虫技巧总结

2016-04-29 j\_hao104 [Python编程](javascript:void(0);)

**来源：开源中国**

**http://my.oschina.net/jhao104/blog/647308 （点击尾部阅读原文前往）**

**作者： j\_hao104**

用python也差不多一年多了，python应用最多的场景还是web快速开发、爬虫、自动化运维：写过简单网站、写过自动发帖脚本、写过收发邮件脚本、写过简单验证码识别脚本。

爬虫在开发过程中也有很多复用的过程，这里总结一下，以后也能省些事情。

**1、基本抓取网页**

**get方法**

import urllib2

url  "http://www.baidu.com"

respons = urllib2.urlopen(url)

print response.read()

**post方法**

import urllib

import urllib2

url = "http://abcde.com"

form = {'name':'abc','password':'1234'}

form\_data = urllib.urlencode(form)

request = urllib2.Request(url,form\_data)

response = urllib2.urlopen(request)

print response.read()

**2、使用代理IP**

在开发爬虫过程中经常会遇到IP被封掉的情况，这时就需要用到代理IP；

在urllib2包中有ProxyHandler类，通过此类可以设置代理访问网页，如下代码片段：

import urllib2

proxy = urllib2.ProxyHandler({'http': '127.0.0.1:8087'})

opener = urllib2.build\_opener(proxy)

urllib2.install\_opener(opener)

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

print response.read()

**3、Cookies处理**

cookies是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)，python提供了cookielib模块用于处理cookies，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.

代码片段：

import urllib2, cookielib

cookie\_support= urllib2.HTTPCookieProcessor(cookielib.CookieJar())

opener = urllib2.build\_opener(cookie\_support)

urllib2.install\_opener(opener)

content = urllib2.urlopen('http://XXXX').read()

关键在于CookieJar()，它用于管理HTTP cookie值、存储HTTP请求生成的cookie、向传出的HTTP请求添加cookie的对象。整个cookie都存储在内存中，对CookieJar实例进行垃圾回收后cookie也将丢失，所有过程都不需要单独去操作。

手动添加cookie

cookie = "PHPSESSID=91rurfqm2329bopnosfu4fvmu7; kmsign=55d2c12c9b1e3; KMUID=b6Ejc1XSwPq9o756AxnBAg="

request.add\_header("Cookie", cookie)

**4、伪装成浏览器**

某些网站反感爬虫的到访，于是对爬虫一律拒绝请求。所以用urllib2直接访问网站经常会出现HTTP Error 403: Forbidden的情况

对有些 header 要特别留意，Server 端会针对这些 header 做检查

1. User-Agent 有些 Server 或 Proxy 会检查该值，用来判断是否是浏览器发起的 Request
2. Content-Type 在使用 REST 接口时，Server 会检查该值，用来确定 HTTP Body 中的内容该怎样解析。

这时可以通过修改http包中的header来实现，代码片段如下：

import urllib2

headers = {

    'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6'

}

request = urllib2.Request(

    url = 'http://my.oschina.net/jhao104/blog?catalog=3463517',

    headers = headers

)

print urllib2.urlopen(request).read()

**5、页面解析**

对于页面解析最强大的当然是正则表达式，这个对于不同网站不同的使用者都不一样，就不用过多的说明，附两个比较好的网址：

正则表达式入门：http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

正则表达式在线测试：http://tool.oschina.net/regex/

其次就是解析库了，常用的有两个lxml和BeautifulSoup，对于这两个的使用介绍两个比较好的网站：

lxml：http://my.oschina.net/jhao104/blog/639448

BeautifulSoup：http://cuiqingcai.com/1319.html

对于这两个库，我的评价是，都是HTML/XML的处理库，Beautifulsoup纯python实现，效率低，但是功能实用，比如能用通过结果搜索获得某个HTML节点的源码；lxmlC语言编码，高效，支持Xpath

**6、验证码的处理**

对于一些简单的验证码，可以进行简单的识别。本人也只进行过一些简单的验证码识别。但是有些反人类的验证码，比如12306，可以通过打码平台进行人工打码，当然这是要付费的。

**7、gzip压缩**

有没有遇到过某些网页，不论怎么转码都是一团乱码。哈哈，那说明你还不知道许多web服务具有发送压缩数据的能力，这可以将网络线路上传输的大量数据消减 60% 以上。这尤其适用于 XML web 服务，因为 XML 数据 的压缩率可以很高。

但是一般服务器不会为你发送压缩数据，除非你告诉服务器你可以处理压缩数据。

于是需要这样修改代码：

import urllib2, httplib

request = urllib2.Request('http://xxxx.com')

request.add\_header('Accept-encoding', 'gzip')        1

opener = urllib2.build\_opener()

f = opener.open(request)

这是关键:创建Request对象，添加一个 Accept-encoding 头信息告诉服务器你能接受 gzip 压缩数据

然后就是解压缩数据：

import StringIO

import gzip

compresseddata = f.read()

compressedstream = StringIO.StringIO(compresseddata)

gzipper = gzip.GzipFile(fileobj=compressedstream)

print gzipper.read()

**8、多线程并发抓取**

单线程太慢的话，就需要多线程了，这里给个简单的线程池模板 这个程序只是简单地打印了1-10，但是可以看出是并发的。

虽然说python的多线程很鸡肋，但是对于爬虫这种网络频繁型，还是能一定程度提高效率的。

from threading import Thread

from Queue import Queue

from time import sleep

# q是任务队列

#NUM是并发线程总数

#JOBS是有多少任务

q = Queue()

NUM = 2

JOBS = 10

#具体的处理函数，负责处理单个任务

def do\_somthing\_using(arguments):

    print arguments

#这个是工作进程，负责不断从队列取数据并处理

def working():

    while True:

        arguments = q.get()

        do\_somthing\_using(arguments)

        sleep(1)

        q.task\_done()

#fork NUM个线程等待队列

for i in range(NUM):

    t = Thread(target=working)

    t.setDaemon(True)

    t.start()

#把JOBS排入队列

for i in range(JOBS):

    q.put(i)

#等待所有JOBS完成

q.join()