21 反爬之客户端仿真

更新时间：2019-06-17 10:56:38

> 我要扼住命运的咽喉，它妄想使我屈服，这绝对办不到。生活是这样美好，活他一千辈子吧！
>
> ——贝多芬

豆瓣读书是带有反爬机制的，如果网络爬虫触发了它的反爬机制IP将会被封导致无法工作。可见一但网络爬虫触发发了网站的反爬机制那将是毁灭性的，归根结底引发反爬机制运作的很多原因，可能是网络爬虫不够“礼貌”，没有尊重对方的规则来取数据。本节将从多个维度解释常见的反爬的方式有哪些，应该如何让我们的爬虫更“礼貌”地爬网。

反爬分析 - 查看豆瓣读书是怎么封IP的

有一些防护措施完备的网站可能会阻止你快速地提交表单，或者快速地与网站进行交互。即使没有这些安全措施，用一个比普通人快很多的速度从一个网站下载大量信息也可能让自己被网站封杀。

因此，虽然多线程程序可能是一个快速加载页面的好办法——在一个线程中处理数据，另一个线程中加载页面——但是这对编写好的爬虫来说是“恐怖”的策略。还是应该尽量保证一次加载页面加载且数据请求最小化。

合理控制速度是不应该被破坏的规则。过度消耗别人的服务器资源会让你置身于非法境地，更严重的是这么做可能会把一个小型网站拖垮甚至下线。拖垮网站是不道德的，是彻头彻尾的错误。所以请控制采集速度！

客户端仿真术

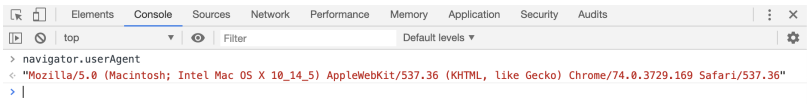
一般来说网站要得知其访客的采用的客户端浏览器是什么最直接的手段就是从请求头中的User-Agent中读取，User-Agent是指包含浏览器信息、操作系统信息等的一个字符串，也称之为一种特殊的网络协议。服务器通过它判断当前访问对象是浏览器、邮件客户端还是网络爬虫。在 `request.headers` 里可以查看User-Agent。

不要轻易地忽略这个不起眼的User-Agent,因为一但你忽视它的存在任意地交由你所使用的工具包或者编程框架来生成的话就会轻易地将你的真实身份暴露出来，如果你使用的是 urllib 那么默认的UA就会被设置为 Python-urllib/2.7 ，如果采用Scrapy的话默认的UA就将是 Scrapy/1.1 (+http://scrapy.org) ，还有些工具包更加老实直接在UA中带有 XX X Cralwer 的字样，这不是明摆着告诉对方：“我是蜘蛛我怕谁？！”

浏览器UA背景知识

一种最有效的方法就是把User-Agent的值设置为标准浏览器，我们可以设置一个User-Agent池（list，数组，字典都可以），存放不同的标准浏览器的UA设置，每次爬取的时候随机取一个来设置 request 的User-Agent，这样User-Agent就会一直在变化，可以提高一定的隐蔽性。

我们可以直接在浏览器中查看浏览器自身的UA是什么，首先打开浏览器，按 F12 进入控制台（Console），然后输入： navigator.userAgent ，即可看到 UA。如下图所示：



上图中就显示了我当前的Chrome 的UA：

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36
```

UA通常格式如下：


```
Mozilla/5.0 (平台) 引擎版本 浏览器版本号
```

由于历史上的浏览器大战，当时想获得图文并茂的网页，就必须宣称自己是 Mozilla 浏览器。此事导致如今 UA 里通常都带有 Mozilla 字样，最早包含该字样的是 Mozilla/1.0 (Win3.1)。斯人已逝，而且现代服务器也不强烈依赖该字符串响应，换言之，现在已经可以不必带上该字样，但几乎每个浏览器依然带有该字样，算是尊重历史吧。

然后是平台部分，这部分可由多个字符串组成，用英文半角分号分开。这部分通常包含操作系统，如果是 Windows 系统，可以参考百度百科 Windows NT 词条。太长不看的版本：

```
Windows NT 5.0 // 如 Windows 2000
Windows NT 5.1 // 如 Windows XP
Windows NT 6.0 // 如 Windows Vista
Windows NT 6.1 // 如 Windows 7
Windows NT 6.2 // 如 Windows 8
Windows NT 6.3 // 如 Windows 8.1
Windows NT 10.0 // 如 Windows 10
Win64; x64 // Win64 on x64
WOW64 // Win32 on x64
```

其中这个 WOW64 (Windows-on-Windows 64-bit)。它是 Windows 的子系统，让大多数 32 位的程序不用修改也能运行在 64 位系统上。

如果是 Linux 系统。

```
X11; Linux i686; // Linux 桌面，i686 版本
X11; Linux x86_64; // Linux 桌面，x86_64 版本
X11; Linux i686 on x86_64 // Linux 桌面，运行在 x86_64 的 i686 版本
```

此外还可以加发行版名：X11; Ubuntu; Linux x86_64;

如果是 macOS（OS X、Mac OS X），形如：

```
Macintosh; Intel Mac OS X 10_9_0 // Intel x86 或者 x86_64
Macintosh; PPC Mac OS X 10_9_0 // PowerPC
Macintosh; Intel Mac OS X 10.12; // 不用下划线，用点
```

最后面的部分就是系统版本。由于 Mac 的系统多次易名，这里只写出 OS X 和 mac OS 的版本号（10.8 之后系统名称均为加州景点），分别是 ：

```
Mountain Lion 10.8.0~10.8.3
Mavericks 10.9.0~10.9.4
Yosemite 10.10.0~10.10.5
El Capitan 10.11.0~10.11.6
Sierra 10.12.0~10.12.4（至今2017.02，更多的内容参考维基百科）
```

你可指明你是 Android、iPod、iPhone、iPad 等：

```
Android; Mobile // Firefox40 及以下
Android; Tablet // Firefox40 及以下
Android 4.4; Mobile // Firefox41 及以上
Android 4.4; Tablet // Firefox41 及以上
iPod touch; CPU iPhone OS 8_3 like Mac OS X
iPhone; CPU iPhone OS 8_3 like Mac OS X
iPad; CPU iPhone OS 8_3 like Mac OS X
```

有的时候还可能看见加密等级的字符：

```
N; 表示无安全
I; 表示弱安全
U; 表示强安全
```


引擎版本和浏览器版本号接下来细说。

据 StatCounter 统计，2017 年 1 月，各桌面浏览器的的使用分布为情况大致如下：

Chrome 的 UA

首先是 Google Chrome。以我的浏览器为例：

```
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.76 Safari/537.36
```

Mozilla/5.0 (Windows NT 10.0; WOW64)，这部分不赘述了。

AppleWebKit/537.36 (KHTML, like Gecko)…Safari/537.36，历史上，苹果依靠了 WebKit 内核开发出 Safari 浏览器，WebKit 包含了 WebCore 引擎，而 WebCore 又从 KHTML 衍生而来。由于历史原因，KHTML 引擎需要声明自己是“类似 Gecko”的，因此引擎部分这么写。再后来，Google 开发 Chrome 也是用了 WebKit 内核，于是也跟着这么写。借用 Littern 的一句话：“Chrome 希望能得到为 Safari 编写的网页，于是决定装成 Safari，Safari 使用了 WebKit 渲染引擎，而 WebKit 呢又伪装自己是KHTML，KHTML 呢又是伪装成 Gecko 的。同时所有的浏览器又都宣称自己是 Mozilla。”。不过，后来 Chrome 28 某个版本改用了 blink 内核，但还是保留了这些字符串。而且，最近的几十个版本中，这部分已经固定，没再变过。

Chrome/56.0.2924.76 ，这部分才是 Chrome 的版本。56.0 是大版本，2924 是持续增大的一个数字，而 76则是修补漏洞的小版本。由于没找到版本号的规律，只能寄希望于别人记录了，查找得如下网站：

1. 谷歌Chrome 旧版本 （3~目前最新）
2. Google Chrome （比较新的五六个版本）

3. 下载旧版本 Google Chrome （0.x ~46）

根据上述网站筛选出的数十个版本号，把版本号看成 xx.0.yyyy.zz ，通常一个 xx 只对应一两个 yyyy，但可能有多个 zz。不强求正确的情况下，可以随意指定 zz（zz通常0~200之间）或者都指定为 0，下列为约近 20 个大版本。

```
58.0.2995.zz
57.0.2986.zz
56.0.2924.zz
55.0.2883.zz
54.0.2840.zz
53.0.2785.zz
52.0.2743.zz
51.0.2704.zz
50.0.2661.zz
49.0.2623.zz
48.0.2564.zz
47.0.2526.zz
46.0.2490.zz
45.0.2454.zz
44.0.2403.zz
43.0.2357.zz
42.0.2311.zz
41.0.2272.zz
40.0.2214.zz
39.0.2171.zz
38.0.2125.zz
37.0.2062.zz
```

Firefox 的 UA

第二部分便是 Firefox。说起来，Firefox 的 UA 相当容易伪造，根据 MDN 一篇文章内容指出格式如下：


```
Mozilla/5.0 (platform; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
```

- rv: GeckoVersion 为 Gecko 内核版本号，rv 是 release version 的缩写。最近的几十个版本中，GeckoVersion 和 FirefoxVersion 一致。
- Gecko/GeckoTrail ，桌面端固定不变为“Gecko/20100101”
- Firefox/firefoxversion，Firefox 的版本，形如xx.0。

不过，随着 Firefox 换 Servo 内核的步伐推进，上述内容可能很快就要发生改变。

IE / Edge 的 UA

第三部分是 IE

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
```

以上三个没啥好说的，都含有 MSIE（Microsoft Internet Explorer），其中 IE 8 开始加入 Trident 字符串。当使用兼容模式时，UA 如下，细看可知仅仅只是 MSIE 部分变了：

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0)
```

然后从 IE9 开始，终于也改为了 “Mozilla/5.0”，前面这部分没变，后面越来越乱。可能包含 NET CLR 等内容。

```
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0; WOW64; Trident/5.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.0.30729)
```

IE10 和 IE9 差不多，可能包含 NET CLR 等内容：

```
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
```

最混乱的是 IE11，看着就像是 Gecko 内核（rv: 11.0），但是显然又不是（like Gecko）同时声明自己是 Trident/7.0 内核。移除了之前版本的“compatible”（兼容）和“MSIE”

Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko

然后是 IE 继任者 Microsoft Edge ， UA 格式：

```
Mozilla/5.0 (Windows NT 10.0; &lt;64-bit tags&gt;) AppleWebKit/&lt;WebKit Rev&gt; (KHTML, like Gecko) Chrome/&lt;Chrome Rev&gt; Safari/&lt;WebKit Rev&gt; Edge/&lt;EdgeHTML Rev&gt;.&lt;Windows Build&gt;
```

Edge 移除了以下内容

```
.NET CLR &lt;version&gt;
.NET &lt;version&gt;
TabletPC &lt;version&gt;
Touch
Infopath &lt;version&gt;
Trident &lt;version&gt;
```

三大家说完了，其余的 Safari 浏览器和 Opera 浏览器也有一定的市场，但是大家应该也知道该怎么分析它们的 UA 了。另外国产的套壳浏览器可能会在 Chrome UA 的基础上再添加几个字符串。例如“QQBrowser”（QQ）、“BIDUBrowser”（百度）、“UBrowser”（UC）、“LBBROWSER”（猎豹）。当然也有某些浏览器 UA 完全等同于 Chrome UA，比如 3Q 大战后 360 浏览器的做法就是完全伪装成 Chrome，丧失了自己的名字。

搜索引擎UA背景知识

告诉你一个非常好的信息——所有的搜索引擎也是通过User-Agent来告诉网站“我是谁”。也就是说我们可以通过设置User-Agent来伪装成为网站欢迎的搜索引擎的蜘蛛。

Google 的爬虫

爬虫名	User-agent
Googlebot News	Googlebot-News
Googlebot Images	Googlebot-Image/1.0
Googlebot Video	Googlebot-Video/1.0
Google Mobile (featured phone)	SAMSUNG-SGH-E250/1.0 Profile/MIDP-2.0 Configuration/CLDC-1.1
UP.Browser/6.2.3.3.c.1.101 (GUI)	MMP/2.0 (compatible; Googlebot-Mobile/2.1; +http://www.google.com/bot.html)
Google Smartphone	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2272.96	Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Google Mobile Adsense	(compatible; Mediapartners-Google/2.1; +http://www.google.com/bot.html)
Google Adsense	Mediapartners-Google
Google AdsBot (PPC landing page quality)	AdsBot-Google (+http://www.google.com/adsbot.html)
Google app crawler (fetch resources for mobile)	AdsBot-Google-Mobile-Apps

搜狗UA

- Sogou Pic Spider/3.0(http://www.sogou.com/docs/help/webmasters.htm#07)
- Sogou head spider/3.0(http://www.sogou.com/docs/help/webmasters.htm#07)
- Sogou web spider/4.0(+http://www.sogou.com/docs/help/webmasters.htm#07)

- Sogou Orion spider/3.0(http://www.sogou.com/docs/help/webmasters.htm#07)
- Sogou-Test-Spider/4.0 (compatible; MSIE 5.5; Windows 98)

其它搜索引擎的UA

引擎	爬虫名	User-agent
必应	Bingbot	Mozilla/5.0 (compatible; Bingbot/2.0; +http://www.bing.com/bingbot.htm)
雅虎	Slurp	Mozilla/5.0 (compatible; Yahoo! Slurp; http://help.yahoo.com/help/us/ysearch/slurp)
DuckDuckBot	DuckDuckBot	DuckDuckBot/1.0; (+http://duckduckgo.com/duckduckbot.html)
百度	Baiduspider	Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)
Yandex	YandexBot	Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)
脸谱	facebot	facebookexternalhit/1.1 (+http://www.facebook.com/externalhit_uatext.php)
Alexa	ia_archiver	ia_archiver (+http://www.alexa.com/site/help/webmasters; crawler@alexa.com)

当伪装成搜索引擎的爬虫时需要注意的是要仔细阅读搜索引擎的爬虫说明，因为有些爬虫为了增强自身的辨识度蜘蛛是采用固定IP+User-agent来共同标识的。

下载器中间件

下载器中间件是介于下载器(Downloader)与Scrapy引擎(Scrapy Engine)之间的请求/响应处理的钩子框架，用于全局修改Scrapy 请求和响应的一个轻量、可插入式的底层系统。

有时某些代码在蜘蛛执行之前要先执行，又或者需要对请求对象(`Request`)进行重新调整，此时就可以使用下载器中间件这种插件系统在不修改代码的前提下直接将新的功能模块接入到Scrapy框架中。

如本节的主题我们要在每个请求发出前随机地修改其UA值，此时就需要采用到下载器中间件了。

Scrapy提供了非常多的标准内置下载器中间件处理不同的场景，

当我们用`scrapy startproject`指令创建scrapy项目时，scrapy已经为我们默认创建了两个中间件，一个为蜘蛛中间件，另一个就是下载器中间件，你可以打开`middlewares.py`这个文件就能找到它们。

Scrapy原生的注释是全英文的对某些读者可能有点难读，我将它们翻译了一下也可以从中看到下载器中间件的实现要求是什么了，具体代码如下：

```python
class DoubanDownloaderMiddleware(object):

    @classmethod
    def from_crawler(cls, crawler):
        """此方法用于构造下载器中间件及可以从配置中读取配置项
        """
        s = cls()
        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
        return s

    def process_request(self, request, response, spider):
        """当每个request通过下载中间件时，该方法被调用
        :param request: 正在处理的请求对象
        :param response:
        :param spider: 发出请求的蜘蛛对象
        :return:返回值必须是以下的任选一种：
         - None 继续处理当前请求
         - 返回 Response 对象直接返回不发出请求
         - 返回 Request 对象替换原有的Request对象
         - 引发一个 IgnoreRequest异常: process_exception() 方法如果实现了就会被调用
        """
        return None

    def process_response(self, request, response, spider):
        """当完成对request的下载并产生response对象时在调用Spider上的parse方法之前被调用
     :param request: 原请求对象
        :param response:正在处理的response对象
        :param spider: 发出请求的蜘蛛对象
        :return:返回值必须是以下的任选一种：
         - 返回 Response 替换原有的响应对象
         - 返回 Request 对象再次发起请求
         - 引发一个 IgnoreRequest异常
    """
        return response

    def process_exception(self, request, exception, spider):
        """当下载处理器（download handler）或process_request()抛出异常（包括 IgnoreRequest异常）时被调用
        :return: 返回值必须是以下的任选一种
         - None: 跳过
         - 返回 Response 对象: 定停执行 process_exception()链
         - 返回 Request 对象: 定停执行 process_exception() 链
        """
        pass

    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)
```


要应用自定义下载器中间件只需要在配置文件的 DOWNLOADER_MIDDLEWARES 配置项中指定下载器中间件在包中的明确位置即可，如下所示：

```python
DOWNLOADER_MIDDLEWARES = {
   'douban.middlewares.DoubanDownloaderMiddleware': 500,
}
```

随机UA下载器中间件

客户端仿真说起来啰嗦但实现起来并不难，其原理不过是从一个装有一堆User-Agent的数组中随机选出其一然后添加到请求头中。Scrapy也提供了一个 UserAgentMiddleware 中间件用于设置User-Agent，但它却没有卵用，因为它只能设置一个User-Agent，一但设置所有的蜘蛛都只会采用这个User-Agent，完全不能达到随机伪装的效果。

为了可以适应以后可以继续增加更多的UA值我们可以将UA的列表加入到配置文件中：

```
USER_AGENTS = [
    'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0',
    'Mozilla/5.0 (Linux; U; Android 2.2) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1',
    'Mozilla/5.0 (Windows NT 5.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1',
    'Mozilla/5.0 (Linux; Android 6.0.1; SM-G532G Build/MMB29T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.83 Mobile Safari/537.36',
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/604.5.6 (KHTML, like Gecko) Version/11.0.3 Safari/604.5.6'
]
```

然后创建一个 RandomUserAgentMiddleware 中间件:

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

import random

class RandomUserAgentMiddleware(object):
    """
    随机User Agent 中间件
    """
    @classmethod
    def from_crawler(cls, crawler):
        return cls(user_agents=crawler.settings.getlist('USER_AGENTS'))

    def __init__(self, user_agents=[]):
        self.user_agents = user_agents

    def process_request(self, request, spider):
        if self.user_agents != None and len(self.user_agents) > 0:
            request.headers.setdefault(
                b'User-Agent', random.choice(self.user_agents))
```

最后在 settings.py 中启用这个下载器中间件，由于Scrapy会默认启用一个 UserAgentMiddleware 中间件设置固定的UA，所以我们可以在配置文件中将其禁止掉，只要将优先级设置为 None 就可以禁止使用：


```
DOWNLOADER_MIDDLEWARES={
  'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
  'douban.middlewares.RandomUserAgentMiddleware':800
}
```

这样就能支持随机的User-Agent了。

小结

本节花了比较多的字数介绍关于UA的背景知识，而UA仿真又仅仅只是反爬技术中最基础的一种，原因是反爬技术其实并不是什么高深的内容，相反地反爬的知识来源就是最基础的互联网知识，只有完全知道HTTP处理的深入原理才能更好地学习反爬，掌握反爬。

20 基于SQL的数据导出机制

22 反爬之反跟踪与随机代理

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论