Python 入门网络爬虫之精华版

Python 学习网络爬虫主要分 3 个大的版块：**抓取**，**分析**，**存储**
另外，比较常用的爬虫框架 Scrapy，这里最后也介绍一下。
先列举一下相关参考：宁哥的小站-网络爬虫

抓取

这一步，你要明确要得到的内容是是什么？是 HTML 源码，还是 Json 格式的字符串等等。

1. 最基本的抓取

一般属于 get 请求情况，直接从服务器上获取数据。
首先，Python 中自带 urllib 及 urllib2 这两个模块，基本上能满足一般的页面抓取。另外，requests 也是非常有用的包，与此类似的，还有 httplib2 等等。

```
Requests:
    import requests
    response = requests.get(url)
    content = requests.get(url).content # string
    print "response headers:", response.headers # dict
    print "content:", content
Urllib2:
    import urllib2
    response = urllib2.urlopen(url)
    content = urllib2.urlopen(url).read() # string
    print "response headers:", response.headers # not dict
    print "content:", content
Httplib2:
    import httplib2
    http = httplib2.Http()
    response_headers, content = http.request(url, 'GET')
    print "response headers:", response_headers # dict
    print "content:", content
```

此外，对于带有查询字段的 url，get 请求一般会将来请求的数据附在 url 之后，以?分割 url 和传输数据，多个参数用&连接。

```
data = {'data1':'XXXXX', 'data2':'XXXXX'} # dict 类型
Requests：data 为 dict，json
    import requests
```

```
    response = requests.get(url=url, params=data) # GET 请求发送
Urllib2：data 为 string
    import urllib, urllib2
    data = urllib.urlencode(data) # 编码工作，由 dict 转为 string
    full_url = url+'?'+data # GET 请求发送
    response = urllib2.urlopen(full_url)
```

2. 对于反爬虫机制的处理

2.1 模拟登陆情况

这种属于 post 请求情况，先向服务器发送表单数据，服务器再将返回的 cookie 存入本地。

```
data = {'data1':'XXXXX', 'data2':'XXXXX'} # dict 类型
Requests：data 为 dict，json
    import requests
    response = requests.post(url=url, data=data) # POST 请求发送，可用于用户名密码登陆情况
Urllib2：data 为 string
    import urllib, urllib2
    data = urllib.urlencode(data) # 编码工作，由 dict 转为 string
    req = urllib2.Request(url=url, data=data) # POST 请求发送，可用于用户名密码登陆情况
    response = urllib2.urlopen(req)
```

2.2 使用 cookie 登陆情况

使用 cookie 登陆，服务器会认为你是一个已登陆的用户，所以就会返回给你一个已登陆的内容。因此，需要验证码的情况可以使用带验证码登陆的 cookie 解决。

```
import requests
requests_session = requests.session() # 创建会话对象 requests.session()，可以记录 cookie
response = requests_session.post(url=url_login, data=data) # requests_session 的 POST 请求发送，可用于用户名密码登陆情况。必须要有这一步！拿到 Response Cookie!
```

若存在验证码，此时采用 response = requests_session.post(url=url_login, data=data)是不行的，做法应该如下：

```
response_captcha = requests_session.get(url=url_login, cookies=cookies)
response1 = requests.get(url_login) # 未登陆
response2 = requests_session.get(url_login) # 已登陆，因为之前拿到了 Response Cookie!
response3 = requests_session.get(url_results) # 已登陆，因为之前拿到了 Response Cookie!
```

相关参考：网络爬虫-验证码登陆
参考项目：爬取知乎网站

2.3 伪装成浏览器，或者反“反盗链”

```
headers = {'User-Agent':'XXXXX'} # 伪装成浏览器访问，适用于拒绝爬虫的网站
headers = {'Referer':'XXXXX'} # 反“反盗链”，适用于有“反盗链”的网站
headers = {'User-Agent':'XXXXX', 'Referer':'XXXXX'}
Requests：
    response = requests.get(url=url, headers=headers)
Urllib2：
    import urllib, urllib2
    req = urllib2.Request(url=url, headers=headers)
    response = urllib2.urlopen(req)
```

3. 使用代理

适用情况：限制 IP 地址情况，也可解决由于“频繁点击”而需要输入验证码登陆的情况。

```
proxies = {'http':'http://XX.XX.XX.XX:XXXX'}
Requests：
    import requests
    response = requests.get(url=url, proxies=proxies)
Urllib2：
    import urllib2
    proxy_support = urllib2.ProxyHandler(proxies)
    opener = urllib2.build_opener(proxy_support, urllib2.HTTPHandler)
    urllib2.install_opener(opener) # 安装 opener，此后调用 urlopen()时都会使用安装过的
opener 对象
    response = urllib2.urlopen(url)
```

4. 对于断线重连

```
def multi_session(session, *arg):
    while True:
        retryTimes = 20
    while retryTimes>0:
        try:
            return session.post(*arg)
        except:
            print '.',
            retryTimes -= 1
```

或者

```
def multi_open(opener, *arg):
```

```
    while True:
        retryTimes = 20
    while retryTimes>0:
        try:
            return opener.open(*arg)
        except:
            print '.',
            retryTimes -= 1
```

这样我们就可以使用 multi_session 或 multi_open 对爬虫抓取的 session 或 opener 进行保持。

5. 多进程抓取

这里针对华尔街见闻进行多进程的实验对比：Python 多进程抓取 与 Java 多进程抓取
相关参考：关于 Python 和 Java 的多进程多线程计算方法对比

6. 对于 Ajax 请求的处理

对于“加载更多”情况，使用 Ajax 来传输很多数据。它的工作原理是：从网页的 url 加载网页的源代码之后，会在浏览器里执行 JavaScript 程序。这些程序会加载更多的内容，“填充”到网页里。这就是为什么如果你直接去爬网页本身的 url，你会找不到页面的实际内容。
这里，若使用 Google Chrome 分析”请求“对应的链接(方法：右键→审查元素→Network→清空，点击”加载更多“，出现对应的 GET 链接寻找 Type 为 text/html 的，点击，查看 get 参数或者复制 Request URL)，循环过程。

- 如果“请求”之前有页面，依据上一步的网址进行分析推导第 1 页。以此类推，抓取抓 Ajax 地址的数据。
- 对返回的 json 格式数据(str)进行正则匹配。json 格式数据中，需从'\uxxxx'形式的 unicode_escape 编码转换成 u'\uxxxx'的 unicode 编码。
  参考项目：Python 多进程抓取

7. 验证码识别

对于网站有验证码的情况，我们有三种办法：

1. 使用代理，更新 IP。
2. 使用 cookie 登陆。
3. 验证码识别。
   使用代理和使用 cookie 登陆之前已经讲过，下面讲一下验证码识别。
   可以利用开源的 Tesseract-OCR 系统进行验证码图片的下载及识别，将识别的字符传到爬

虫系统进行模拟登陆。如果不成功，可以再次更新验证码识别，直到成功为止。
参考项目：Captcha1

分析

抓取之后就是对抓取的内容进行分析，你需要什么内容，就从中提炼出相关的内容来。
常见的分析工具有正则表达式，BeautifulSoup，lxml 等等。

存储

分析出我们需要的内容之后，接下来就是存储了。
我们可以选择存入文本文件，也可以选择存入 MySQL 或 MongoDB 数据库等。

Scrapy

Scrapy 是一个基于 Twisted 的开源的 Python 爬虫框架，在工业中应用非常广泛。
相关内容可以参考基于 Scrapy 网络爬虫的搭建