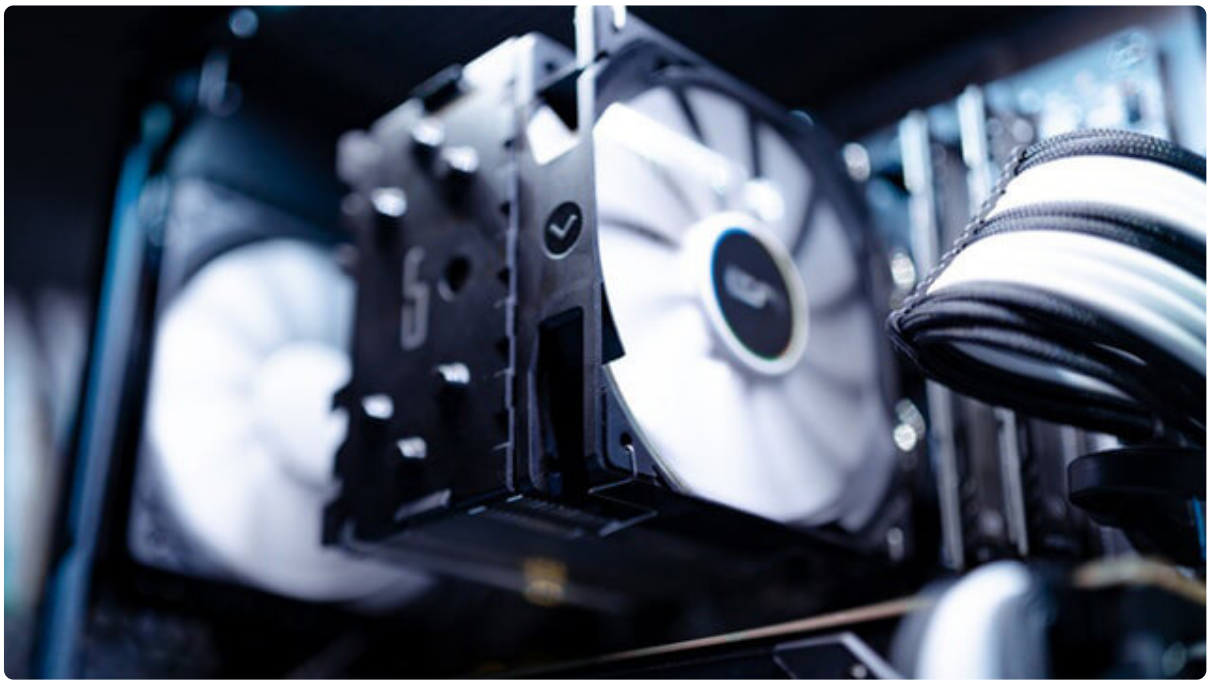
22 反爬之反跟踪与随机代理

更新时间：2019-06-17 10:56:38



> 学习要注意到细处，不是粗枝大叶的，这样可以逐步学习、摸索，找到客观规律。
>
> —— 徐特立

重新认识 Scrapy 的自动降速中间件

Scrapy的蜘蛛的工作效率非常高。它们可以同时处理多个并发请求并充分利用带宽和计算能力。但这种高效在很多情况下并不是好事，因为它很容易造成目标网站的性能下载，或者由于请求过于频繁、数据下载量过大而被对方网站发觉甚至直接被禁止访问。我们的豆瓣爬虫之所以在运行一段时间后就被封IP，同一IP的过于频繁与大量的请求就是其中最大的问题。站在网站开发的角度上对于这种"非人"的请求必然可以判断是由爬虫或某些带有攻击性的机器人程度所为，当然将其IP封禁是理所当然的。那么要避免被封，降速访问就是我们第一个能想到的"无奈之举"了。

那么我们应该如何对Scrapy进行调速呢？

Scrapy的调速的基本原理就是调整并发数量与发出请求间的延时。对此Scrapy有以下几个标准配置选项进行设置：

- `CONCURRENT_REQUESTS` - Scrapy 下载器并发请求(concurrent requests)的最大值。(默认为16)
- `CONCURRENT_REQUESTS_PER_DOMAIN` - 对单个网站进行并发请求的最大值。(默认为8)
- `CONCURRENT_REQUESTS_PER_IP` - 对单个IP进行并发请求的最大值。如果非0，则忽略 `CONCURRENT_REQUESTS_PER_DOMAIN` 设定， 使用该设定。 也就是说，并发限制将针对IP，而不是网站。
- `DOWNLOAD_DELAY` - 下载延时
- `RANDOMIZE_DOWNLOAD_DELAY` - 如果启用，当从相同的网站获取数据时，Scrapy将会等待一个随机的值 (0.5到1.5之间的一个随机值 * `DOWNLOAD_DELAY`)

要避免出现过分频繁的访问请求，就需要在实际部署爬虫系统时设置`DOWNLOAD_DELAY`。其实Scrapy是默认启用了`RANDOMIZE_DOWNLOAD_DELAY`计算一个随机的请求延时间隔。

如果想设置明确的`DOWNLOAD_DELAY`，就必须禁用`RANDOMIZE_DOWNLOAD_DELAY`选项。

默认情况下`DOWNLOAD_DELAY`被设置为0。如果要将每个请求之间的间隔设置为5秒，则可以按以下方式来设置：

```
RANDOMIZE_DOWNLOAD_DELAY = False
DOWNLOAD_DELAY = 5.0
```

> **技巧**:如果项目中正在同时运行多个不同的蜘蛛，而不同蜘蛛之间的请求间隔又不尽相同，则可以直接在蜘蛛中设置其`download_delay`属性以对配置进行单独的覆盖。
>
> ```
> class MySpider(scrapy.Spider):
>  name = 'myspider'
>  download_delay = 5.0
>  ...
> ```

下载延时是不是越大越好呢？理论上是，但你要相对付出的就是"等待时间"，下载延时越久你要等待爬虫完成爬取的时间就越久，调整这个参数一般需要靠**猜**对方网站的极限值，从`5.0`开始一直向下调整，直到被封IP为止，这样就大约可以知道对方大约能接受多大的极限值。

调整并发请求数

除了调整下载延时，另一个做法就是可以调整对每个域的并发请求数，使得蜘蛛变得更有“礼貌”。默认情况下，Scrapy将最多同时向任何给定的域发送8个请求，但可以通过更新`CONCURRENT_REQUESTS_PER_DOMAIN`设置来更改此值。

> 注意：`CONCURRENT_REQUESTS`用于设定Scrapy的下载器在同一时间的并发请求数。调整此设置对于服务器性能/带宽来说比在同一时间抓取多个域时的目标更高。

这个参数的调整办法与延时是相同的，先试验对方的**极限**，我在前两章设计网易爬虫时，开始是直接将`CONCURRENT_REQUESTS`上限调整到32。

AutoThrottle

网站可以处理的请求数量差别很大。如果对每个爬取的网站进行手动调整，则可能会让你抓狂。Scrapy提供了一个自动降速(`AutoThrottle`)的扩展来解决这个问题。`AutoThrottle`根据当前Web服务器负载自动调整请求之间的延迟。它首先计算一个请求的延迟。 然后调整同一个域的请求之间的延迟，使得不超过`AUTOTHROTTLE_TARGET_CONCURRENCY`的请求将同时激活。它还确保请求在给定的时间范围内均匀分布。

自动降速扩展是由以下几个配置项所控制

- `AUTOTHROTTLE_ENABLED` - 启用AutoThrottle扩展(默认为`False`)。
- `AUTOTHROTTLE_START_DELAY`- 初始下载延迟(单位:秒)。
- `AUTOTHROTTLE_MAX_DELAY`- 在高延迟情况下最大的下载延迟(单位秒)。
- `AUTOTHROTTLE_TARGET_CONCURRENCY` - 显示每个响应的降速阀值状态。
- `AUTOTHROTTLE_DEBUG` - 起用AutoThrottle调试(debug)模式，展示每个接收到的response。 您可以通过此来查看限速参数是如何实时被调整的。

我一般情况下会这样配置爬虫的自动降速设定：

```
AUTOTHROTTLE_ENABLED = True # 一定要打开，否则自动降速就会禁用
AUTOTHROTTLE_START_DELAY = 5
AUTOTHROTTLE_MAX_DELAY = 60
AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0
AUTOTHROTTLE_DEBUG = False
```

化身术

所谓的化身术就是将爬虫的行迹进行隐藏，将其**模仿**人或搜索引擎蜘蛛的行为。除了上一节重点讲述的 UA，再配合上整调好的限速设定后最重要的是将请求的IP更换掉。

随机代理 `RandomProxyMiddleware`

更换IP的最佳办法当然是使用代理来访问目标网站了，这个效果就正如我们在浏览器上设置代理上网的原理是一样的。

要让爬虫能化身万千就要有**万千**个UA和**万千**个IP，这是必备的条件。UA只要上网百度一下就可以捞出一个大全，毕竟UA的变换频次不高。但IP的变换则需要一定的代价，主要我们可以想到如下几个办法:

1. 找一个会自动变更IP的代理
2. 用程序控制ADSL Moden重启，每次重启成功都会获得新IP
3. 购买付费代理服务。
4. 收集稳定可用的代理，建立一个代理池，编写一个随机代理中间件在每次发出请求前随机更换代理

在上述的办法中我们显然会直接先第4种，采用随机代理。

Scrapy提供了一个代理插件扩展，但只能设置一个代理地址。所以我们只能重写一个随机代理中间件下载器。

在Scrapy中更换代理地址的办法很简单，只要设置`request.meta['proxy']`属性即可，与随机UA的做法相同，先在`settings.py`文件中增加一个配置项`HTTP_PROXIES`作为代理池存放一个可用的代理地址列表。

然后在 `middlewares` 中加入以下代码:

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

import random


class RandomProxyMiddleware(object):
    """
    随机代理。在运行时会从settings.py设置的PROXIES中随机抽取一个作为当前代理地址。
    """
    @classmethod
    def from_crawler(cls, crawler):
        return cls(proxies=crawler.settings.getlist('HTTP_PROXIES'))

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

    def process_request(self, request, spider):
        request.meta['proxy'] = random.choice(self.proxies)
```

最后，在配置中将随机代理启用起来：

```
DOWNLOADER_MIDDLEWARES = {
    'douban.middlewares.RandomProxyMiddleware': 501,
}
```

> 如果你真的找不到什么代理资源，又不希望付费购买，那么还可以采用一种不可描述的大招，尽破一切IP封禁机制 (专栏结束后可加群索取或在我出版的《Python绝技 - 虫术》电子工业出版社 一书中也有此招)

反cookie跟踪

要模拟出状态化的效果只能通过cookie或者服务端会话（session）实现，如果开启真正的服务端会话，会使服务器的容量到达低谷。因为一旦开启会话，服务器会为每个访问的客户开启一个专用的空间来保存会话。会话一般以20分钟为可用时长，而一旦访问量过大，会话空间就有可能拖垮服务器。禁用会话是开发高性能网站的基本常识，因此很多开发语言中的会话功能都是一种“伪会话”，大多都是通过向cookie写入一个会话ID来进行识别的。这样说来，cookie也就成为了唯一能跟踪客户行为的手段。

比起会话ID，用户的登录信息才是cookie的常客，几乎所有的用户登录功能都要用Cookie来存储。如果网站开发者重视安全性，往往会对cookie中存储的值进行非对称加密，这样就会让客户端无法伪造cookie中的值。

那么在爬虫中应该如何应对呢？

1.禁用cookie

通过禁止cookie，这是客户端主动阻止服务器写入。禁止cookie可以避免那些采用cookies识别爬虫的网站的封杀。在Scrapy爬虫中可以设置`COOKIES_ENABLES=FALSE`，即不启用cookies middleware，不向Web Server发送cookies。

```
# settings.py
COOKIES_ENABLES=False
```

这种方法直接、粗暴、简单。但也遇到过有些网站会发送随机cookie，如果检测到客户端没有回发随机生成的cookie，则将客户端认定为爬虫而直接封掉。另外，这种做法一旦遇到需要登录的场景就完全失效，所以说它**仅仅适用于那些不需要验证登录身份**的开放式网站或者页面。

2.合并cookie

也就是将每次响应收到的cookie先存起来，在下次发出请求时使用。这种做法在Scrapy中是不用设置的，正如前面所述cookies中间件是默认启用的，它会自动进行处理。

反跟踪技术 - `ReferMiddleware`

要从服务器端主动跟踪和分析客户端的异常行为，其实手段相当有限。只要深入地反思一下这个问题就能找到答案：“你可以用何种手段在网站上跟踪访客？”首先我们知道Web服务器是无状态的，要跟踪访客的行为就需要知道客户端的某一种状态。服务端只能从请求头上得知客户端的一些基本信息。例如，通过UserAgent获取浏览器的名称和版本，通过IP地址计算归属地，通过Referer了解这个请求是从哪个页面引入的，通过查询字符串（Query String）显式地传入参数进行某些判断。而这些都是一次性信息，并不带有任何状态。

referer策略

referer用于告诉服务器当前这个请求是由哪个页面转入的，这是一个URL值，经常用于反盗链检测（是反盗链而不是反爬）。这种手段常见于一些大型网站或者老牌网站（如百度），一旦触发了反盗链机制，所获得的网页内容就与原来的内容完全不同了，但爬虫却可能完全不知。它不会使请求失败，而是会响应写入其他版权保护信息。例如，爬取的是一张图片，但由于referer的设置违反了对方的引用策略，那么得到的可能就是一个“该图片来自XXX网站”的占位图。

referrer策略包含以下值：

- no-referrer - 最简单的策略是“no-referrer”，表示所有的请求都不带referrer。
- no-referrer-when-downgrade - 主要针对于受TLS保护的URL（如HTTPS），简单地说就是在HTTPS的页面中，如果连接的资源也是HTTPS的，则发送完整的referrer，如果连接的资源是HTTP的，就不发送referrer。这是在没有特别指定referrer策略时，浏览器的默认行为。
- same-origin - 对于同源的链接，会发送referrer，其他的不会。同源意味着域名需要相同，example.com和not.example.com是非同源的。
- origin - 这个策略对于任何资源来说只发送源的信息，不发送完整的URL。
- strict-origin - 这个策略类似于origin和no-referrer-when-downgrade的合体，如果一个HTTPS页面中链接到HTTP的页面或资源，则不会发送referrer。HTTP页面链接和HTTPS链接到HTTPS都只发送来源页面的源信息。
- origin-when-cross-origin - 该策略在同源的链接中发送完整的URL，其他情况仅发送源信息。相同的域名，HTTP和HTTPS协议被认为是非同源的。
- strict-origin-when-cross-origin - 对于同源请求，发送完整的URL；对于同为HTTPS的，只发送源信息；对于HTTPS页面只发送源信息；HTTPS页面中的HTTP请求不发送referrer。
- unsafe-url - 这个主要是解决HTTPS页面中的HTTP资源不发送referrer的问题，它会使在HTTPS页面中的HTTP资源发送完整的referrer。
- 空字符串 - 空字符串表示没有referrer策略，默认为no-referrer-when-downgrade。

那么怎么才知道目标网站采用了哪个referrer策略呢？

referrer策略会通通过以下方法声明：

1. 通过HTTP请求头中的Referrer-Policy字段。
2. 通过meta标签，name为referrer，如：`<meta name="referrer" content="same-origin" />`。
3. 通过`<a>`、`<area>`、`<img>`、`<iframe>`、`<link>`元素的referrerpolicy属性。
4. 通过`<a>`、`<area><link>`元素的`rel=noreferrer`属性。
5. 通过隐式继承。

因此，一旦遇到具有反盗链策略的网站，就要先对上述地方进行检测，得出对方的引用策略后才能正确应对。Scrapy提供了一个`RefererMiddleware`中间件用于处理referer，但它只能用于统一地将所有请求设置为配置中指定的referrer策略，默认情况下它是被启用的。如果要关闭它，则只需要将配置文件中的`REFERER_ENABLED`设置为`False`即可。

```
REFERER_ENABLED=False
```

通过设置`REFERRER_POLICY`可以默认启用referrer策略：

```
REFERRER_POLICY='origin'
```

另外，还可以在代码中动态设置referrer策略：

```
request.meta['referrer_policy'] = 'origin-when-cross-origin'
```

小结

主要的放反技术在这两节中都分重点进行了一个介绍，这两节书的内容对于初学者的学习曲线可能会稍显陡峭，毕竟防反是一种基础网络技术的范畴，要运用得好得活学活用而且还需要对多个基础领域深入了解。所以从"基于SQL数据导出机制"一节开始我就在专栏内容里面留下一些思考的内容，而且没有给出答案。只有自己独立面对问题并经过思考寻找答案才是真正掌握一门技术的关键。同样地，在本节我也给你留了一个思考题：在如此多的防反技巧中应该如何来配置你的`settings.py`文件呢？

21 反爬之客户端仿真

23 分布式网络爬虫

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