前言

我们在做爬虫的过程中经常会遇到这样的情况：最初爬虫正常运行，正常抓取数据，然而一杯茶的功夫可能就会出现错误，比如403 Forbidden；这时候网页上可能会出现 “您的IP访问频率太高”这样的提示，过很久之后才可能解封，但是一会后又出现这种情况。

造成这种现象的原因是该网站已采取了一些防爬虫措施。 例如，服务器将在一个时间单位内检测IP请求的数量。 如果超过某个阈值，服务器将直接拒绝该服务并返回一些错误信息。 这种情况可以称为封IP，因此该网站成功禁止了我们的抓取工具。

想象一下，由于服务器检测到IP单位时间内的请求数量，因此我们使用某种方式来伪装IP，以使服务器无法识别由本地计算机发起的请求，因此我们可以成功地阻止IP被封。

所以这时候代理池就派上用场了。

网络上有大量免费且公开的代理可以供我们使用，但这些单利并不能保证都可以使用，因为同样的代理可能被其他人拿来爬虫使用而遭到封禁，因此，在真正使用之前，我们需要对这些免费代理进行筛选，剔除那些不能使用的。保留下可以用的，来构建一个代理池，供我们爬虫使用。

1、代理池设计

- 构建一个代理IP池，可能有下面这些问题：

1、代理IP从何而来?

许多刚接触爬虫的，都试过去西刺、快代理之类有免费代理的网站去抓些免费代理，还是有一些代理能用。 当然，如果你有更好的代理接口也可以自己接入。

免费代理的采集也很简单： 访问页面 —> 正则/xpath提取 —> 保存

2、如何保证代理质量?

可以肯定免费的代理IP大部分都是不能用的，不然别人还提供付费接口干嘛(不过事实上很多代理商的付费IP也不稳定，也有很多是不能用)。 所以采集回来的代理IP不能直接使用，检测的办法也很简单：可以写个程序不断的用代理访问一个稳定的网站，看是否可以正常访问即可。 这个过程可以使用多线/进程或异步的方式，因为检测代理是个很慢的过程。

3、采集回来的代理如何存储?

这里选择 Redis 作为数据存储。

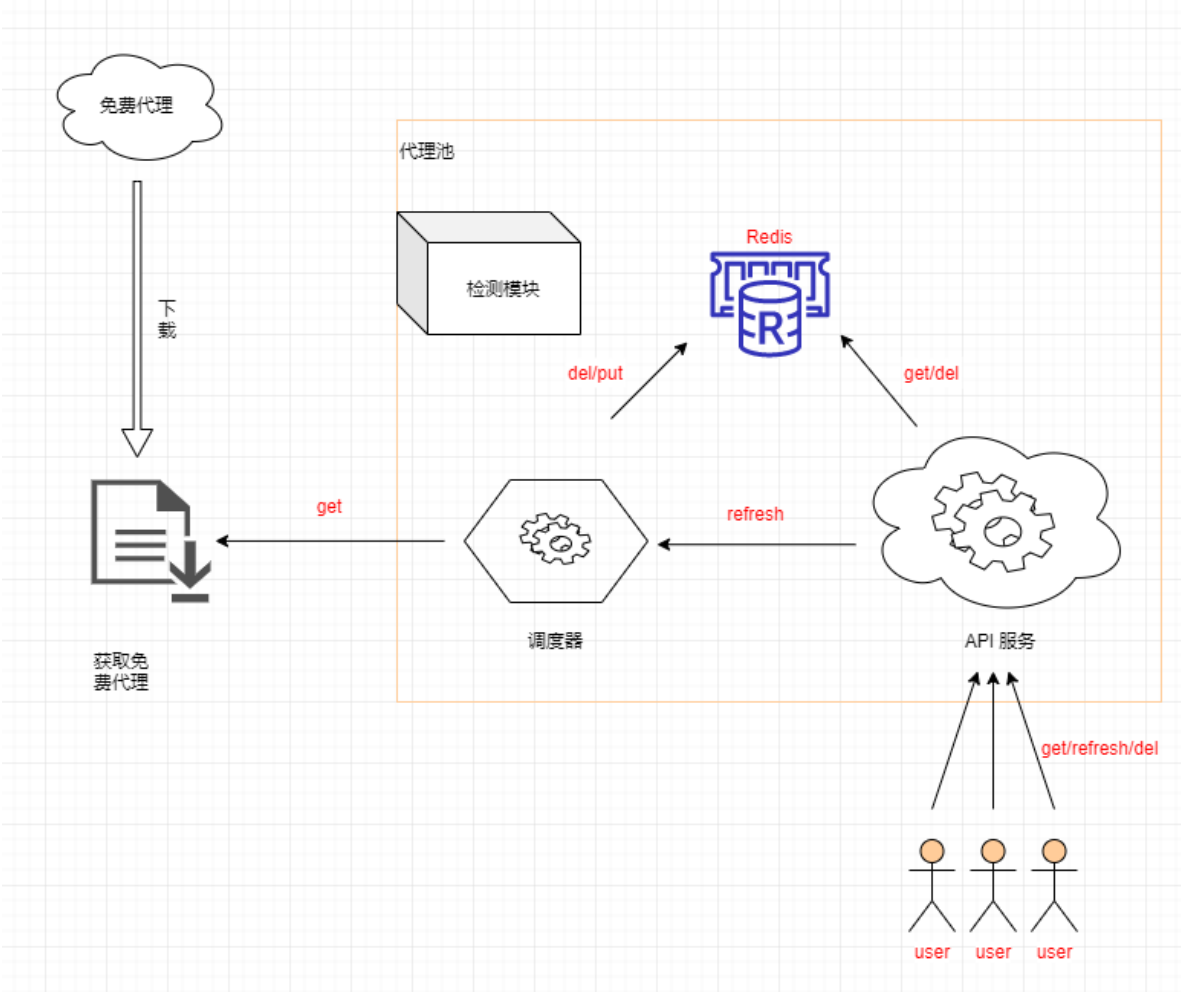
4、如何让爬虫更方便的用到这些代理?

答案是服务，使用Python的web框架，写个api供爬虫调用。这样代理和爬虫架构分离，同事有很多好处， 比如：当爬虫完全不用考虑如何校验代理，如何保证拿到的代理可用，这些都由代理池来完成。这样只需要安静的码爬虫代码就行啦。

代理池由四部分组成：

- **获取模块**：代理获取接口，负责到各大免费代理网站爬取代理。代理形式都是IP+端口
- **存储模块**：负责存储抓取下来的代理。保存到数据库。
- **检测模块**：获取到的代理不一定都能使用，因此需要对抓到的每个代理，针对未来将要爬取的网站进行检测。 测试过程中，如果可用就保存，不可用就删除。并且根据每个代理的表现，给代理进行评分，分数越高可用性越高。

- **接口模块**：使用轻量级的 Flask 来实现一个 webAPI 接口。



2、Redis数据库

2、1 Redis是什么

Redis 常被称作是一款数据结构服务器（data structure server）。Redis 的键值可以包括字符串（strings）类型，同时它还包括哈希（hashes）、列表（lists）、集合（sets）和 有序集合（sortedsets）等数据类型。

对于这些数据类型，你可以执行原子操作。例如：对字符串进行附加操作（append）；递增哈希中的值；向列表中增加元素；计算集合的交集、并集与差集等。

2、2 redis应用场景

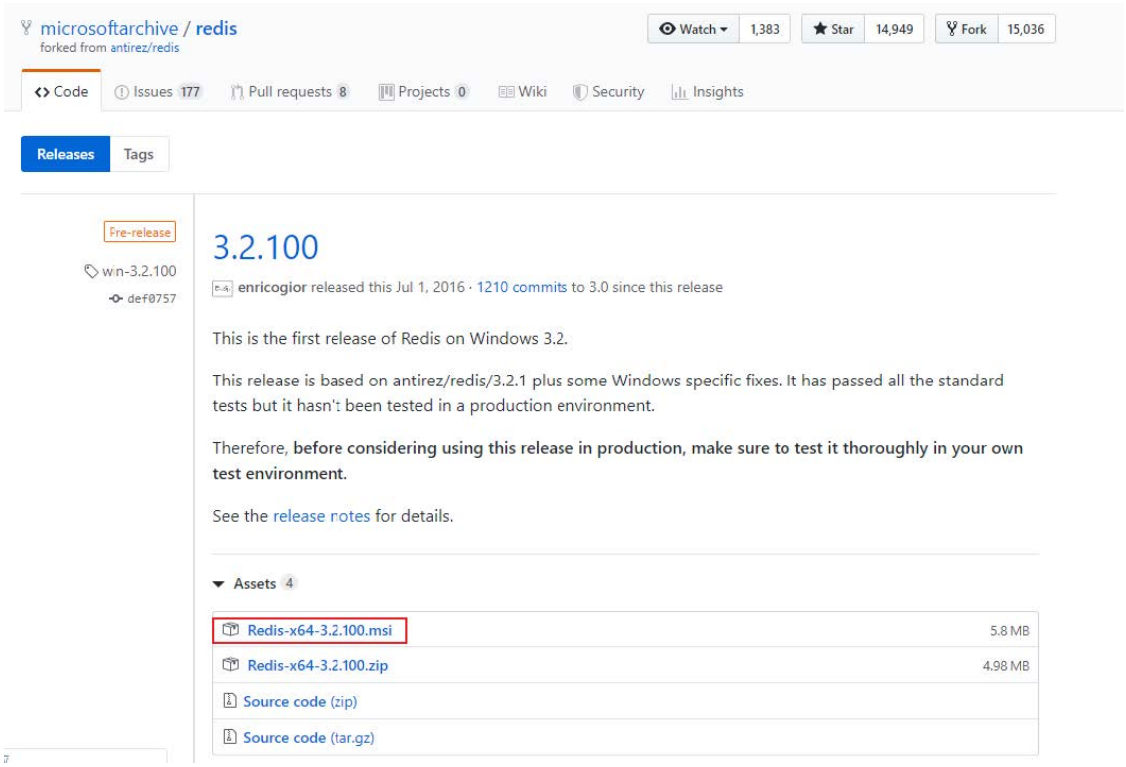
- 用来做缓存(ehcache/memcached)——redis的所有数据是放在内存中的（内存数据库）
- 可以在某些特定应用场景下替代传统数据库——比如社交类的应用
- 在一些大型系统中，巧妙地实现一些特定的功能：session共享、购物车
- 只要你有丰富的想象力，redis可以用在可以给你无限的惊喜.....

2、3 Redis安装

- Linux
- 直接输入命令 sudo apt-get install redis-server 。安装完成后，Redis服务器会自动启动。使用redis-cli 命令即可进入Redis数据库。
- Mac电脑请参考如下教程

    - https://blog.csdn.net/wto882dim/article/details/99684311
- windows

  点击: https://github.com/microsoftarchive/redis/releases
  然后选择 msi 文件



下载完成后双击打开安装（安装过程中一定要添加到 path 环境）

2、4 Redis数据库有序集合操作

首先咱们要安装一下Redis数据库操作模块, 通过如下命令:

```
pip install redis
```

- zadd： 添加一个有序的集合
- zrangebyscore： 获取指定序列(评分)的数据
- zscore： 查询数据在集合中的排序(评分)
- zincrby： 修改集合序列值
- zcard： 获取有序集合的数量

3、代理池实现

3、1 采集模块

搭建代理池，第一步是采集代理。可以在搜索引擎搜索“代理” 关键字 ，就可以到许多代理服务网站，网站上会有很多免费代理。

但是这些免费代理大多数情况下都是不好用的，所以比较靠谱的方法是购买付费代理付费代理在很网站都有售卖，数量不用多，稳定可用即可，我们可以进行选购。

这里我们**以云代理**为例，使用了设置了 headers 模拟请求头之后就能直接拿到数据。

```
import requests
import re


headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)Chrome/86.0.4240.193 Safari/537.36'}

proxies_pattern = re.compile('.*?(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}).*?(\d{1,5}).*?', re.S)


def spider_yun_proxies():
    """
```

```
    爬取云代理ip
    目标网址: http://www.ip3366.net/
    """
     for page in range(1, 11):
            time.sleep(1)
            print('云代理', f'http://www.ip3366.net/?stype=1&page={page}')
            html_data = requests.get(url=f'http://www.ip3366.net/?stype=1&page={page}',
headers=headers).text
            ip_port = re.findall(proxies_pattern, html_data)
            # print(ip_port)
            for ip, port in ip_port:
                  yield ip + ":" + port
```

这样我们就将数据给下载下来了。这里我只使用了一家做示范，到项目真实部署，则需要更多地免费代理，否则当大规模采集时，ip可能不够用。

代理名称	状态	更新速度	可用率	地址
无忧代理	√可用	★	*	**地址**
66代理	√可用	★★	*	**地址**
~~西刺代理~~	~~已关闭~~	——	——	**~~地址~~**
全网代理	√可用	★	*	**地址**
快代理	√可用	☆	*	**地址**
代理盒子	√可用	★★★	*	**地址**
云代理	√可用	★	*	**地址**
~~IP海~~	~~已关闭~~	——	——	**~~地址~~**
免费代理库	√可用	☆	*	**地址**

3、2 存储模块

数据选择保存在 REDIS 数据库中Redis 数据库非常快而且安全。操作足够简单。真正的代理池都是用的是 Redis 的数据库。
封装的方法主要是支持基本的链接数据库、添加数据、获取代理。

- 定义 RedisClient() 类

  1、一旦实例化该类，自动初始化【连接数据库】，并作为该实例的一个属性

  2、类实现以下方法：

  - init初始化方法链接数据库
  - add() 添加代理并将其设置为初始分数
  - random() 随机选一个代理
  - decrease() 代理质量检测
  - max() 代理设置最大分数
  - count() 获取代理数量
  - all() 获取所有代理
  - count_for_num() 指定数量获取代理

```
"""
数据库模块
```

```
"""
import random
import redis


class RedisClient(object):
    def __init__(self, host='127.0.0.1', port=6379, db=0, **kwargs):
        """初始化redis客户端"""
        self.db = redis.Redis(host=host, port=port, db=db, decode_responses=True, **kwargs)

    def exists(self, proxy):
        """判断传入的的代理有没有存在于数据库"""
        return not self.db.zscore('PROXIES', proxy) is None

    def add(self, proxy, score=10):
        """添加代理到数据库, 并将代理设置为初始分数"""
        # 判断数据库有没有当前传进来的代理
        if not self.exists(proxy):
            # 如果数据库没有存储该代理, 执行数据插入
            return self.db.zadd('PROXIES', {proxy: score})

    def random(self):
        """随机选一个代理"""
        proxies = self.db.zrangebyscore('PROXIES', 100, 100)
        if len(proxies):
            return random.choice(proxies)
        proxies = self.db.zrangebyscore('PROXIES', 10, 99)
        if len(proxies):
            return random.choice(proxies)
        print('########---数据库为空---#########')

    def decrease(self, proxy):
        """传入一个代理, 如果检测不过关, 则降低代理的分数"""
        self.db.zincrby('PROXIES', -9, proxy)
        score = self.db.zscore('PROXIES', proxy)
        if score <= 0:
            self.db.zrem('PROXIES', proxy)

    def max(self, proxy):
        """如果检测的代理可用, 将代理设置最大分数"""
        return self.db.zadd('PROXIES', {proxy: 100})

    def count(self):
        """获取代理的数量"""
        return self.db.zcard('PROXIES')   # zcard()  查询数据库集合中的所有代理

    def all(self):
        """获取所有代理, 返回列表"""
        proxies = self.db.zrangebyscore('PROXIES', 1, 100)
        if proxies:
            return proxies
        else:
            raise Exception('代理池无代理可用')

    def count_for_num(self, number):
        """指定数量获取代理, 返回一个列表"""
        all_proxies = self.all()
        proxies = random.sample(all_proxies, k=number)
```

```
            return proxies


if __name__ == '__main__':
      proxies = ['927.72.91.211:9999', '927.12.91.211:8888', '927.792.91.219:7777',
'927.732.91.211:6666', ]
      redis_client = RedisClient()
      # for proxy in proxies:
      #       redis_client.add(proxy)

      result = redis_client.count_for_num(3)
      print(result)
```

3、3 检测模块

已经成功将各个网站的代理获取下来了，接下来就是对这些代理进行筛选，选出那些比较好用的，对于不好的可以适当降低评分。当分数为 0 时彻底把代理删除。

reqeusts是一个同步请求库，当发送一个请求之后，程序就会阻塞在等待服务器返回内容的过程中。整个过程大概会阻塞一到五秒，如果网络不好或者遇上了服务器请求过载，需要的时间可能会更多。在等待服务器响应的期间，程序不会自动继续向下执行。而程序员完全可以把cpu充分利用起来。

为了提高代理的检测效率，可以使用多线程/多进程的方式来提升检测的速度。

定义一个 verify_proxies() 方法

- 定义一个方法，检测代理是否可用
    - 可用将代理评分设置最高
    - 不可用将代理评分减1

```
"""
     检测模块
"""
import requests
from db import RedisClient
import concurrent.futures


TEST_URL = "https://www.baidu.com"

client = RedisClient()


def verify_proxy(proxy_list):
      """
      检测代理是否可用，传入一个列表参数
      """
      print('############--正在检测代理--##########')

      for proxy in proxy_list:
            proxies = {
                  "http": "http://" + proxy,
                  "https": "http://" + proxy,}

            try:
                  response = requests.get(url=TEST_URL, proxies=proxies, timeout=2)
                  if response.status_code in [200, 206, 302]:   # 200 206 302
```

```
                        print('******代理可用******', proxy)
                        client.max(proxy)
                  else:
                        print('--请求状态码不合法--', proxy)
                        client.decrease(proxy)
            except Exception as e:
                  client.decrease(proxy)
                  print('-----请求失败-----', proxy)

# 速度太慢了 ？ 多任务: 多线程,多进程,多进程嵌套多线程
def verify_thread_pool():
      """线程池测试代理"""
      proxies_list = client.all()
      with concurrent.futures.ThreadPoolExecutor(max_workers=10) as executor:
            executor.submit(verify_proxy, proxies_list)
```

3、4 web接口

为了方便爬虫接入代理池，利用Flask框架搭建一个简陋的接口。这样当爬虫需要使用时，调用接口就可以获取免费代理。

1. 实例化一个 web 对象
2. 编写请求API

```
"""
      编写接口: 提供代理给客户端去使用
      客户端访问网址就能拿到代理数据
"""
import flask
from flask import request
from db import RedisClient
from flask import jsonify


app = flask.Flask(__name__)
client = RedisClient()


@app.route('/')
def index():
      # 视图函数只能返回字符串
      return '<h2>欢迎来到代理池</h2>'

@app.route('/get/')
def get_proxy():
      """随机获取一个代理, 调用数据库模块 random() 方法"""
      one_proxy = client.random()
      return one_proxy

@app.route('/getcount')
def get_any_proxy():
      """获取指定数量的代理, 调用数据库模块 count_for_num() 方法"""
      num = request.args.get('num', '')
      if not num:
            num = 1
      else:
            num = int(num)
```

```
    any_proxy = client.count_for_num(num)

    return jsonify(any_proxy)

@app.route('/getnum')
def get_count_proxy():
    """获取所有代理的数量，调用数据库模块 count() 方法"""
    count_proxy = client.count()
    return f'代理池还有 {count_proxy} 个代理可用!!!'

@app.route('/getall')
def get_all_proxy():
    """获取所有代理，调用数据库模块 all() 方法"""
    all_proxy = client.all()
    return jsonify(all_proxy)


if __name__ == '__main__':
    app.run(debug=True)
```

3、5 调度模块

目的：定义scheduler()类。实现多进程运行以上三个模块：获取、检测、web api 的方法

```
"""
    调度模块
"""
import time
from db import RedisClient
from getter import proxies_func_list
from verify import verify_thread_pool
from api import app
import multiprocessing
import concurrent.futures


client = RedisClient()

class Schedule(object):
    def getter_proxy(self):
      """
          # 1.获取代理
          免费的代理会更新，定时请求代理 五分钟
      """
      while True:
          for func in proxies_func_list:
              proxies = func()
                 for proxy in proxies:
                  print('--代理写入数据库--', proxy)
                      client.add(proxy)
          time.sleep(GETTER_PROXY)

    def verify_proxy(self):
          """
          # 2.验证代理
          代理验证一次就不验证了，我们也要设置一个时间阈值，不断的检测代理 十分钟
        """
```

```
            while True:
                verify_thread_pool()
                time.sleep(60 * 8)

    # 3. 调用api服务模块
    def api_server(self):
        """
        # 3.提供api服务
        """
        app.run()

    # 调用三个方法一起去执行
    def run(self):
        # 各自有各自的逻辑
        print('################--代理池开始运行--################')
        getter_proxy_process = multiprocessing.Process(target=self.getter_proxy)
        getter_proxy_process.start()

        verify_proxy_process = multiprocessing.Process(target=self.verify_proxy)
        verify_proxy_process.start()

        api_server_process = multiprocessing.Process(target=self.api_server)
        api_server_process.start()


if __name__ == '__main__':
    work = Schedule()
    work.run()
```

3、6 配置文档

创建config.py 文件，方便我们修改项目的配置：

- redis数据库的地址，端口，密码等
- 请求代理和验证代理时间间隔设置
- 服务器配置
- 测试的 url 地址配置

```
"""数据库配置"""
REDIS_HOST = '127.0.0.1' # 数据库所在ip地址
REDIS_PORT = 6379 # 数据库端口
REDIS_DATABASE = 0 # 操作哪一个数据库
REDIS_OBJECT = 'PROXIES' # 数据库操作的对象

"""时间间隔配置"""
GETTER_PROXY_TIME = 60 * 10 # 获取代理的时间间隔
VERIFY_PROXY_TIME = 60 * 8 # 验证代理的时间间隔"""服务器配置"""
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 80

"""服务器配置"""
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 80

"""所需测试的 url 地址"""
TEST_URL = 'http://www.baidu.com/'
```