常见反爬策略及应对方案

1. 构造合理的HTTP请求头。
    - Accept
    - User-Agent - 三方库fake-useragent

```
from fake_useragent import UserAgent
ua = UserAgent()

ua.ie
# Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US);
ua.msie
# Mozilla/5.0 (compatible; MSIE 10.0; Macintosh; Intel Mac OS X 10_7_3;
Trident/6.0)'
ua['Internet Explorer']
# Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0;
GTB7.4; InfoPath.2; SV1; .NET CLR 3.3.69573; WOW64; en-US)
ua.opera
# Opera/9.80 (X11; Linux i686; U; ru) Presto/2.8.131 Version/11.11
ua.chrome
# Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.2 (KHTML, like Gecko)
Chrome/22.0.1216.0 Safari/537.2'
ua.google
# Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/537.13
(KHTML, like Gecko) Chrome/24.0.1290.1 Safari/537.13
ua['google chrome']
# Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML,
like Gecko) Chrome/20.0.1132.57 Safari/536.11
ua.firefox
# Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:16.0.1) Gecko/20121011
Firefox/16.0.1
ua.ff
# Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:15.0) Gecko/20100101
Firefox/15.0.1
ua.safari
# Mozilla/5.0 (iPad; CPU OS 6_0 like Mac OS X) AppleWebKit/536.26
(KHTML, like Gecko) Version/6.0 Mobile/10A5355d Safari/8536.25

# and the best one, random via real world browser usage statistic
ua.random
```

    - Referer
    - Accept-Encoding
    - Accept-Language
2. 检查网站生成的Cookie。
    - 有用的插件：EditThisCookie
    - 如何处理脚本动态生成的Cookie
3. 抓取动态内容。
    - Selenium + WebDriver
    - Chrome / Firefox - Driver

4. 限制爬取的速度。
5. 处理表单中的隐藏域。
    - 在读取到隐藏域之前不要提交表单
    - 用RoboBrowser这样的工具辅助提交表单
6. 处理表单中的验证码。
    - OCR（Tesseract） - 商业项目一般不考虑
    - 专业识别平台 - 超级鹰 / 云打码

```
from hashlib import md5

class ChaoClient(object):

    def __init__(self, username, password, soft_id):
        self.username = username
        password =  password.encode('utf-8')
        self.password = md5(password).hexdigest()
        self.soft_id = soft_id
        self.base_params = {
            'user': self.username,
            'pass2': self.password,
            'softid': self.soft_id,
        }
        self.headers = {
            'Connection': 'Keep-Alive',
            'User-Agent': 'Mozilla/4.0 (compatible; MSIE 8.0; Windows
NT 5.1; Trident/4.0)',
        }

    def post_pic(self, im, codetype):
        params = {
            'codetype': codetype,
        }
        params.update(self.base_params)
        files = {'userfile': ('captcha.jpg', im)}
        r =
requests.post('http://upload.chaojiying.net/Upload/Processing.php',
data=params, files=files, headers=self.headers)
        return r.json()


if __name__ == '__main__':
    client = ChaoClient('用户名', '密码', '软件ID')
    with open('captcha.jpg', 'rb') as file:

        print(client.post_pic(file, 1902))
```

7. 绕开“陷阱”。
    - 网页上有诱使爬虫爬取的爬取的隐藏链接（陷阱或蜜罐）
    - 通过Selenium+WebDriver+Chrome判断链接是否可见或在可视区域
8. 隐藏身份。
    - 代理服务 - 快代理 / 讯代理 / 芝麻代理 / 蘑菇代理 / 云代理

      《爬虫代理哪家强？十大付费代理详细对比评测出炉！》

- 洋葱路由 - 国内需要翻墙才能使用

```
yum -y install tor
useradd admin -d /home/admin
passwd admin
chown -R admin:admin /home/admin
chown -R admin:admin /var/run/tor
tor
```

反爬虫策略总结

今日终于有点时间了，总结一下网络爬虫领域比较常见的反爬虫策略，希望在我们抓取数据过程中遇到问题时，提供解决方法。话不多说，开讲：

1、最为经典的反爬虫策略当属"验证码"了。因为验证码是图片，用户登录时只需输入一次便可登录成功，而我们程序抓取数据过程中，需要不断的登录，比如我们需要抓取1000个用户的个人信息，则需要填1000次验证码，而手动输入验证码是不现实的，所以验证码的出现曾经难倒了很多网络爬虫工程师。

解决方法：

1）分析网站验证码机制，从网站后台或者前端获取验证码(文本形式)，该方法只适用于少量网站，因为一般验证码我们很难拿到。

2）利用图像识别技术，识别验证码文本(例如最近比较火的深度学习Tensorflow等)。

3）往往一些网站不只有pc端，还有手机端网站，很有可能手机端是不包含验证码的。所以不妨试试手机端网站，也许会有意外收获。

2、 另外一种比较恶心的反爬虫策略当属封ip和封账号了。本人初期曾经用一台机器抓取新浪微博，导致短时间内账号被封，IP被封，所以遇到类似问题一定要多加小心。

解决方法：

1）最简单的解决办法：限制程序抓取频率，每隔几秒登录一次（如果对抓取数量没有要求，则可以采用这种方法，如果想抓取大量数据，还不得抓取到猴年马月啊）。

2）既然封账号封IP，那我就用多个账号、多台机器抓取呗，既解决了反爬虫问题，也相当于做了分流处理，降低单台机器带宽压力。

3）事实证明，有些网站即使我们采用了1）2）方法，还是会被封，这种情况下我们只能去抓取IP代理了，可以写一个专门的爬虫程序用来抓取代理，用这些代理去抓取我们想要的数据。到此为止，基本上封账号、封IP的问题就可以解决了。

3、还有一种比较普通的反爬虫策略：通过cookie限制抓取信息，比如我们模拟登陆之后，想拿到登陆之后某页面信息，千万不要以为模拟登陆之后就所有页面都可以抓了，有时候还需要请求一些中间页面拿到特定cookie，然后才可以抓到我们需要的页面。

解决方法：

1）通过浏览器的F12查看器，观察具体整个过程都请求了哪些URL(主要包括HTML、JS、XHR)，挨个试吧，试到成功为止。

4、另外一种比较常见的反爬虫模式当属采用JS渲染页面了。什么意思呢，就是返回的页面并不是直接请求得到，而是有一部分由JS操作DOM得到，所以那部分数据我们也拿不到咯。

> 解决方法：
>
> 1）一般JS或者XHR都是请求某个链接得到数据，所以需要我们去对应JS、XHR代码里面分析具体请求了哪个链接，然后采用同样的方式，用程序模拟请求。
>
> 2）有一些开源工具包是可以执行js的，例如HtmlUnit，可以试试哦（不过执行起来比较慢，这是缺点）

5、还有一种反爬虫，实在登录部分做了手脚，对用户名或者密码进行加密处理。而我们模拟登陆的时候如果用原始用户名密码的话，肯定登录失败的。

> 解决方法：
>
> 1）一般加密都在某个JS里面，所以只能分析JS，在我们的程序里面进行加密了。
>
> 2）用HtmlUnit等开源工具，直接执行JS，直接就可以登录啦。

6、最近刚发现的一种反爬虫技术：绑定IP。啥意思呢，意思就是整套请求流程必须以一个IP去访问，如果换IP了，对不起，登录失败。

> 解决办法：
>
> 1）没啥解决办法，只能用一个IP登录。。

7、最恶心最恶心的反爬虫，把页面全部转换成图片，你抓取到的内容全部隐藏在图片里。想提取内容，休想。

> 解决办法：
>
> 1）唯一解决的办法：图像识别吧，但是感觉代价很大。。。

先写到这里吧，持续更新ing。

常见的反爬策略汇总

1. 限制IP地址单位时间的访问次数
2. 用户登录才能访问网站内容, 若识别为爬虫账号,封禁IP
3. header, User-Agent检查用户所用客户端的种类和版本, 在请求头中加入CSRF_token识别用户请求(参考form表单验证)
4. Referer, 检查请求由哪里来，通常可以做图片的盗链判断
5. Cookies,检测Cookie中session_id 的使用用次数，如果超过限制，就触发反爬策略略
6. 动态加载,网站使用ajax动态加载内容
7. 对前端请求的API的参数进行加密
8. 对网站JS进行混淆加密(适用于对API参数加密的情况,对用于加密的JS进行混淆)
9. 在用户登录时,进行验证码验证(图片验证码或滑动验证码或短信验证码等)
10. 对网页数据展示的总页数进行限制,比如用户只能浏览200页