普通反爬虫机制的应对策略

2017/04/26 · 基础知识 · 反反爬虫, 网络爬虫

原文出处： Melwood

爬虫与反爬虫，这相爱相杀的一对，简直可以写出一部壮观的斗争史。而在大数据时代，数据就是金钱，很多企业都为自己的网站运用了反爬虫机制，防止网页上的数据被爬虫爬走。然而，如果反爬机制过于严格，可能会误伤到真正的用户请求；如果既要和爬虫死磕，又要保证很低的误伤率，那么又会加大研发的成本。
简单低级的爬虫速度快，伪装度低，如果没有反爬机制，它们可以很快的抓取大量数据，甚至因为请求过多，造成服务器不能正常工作。而伪装度高的爬虫爬取速度慢，对服务器造成的负担也相对较小。所以，网站反爬的重点也是那种简单粗暴的爬虫，反爬机制也会允许伪装度高的爬虫，获得数据。毕竟伪装度很高的爬虫与真实用户也就没有太大差别了。
这篇文章主要讨论使用Scrapy框架时，如何应对普通的反爬机制。

header检验

最简单的反爬机制，就是检查HTTP请求的Headers信息，包括User-Agent, Referer、Cookies等。

User-Agent

User-Agent是检查用户所用客户端的种类和版本，在Scrapy中，通常是在下载器中间件中进行处理。比如在setting.py中建立一个包含很多浏览器User-Agent的列表，然后新建一个random_user_agent文件：

```
class RandomUserAgentMiddleware(object):
    @classmethod
    def process_request(cls, request, spider):
        ua = random.choice(spider.settings['USER_AGENT_LIST'])
        if ua:
            request.headers.setdefault('User-Agent', ua)
```

这样就可以在每次请求中，随机选取一个真实浏览器的User-Agent。

Referer

Referer是检查此请求由哪里来，通常可以做图片的盗链判断。在Scrapy中，如果某个页面url是通过之前爬取的页面提取到，Scrapy会自动把之前爬取的页面url作为Referfer。也可以通过上面的方式自己定义Referfer字段。

Cookies

网站可能会检测Cookie中session_id的使用次数，如果超过限制，就触发反爬策略。所以可以在Scrapy中设置`COOKIES_ENABLED = False`让请求不带Cookies。
也有网站强制开启Cookis，这时就要麻烦一点了。可以另写一个简单的爬虫，定时向目标网站发送不带Cookies的请求，提取响应中Set-cookie字段信息并保存。爬取网页时，把存储起来的Cookies带入Headers中。

X-Forwarded-For

在请求头中添加X-Forwarded-For字段，将自己申明为一个透明的代理服务器，一些网站对代理服务器会手软一些。
X-Forwarded-For头一般格式如下

```
X-Forwarded-For: client1, proxy1, proxy2
```

这里将client1，proxy1设置为随机IP地址，把自己的请求伪装成代理的随机IP产生的请求。然而由于X-Forwarded-For可以随意篡改，很多网站并不会信任这个值。

限制IP的请求数量

如果某一IP的请求速度过快，就触发反爬机制。当然可以通过放慢爬取速度绕过，这要以爬取时间大大增长为代价。另一种方法就是添加代理。
很简单，在下载器中间件中添加:

```
request.meta['proxy'] = 'http://' + 'proxy_host' +  ':' + proxy_port
```

然后再每次请求时使用不同的代理IP。然而问题是如何获取大量的代理IP?
可以自己写一个IP代理获取和维护系统，定时从各种披露免费代理IP的网站爬取免费IP代理，然后定时扫描这些IP和端口是否可用，将不可用的代理IP及时清理。这样就有一个动态的代理库，每次请求再从库中随机选择一个代理。然而这个方案的缺点也很明显，开发代理获取和维护系统本身就很费时费力，并且这种免费代理的数量并不多，而且稳定性都比较差。如果必须要用到代理，也可以去买一些稳定的

在requests库中添加带认证的代理很简单，

```
proxies = {
    "http": "http://user:pass@10.10.1.10:3128/",
}
```

然而Scrapy不支持这种认证方式，需要将认证信息base64编码后，加入Headers的Proxy-Authorization字段：

```
import base64

# Set the location of the proxy
proxy_string = choice(self._get_proxies_from_file('proxies.txt')) # user:pass@ip:port
proxy_items = proxy_string.split('@')
request.meta['proxy'] = "http://%s" % proxy_items[1]

# setup basic authentication for the proxy
user_pass=base64.encodestring(proxy_items[0])
request.headers['Proxy-Authorization'] = 'Basic ' + user_pass
```

动态加载

现在越来越多的网站使用ajax动态加载内容，这时候可以先截取ajax请求分析一下，有可能根据ajax请求构造出相应的API请求的URL就可以直接获取想要的内容，通常是json格式，反而还不用去解析HTML。
然而，很多时候ajax请求都会经过后端鉴权，不能直接构造URL获取。这时就可以通过PhantomJS+Selenium模拟浏览器行为，抓取经过js渲染后的页面。具体可以参考：
Scrapy+PhantomJS+Selenium动态爬虫
需要注意的是，使用Selenium后，请求不再由Scrapy的Downloader执行，所以之前添加的请求头等信息都会失效，需要在Selenium中重新添加

```
headers = {...}
for key, value in headers.iteritems():
    webdriver.DesiredCapabilities.PHANTOMJS['phantomjs.page.customHeaders.{}'.format(key
```

另外，调用PhantomJs需要指定PhantomJs的可执行文件路径，通常是将该路径添加到系统的path路径，让程序执行时自动去path中寻找。我们的爬虫经常会放到crontab中定时执行，而crontab中的环境变量和系统的环境变量不同，所以就加载不到PhamtonJs需要的路径，所以最好是在申明时指定路径：

```
driver = webdriver.PhantomJS(executable_path='/usr/local/bin/phantomjs')
```