**系统成果交付**

--线下最终版

--24组

--姓名：葛萌萌（组长）

--学号：2017210461

--班级：2017211102

--小组其他成员：罗平

--时间：2019.06.13

**--NOTE:项目地址请见6部分**

目录

[1、系统概述 1](#_Toc11361432)

[2、系统说明 4](#_Toc11361433)

[3、功能测试 24](#_Toc11361434)

[4、性能测试 30](#_Toc11361435)

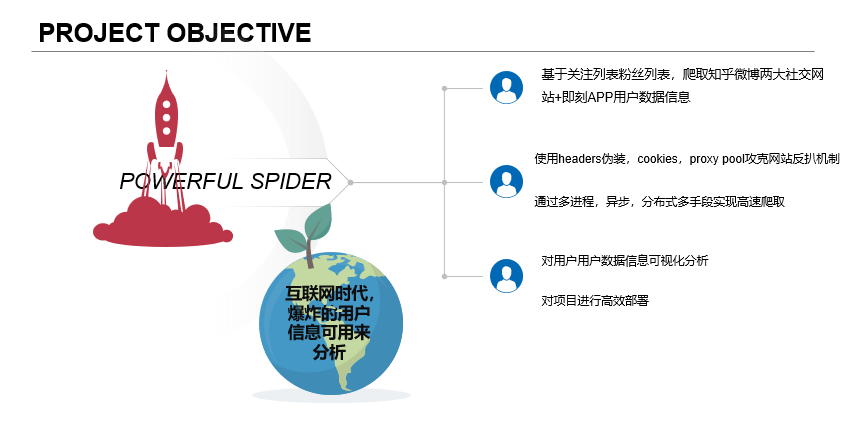
[5、测试结论和建议 31](#_Toc11361436)

[6、项目地址： 32](#_Toc11361437)

# 1、系统概述

**1.1、系统简介**

**背景+目标**：互联网时代为我们提供了海量的数据资源，而这些资源并非直接呈现，需要采用合法合理高效的手段进行提取，所以爬虫应运而生，而克服多种反扒机制以及提升爬虫性能和简易化实现爬虫项目分发部署在实际中尤为重要。用户信息也是现在的热门话题，而各大社交平台提供了海量的公开用户信息，通过挖掘分析这些信息可以发现跟多有趣的是事情。因此便有了我们**PowerFul crawler**项目组的“**用户信息爬虫**”项目。



**项目基本内容**：专注于爬虫方面开发，爬取知乎、微博、即刻APP上的用户信息，并且进行性能优化，项目部署管理，以及简要数据分析。

**基本完成情况：**



**1.2、术语表**

|  |  |  |
| --- | --- | --- |
| **序号** | **术语或缩略语** | **说明性定义** |
| 1 | Pycharm vscode | 集成开发环境作为基本的编码调试运行工具 |
| 2 | Scrapy | 项目主要依赖的python高效爬虫库 |
| 3 | Redis mongodb | 数据库 |
| 4 | Robomongo RedisDesktopManager | 数据库连接工具 |
| 5 | Mongodb-charts | 数据库可视化工具 |
| 6 | Appium | APP自动化操作工具 |
| 7 | Charles mitmproxy | App抓包工具 |
| 8 | Xshell | 远程ssh连接工具 |
| 9 | Git | 项目管理工具 |
| 10 | Docker | 项目发布部署工具 |
| 11 | Genymotion | PC端的安卓模拟器 |
| 12 | Andriod\_SDK | 主要借助其中adb工具进行移动端控制 |
| 13 | VMware VirtualBox | PC端的虚拟机支持软件 |
| 14 | GitHub | 程序员交友平台。。。 |
| 15 | Python库 | 项目开发需要用到的python库不一一列举 |

**1.3、系统运行环境**

**硬件平台：**

安装docker的主机（容器的存在使项目部署如此便捷）+云主机（分布式支持）

**操作系统：**

不限系统，项目最终发布的容器内集成了需要的python环境

**数据库系统**：

redis mongodb

有一台主机必须要有redis（分布式调度主机）

其余主机需要有容器技术实现redis mongodb环境的安装

**编程平台（开发工具）：**

Pycharm Vscode Git

**网络协议：**

http https

**1.4、开发环境**

**系统分析、程序设计工具：**

云笔记，xmind8，Visio

**程序开发：**

工程工具：

pycharm vscode git docker redis mongodb 数据库连接可视化工具appium charles mitmproxy Andriod\_SDK xshell Genymotion VMware VirtualBox

开发语言：

python3

**工具软件的名称、版本：**

Pycharm vscode 集成开发环境作为基本的编码调试运行工具

Redis mongodb 以及一些链接客户端作为数据库以及数据库操作可视化工具

Appium app 自动化操作工具

Charles mitmproxy App抓包工具

Xshell 远程ssh连接工具

Git 项目管理工具

Docker 项目发布部署工具

Genymotion PC端的安卓模拟器

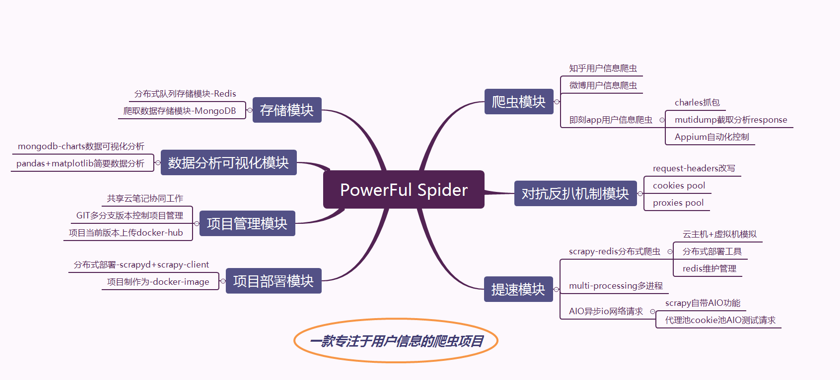
Andriod\_SDK 主要借助其中adb工具进行移动端控制

VMware VirtualBox PC端的虚拟机

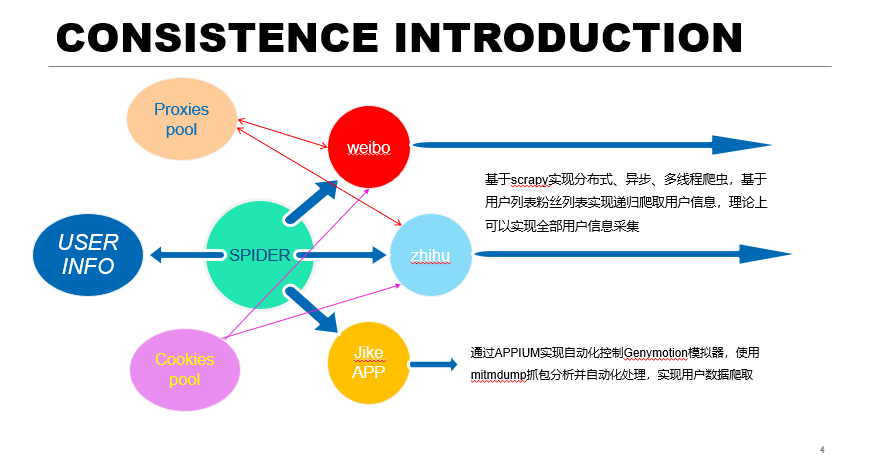
# 2、系统说明

**2.1、整体说明**

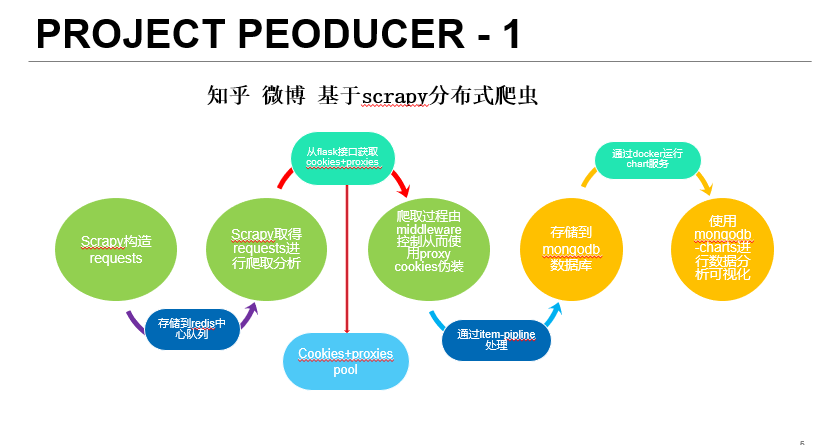
**系统架构**

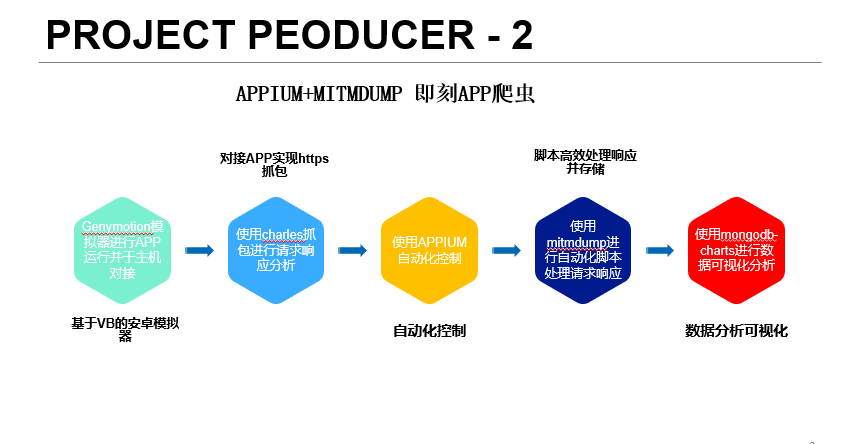


**系统主要组成部分**



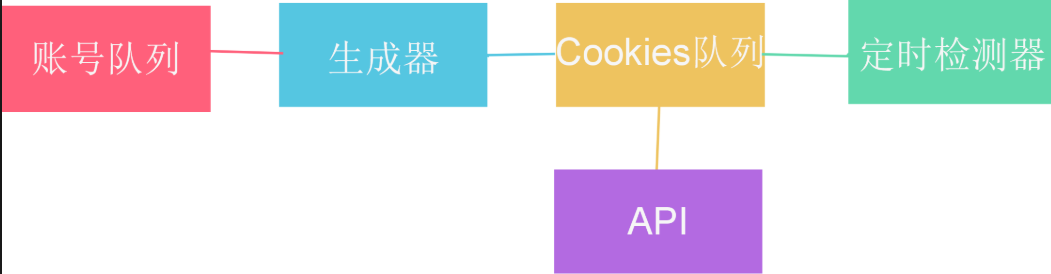
**系统三大爬虫基本流程**



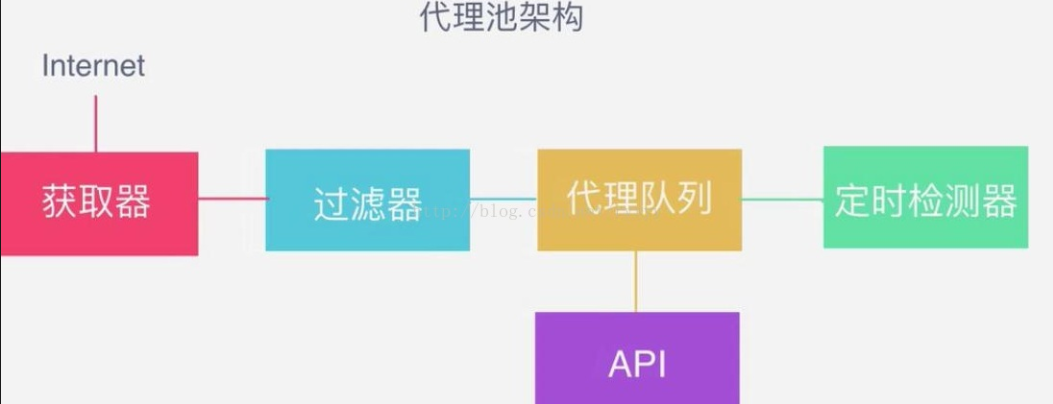


**系统cookies+proxy pool处理流程**

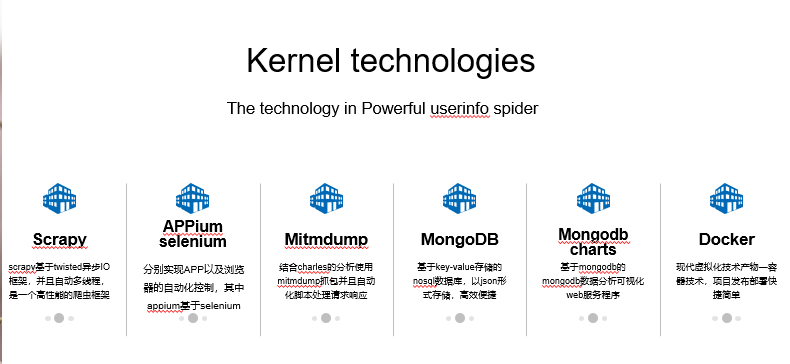
**Cookies池**



**代理池**



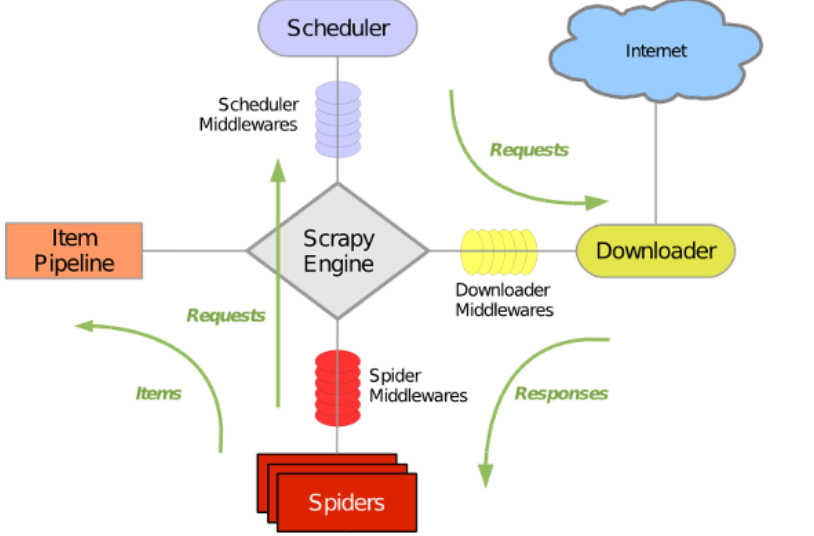
**系统所用核心技术**





**声明：以下部分与详细设计中的内容差别不大**

**2.2、模块0—核心基础模块**



**2.2.1基本组成：**

* 引擎(Scrapy Engine)，用来处理整个系统的数据流处理，触发事务。
* 调度器(Scheduler)，用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回。
* 下载器(Downloader)，用于下载网页内容，并将网页内容返回给蜘蛛。
* 蜘蛛(Spiders)，蜘蛛是主要干活的，用它来制订特定域名或网页的解析规则。编写用于分析response并提取item(即获取到的item)或额外跟进的URL的类。 每个spider负责处理一个特定(或一些)网站。
* 项目管道(Item Pipeline)，负责处理有蜘蛛从网页中抽取的项目，他的主要任务是清晰、验证和存储数据。当页面被蜘蛛解析后，将被发送到项目管道，并经过几个特定的次序处理数据。
* 下载器中间件(Downloader Middlewares)，位于Scrapy引擎和下载器之间的钩子框架，主要是处理Scrapy引擎与下载器之间的请求及响应。
* 蜘蛛中间件(Spider Middlewares)，介于Scrapy引擎和蜘蛛之间的钩子框架，主要工作是处理蜘蛛的响应输入和请求输出。
* 调度中间件(Scheduler Middlewares)，介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。

**2.2.2基本流程：**

1. 引擎打开一个网站(open a domain)，找到处理该网站的Spider并向该spider请求第一个要爬取的URL(s)。
2. 引擎从Spider中获取到第一个要爬取的URL并在调度器(Scheduler)以Request调度。
3. 引擎向调度器请求下一个要爬取的URL。
4. 调度器返回下一个要爬取的URL给引擎，引擎将URL通过下载中间件(请求(request)方向)转发给下载器(Downloader)。
5. 一旦页面下载完毕，下载器生成一个该页面的Response，并将其通过下载中间件(返回(response)方向)发送给引擎。
6. 引擎从下载器中接收到Response并通过Spider中间件(输入方向)发送给Spider处理。
7. Spider处理Response并返回爬取到的Item及(跟进的)新的Request给引擎。
8. 引擎将(Spider返回的)爬取到的Item给Item Pipeline，将(Spider返回的)Request给调度器。
9. (从第二步)重复直到调度器中没有更多地request，引擎关闭该网站。

**3.3、模块1**

**爬虫模块**

三个爬虫，分别是基于scrapy的知乎以及微博用户信息爬虫，以及基于charles抓包分析，appium自动化操作，mitmdump抓包脚本处理的即刻APP爬虫。

[1] 知乎爬虫 微博爬虫

**思路：**

基于用户🡪粉丝列表关注列表+翻页🡪用户

实现递归爬去 递归生成请求 保存到scrapy的request队列中

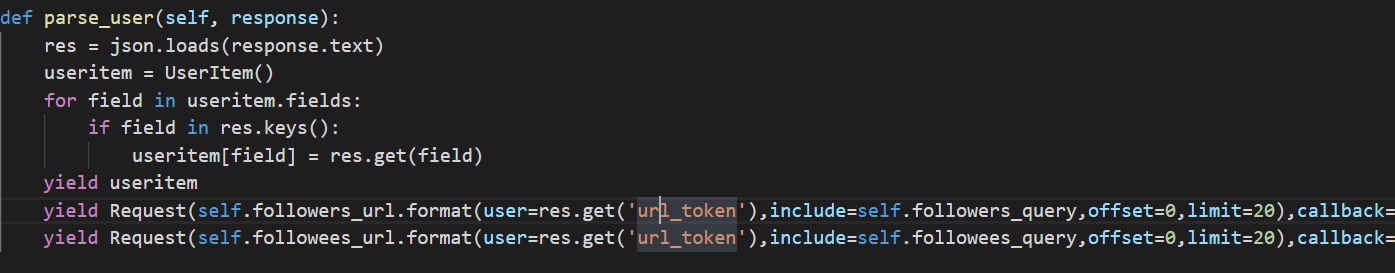
**具体代码剖析：**

**（1）爬虫代码**

知乎爬虫：

以下是在用户解析，粉丝列表关注列表解析的函数

用户解析函数需要填充对象field 从而生成需要的item

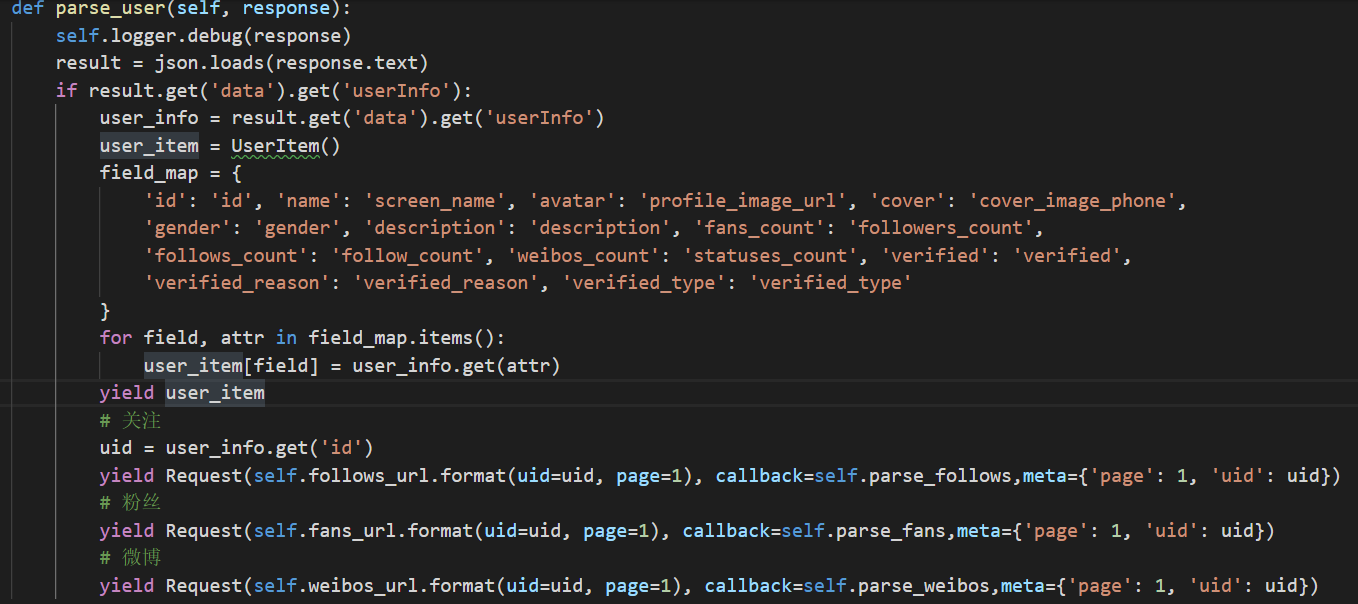


粉丝列表关注列表函数需要对于每个用户回调用户解析函数，并且翻页



微博爬虫：

类似上面的逻辑，只是生成对象item使用了预先定义的字典映射的方式，从而填充item的每个field，返回item并且发起粉丝列表关注列表以及微博正文的请求



粉丝关注列表解析函数类似知乎爬虫，只是额外改进添加了对于粉丝列表关注列表进行单独保存的功能，并且为了保证能够正确异步存储粉丝关注列表，需要在构造的请求中通过meta传递元数据，保证正确存储。

基本功能：生成用户request ，关注列表/粉丝列表生成 ，请求下一页



粉丝列表函数类似不再赘述

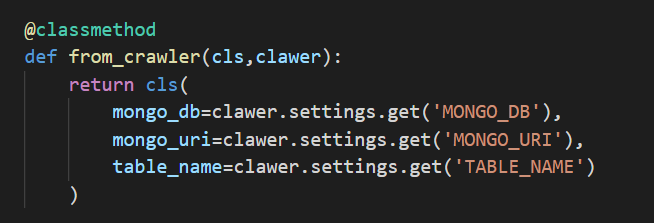
**（2）处理代码**

处理部分包括对于item的处理，以及downloader的处理

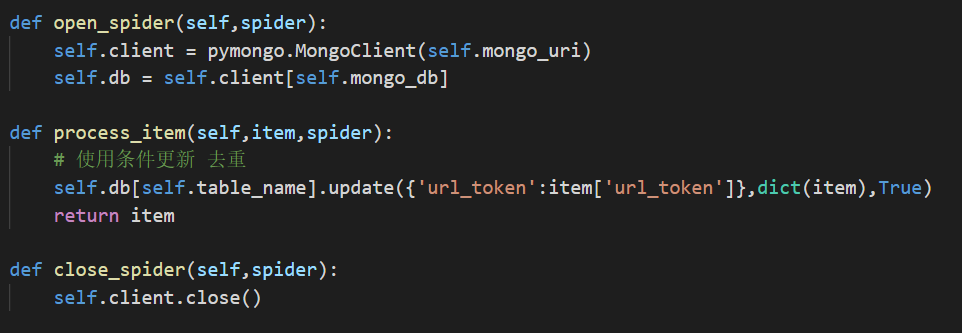
所以需要用到ItemPipeline + DownloaderMiddleware

在itempipline中需要做到存储item到mongodb中，并且对于数据进行清洗格式化，并且对于微博爬虫来说，需要对于生成的关注列表粉丝列表item对象进行单独处理，实现异步存储，以下是知乎中itempipline代码：

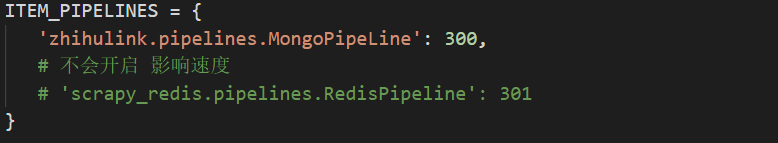
读取配置



过程处理+mongodb存储



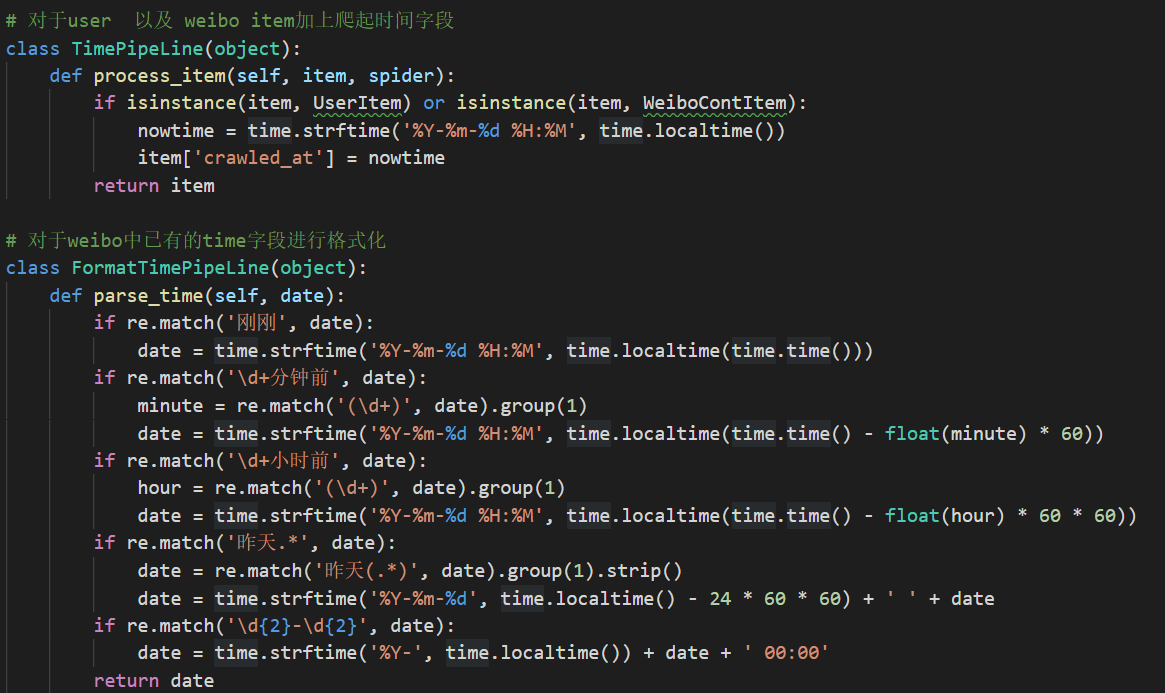
Settings.py中调用



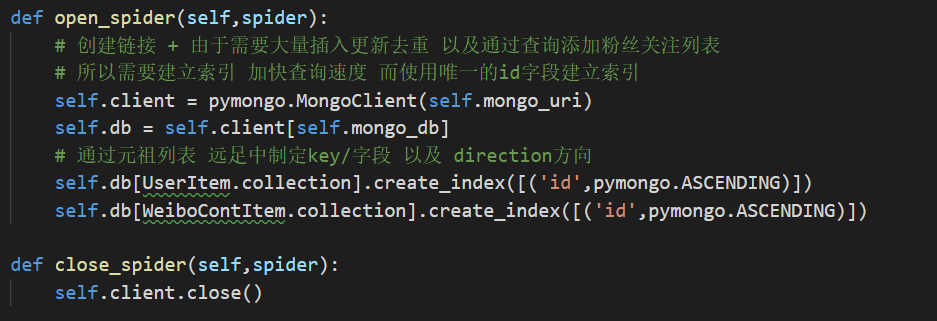
------------------------------------------------------------------------------------------

以下是微博中itempipeline代码

数据清洗时间处理



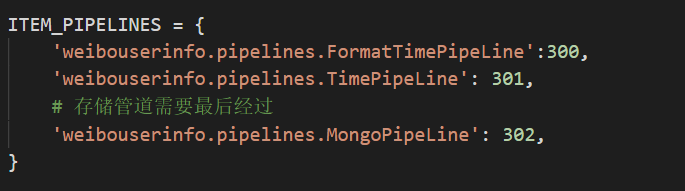
打开关闭mongodb链接



进行实际的mongodb存储---需要针对不同的item进行类别检查，从而选用合理的处理方式存储方式--------useritem/weibocontentitem--------------RelationItem

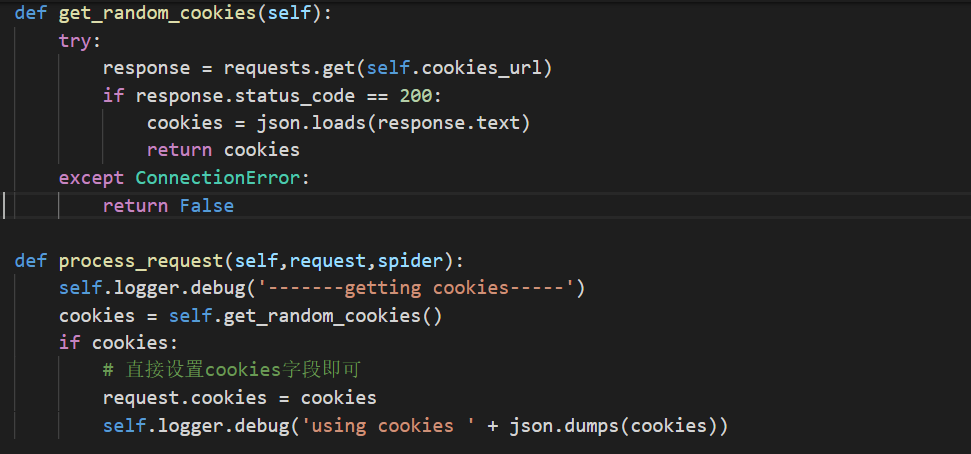


Settings.py中的通过配置调用



对于downloadmiddleware需要在爬取过程中检查响应状态从而实时通过flask接口获取代理/cookies从而切换cookies/ip伪装headers，实现批量爬去，应对微博知乎的强反爬虫机制

随机获取cookies并且直接每次请求更换cookies

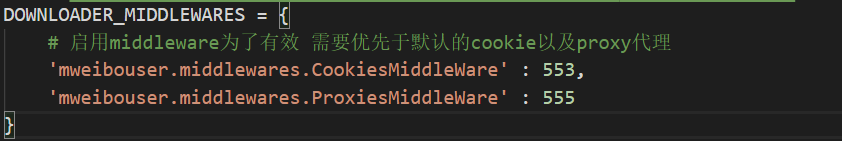


通过检查当前retry\_times字段值从而确定是否需要获取代理，更换ip

微博爬虫



在settings中进行配置性的调用



知乎爬虫类似，只是通过检查下载过程中exception实现使用代理切换ip



**3.4、模块2**

**代理池，cookies池模块**

由于知乎微博强反扒机制，通过购买微博账号搭建cookies池

从网络上获取免费代理ip构建代理池，实现ip,cookies切换从而有效爬取

Cookies池尚未完成，所以使用的是手动登录，实现cookies获取

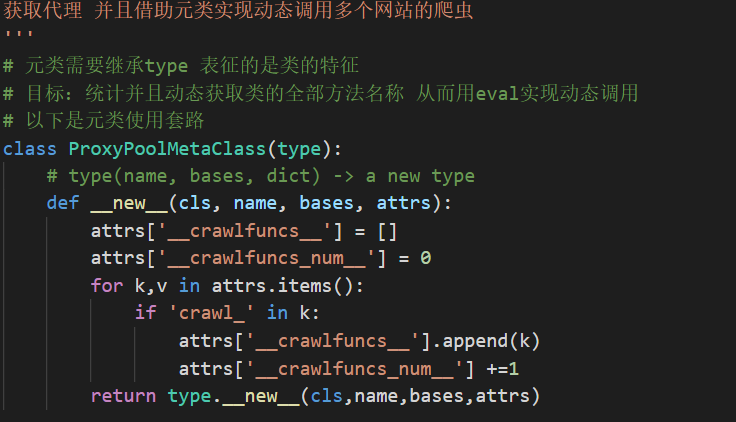
Ip/cookies pool二者逻辑相同，分为三个子成分，getter tester interface

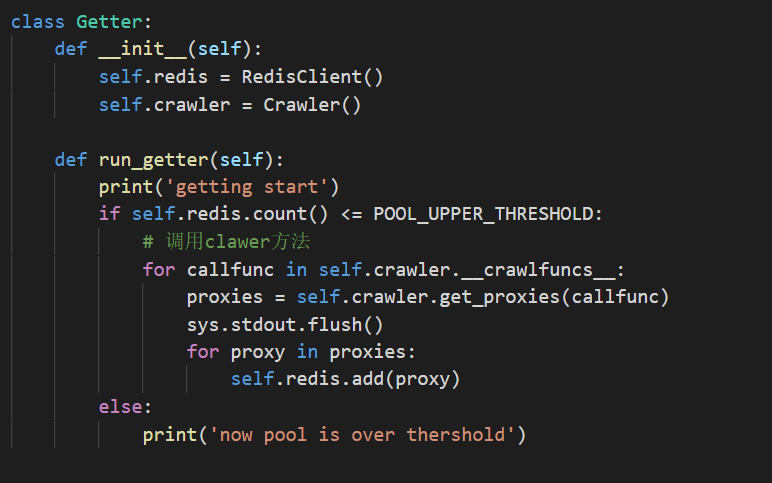
**[1] getter**

代理池通过爬虫获取免费代理网站代理

Cookies通过selenium使用redis中的账号进行模拟登陆，从而获取cookies

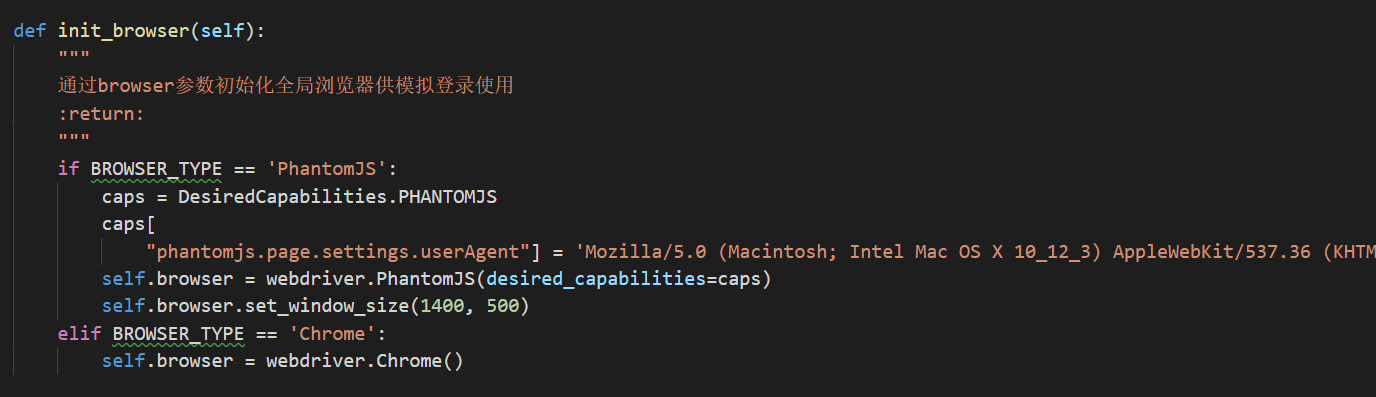
代理池getter





Cookies池getter

初试化selenium浏览器



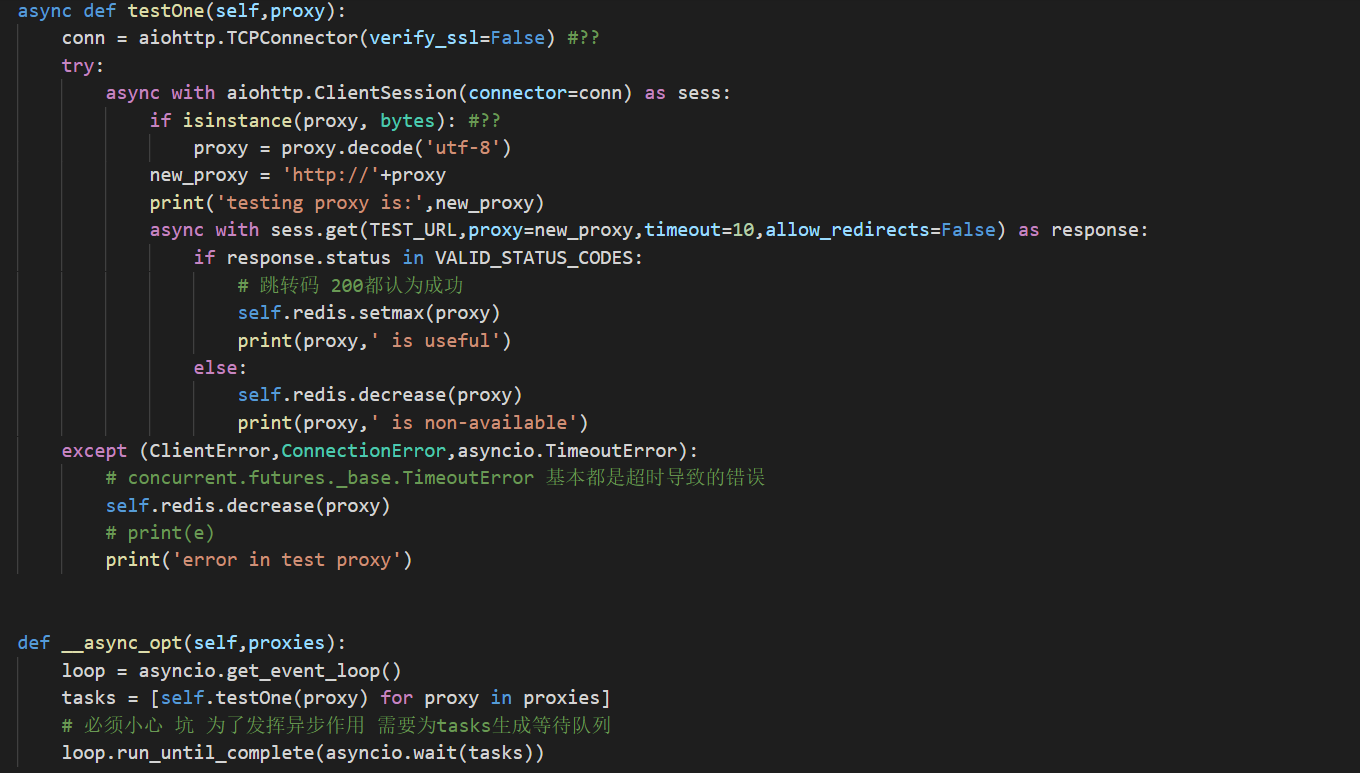
进行模拟登陆



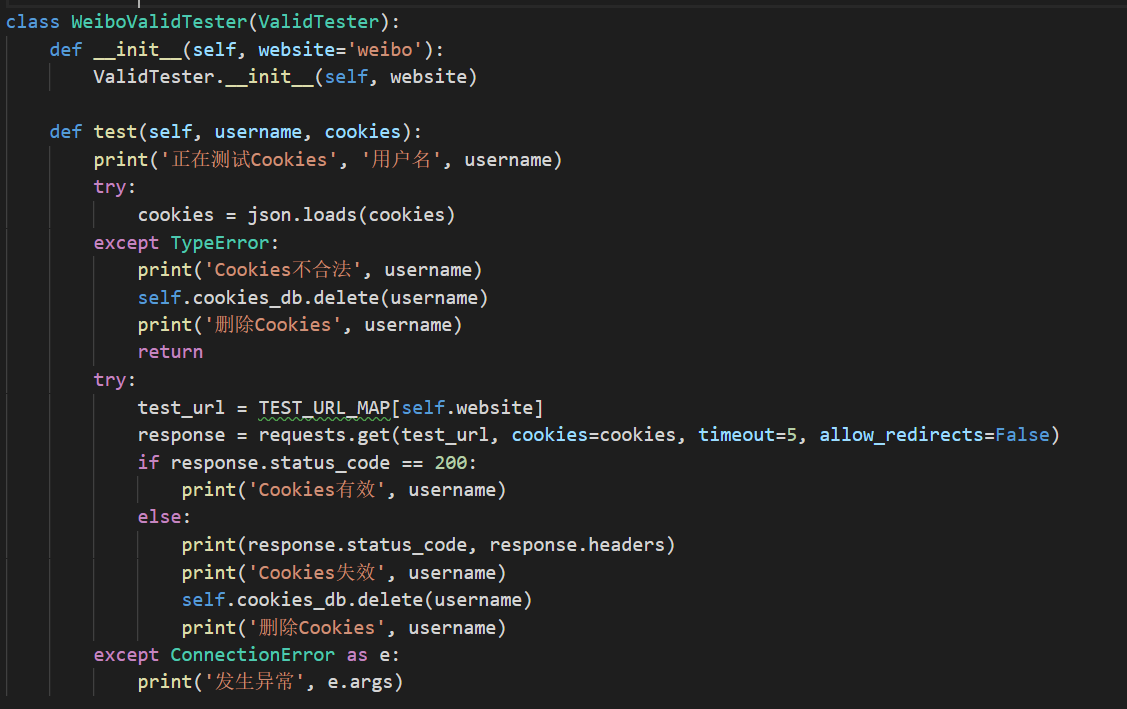
**[2]tester**

Proxy pool tester

使用异步方式进行网络请求，测试代理/cookies可用性，由于网络IO是主要性能障碍，所以异步aiohttp+asyncio有效提升了速度



Cookies pool tester



通过继承关系，使cookies不局限于一个微博网站，可以通过继承父类实现子类，边界扩展，但是代理池由于切换的是通用的IP所以不需要进行扩展的考虑

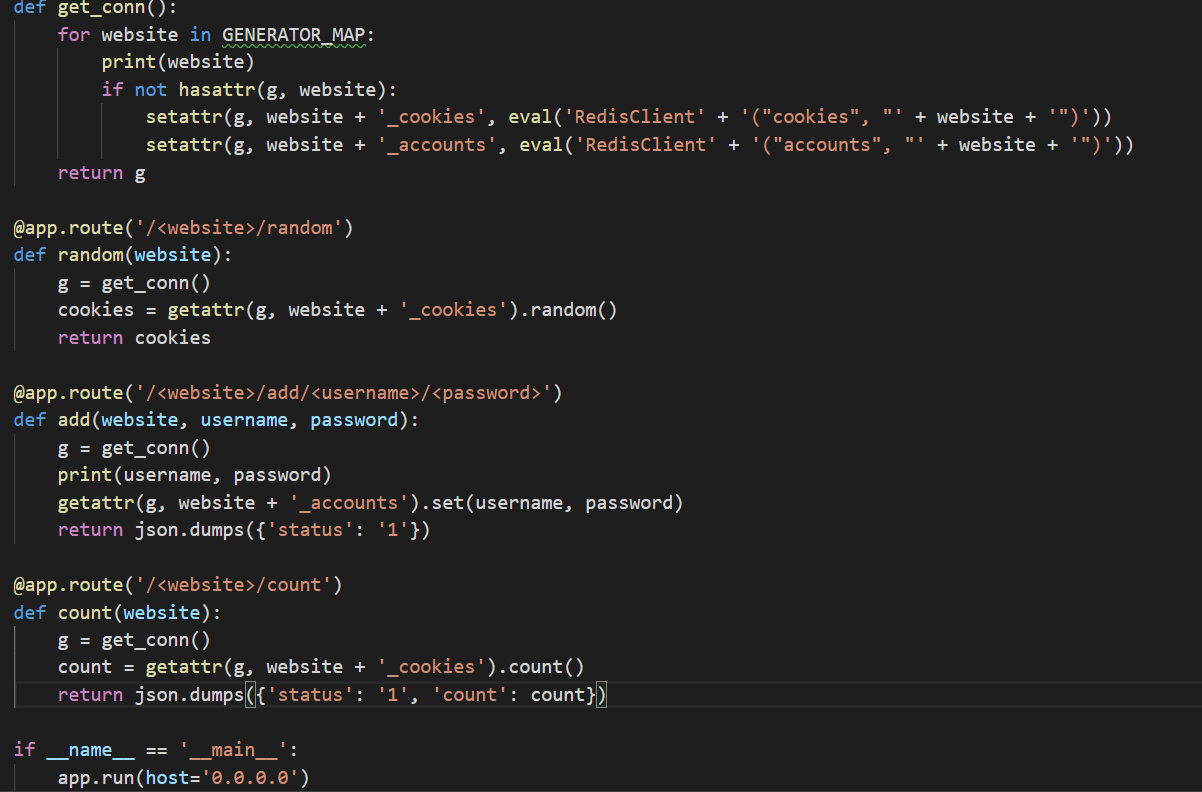
**[3]interface**

使用flask这个简便的web框架实现为redis中的proxy/cookies提供对外访问获取接口，而爬虫只需要通过访问接口拿到数据即刻获得proxy/cookies

代理池接口

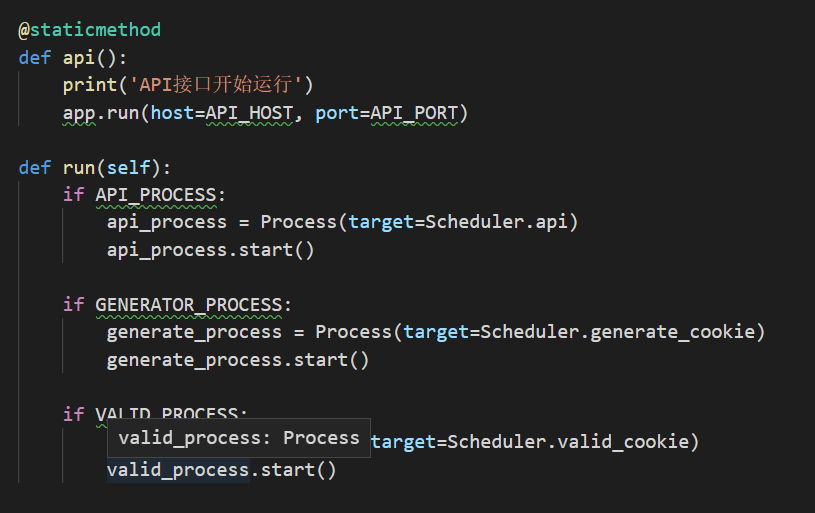


Cookies池接口



**最后通过调度器使用多进程multiprocessing，并且读取配置文件中的启动信息，从而进行统一调度**





**3.5、模块3**

**提速模块**

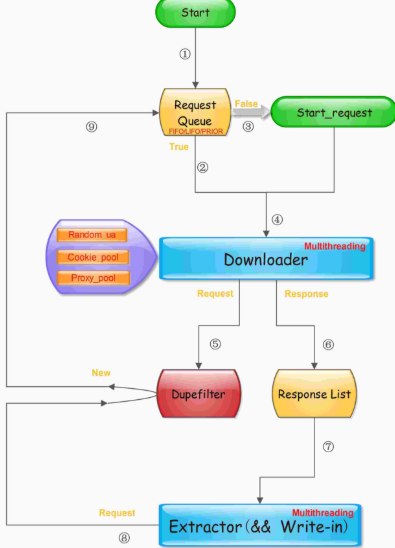
在此模块中，我们通过三方面对爬虫爬取速度进行提速：

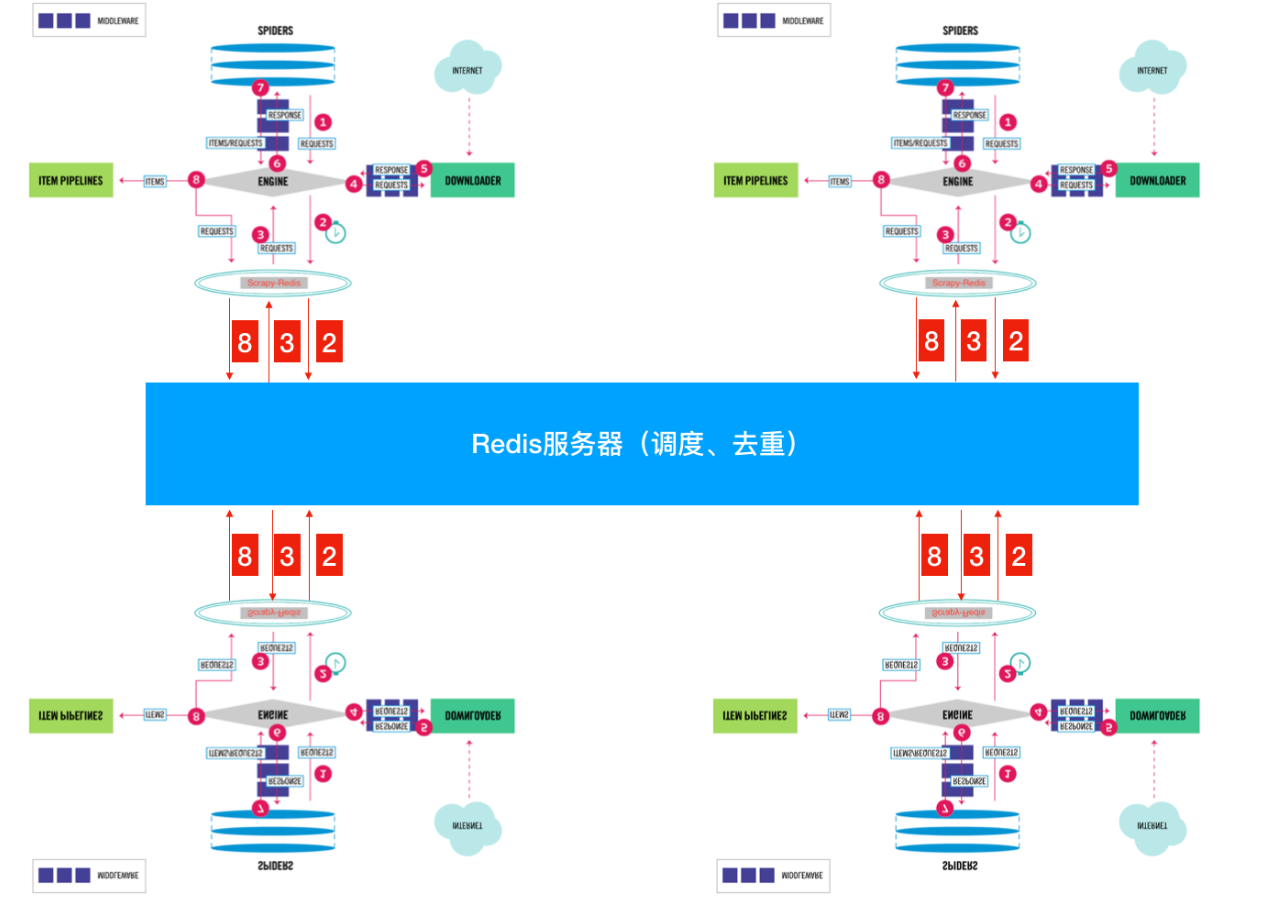
scrapy-redis分布式爬虫

multi-processing多进程

AIO异步IO网络请求

**[1]Scrapy-redis分布式爬虫:**

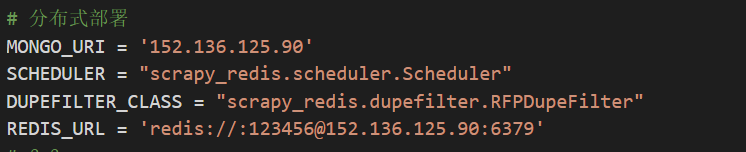




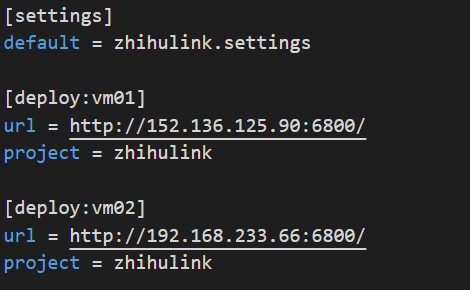
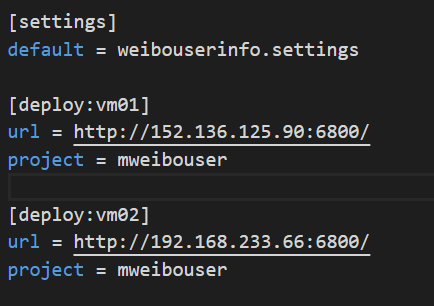
**原理图如上**

**以下介绍分为分布式代码配置部分+分布式部署部分**

**配置**



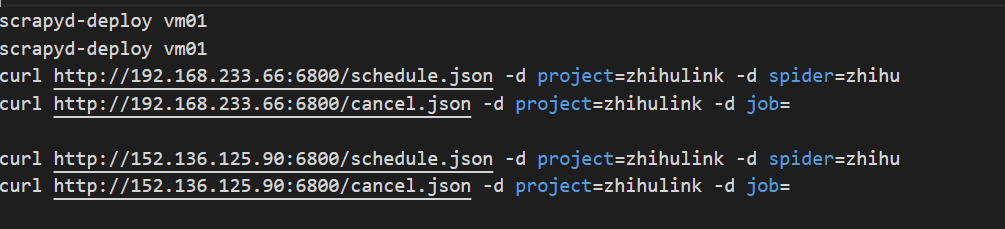
**部署**



分布式调度

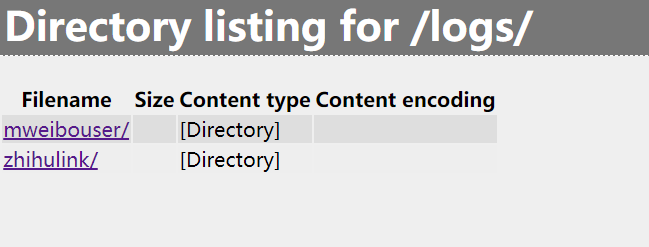
首先需要在目标主机上运行scrapyd并且监听来自外部的请求在6800端口

之后再本地主机上运行以下命令，即可实现从项目打包部署，到爬虫运行启动终止调度



并且可以通过浏览器访问相应分布式节点的scrapyd接口，查看jobs 以及 logs信息

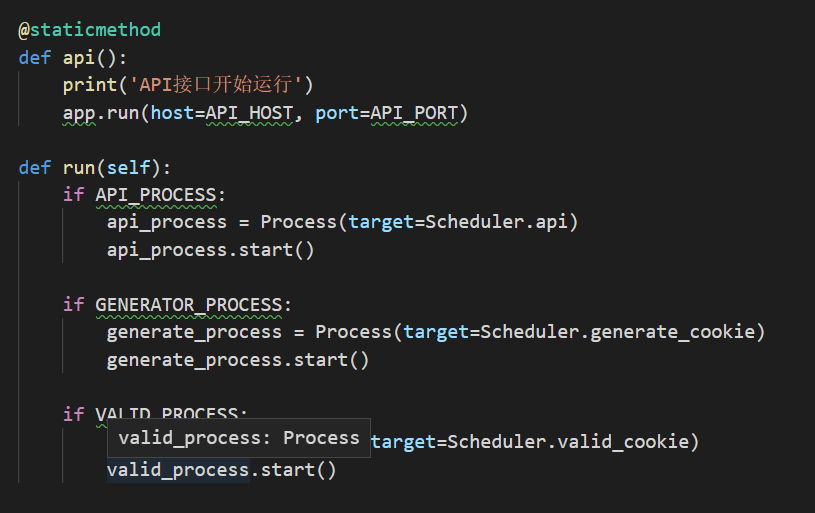




**[2]Multi-processing多进程实现pool调度：**

通过多进程实现多个模块同时开启进行调用，减少阻塞所等待的时间，实现同步运行

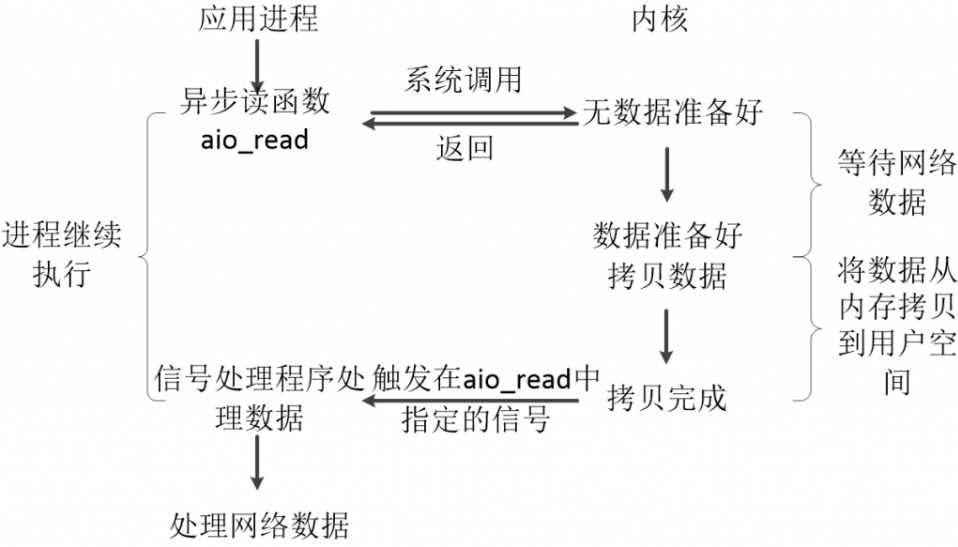
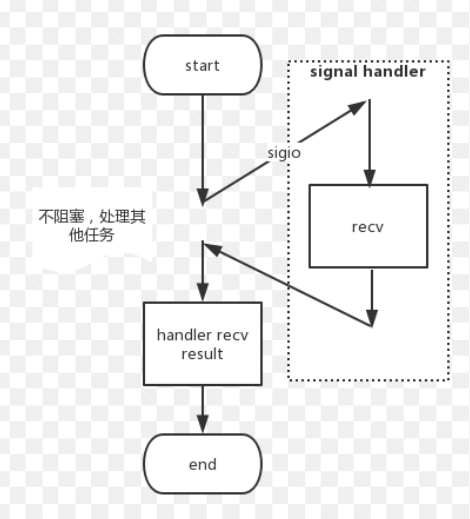




**[3]AIO异步io网络请求：**

**AIO原理**

在一个进程中为了执行多个 I/O 请求而对计算操作和 I/O 处理进行重叠处理的能力利用了处理速度与 I/O 速度之间的差异。当一个或多个 I/O 请求挂起时，CPU 可以执行其他任务；或者更为常见的是，在发起其他 I/O 的同时对已经完成的 I/O 进行操作。



**AIO实现**

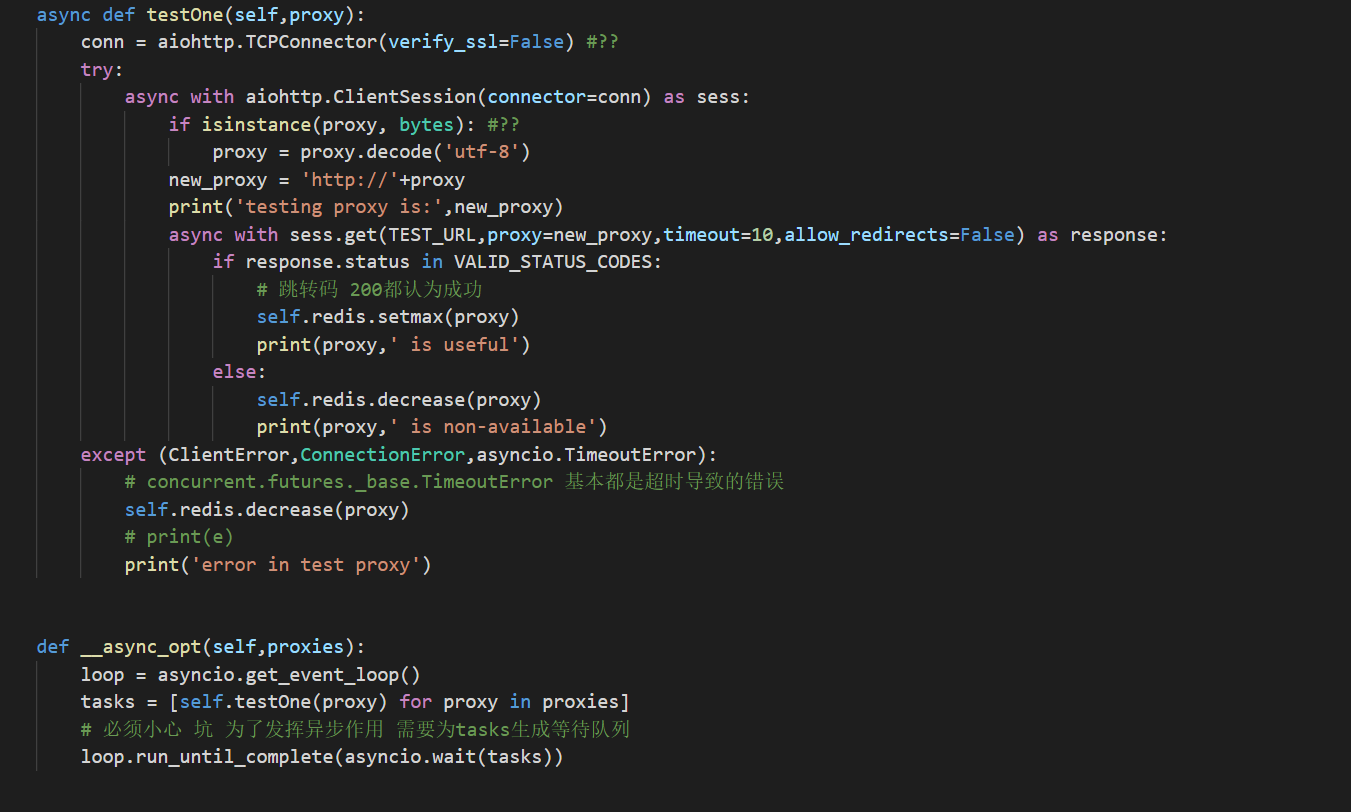
爬虫部分的AIO的通过scrapy自带的爬虫异步请求功能得以实现

scrapy基于异步框架twistted

**NOTE:分布式部署需要使用18.9.0而不能是19.2.0的twisted，否则无法分布式部署**

同时在代理池cookies池中使用异步请求aiohttp模块实现异步网络请求验证cookie以及ip有效性，从而缓解网络IO带来的性能瓶颈。





**3.6、模块4**

**数据存储及可视化模块**

**[1]数据存储：**

1. 分布式队列存储-redis：

redis是一个开源的、使用C语言编写的、支持网络交互的、可基于内存也可持久化的Key-Value数据库。Redis将数据存储到内存中，使得读写速度非常快。使用redis列表类型保存数据，队列采用左进右出的模式保证队列消息的顺序性。

1. 将爬取到的数据存储到MongoDB中：

MongoDB 是由C++语言编写的，是一个基于分布式文件存储的开源数据库系统。MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成



**[2]数据分析可视化：**

1. mongodb-charts可视化分析：mongodb-charts是在mongodb数据上构建可视化最快的方法。我们通过添加数据源、创建一个仪表板和创建图表三个步骤连接到mongodb图表服务器，从而实现数据的可视化
2. pandas+matplotlib对数据进行可视化简要分析：Pandas与Matplotlib配合绘制出折线图, 散点图, 饼图, 柱形图, 直方图等五大基本图形，对数据进行了可视化，从而方便我们的简单分析。（目前要根据进度来看，不一定会实现）

**3.7、模块5**

**项目管理及部署模块：**

[1]我们通过共享云笔记进行协同合作，推进实验的进行，有什么问题在云端进行讨论与解决，当然，实验课时间通过面对面交讨论问题。

[2]通过GIT多分布版本控制项目管理：从master->zhihu->weibo->v1->master

通过分支合并保证master分支可用性，而在从分支上进行拓展开发，当开发完成与master分支进行合并，模拟了产品更新升级的过程，而且必要时候可以实现版本回退，git分支管理的意义不言而喻

[3]将项目打包成docker-image提交到docker-hub以及阿里云提供的docker registry

利用容器技术实现项目部署，分别有scrapyd的docker-images，以及知乎爬虫，微博爬虫的docker-images

[4]分布式部署scrapyd+scrapy-client

通过多台机器爬取使用scrapyd来监听请求命令，进行对多个爬虫的上传、启动、暂停等调度管理。

# 3、功能测试

**3.1、模块1/功能1-----爬虫模块**

**3.1.1、测试结果**

以下是爬取到的数据





**3.1.2**、**结果分析**

爬取可以顺利进行，并且将数据结果保存到mongodb数据库中

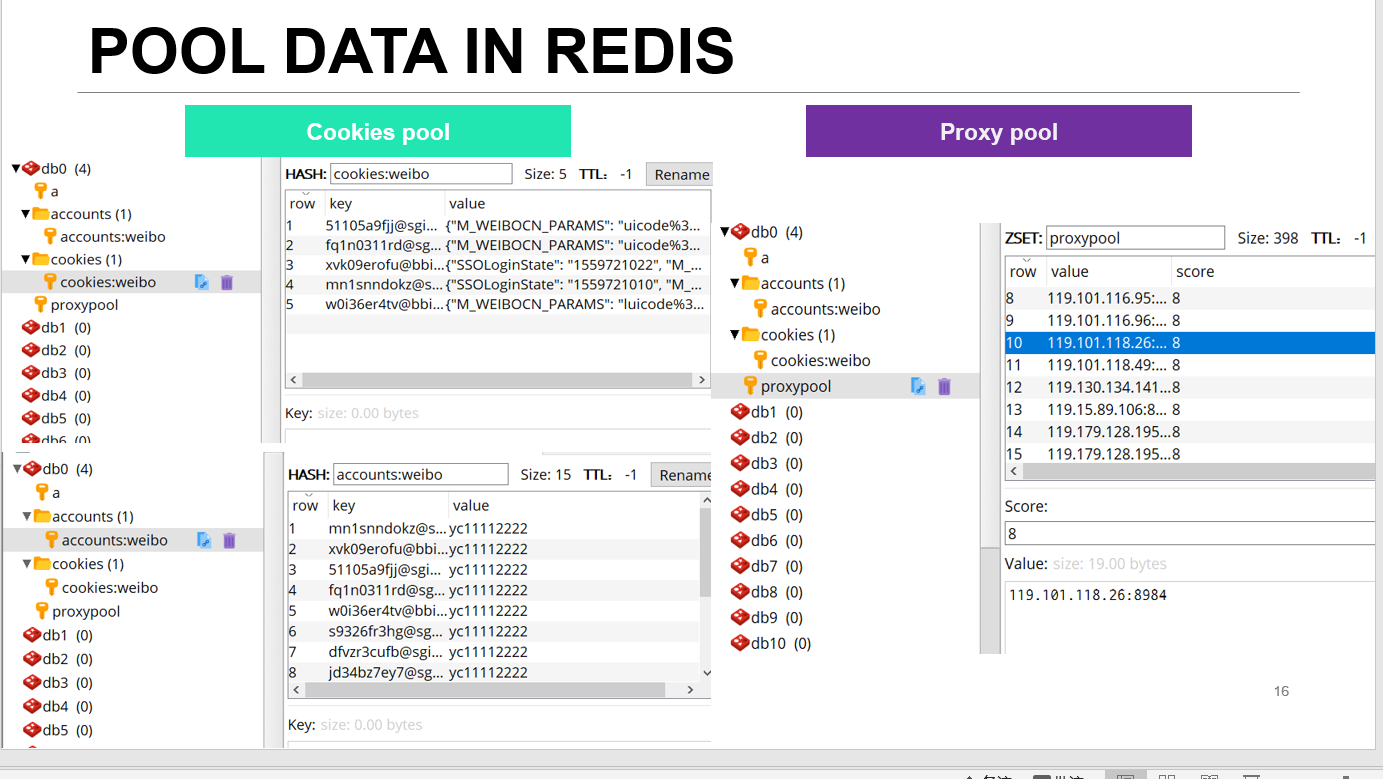
**3.2、模块2/功能2-----cookies/proxy pool模块**

**3.2.1测试结果**

以下是在cookies以及proxy pool模块中获取到的cookies以及proxy ip并存储到redis中的结果：

cookies池有两部分组成：账号，cookies

proxy池有：代理ip



**3.2.2结果分析**

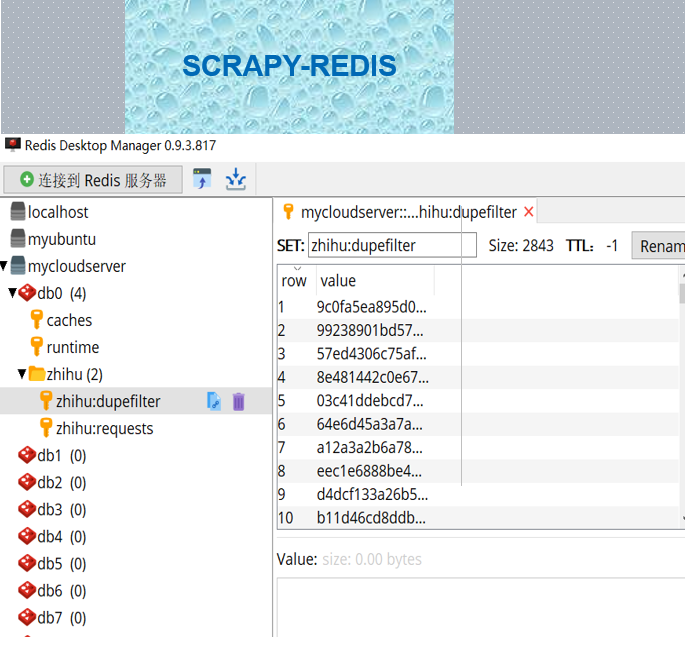
[1]代理池可以顺利从免费网站上抓取免费代理，进行测试评分，优先级排序保存，并且提供对外获取接口，功能基本实现，且性能由于使用AIO，较可观

[2]cookies池可以顺利通过购买的账号导入到redis中，并基于账号进行模拟登陆。从而获取cookies，并且测试可用性，保存到redis中，并对外提供获取接口，但是由于微博验证码机制变更，目前模拟登陆受到阻碍，有待改进。

**3.3、模块3/功能3-----提速模块--- Scrapyredis分布式**

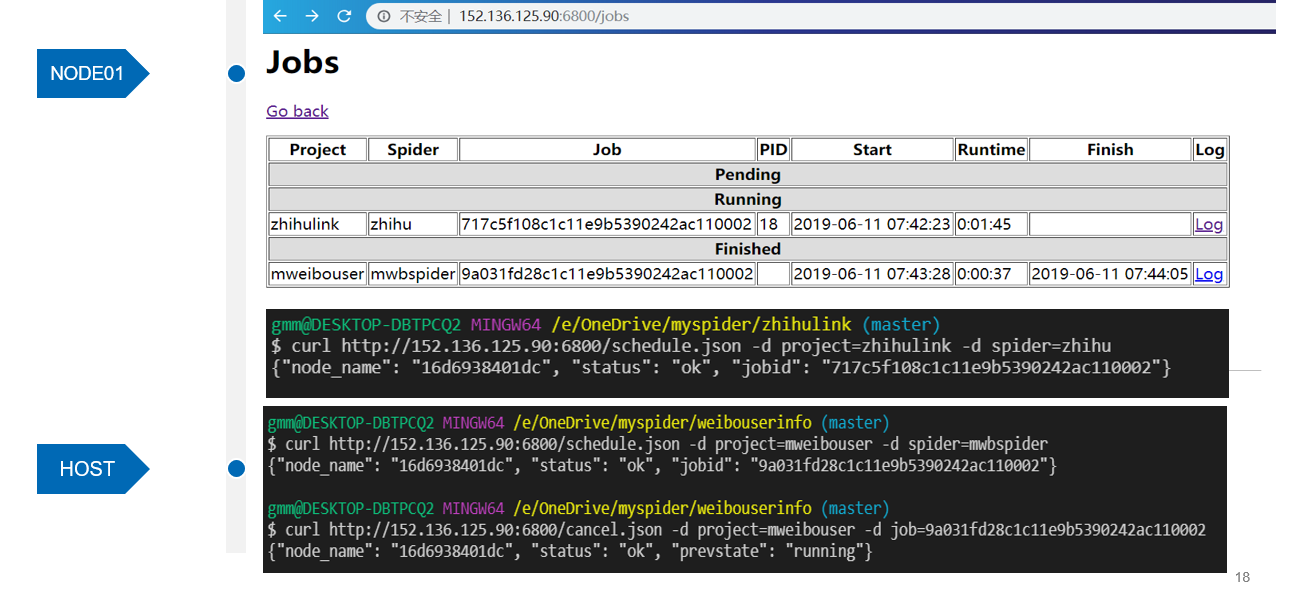
**3.3.1测试结果**

Redis中保存的分布式爬取请求队列以及指纹队列

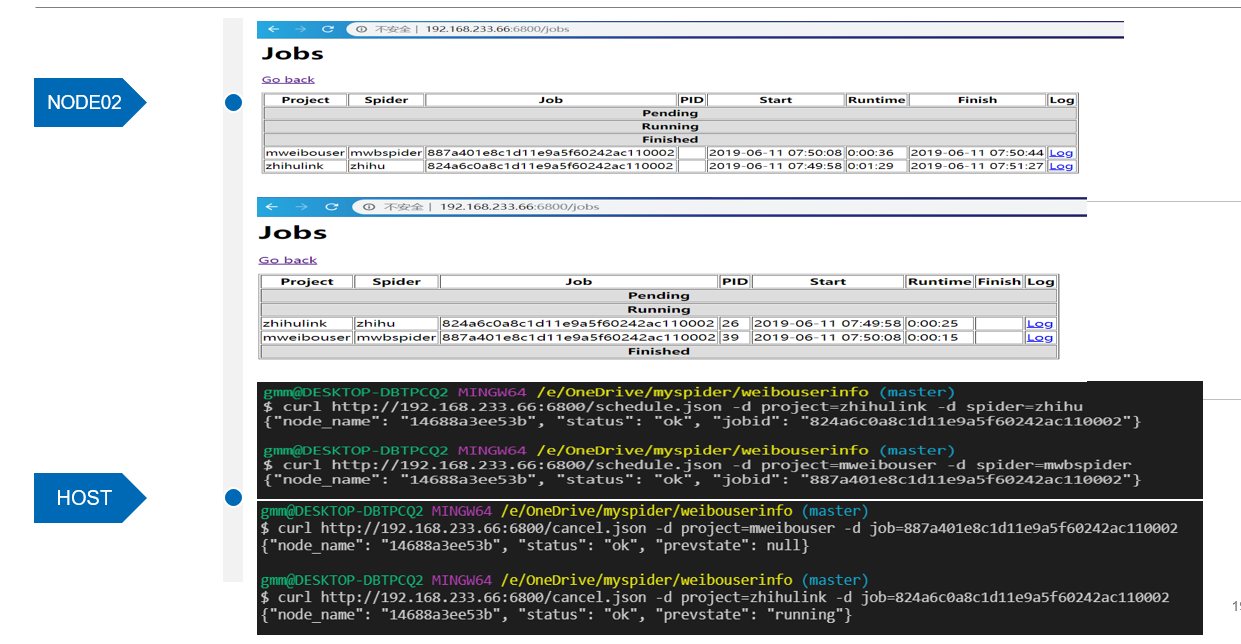


两台分布式主机在本地主机命令下运行情况监控

**Host1**

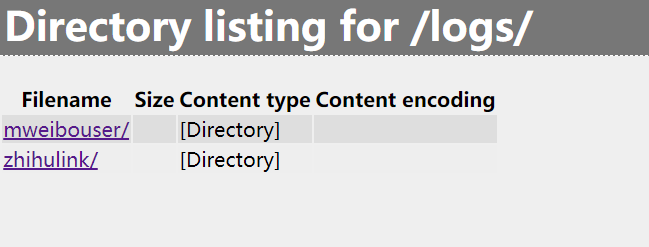


**Host2**



**通过浏览器查看相应的jobslogs进行监控**





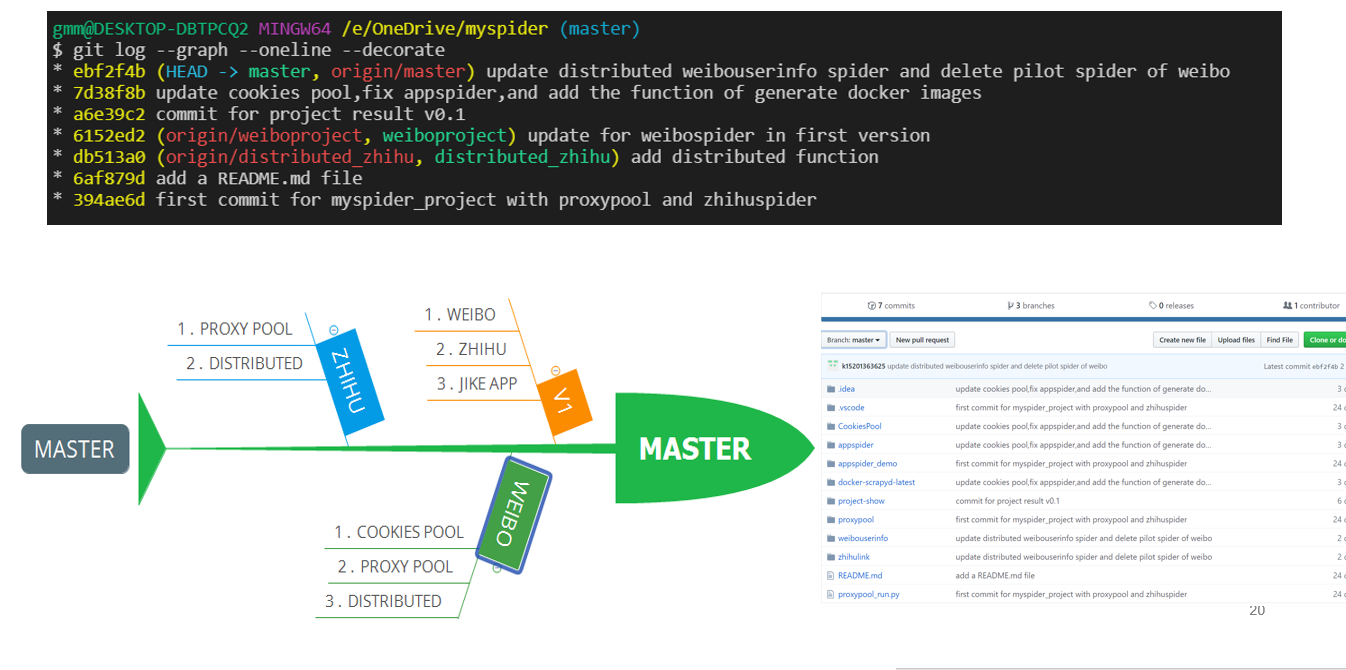
**[2]结果分析：**

基于scrapyd的服务实现的分布式部署已经基本完成，并未实现了基本的可控性，可以通过本地命令进行调度，本地浏览器访问相应的网络服务实现监控，并将爬取中心的请求指纹队列以及数据存储到了redis以及mongodb中，双节点分布式已经实现，通过购买云主机可以进一步扩展分布式规模，但是代理池cookies池还没有很好的运用到分布式爬取中，有待改进。

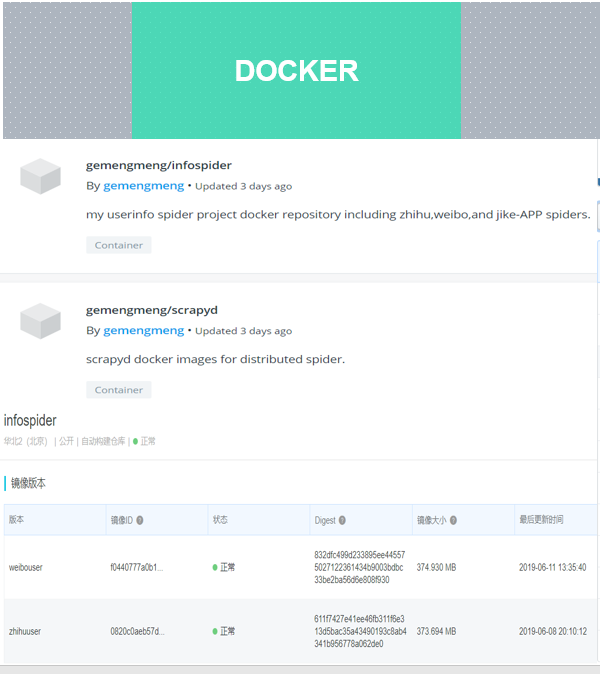
**3.4、模块4/功能4----项目分支管理以及部署模块**

**3.4.1测试结果**

**Github进行分支管理**



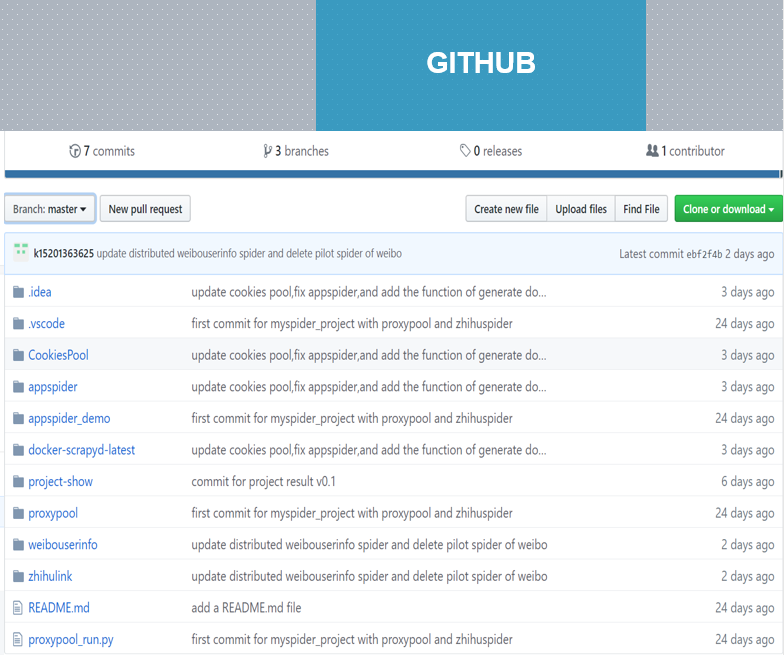
**发布到Docker registry的docker image**



**云主机拉取获得daocker镜像并且进行运行**



**发布到github上的项目代码**



**3.4.2结果分析**

通过云主机上的docker服务已经正常运行，以及分布式任务的正常执行， 可以确定docker实现的项目分布式部署目前为止没有可以稳定运行，此部分基本实现，但在代理池cookies池的打包部署上需要完善，并且目前的docker images原生部署方式修改配置文件不太方便，可以通过整合k8s，以及docker-compoes实现项目更便捷的部署。

Github方面分支管控，版本控制很好的辅助项目推进，master始终保障高可用性，在其他分支进行功能拓展，最终合并到master分支从而更新，并且项目发布到github上便于项目的发展完善以及公开。

# 4、性能测试

4.1、精度分析

[1]爬虫有效爬取到了网站/APP的response中的全部有效信息，并对于数据进行一定的格式化处理存储到了mongodb数据库，并且对于空字段保留原有字段值置为空。

[2]代理池内置的爬虫有效抓取免费代理并且存储到redis中，并可通过测试打分后有优先顺序的返回可用性较高的代理。

[3]Cookies池由于验证码问题暂无法自动登录，需要人手工破解验证码，但可以有效的保存获取的cookies保存到redis中，并且进行测试可用性，提供可用的cookies给爬虫。

[4]分布式中，中心redis队列精确地收集到了从节点生成的request，并且将指纹以及请求无重复的保存到数据库中。

综上，数据库中存储的信息均做到不重复不遗漏，精度满足当前需求。

4.2、时间特性分析

**爬虫部分：**

由于引入scrapy自带的异步特性，并且在爬虫逻辑中有效检查重复爬取问题，爬虫本身速度较快。之后加入了scrapy-redis分布式实现了由于硬件限制导致瓶颈的突破，速度进一步提升。

**Cookies以及proxy池部分**：

在获取方面，代理池由于使用爬虫实现获取代理，速度较快，但免费代理可用性较差，cookies池模拟登陆使用selenium本身速度受到限制，不过cookies有较长有效期，得以弥补.

在可用性测试方面，均使用异步AIOHTTP请求方式（python3.6才开始支持）进行测试可用性，性能较高，而proxy需要额外进行的打分机制也可较快完成，与网络IO相比，不会造成速度瓶颈。

在提供给爬虫的web接口中使用了flask，对于当前小规模爬虫不会造成问题，但是对于大规模情况下，还需要nginx保证高可用负载均衡，tarnado提供异步高速支持等方式进一步提升性能。

**Docker部分：**

当前制作的docker镜像，可以实现scrapyd以及zhihu，weibo单机爬虫的边界部署，再部署效率上较高，而且虚拟化技术保证了容器运行的高效性。

4.3、灵活性分析

爬虫以及分布式部分由于使用docker部署，系统的灵活性得到了充分的保证。但是在APP爬虫方面由于目前对于环境要求较高，所以灵活性较差。

在cookies以及proxy池部分，由于配置选项较多，并且有在运行时修改配置的需求，有限于当前的docker部署技术未能实现有效部署，仅能手工使其在本地/远程主机上运行， 无法做到即插即用，不过目前有github拉取运行勉强够用。

# 5、测试结论和建议

5.1 测试结论

**由测试结果得出基本完成情况：**



当前项目初步已经完成，但还有很多需要优化改进的方面，也有很多可以拓展研发的方面，具体的方向在展示ppt以及之前的文档中都有所提及，所以不再赘述。但目前项目初步目标已经基本达成，从开发到发布再到部署基本流程皆有，功能基本实现，性能方面也在多次版本迭代中不断改进，仍有的不足也已认识到，并有了初步的改进思路与优化方向。

5.2心得体会

从本次程序设计实验中爬虫知识得到了巩固提升， get到很多对抗反扒的技术，在搭建pool方面这种系统内部功能组件上学到很多，在系统架构组成设计方面有了一些思考，在APP爬虫方面初入门路，在分布式原理，配置，部署方面都有所掌握，在docker技术运用上实现从0->0.1的突破，在文档书写项目ppt制作过程中也有了不少心得及收货，最重要的是学到了程序设计有关的很多知识。

感谢这次课程的机会，也感谢老师助教的辛苦付出，谢谢。

# 6、项目地址：

Github地址：

<https://github.com/k15201363625/myspider>

相应的docker镜像的地址：

Docker hub：

Docker pull gemengmeng/scrapyd:latest

Docker pull gemengmeng/ infospider:weibouser

Docker pull gemengmeng/ infospider:zhihuuser

阿里Docker registry：

Docker pull registry.cn-beijing.aliyuncs.com/kgmm/scrapyd

Docker pull registry.cn-beijing.aliyuncs.com/kgmm/infospider:weibouser

Docker pull registry.cn-beijing.aliyuncs.com/kgmm/infospider:zhihuuser

Docker镜像中代理池cookies池部分由于部署不便暂未实现docker部署

**NOTE:程序包的完整抄送(包括ppt 文档)也已通过邮件打包发送**