**系统概要设计**

--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：2019.06.11

目录

[1、系统概述 1](#_Toc11233868)

[2、总体结构设计 3](#_Toc11233869)

[3、模块设计 5](#_Toc11233870)

[4、数据库与数据结构设计 11](#_Toc11233871)

[5、接口设计 13](#_Toc11233872)

[6、其他设计 13](#_Toc11233873)

# 系统概述

**1.1、系统简介**

基本情况：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。

背景：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。

**1.2、系统目标**

[1] 基于关注列表粉丝列表，爬取知乎微博两大社交网站+即刻APP用户数据信息

[2] 使用headers伪装，cookies，proxy pool攻克网站反扒机制

[3] 通过多进程，异步，分布式多手段实现高速爬取

[4] 对用户用户数据信息可视化分析

[5] 对项目进行高效部署

本概要设计意在让用户和系统开发者双方对整个系统的运行环境、功能、性能要求的初始规定有一个共同理解，为爬虫系统的开发工作奠定一定的基础；同时也为详细设计提供统一的参照标准，其中包括系统的内外部接口、系统架构以及其他各种主要问题的解决方案。在此文档确立后，所有有关本系统的详细设计必须遵照此文档的相关标准和约束来进行。另外，此文档也作为对详细设计文档进行同行评审所依照的标准之一。

**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）+ 云存储/中心具有公网IP的存储设备(redis+mongodb)

**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

（必选）有一台主机必须要有redis（分布式调度主机）

（三选一）其余主机需要有容器技术实现redis mongodb环境的安装/使用具有公网IP的装有两大数据库的可访问存储主机/云存储系统

**编程平台（开发工具）：**

Pycharm Vscode Git docker

**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 git docker命令

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

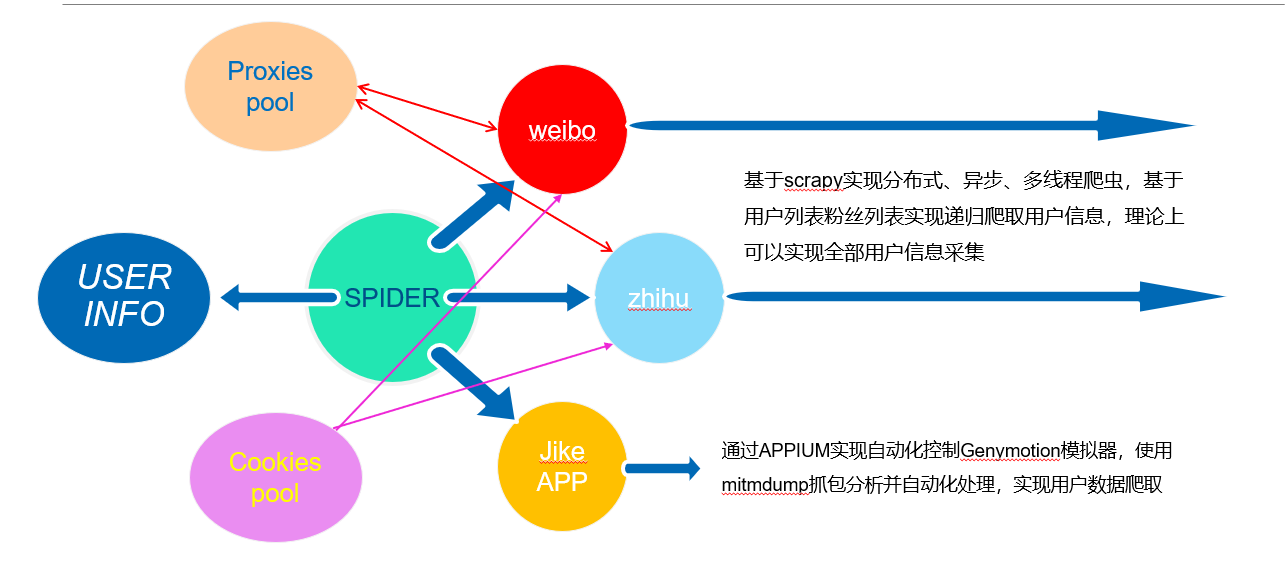
Andriod\_SDK 主要借助其中adb工具进行移动端控制

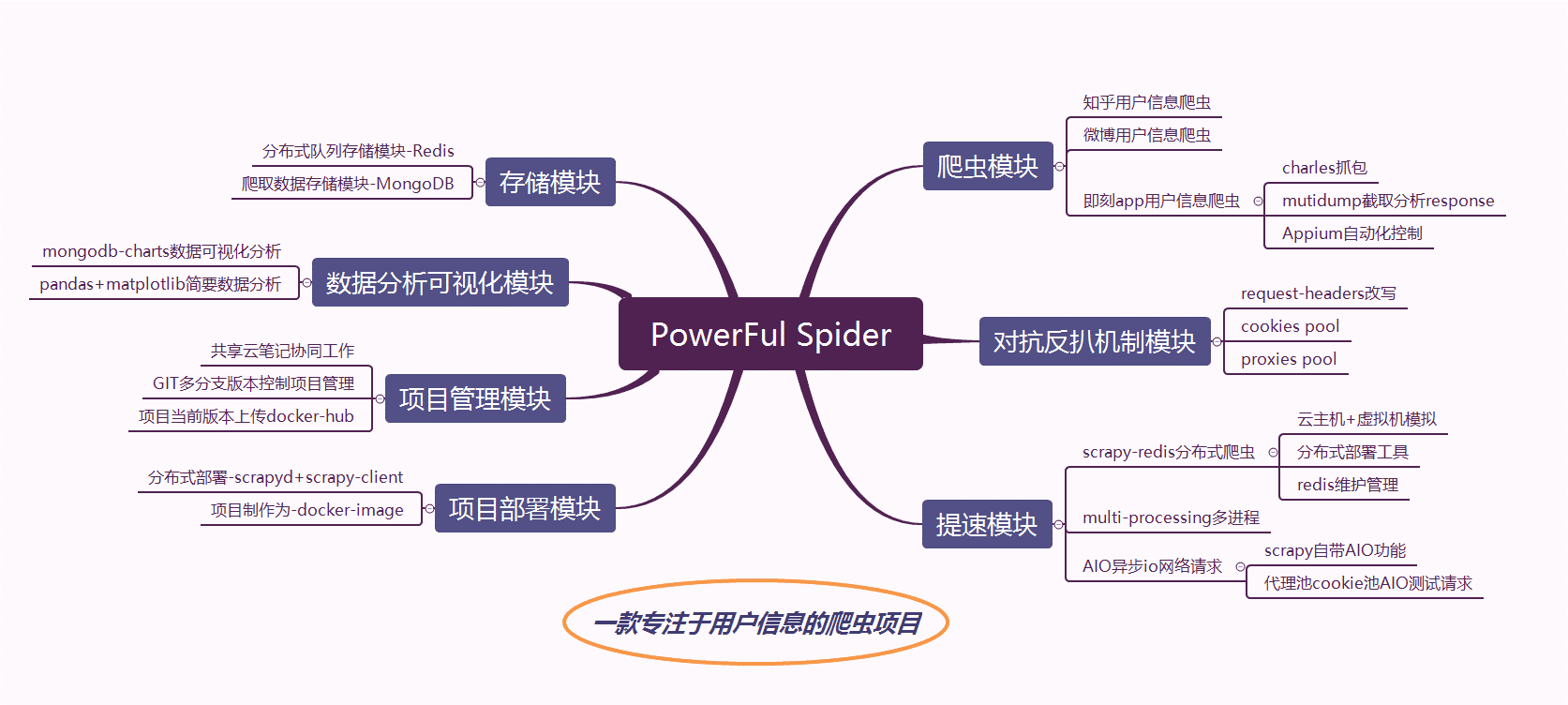
VMware VirtualBox PC端的虚拟机

# 2、总体结构设计

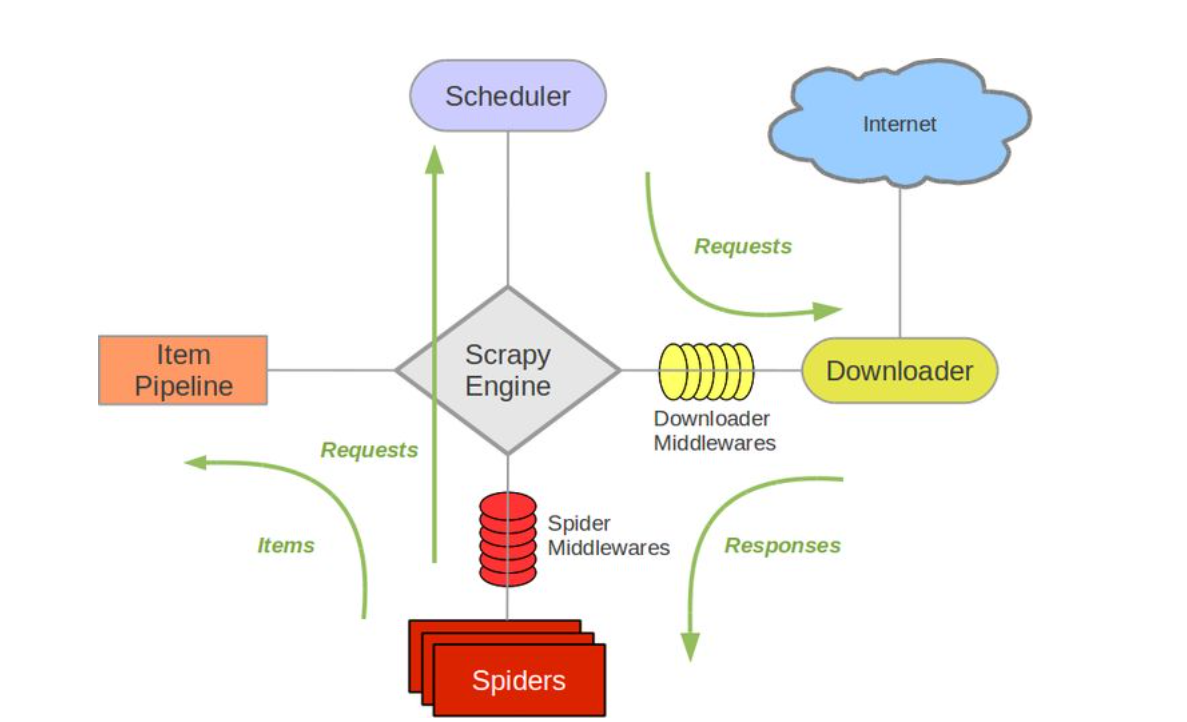
**2.1、软件结构**

**软件系统结构**



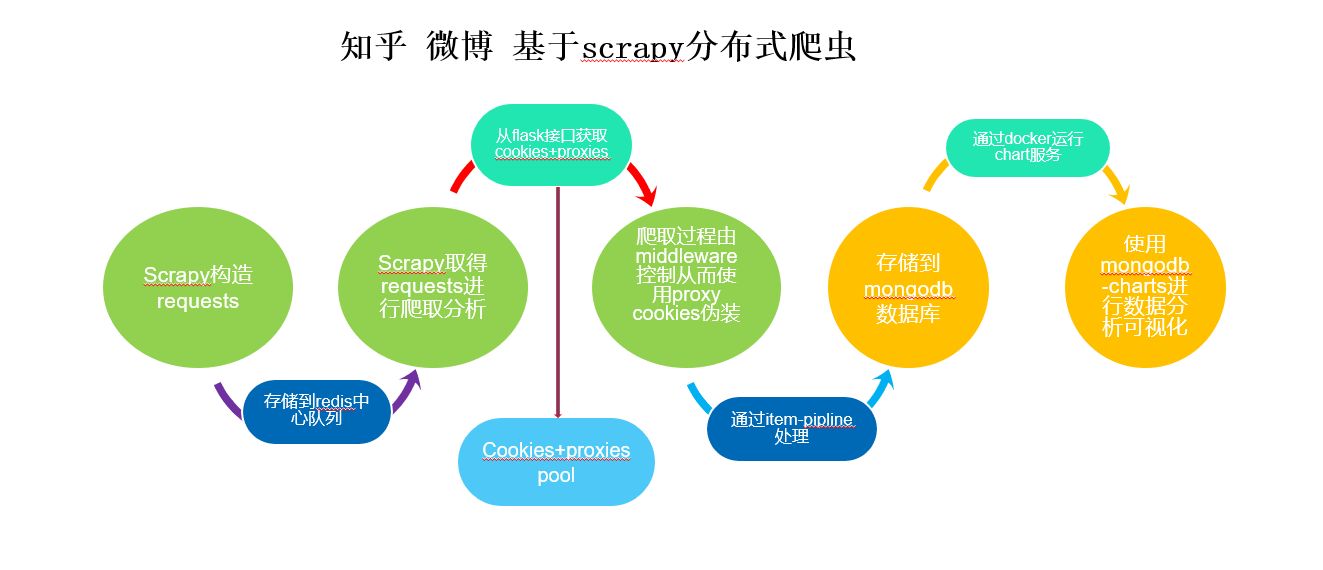


**爬虫scrapy内部结构**

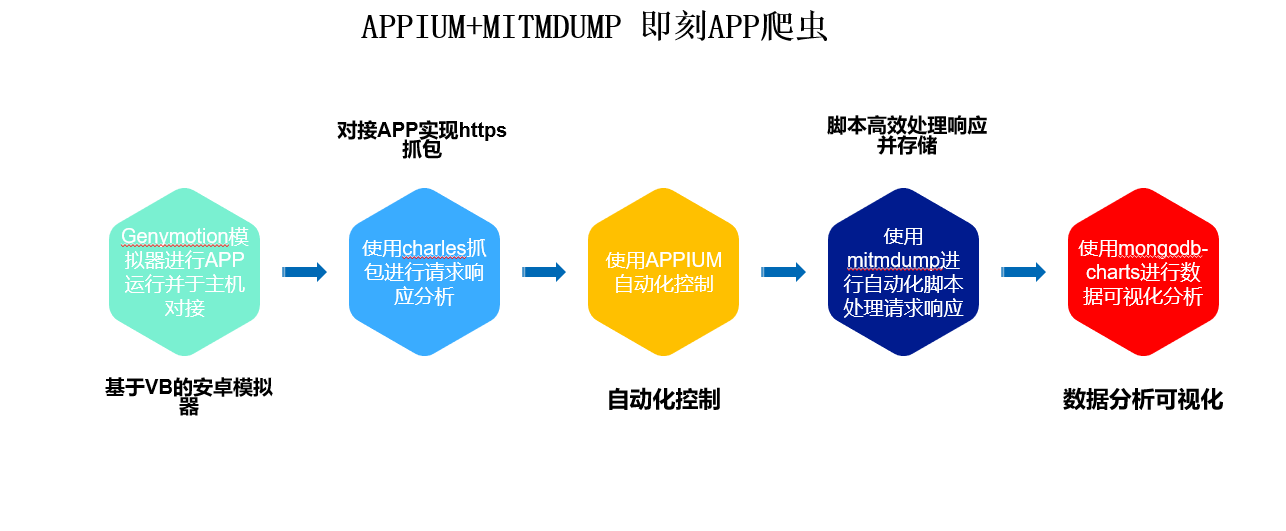


**2.2、设计思想**

**知乎微博爬虫**



**APP爬虫**



我们所做的项目在爬虫方面可以实现三部分内容：知乎用户信息爬虫、微博用户信息爬虫以及即刻app用户信息爬虫。

而在爬取微博、知乎平台时会遇到反爬机制，我们通过request-headers改写请求头以及proxypool，cookiespool实现ip以及cookies切换,从而对抗知乎微博反爬机制。

在爬取过程中由于网络IO以及CPU限制，导致爬取速度受到限制，于是我们采用scrapy-redis实现分布式爬虫、multi-processing多进程和AIO异步IO网络请求来进行提速。

在数据存储方面，我们用分布式队列存储，将scrapy生成的请求存储到中心主机的redis队列中，每个从节点主机获取中心主机的request队列进行爬取，并更新该队列，实现分布式调度，并将爬取到的数据存储到数据库MongoDB中。

在数据可视化分析方面，用mongodb-charts以及pandas+matplotlib（待完成）对数据进行可视化方便简要分析。

在项目管理部分，用共享云笔记协同工作、GIT多分布版本控制项目管理，然后将项目上传至docker-hub。

在项目部署方面，采用scrapyd+scrapy-client进行分布式部署并且通过docker容器技术，制作docker-image实现项目发布，便捷部署。

# 3、模块设计

**3.0、模块0 Scrapy框架基础：**

主要由五大组件组成，它们分别是调度器(Scheduler)、下载器(Downloader)、爬虫（Spider）和实体管道(Item Pipeline)、Scrapy引擎(Scrapy Engine)。下面我们分别介绍各个组件的作用。

(1)、调度器(Scheduler):

调度器，说白了把它假设成为一个URL（抓取网页的网址或者说是链接）的优先队列，由它来决定下一个要抓取的网址是 什么，同时去除重复的网址（不做无用功）。用户可以自己的需求定制调度器。

(2)、下载器(Downloader):

下载器，是所有组件中负担最大的，它用于高速地下载网络上的资源。Scrapy的下载器代码不会太复杂，但效率高，主要的原因是Scrapy下载器是建立在twisted这个高效的异步模型上的(其实整个框架都在建立在这个模型上的)。

(3)、 爬虫（Spider）:

用户定制自己的爬虫，用于从特定的网页中提取自己需要的信息，即所谓的实体(Item)。用户也可以从中提取出链接,让Scrapy继续抓取下一个页面。

(4)、 实体管道(Item Pipeline):

实体管道，用于处理爬虫提取的实体。主要的功能是持久化实体、验证实体的有效性、清除不需要的信息。

(5)、Scrapy引擎(Scrapy Engine):

Scrapy引擎是整个框架的核心.它用来控制调试器、下载器、爬虫。实际上，引擎相当于计算机的CPU,它控制着整个流程。

**3.1、模块1 爬虫模块**

3.1.1、功能描述

三个爬虫，分别是：

基于scrapy的知乎+微博用户信息爬虫

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

3.1.2、数据结构描述

用户信息爬虫套路相似，基于一个用户的粉丝关注列表实现递归爬去，本项目的知乎微博爬虫都是采用队列维护生成的request从而顺序爬取，而request的生成基于当前用户的关注粉丝列表，所以整体上呈现广度优先搜索----由于基于队列维护请求并且进行爬取；APP爬虫由于需要自动化操作所以不便与实现递归爬去，所以采用生成初始用户队列基于这些用户的粉丝关注列表进行用户信息获取，相当于简单的队列遍历。

**3.2、模块2 代理池，cookies池模块**

3.2.1 、功能描述

由于知乎微博强反扒机制，通过构建购买大量微博账号搭建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，主要是用于对抗反扒机制，实现大量长期有效爬取

3.2.2接口描述

Proxypool以及cookiespool通过Flask这个简便的web框架，提供爬虫爬取过程中需要的cookie以及proxy的获取接口，爬虫只需要对接口进行请求之后对响应进行简单的分析就可以得到ip/cookeis

3.2.3数据结构描述

**Pool数据结构**

在proxypool cookiespool部分使用redis中的sortedset以及基于key-value的hashset从而实现去重存储并对于代理ip进行打分排序确定获取优先级。

**3.3、模块3 提速模块**

3.3.1、功能描述

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

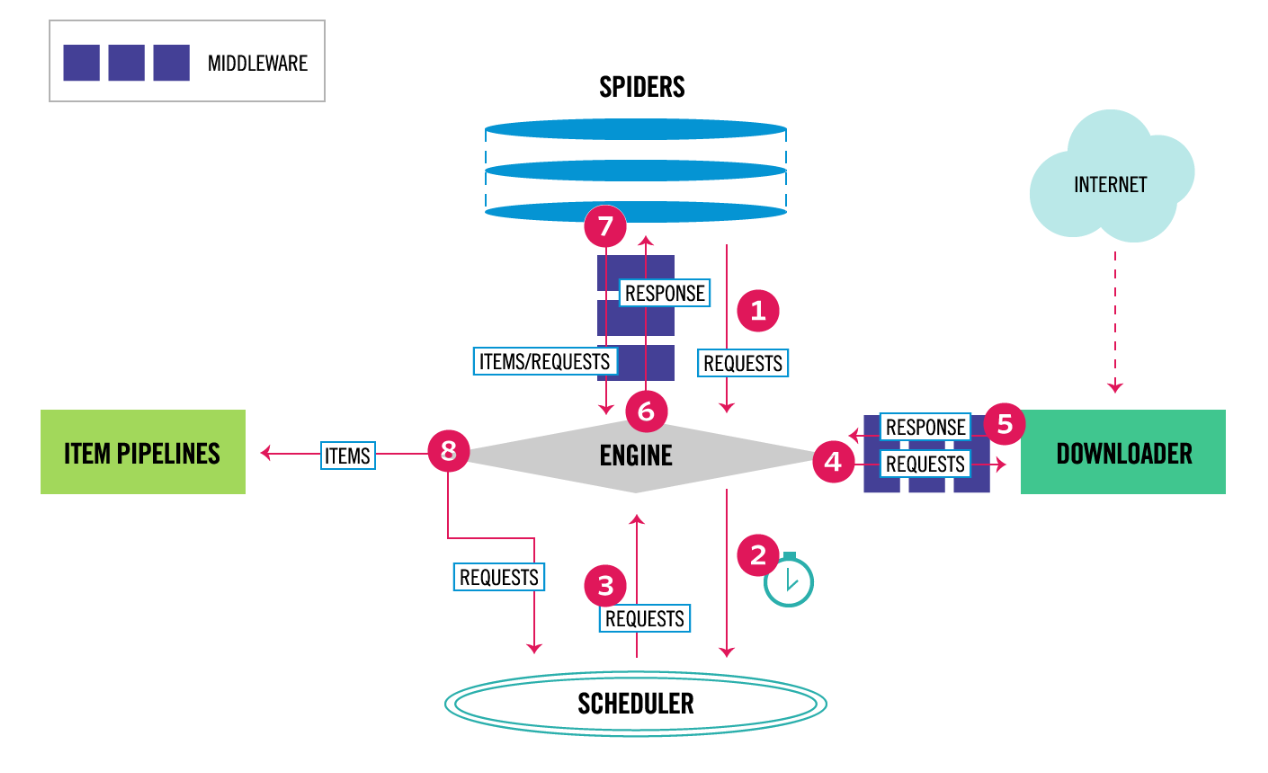
multi-processing多进程

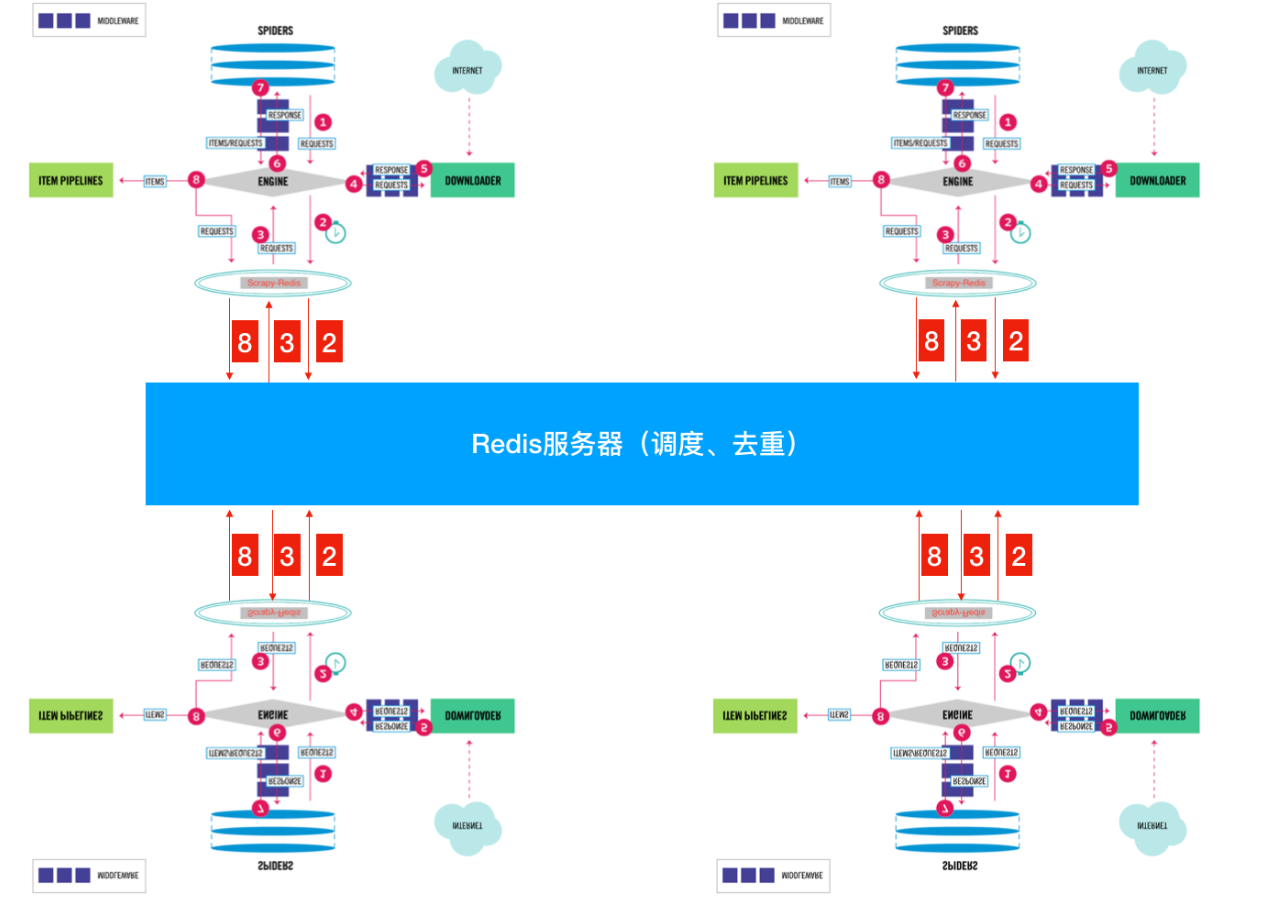
AIO异步IO网络请求。

从而实现爬虫的高速爬取，pool的告诉获取测试以及数据提供

**Scrapy-redis分布式爬虫:**

将scrapy使用的queue数据结构替换成中心redis数据库中的请求队列，将要爬取的requests存放到这个redis队列中，实现多个爬虫去一个数据库里获得请求进行爬取，并将生成的请求通过指纹进行辨识存储到数据库中来更新请求队列，并保证在没有请求的时候使用scrapy本身的请求获取机制获取请求从而防止分布式爬虫因为中心队列为空而导致的停止/中断。





**Scrapy-redis工作原理**：假设有100台机器可以用，我们把这 100 台中的 99 台运算能力较小的机器叫作 slave，另外一台较大的机器叫作 master。如果我们能把任务队列 queue 放到这台 master 机器上，所有的 slave 都可以通过网络跟 master 联通，每当一个 slave 完成一个请求，就从 master 获取一个新的请求。而每次 slave 新抓到一个网页，就把这个网页上所有的链接送到 master 的 queue 里去。

其中的分布式体现在多台机器上的 spider 同时爬取，并且这种分布式是通过 scrapy\_redis 实现的。Redis 中存储了工程的 request，stats 信息，能够对各个机器上的爬虫实现集中管理，这样可以解决爬虫的性能瓶颈，利用 Redis 的高效和易于扩展能够轻松实现高效率下载：当 Redis 存储或者访问速度遇到瓶颈时，可以通过增大 Redis 集群数和爬虫集群数量改善。

本质上说，就是大家（所有机器，所有爬虫）把拿到的东西（url，request）放在一起（redis queue）去调度。

**Multi-processing多进程：**

用于cookies/proxy pool中，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

**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实现细节设计**

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；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。

**3.4、模块4 数据存储及可视化模块**

3.4.1、模块功能

**[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）

3.4.2、接口描述

通过数据库接口可以进行数据的查询获取更新

通过可是化接口可以对数据特征以及内在蕴含的信息做初步的了解与挖掘

3.4.3数据结构描述

**数据库存储数据结构**

数据库（mongodb redis）内部所使用的高效数据存储结构比如B,B-,B+树等等不赘述，nosql中比如mongodb的keyvalue存储方式很高效，并且mongodb本身也是类似json的形式存储数据可读性解析性强，与网络请求返回的数据极其相似，无需过多的转换便可以存储，并且符合面对对象的逻辑，嵌套的kv结构使数据意义明确且编程控制方便；redis则是内存数据库，具有丰富的数据存储结构， 常用的sorted set可以实现去重存储以及以及有带优先级查询等丰富功能。

Redis依赖于链表 字典 跳跃表等数据结构dict Sds ziplist quicklist skiplist

Mongodb依赖于**B-树实现索引**

**3.5、模块5 项目管理及部署模块：**

3.5.1、功能描述

[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。

3.5.2、接口描述

通过docker container由docker image打包的服务即即可

通过主机向分布式节点上运行的scrapyd进程发送请求从而实现分布式任务调度调控

# 4、数据库与数据结构设计

**4.1、 数据库及数据表**



**Mongodb:**

它是一个内存数据库，数据都是放在内存里面的。

对数据的操作大部分都在内存中，但 MongoDB 并不是单纯的内存数据库。

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。

在高负载的情况下，增加更多的节点，可以保证服务器性能。

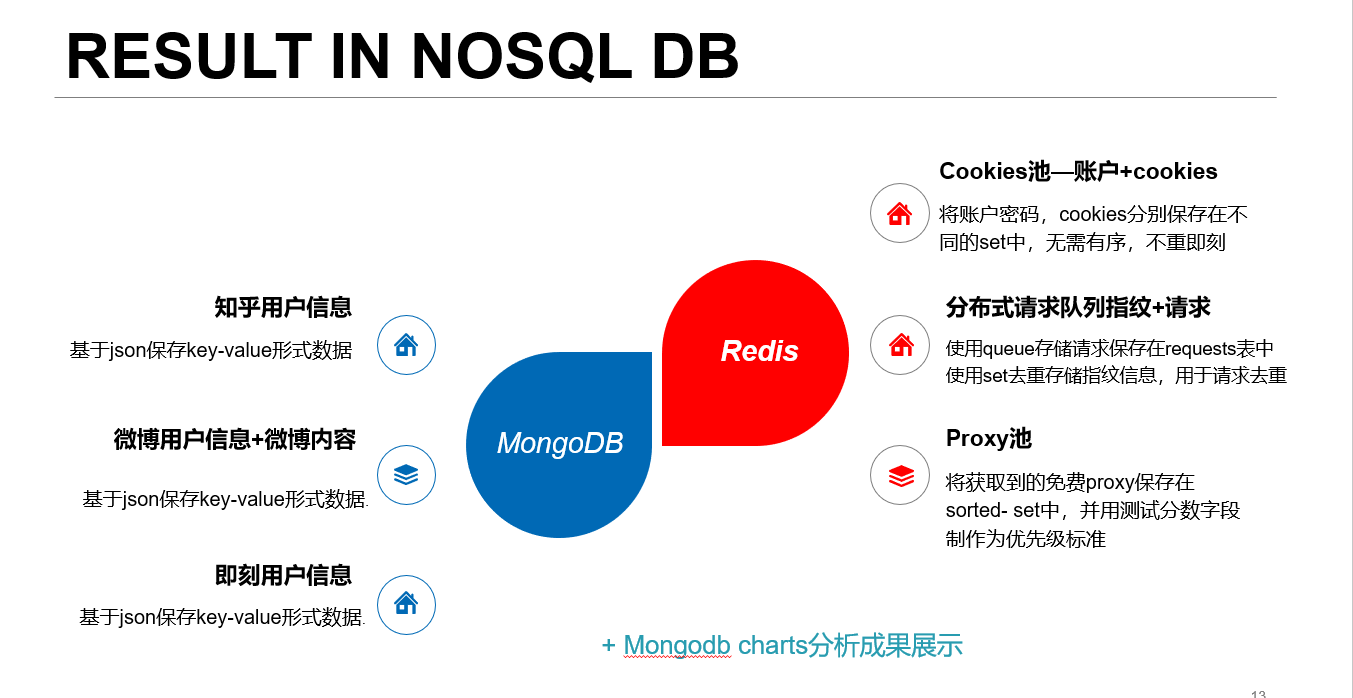
MongoDB 旨在为 WEB 应使用提供可扩展的高性能数据存储处理方案。 **Redis:**

是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。目前由VMware主持开发工作。

redis是一个key-value[存储系统](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%B3%BB%E7%BB%9F)。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list([链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8))、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B" \t "_blank)都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

基于以上数据库的特性

从而有了我们的基本数据库及表结构



**4.2、 数据结构设计**

**获取并管理url方面**

在爬虫程序中, 用到了广度优先搜索(BFS)算法. 这个算法用到的数据结构就是队列.

**去重方面**：

在爬虫程序中, 为了不重复爬那些已经爬过的网站, 我们需要把爬过的页面的url放进集合中, 在每一次要爬某一个url之前, 先看看集合里面是否已经存在. 如果已经存在, 我们就跳过这个url; 如果不存在, 我们先把url放入集合中, 然后再去爬这个页面. python提供了set这种数据结构. set是一种无序的, 不包含重复元素的结构. 一般用来测试是否已经包含了某元素, 或者用来对众多元素们去重，当然这里使用的是redis的去重。

数据去重同url去重，借助redis/mongodb的update而不是单纯的添加实现去重，内部实现也是基于set等数据结构，set的实现可以使二叉排序树等等

**爬虫逻辑数据结构**

用户信息爬虫套路相似，基于一个用户的粉丝关注列表实现递归爬去，本项目的知乎微博爬虫都是采用队列维护生成的request从而顺序爬取，而request的生成基于当前用户的关注粉丝列表，所以整体上呈现广度优先搜索----由于基于队列维护请求并且进行爬取；APP爬虫由于需要自动化操作所以不便与实现递归爬去，所以采用生成初始用户队列基于这些用户的粉丝关注列表进行用户信息获取，相当于简单的队列遍历。

**Pool数据结构**

在proxypool cookiespool部分使用维护的redis中的sortedset以及基于key-value的hashset从而实现去重存储。

**数据库存储数据结构**

数据库（mongodb redis）内部所使用的高效数据存储结构比如B,B-,B+树等等不赘述，nosql中比如mongodb的keyvalue存储方式很高效，并且mongodb本身也是类似json的形式存储数据可读性解析性强，redis则是内存数据库，具有丰富的数据存储结构， 常用的sorted set可以实现去重存储以及以及有带优先级查询等丰富功能。

**响应内容分析及数据存储数据结构**

本项目中由于现有网络请求返回的response.text大多为json字符串，所以json数据结构的分析提取是本项目中最关键最基础的部分，所谓json即嵌套的keyvalue字典结构，通过遍历key可以迅速得到value部分进行数据提取，并且支持空数据字段，使用python提供的json模块可以与txt格式文本迅速转换实现分析及存储。而在数据存储中由于mongodb支持keyvalue形式存储，所以将提取到的信息转换为json直接实现数据库存储，方便快捷。

# 5、接口设计

**5.1、外部接口**

用户只需要用安装有docker的主机即可通过pull docker-image的方式获取项目运行环境，并通过架设多台主机以及一台装有redis的主机实现分布式部署，但是过程中需要对得到的docker镜像中的项目代码中的配置文件进行简要修改，从而适应于分布式以及远程连接环境。

**5.2、内部接口**

[1]Proxypool以及cookiespool通过Flask提供爬虫爬取过程中需要的cookie以及proxy的获取接口

[2]数据库接口 通过数据库连接可以对数据库进行读取查询以及写入更新

**5.3、调用方式**

[1]进入项目目录中，之后单独进入cookies池以及proxy池中，修改config.py中设置的开关接口，打开api接口，运行池，之后通过在爬虫通过访问flask提供的网络借口比如random获取网页内容—proxyip/cookies。

后续可以将这些调用过程单独做成项目中的一个启动模块实现快速调度。

[2]数据库接口通过pymongo/Redis等python模块获取数据库连接并且进行数据存取操作，还可以通过数据库客户端实现数据库数据可视化分析。

# 6、其他设计

**6.1当时的（项目进行过程中的）：**

为了方便分布式管理爬虫节点主机，借助开端实现scrapyd+scrapyd-client实现快速部署项目并且发布到节点主机

对于验证码的识别以及将数据存储下来后的呈现。

**6.2项目完成后的：**

[1]尚需解决优化的问题：

**数据分析可视化**

爬取更多以足够分析的数据，使用pandas,matplotlb,pandas,charts进行更加灵活全面的数据分析可视化并以web页面方式展示

**Cookies Pool完善**

补充验证码破解模块，可以通过调用远程验证码破解接口破解困难的验证码，从而完善cookies池自动获取cookies功能.

**Pool接口优化**

通过将flask网络借口改变成tarnado等异步式式web框架提高获取速度

使用nginx反向代理应对大批量分布式访问

**部署方面**

通过熟悉docker的高级使用，进而将cookies,proxy pool部署成为docker镜像， 进而整合到知乎微博的爬虫docker镜像中，并尝试实现APP爬虫的Docker部署

**分布式部署完善**

通过熟悉docker的高级使用，进而将cookies,proxy pool部署成为docker镜像， 进而整合到知乎微博的爬虫docker镜像中，并尝试实现APP爬虫的Docker部署

[2]项目未来发展方向：

**爬取方向扩展**

通过facebook+twitter提供的接口实现国外社交网站更全面用户信息的爬取

**ADSL Proxy pool**

通过拨号主机ADSL拨号申请动态IP地址，并且搭建动态DNS解析服务器，实现代理池V2.0,从而解决付费IP成本，以及免费IP不可用性等问题

**验证码破解**

当前仅通过tensorflow实现了对于字母数字这种传统验证码的CNN识别，需要拓展其他破解方法，e.g.训练RNN/CNN模型，算法破解，而不只是上传人工服务平台

**自动化爬取**

向着不局限于站点，而是通用性，自动化提取数据信息的智能化爬虫方向发展