**《基于网络爬虫的数据分析系统的实现》**

**中文摘要:** 随着以互联网为代表的大数据信息时代的到来信息数量急速膨胀，网上信息用海量形容，使得专门负责信息采集的网络爬虫技术面临了一个巨大的挑战。网络爬虫以其灵活的可定制性和信息采集速度和规模，满足了人日益益增长的对信息获取的手段，用以满足对大数据的数据挖掘与数据分析提供数据获取、搜索引擎的需求。

针对上述问题，本文以Python2.7和Scrapy环境为基础，基于Scrapy-Redis分布式框架的网络爬虫，并以“新浪微博”Web为爬取的对象，在学习并分析当前爬虫技术的原理、核心模块以及运行流程的基础上，探索性地实现一个基于Scrap-Redis框架,多线程，高并发，强鲁棒性的网络爬虫，完成数据抓取等目标，然后对爬到的数据进行简单的数据分析。

首先，本文简明给出了爬虫技术的原理和发展现状，介绍爬虫工程中一些关键技术，并着重介绍了在本研究中有深刻影响的Cookie池和user-agent欺骗来突破站点限制、信息过滤、搜索策略。

其次，通过使用基于Python语言开发的Scrapy开源爬虫框架来进行爬虫开发，指出了Redis的内存数据库做去重、任务调度、提高爬取速度、支持“断点继爬”，同时指出了MongoDB为代表的NoSQL数据库在元数据存储中的巨大作用。详细介绍了Scrapy-Redis开发爬虫的流程和实现细节。

再次，讨论了对于爬虫设计领域的关键问题，本文实现的自定义爬虫的解决方法。与对抗微博反爬技术，破解验证码，URL去重防止环路的出现，和多线程并发的问题，则采用并分析Scrapy-Redis自带的解决方案。

最后对爬虫爬取到的数据做简单的数据分析得出一系列小结论。

**关键词: 新浪微博, Scrapy-Redis ,Python, Web, 爬虫, 数据分析**

**Abstract**

With the rapid expansion of the number of incoming information in the era of big data information represented by the Internet, online information is described in terms of mass, making the web crawler technology responsible for information collection face a great challenge. With its flexible customizability and information collection speed and scale, the web crawler meets people's increasingly growing means of information acquisition to meet the needs of data acquisition and search engine for data mining and data analysis of big data.

In view of the above problems, based on the Python2.7 and Scrapy environment, this article is based on Scrapy-Redis distributed framework web crawler, and uses "Sina Weibo" Web as crawling object to learn and analyze the current crawling technology principles. Based on the core modules and the running process, an exploratory implementation of a Scrap-Redis framework, multithreading, high concurrency, and robust web crawler is performed to complete the data capture and other objectives, and then the crawled data is simply data analysis.

    First of all, this paper gives a brief description of the principle and development status of crawler technology, introduces some key technologies in crawler engineering, and focuses on the deep impact of the Cookie pool and user-agent deception in this study to break site restrictions, information filtering, Search strategy.

    Secondly, by using the Scrapy open source crawler framework developed based on the Python language for crawler development, Redis's in-memory database is deduplicated, tasked, crawled, and crawled, and supported by "breakpoints, climbs," and MongoDB is indicated. The NoSQL database has a huge role in metadata storage. Details the process and implementation details of the Scrapy-Redis development crawler.

     Again, the key issues in the field of crawler design are discussed, and the solutions for custom crawlers implemented in this article are discussed. With anti-climb technology against microblogging, crack verification code, URL deduplication to prevent the emergence of loops, and multi-threaded concurrent issues, then use and analyze Scrapy-Redis's own solution.

Finally, a simple data analysis of the data crawled by the crawler yields a series of small conclusions.

**Keywords：Sina Weibo, Scrapy-Redis, Python, Web, Crawler, Data Analysis**

目 录

## 摘要

## ABSTRACT

## 绪论

#### 课题背景

#### 发展历史

#### 发展现状

* 1. **研究意义**
  2. **研究目标**
  3. **研究中遇到的关键问题**
  4. **论文主要的组织结**

## 网络爬虫基本知识与相关技术

#### 2.1爬虫的分类与作用

###### 2.1.1爬虫的定义

###### 2.1.2爬虫的作用与分类:

#### 2.2HTTP协议

#### 2.3Rebots协议

###### 2.3.1网络爬虫引发的问题

###### 2.3.2Rebots协议

#### 2.4爬虫搜索策略-防止环路的出现

###### 2.4.1网站的树结构

###### 2.4.2网络爬虫图的搜索策略

## 第3章 爬虫系统架构设计

#### 3.1需求分析

###### 3.1.1需求分析

#### 3.2非关系型数据库

###### 3.2.1NOSQL的介绍

###### 3.2.2Mongodb简介

###### 3.2.3Mongodb搭建

###### 3.2.4Redis简介

###### 3.2.5Redis搭建

#### 3.3 Scrapy框架

###### 3.3.1Scrapy框架简介

###### 3.3.2Scrapy框架解析

###### 3.3.3Scrapy创建

#### 3.4 Scrapy+redis架构

###### 3.4.1 Scrapy+Redis架构定义

###### 3.4.2 组件间关系:

## 第4章 爬虫遇到的关键问题解决

#### 4.1网站的分析

###### 4.1.1微博移动版网站的分析

#### 4.2 爬虫伪装

###### 4.2.1User-agent伪装

#### 4.3信息过滤规则-正则表达式

###### 4.3.1 HTML页面解析:

###### 4.3.2正则表达式简介

###### 4.3.3正则表达式相关注解

###### 4.3.4Chrome编写正则表达式

#### 4.4去重

###### 4.4.1去重原理

###### 4.4.2 Redis去重

#### 4.5反扒技术

###### 4.5.1企业中常见的反爬技术手段

###### 4.5.2对抗AJAX技术

###### 4.5.3 JSON格式爬取

###### 4.5.4验证码处理

#### 4.6 Cookies池

###### 4.6.1 Cookies池简介

###### 4.6.2构建Cookies池

## 第5章 实验测试

#### 5.1测试环境

#### 5.2运行状态及测试

#### 5.3测试过程其他组件过程展示

#### 5.4数据抓取结果展示与分析

## 第6章 数据库设计与数据分析

#### 6.1数据模型

###### 6.1.1数据库概念设计——E-R模型

###### 6.1.2数据库逻辑设计

###### 6.1.3数据表设计

#### 6.2数据的分析

###### 6.2.1数据分析

## 第7章 总结与展望

###### 7.1总结

###### 7.2不足

###### 7.3展望

## 参考文献

## 致谢

## 附录

**1.1课题背景：**

随着以互联网为代表的各种网络技术的飞速发展，网络上存储的各种信息已经可以用海量来形容, 并呈现出指数增长的趋势，成为21世纪最为重要的经济资源之一。如何在海量的信息中快速检索到所需要的信息，成为了网络技术研究领域的一大热点。最初始的搜索引擎采用人工方式收集信息，由编辑员查看信息之后，人工形成信息摘要，并将信息置于事先确定的分类框架中。这种方法虽然准确可靠，但随着网络的飞速发展，在信息收集的数量和更新速度上已经远远无法满足要求，因此目前绝大多数搜索引擎都采用自动化的信息收集方式，由一个称为网络爬虫的机器人程序以某种策略自动地在互联网中搜集和发现信息，由索引器为搜集到的信息建立索引。该类搜索引擎的优点是信息量大、更新及时、毋需人工干预。可以看到，网络爬虫程序的质量直接关系到搜索引擎的质量，因此是目前网络大数据技术研究领域的一个热点。  
 网络爬虫，又称网页蜘蛛、网络机器人。随着计算机技术的高速发展，互联网中的信息量越来越大，搜索引擎应运而生。传统的搜索引擎会有返回结果不精确等局限性。为了解决传统搜索引擎的局限性，专用型网络爬虫在互联网中越来越常见。同时，专用型网络爬虫具有专用性，可以根据制定的规则和特征，最后只体现和筛选出有用的信息，并且可以对爬取得数据进行有效的数据分析实现快速的获取数据获取加快大数据在现实世界的应用。

**1.2发展历史：**

一开始世界上没有人写爬虫，后来网络上的数据多了，人看不过来了就有了最初的爬虫程序，进而出现早起的搜索引擎。

之后爬虫最大的应用还是在搜索引擎，一般的企业都忙着做网站，做系统。后来一些咨询公司需要分析网上的数据，出现了一些爬虫岗位，后来网络上的舆情事件层出不穷，针对网络舆情有了大量的需求，于是很多出现了很多爬虫岗位，这个阶段，爬虫的岗位技术门槛不高，采集的对象都是些新闻资讯站点。

再后来 大数据和数据挖掘火了，很多人都意识到数据中的财富，一时间各大小公司都组件大数据部门，结果发现没数据，于是想到了爬虫，这时候爬虫的采集对象多了，电商，论坛，机票，本地服务其中某两家本地信息平台互相采集对方数据，形成最早的爬虫与反爬虫。

逐渐的社区出现一大批开发爬虫，采集技术门槛越来越低，用心研究一下，就能实现大规模的数据采集很多公司往往设置一个爬虫岗位，领导没有长远规划，只是下达采集任务。

由于爬虫的岗位稀少，需求量一时很大，导致薪资比一般开发偏高，大批人员涌向爬虫岗位。

目前招聘爬虫的以大公司为主，一般划分为大数据岗位，另一些稳定岗位是互联网金融公司，长期需要采集数据目前看不到爬虫岗位增加的趋势，一般采集需求，现有人员利用开源爬虫即可完成互联网金融公司的爬虫一般服务于业务，但基本上各个维度的信息都有团队攻克，业界都有商业解决方案，大公司基于数据安全考虑会自建采集团队，一般小公司基于成本考虑会购买成熟服务，打了一定体量才会构建采集能力。

随着竞争的激烈，爬虫的技术会逐渐公开化，各家的竞争点将是稳定性和其他数据附加。

**1.3发展现状：**

随着大数据时代的来临，互联网对人类的生活影响越来越深入，已经成为人类获取信息的主要来源之一。互联网为用户带来海量数据的同时也带来了困扰，如何及时获得有效信息成为研究重点。搜索引擎根据预定的策略从互联网上发现和抓取数据，存入本地；对数据进行去噪、抽取和生成索引等处理，最终为用户提供信息检索服务，将相关信息展示给用户的系统。

爬虫（Crawler）是搜索引擎架构中的最底层模块，以一定的策略从互联网上抓取数据，并对数据进行预处理，然后将处理后的数据提交给搜索引擎其他模块，数据的质量和数量直接影响用户的体验。但是，随着大数据时代互联网数据爆炸式增长，爬虫抓取数据的速度越来越不能满足实际应用的需要。解决这个问题主要从软硬件考虑：一是升级爬虫的硬件，使用性能更好的硬件设备，但性价比不高，且不易扩展；二是利用分布式方式提高爬虫的并行处理能力，但这种方法会增加爬虫系统设计的复杂度。

目前，大多数大型爬虫系统采用的是分布式方式，但仍然不能满足用户的实际需要。其次，爬虫系统还需解决网页动态变化导致本地副本过期的问题。网页随时都可能变化，有的几小时更新一次，爬虫系统必须及时的发现并更新本地的网页，但互联网海洋中网页数量多、分布广，爬虫系统更新一遍需要数周甚至更长的时间，使得本地库中网页副本时新性较低。因此，一个采集速度快，网页更新及时的高可靠爬虫系统，不仅仅为搜索引擎提供基础数据，也可以为数据分析、挖掘提供基础数据，从而获得信息、知识。

**1.4研究意义**

1.深入学习巧Python和Soapy-Redis开源框架，自己动手实现有良好拓展性的网络爬

虫原型，将对我们学习新技术和拓宽眼界有着积极的作用。

2.虽然实现的只是一个原型程序，但是探索简单易模改易拓展的思路是非常正

确的。针对不同用户的特定需求提供特定的功能，正是如今程序开发领域流行的思

想。简单易模改的爬虫将是很好的实践。

**1.5研究目标**

本课题的研究目的在于使用基于Python的开源技术，结合其他网络编程的知识等，实现一个自定义的拓展能力强的网络爬虫原型程序。

1.基于Soapy-Redis框架，自定义的爬取规则。

2.选择合适的数据库进行数据存取。

3.采用多线程并发结构，提高运行效率。

4.由于是本科阶段不管是学习时间上的不足或者难度比较大工程量较大，所以本课题主要是研究高效强鲁棒性的爬虫系统，并且爬取到的数据进行一个非常简单的数据分析。

**1.6研究中遇到的关键问题**

1.突破目标网站对爬虫的限制，如拒绝访问、强制登出、封锁IP 、验证码、用户Cookie封锁等。破解网页的限制将是爬虫是否能高效运作的关键。

2.解决URL重复问题。在高速运行的爬虫工作过程中，如何鉴别即将爬取的网页是否己经爬取过极大影响了运行效率和资源的利用程度。

3.爬虫爬取有记忆可以中断，再启动时候可以继续中断时的爬取。这样做可以防止出现意外中断爬虫程序就前工尽废，增强爬虫程序的鲁棒性。

4.多线程并发实现。根据爬取目标网页的设计，如何设计更加高效利用CPU能力，怎样设计多线程的并发代码也需要学习解决。

5.编码问题。为了爬虫的数据表示有效使用Base64编码传输，为了应对动态交互AJAX网站表单等部分数据不可见使用JSON格式，数据库也是用MongoDB也是JSON格式。

**1.7 论文主要组织结构：**

本论文分八章节进行阐述:

第一章,绪论。首先介绍本文的研究背景网络爬虫的产生与发展以及在数据分析等邻域中数据获取的手段，接着是本课题的研究目的和意义。最后给出了论文的研究内容及论文结构安排。

第二章,分别介绍了网络爬虫系统的基本知识和相关技术。在这一章介绍爬虫的分类及作用，随后介绍网络爬虫涉及到的网络协议，最后介绍爬虫搜索策略。

第三章,爬虫总体架构设计。本章主要是爬虫系统的需求与架构设计进行了 详述的描述。首先介绍爬虫系统的业务需求，性能需求和其他需求，然后完成爬虫系统使用的Scrapy框架，再来是分别介绍两个NOSQL数据库，一个MongoDB是放在磁盘层作为获取的数据的持久化操作，另一个Redis内存数据库放在内存中一方面进行查重，另一方面进行爬虫的调度。最后介绍Scrapy+redis网络爬虫的架构以及实现。

第四章,爬虫关键问题实现。介绍爬虫系统的具体实现的时候会遇到的一些难题比如User-agent伪装，用正则表达式编写信息的过滤规则，怎么样对抗微博的反爬技术，如何打码，如何查重，并建立Cookie池,讨论并得到解决方案。

第五章,系统测试。介绍了系统实现的软件与硬件环境，爬虫爬取数据等进行简单的测试。

第六章,数据库设计与分析。对爬虫系统设计数据库，在MongoDB数据库上设计微博用户信息等三个表，E-R图，具体结构设计，逻辑设计等。然后对爬虫爬取到的微博的用户信息进行简单的数据分析得出一些简单的统计等结论。

第七章,总结与展望。总结本次论文研究课题的完成情况。对系统的整体设计与实现做回顾，分析自己的设计存在不足，展望进一步改进的主要方向。对本课题下一步发展方向做一些分析。

最后，向所有的在本人本科阶段研究哦期间给我支持帮助，指导点拨的家人，前辈，同学，朋友和老师致谢。

2.1.1爬虫的定义:

网络爬虫(Web Crawler)又被称为Ants, Automatic Indexer, Spiders, Robots

或者Wanderers，就是抓取特定网站网页的HTML数据。网络爬虫从一个存放URL

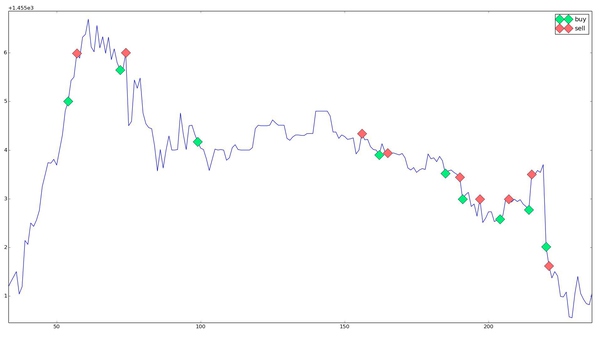
的集合开始进行爬取，利用HTTP协议连续地从不同的服务器中获取Web页面,首先从队列中获取一个URL并下载此网页，提取该网页中的其它URL并放入队列中。此过程将重复直至关闭的计算机程序。现在很多网站，尤其是搜索引擎都将网络爬虫作为一种获取、提供互联网最新数据的手段川。

2.1.2爬虫的作用与分类:

爬虫能用来干什么.如今每天互联网上的流量足足有10亿GB左右，不可能毫无选择的全部爬取下来，这是要根据我们自己的需求来有选择的爬取相对应的数据.那爬虫可以用来干什么呢？我们根据实际需要将爬虫的目标分为三类.

第一类是通过搜索引擎的网络爬虫来充实搜索引擎的索引列表.这部分需求所需要的数据是各种网页的地址，标题，主题字等等.这方面比较知名的有google爬虫，baidu爬虫，Yahoo爬虫等等。可以大致分为 批量型爬虫（Batch Crawler），增量型爬虫（Incremental Crawler），垂直型爬虫(Focused Crawter）.

第二类建立自己的数据仓库，然后用各种机器学习模型来得出一些寻常无法得出的预测和分析.我们可以通过这些数据集来进行预测.比如说股票，比特币和各种风险交易。

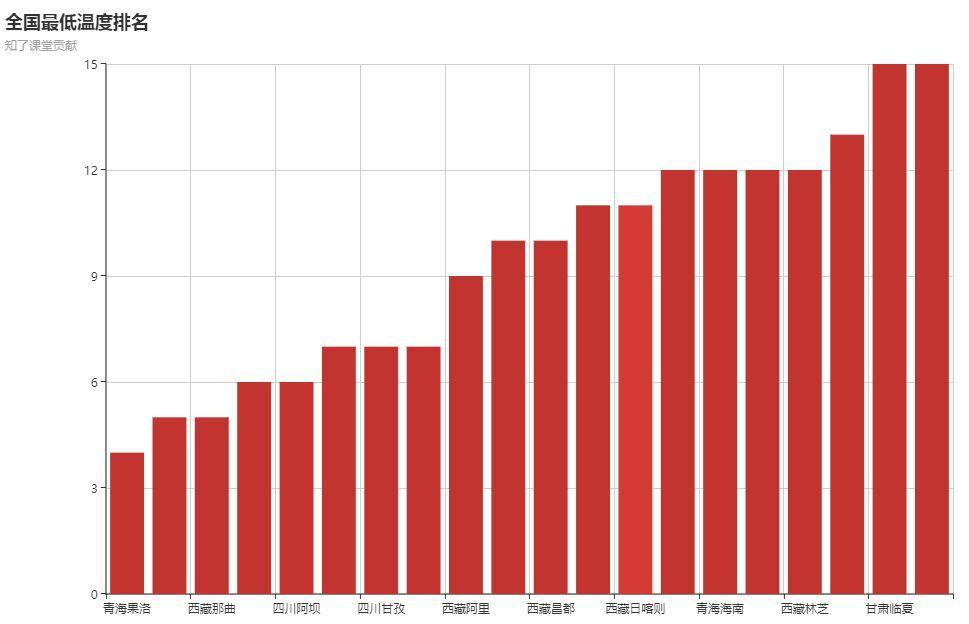


只不过要达到这种水平，那么需要爬取到特别优质的数据集.而比特币市场比起股票来说是一个很简单的市场，但是这也能证明爬虫在该方面的运用.

注：该MIT论文的链接是[https://arxiv.org/pdf/1410.1231v1.pdf](https://arxiv.org/pdf/1410.1231v1.pdf%29.这篇论文以比特币市场为例，用爬虫爬取的各种高频交易的信息，通过机器学习建立的人工智能成功的预测比特币短期价格的涨跌，然后做高频交易.下面就是一张MIT只用了3天的training)

第三类为各种论文以及文章提供有力的数据支撑。

我们一般在报告中看到的有关于各行各业的数据分析图，那都是通过抓取特定行业的专业数据，然后用各种数据分析也好，自己使用第三方可视化JavaScript库也好，最终得出的这些表图.同样用例子来说明.比例如，我们要找出全国气温最低的地方并直观的表示出来，那么我们仅仅需要爬取一下全国的气象信息，然后通过类似于D3.js这种可视化JavaScript库就可以得到一张柱状图.如下图所示，这样，全国最低气温就能直观的显示出来。同时也可以用过各种气象论文中论点的有力支撑.



2.2.1 Http协议定义

HTTP，Hypertext Transfer Protocol，超文本传输协议

HTTP是一个基于“请求与响应”模式的、无状态的应用层协议 HTTP协议采用URL作为定位网络资源的标识，URL格式如下：

http://host[:port][path]

host: 合法的Internet主机域名或IP地址 port: 端口号，缺省端口为80 path: 请求资源的路径

对HTTP的URL的理解：

URL是通过HTTP协议存取资源的Internet路径，一个URL对应一个数据资源,HTTP URL的实例:

1. <https://www.zhihu.com/>
2. <https://www.csdn.net/>
3. <https://weibo.com/>

2.2.2 POST与GET的请求

* GET请求数据参数会拼接在URL中，一是不安全太容易被发现，二是URL的长度有限制。而POST请求会把数据放到请求体中，安全稍微提高那么一点，因为截获报文扔能看到，长度没有限制。
* GET请求可以被缓存，比如登录页面，如果通过GET请求提交，用户名和密码都会被添加到URL上，这个页面可以被缓存下来，或在历史记录中被查看。
* GET请求可以被缓存，比如登录页面，如果通过GET请求提交，用户名和密码都会被添加到URL上，这个页面可以被缓存下来，或在历史记录中被查看。

2.2.3 Cookie和Session

HTTP是一种无状态的协议，一旦数据交互完毕，客户端与服务端的连接就会断开。但有些时候，我们需要持久的保存一些信息，比如上次连接的信息，如用户信息：用户名、密码等。   
 Cookie就是这样一种机制，它可以弥补HTTP无状态的不足。在Session出来之前，基本所有网站都是采用Cookie来跟踪会话。Cookie不能跨域使用。   
 Session是服务端记录客户端状态的一种机制，使用上比Cookie简单一些，相应的也会增加服务器的存储压力。

* cookie数据存放在客户的浏览器上，session数据放在服务器上；
* cookie不是很安全，别人可以分析存放在本地的COOKIE并进COOKIE欺骗，考虑到安全应当使用session；
* session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能。考虑到减轻服务器性能方面，应当使用COOKIE；
* cookie总大小在客户端也有限制（基本是4k），Firefox和Safari允许cookie多达4097个字节，包括名（name）、值（value）和等号。
* 每个域名cookie有限制，Firefox每个域名cookie限制为50个。

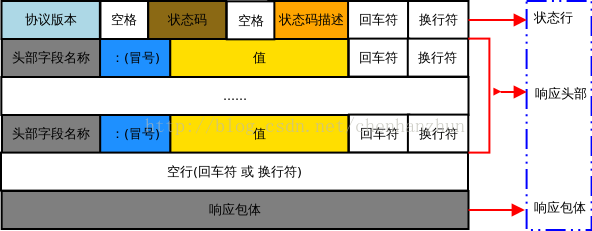
2.2.4 HTTP响应报文

状态行：HTTP/1.1 200 OK（CRLF）

协议和版本

状态码

状态码的描述



常见状态码：

100-199 : 表示成功接收请求, 要求客户端继续提交下一次请求才能完成整个处理过程   
200-299: 表示成果接收请求并已完成整个处理过程. 常用200   
300-399: 为完成请求, 客户需进一步细化需求: 例如: 请求的资源已经移动一个新地址, 常用302(重定向), 307和304(拿缓存)   
400-499: 客户端的请求有错误, 包含语法错误或者不能正确执行. 常用404(请求的资源在web服务器中没有) 403(服务器拒绝访问, 权限不够)   
500-599: 服务器端出现错误

**2.3.1网络爬虫引发的问题:**

* 网络爬虫在法律上的风险: 服务器上的数据有产权归属 网络爬虫获取数据后牟利将带来法律风险
* 网络爬虫在隐私的泄漏: 网络爬虫可能具备突破简单访问控制的能力，获得被保护数据 从而泄露个人隐私
* Web服务器默认接收人类访问 受限于编写水平和目的，网络爬虫将会为Web服务器带来巨大的资源开销

**2.3.2 Robots协议**：

Robots协议（也称为爬虫协议、机器人协议等）的全称是“网络爬虫排除标准”（Robots ExclusionProtocol），网站通过Robots协议告诉搜索引擎哪些页面可以抓取，哪些页面不能抓取.

作用：网站告知网络爬虫哪些页面可以抓取，哪些不行

形式：在网站根目录下的robots.txt文件

网络爬虫：自动或人工识别robots.txt，再进行内容爬取

约束性：Robots协议是建议但非约束性，网络爬虫可以不遵守，但存在法律风险

**1．）为什么需要Robots协议**

互联网上的网页是通过超级链接互相关联起来的，从而形成了网页的网状结构。爬虫的工作方式就像蜘蛛在网上沿着链接爬来爬去，最基本的流程可以简化如下：

1)喂给爬虫一堆URL，我们称之为种子(seeds)；

2)爬虫抓取seeds，解析html网页，抽取其中的超级链接；

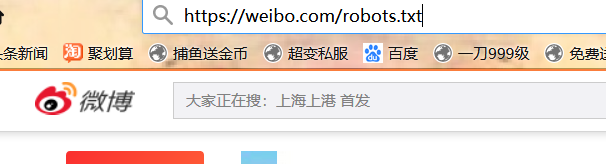
3)爬虫接着抓取这些新发现的链接指向的网页。

步骤2和步骤3循环往复。

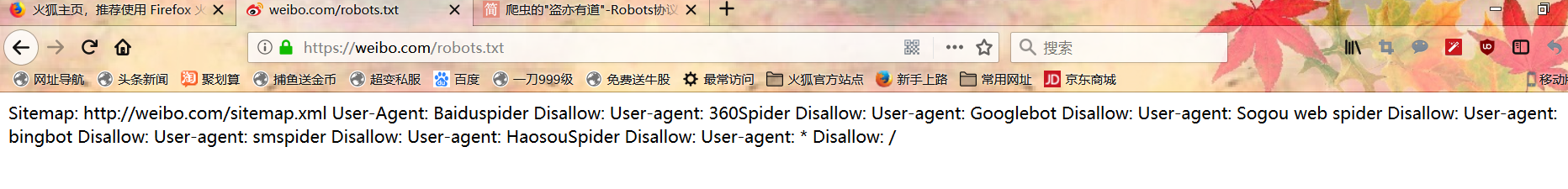
了解了上面的流程就能发现：对爬虫来说网站非常被动，只有老老实实被抓取的份。所以，对于网站的管理者来说，就存在这样的需求：

某些路径下是个人隐私或者网站管理使用，不想被搜索引擎抓取，比如说日本爱情动作片；不喜欢某个搜索引擎，不愿意被他抓取，最有名的就是之前淘宝不希望被百度抓取；小网站使用的是公用的虚拟主机，流量有限或者需要付费，希望搜索引擎抓的温柔点；某些网页是动态生成的，没有直接的链接指向，但是希望内容被搜索引擎抓取和索引。网站内容的所有者是网站管理员，搜索引擎应该尊重所有者的意愿，为了满足以上等等，就需要提供一种网站和爬虫进行沟通的途径，给网站管理员表达自己意愿的机会。有需求就有供应，robots协议就此诞生。

**2.）微博的Robots协议**

首先打开浏览器访问微博主页<https://weibo.com/> ,如下图:

在URL’/’后输入robots.txt回车 就能看到微博的robots协议如图所示:



就能看到微博的对爬虫的限制:

Sitemap: http://weibo.com/sitemap.xml User-Agent: Baiduspider Disallow: User-agent: 360Spider Disallow: User-agent: Googlebot Disallow: User-agent: Sogou web spider Disallow: User-agent: bingbot Disallow: User-agent: smspider Disallow: User-agent: HaosouSpider Disallow: User-agent: \* Disallow: /

显示360,bing,google,baidu,sogou,等等搜索引擎都不允许。

**爬虫搜索策略-防止环路的出现:**

现在看看图论的遍历算法和搜索引擎的关系。互联网虽然很复杂，但是说穿了其实就是一张大图而已一可以把每一个网页当作一个节点，把那些超链接(Hyperlinks )当作连接网页的弧。网页中那些蓝色、带有下划线的文字背后其实藏着对应的网址，当你点击的时候，浏览器通过这些隐含的网址跳转到相应的网页。这些隐含在文字背后的网址称为”超链接”。有了超链接，我们可以从任何一个网页出发，用图的遍历算法，自动地访问到每一个网页并把它们存起来。完成这个功能的程序叫做网络爬虫( Web Crawlers ).或者在一些文献中称为“机器人”( Robot)。世界上第一个网络爬虫是由麻省理工学院的学生马休·格雷( Matthew Gray)在1993年写成的。他给自己的程序起了个名字叫“互联网漫游者，(WWW Wanderer)。以后的网络爬虫越写越复杂。但原理是一样的。

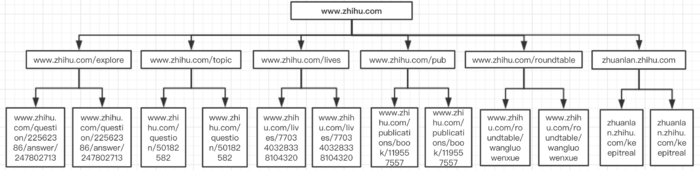
我们来看看网络爬虫如何下载整个互联网。假定从一家门户网站的首页出发。先下载这个网页，然后通过分析这个网页，可以找到页面里的所有超链接。也就等于知道了这家门户网站首页所直接链接的全部网页，诸如雅虎邮件、雅虎财经、雅虎新闻等。接下来访向、下载并分析这家门户网站的邮件等网页，又能找到其他相连的网页。让计算机不停地做下去，就能下载整个的互联网。当然，也要记载哪个网页下载过了，以免重复。在网络爬虫中，使用一个称为“哈希表”( Hash Table )的列表而不是一个记事本记录网页是否下载过的信息。

现在的互联网非常庞大，不可能通过一台或几台计算机服务器就能完成下载任务。比如Google在2010。年时整个的索引大小大约有S 000亿个网页，即使更新最频繁的基础索引也有100亿个网页，假如下载一个网页需要一秒钟，下载这100亿个网页则需要317年，如果下载5 000亿个网页则需要16 000年左右，是我们人类有文字记载历史的三倍时间。因此，一个商业的网络爬虫需要有成千上万台服务器，并且通过高速网络连接起来。如何建立起这样复杂的网络系统，如何协调这些服务器的任务，就是网络设计和程序设计的艺术了。

**2.4.1网站的树结构**

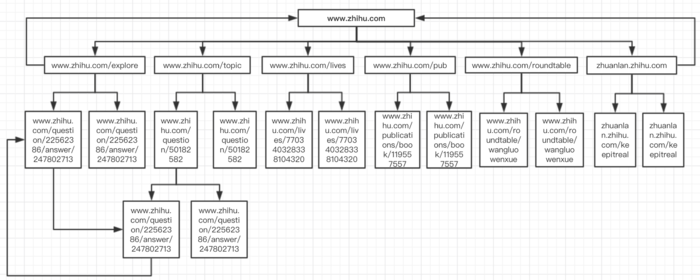
1、一个网站的URL结构图

以知乎为例，知乎目前有发现、话题、Live、书店、圆桌、专栏主要的6个tab页。每个网站的url都是有一定的层次，如下图：发现explore、话题topic、Live lives、书店pub、圆桌roundtable、专栏zhuanlan都是在主域名zhihu的下一级，而具体的Live在…/67006058/answer内容又在话题之下zhihu/question/67006058/answer/250037350，网站的所有内容都一层一层的类似一个树形结构。



2、网站URL链接的结构图

当然，如果我们要做爬取整个网站的url时，我们必须要知道每个网站的url链接一般情况下都是存在环路的，也就是在下一级页面存在上一级页面的url链接，这样形成一个环路。当遇到这个情况时我们需要做url去重，一般的处理方式是把已经爬过的url放到一个list，每次爬取url的时候都去这个list查看下是否已经爬过，爬过的就跳过。这块url去重我下次再详细介绍。



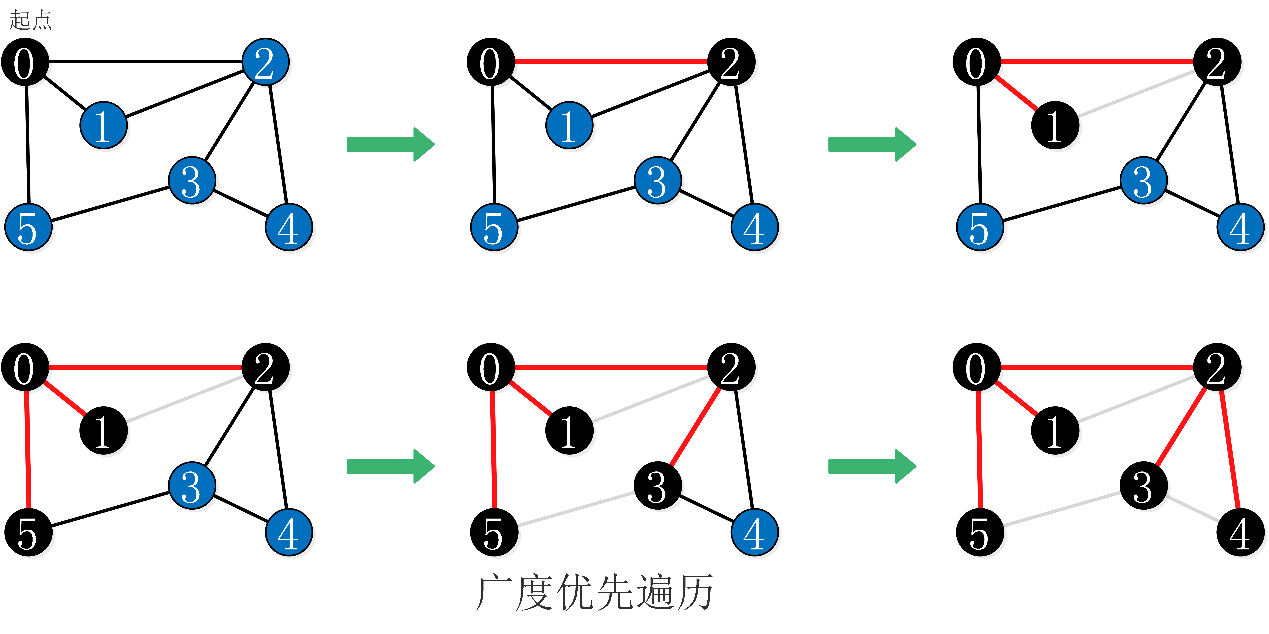
**2.4.2网络爬虫图的搜索策略**：

常见的爬虫爬取策略有以下两种：

1.广度优先遍历策略

广度优先遍历策略，可理解为一种基于网络层次分析的遍历策略，是将网络划分成若干层次，其中，种子站点是处于结构的最顶层。在遍历时，需要处理Web页之间的抓取优先级的问题，一般规定层次结构中层次越高，优先级也就越高;同一层次中，从左到右优先级依次降低。因此，如果只用一个网络爬虫进行遍历操作，那么就按不同层次之间先高后低的次序，以及同层次之间先左后右的次序进行。当然，也可以将多个网络爬虫设计成分布式的结构，它们分别负责其中一层或几层Web站点的遍历，这样不但技术实现的难度不高，而且可以较好地避免重复遍

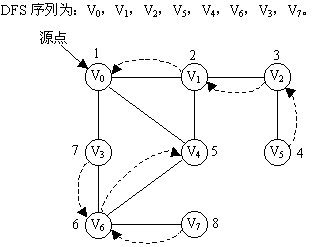
BFS遍历如下图:



2.深度优先策略

深度优先遍历类似于树的前序遍历。它将遍历的侧重点放在网络爬虫爬行的深度上，往往为了抓取更多的Web网页，允许网络爬虫访问站点中某一超链接相关的所有链接。从互联网的结构看，Web页之间通过数量不等的超链接相互连接，形成一个彼此关联、庞大复杂的有向图。所以，按照深度优先的原则，网络爬虫通常需要多次爬行一些Web页，即出现重复爬行现象。如何规避此类问题也就成为修正深度优先策略的重中之重，在实践中，一般会建立一个爬行路径优选算法来简化网络爬虫的行走路线，同时需要依照具体情况确保一个合适的遍历深度。对比以上两种遍历策略errew，广度优先策略更适合大规模Web信息的搜集，可充分发挥网络爬虫程序算法的功效，适合水平型搜索引擎的信息抓取;而宽度优先策略一般适用于垂直搜索引擎的信息搜集。但在实际应用中，二者可相互嵌套。

DFS遍历如下图：



分析：

深度优先搜素算法：不全部保留结点，占用空间少；有回溯操作(即有入栈、出栈操作)，运行速度慢。

广度优先搜索算法：保留全部结点，占用空间大； 无回溯操作(即无入栈、出栈操作)，运行速度快。

通常深度优先搜索法不全部保留结点，扩展完的结点从数据库中弹出删去，这样，一般在数据库中存储的结点数就是深度值，因此它占用空间较少。

所以，当搜索树的结点较多，用其它方法易产生内存溢出时，深度优先搜索不失为一种有效的求解方法。

广度优先搜索算法，一般需存储产生的所有结点，占用的存储空间要比深度优先搜索大得多，因此，程序设计中，必须考虑溢出和节省内存空间的问题。

但广度优先搜索法一般无回溯操作，即入栈和出栈的操作，所以运行速度比深度优先搜索要快些。

综上所述: 我采取了广度优先搜索的爬行，避免深度优先陷入某个站点的环路中，无法访问其他站点。

**3.1.1需求分析**

随着微博应用在众多网络用户心中的地位越来越重，越来越多的研究者重

视对微博数据的研究。要充分对庞大的微博数据进行分析研究，首先必须要构

建一个能够抓取微博信息的微博爬虫。目前存在的微博爬虫一方面是基于微博

平台API的，在数据获取数量上有限制;另一方面获取的微博数据也是杂乱无

章的，没有规律性可言。基于上面的两个当前微博爬虫存在的问题，本文提出

了基于主题的微博Web爬虫。那么根据这些描述，本文研究的爬虫需要实现以

下的功能:

(1)模拟微博平台的登陆过程,对抗微博的反爬技术，获得微博平台的合法URL地址，以便后面URL地址的扩大采集。

（2)本毕设不使用微博平台的API，利用网页分析技术分析微博网页页面，回避现有的API的种种限制，尽可能的抓取更多的微博数据。

(3)爬取的数据包括对用户发表的微博点赞、转载、评论数量与发表时间、定位、发表平台等等进行爬取；爬取微博用户ID、粉丝、省市、关注、首页链接等等基本信息；爬取用户间关系方便以后建立关系分析等。

（4）

(4)设计数据库Mongodb，存储微博数据以及微博用户信息。

根据上述功能需求的分析，可以将整个爬虫大致分为三个模块。这三个模

块分别为:模拟微博平台登陆模块、微博信息抓取模块、微博数据存储模块。

其中，第一个模块为起始模块，是实现后续模块的基础，而第二个模块为核心

模块，是实现抓取主题相关微博的主要模块，其中最重要的是微博主题相关性

分析子模块，而第三个模块便是爬虫的收尾模块，是对于前几个模块成果的体

现。每个模块实现的具体功能为:

（1）模拟微博平台登陆模块:根据微博平台登陆的流程，模拟其登陆过程，

获取合法的初始URL地址，将这个URL地址作为seed，后续模块将根据这

个“引子’，获取更多URL地址;

（2）微博信息抓取模块:利用合法的URL地址，获取对应的HTML代码，

该模块主要是分析这一HTML代码，从中抽取出微博数据以及相关的微博用户

信息。同时这一模块也需实现获取新的URL地址的功能，并且需要判断微博是

**3.2.1NOSQL的介绍**

NoSQL，指的是非关系型的数据库。NoSQL有时也称作Not Only SQL的缩写，是对不同于传统的关系型数据库的数据库管理系统的统称。NoSQL用于超大规模数据的存储。（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

1）为什么使用NoSQL ?

今天我们可以通过第三方平台（如：Google,Facebook等）可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了, NoSQL数据库的发展也却能很好的处理这些大的数据。2）NoSQL的优点/缺点

优点:

* 高可扩展性
* 分布式计算
* 低成本
* 架构的灵活性，半结构化数据
* 没有复杂的关系

缺点:

* 没有标准化
* 有限的查询功能（到目前为止）
* 最终一致是不直观的程序

**3.2.2Mongodb简介**

MongoDB 是一个基于分布式文件存储的数据库。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的MongoDB 将数据存储为一个文档，数据结构由键值(key=>value)对组成。MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档，数组及文档数组。

1）应用场景分析

1、MongoDB的应用场景

（1）表结构不明确且数据不断变大

MongoDB是非结构化文档数据库，扩展字段很容易且不会影响原有数据。内容管理或者博客平台等，例如圈子系统、存储用户评论之类的。

（2）更高的写入负载

MongoDB侧重高数据写入的性能，而非事务安全，适合业务系统中有大量“低价值”数据的场景。本身存的就是json格式数据。例如做日志系统。

（3）数据量很大或者将来会变得很大

MySQL单表数据量达到5-10G时会出现明细的性能降级，需要做数据的水平和垂直拆分、库的拆分完成扩展，MongoDB内建了sharding、很多数据分片的特性，容易水平扩展，比较好的适应大数据量增长的需求。

（4）高可用性

自带高可用，自动主从切换（副本集）

不适用的场景

（1）MongoDB不支持事务操作，需要用到事务的应用建议不用MongoDB。

（2）MongoDB目前不支持join操作，需要复杂查询的应用也不建议使用MongoDB

**1.4 Mongodb的搭建**

从MongoDB官网下载安装mongodb3.4 64位

创建数据目录

MongoDB将数据目录存储在db目录下。但是这个数据目录不会主动创建，我们在安装完成后需要创建它。请注意，数据目录应该放在根目录下。

CMD下命令：

c：\> cd c：\

c：\> mkdir data

c：\> cd data

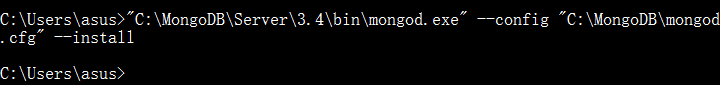
c：\ data> mkdir db

c：\ data> cd db

在C盘安装了mongodb，现在让我们创建一个数据的目录然后在数据目录里创建db目录。

从命令提示符下运行MongoDB服务器，您必须从MongoDB目录的bin目录中执行mongod.exe文件

如下图：



配置MongoDB服务：

在管理员模式打开命令行窗口创建目录，执行下面的语句来创建数据库和日志文件的目录

mkdir c：\ data \ db：\ data \ db

mkdir c：\ data \ log：\ data \ log

创建配置文件

创建一个配置文件。该文件必须设置systemLog.path参数，包括一些附加的配置选项更好。创建一个配置文件位于C：\ mongodb \ mongod.cfg，其中指定systemLog.path和storage.dbPath。具体配置内容如下：

systemLog：：

目标：文件：文件

路径：c：\ data \ log \ mongod.log：c ：\ data \ log \ mongod 。日志

存储：：

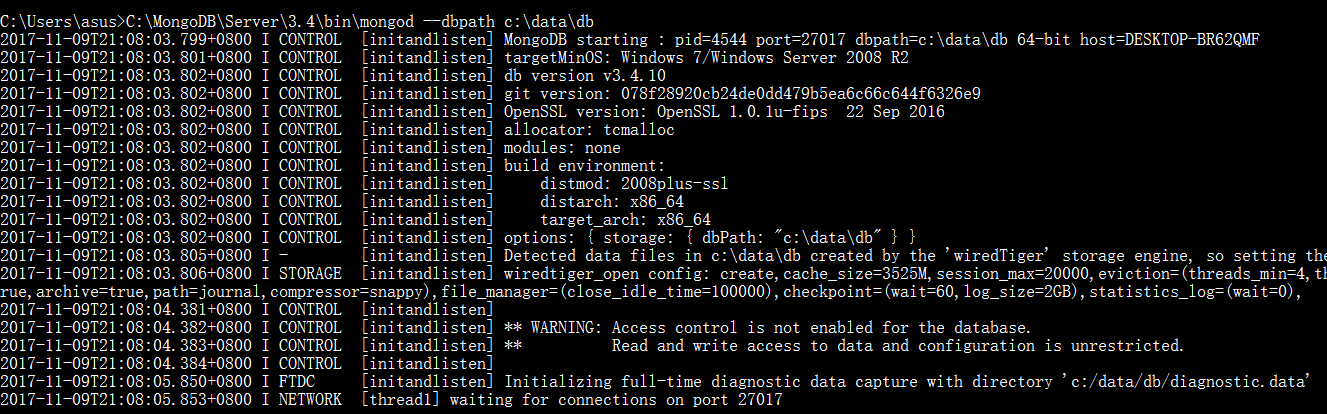
dbPath：c：\ data \ db：c ：\ data \ db

安装Mongodb:

通过执行mongod.exe，使用--install选项来安装服务，使用--config选项来指定之前创建的配置文件。

“C：\ mongodb \ bin \ mongod.exe”--config“C：\ mongodb \ mongod.cfg”--install - 配置“C：\ mongodb \ mongod.cfg” - 安装

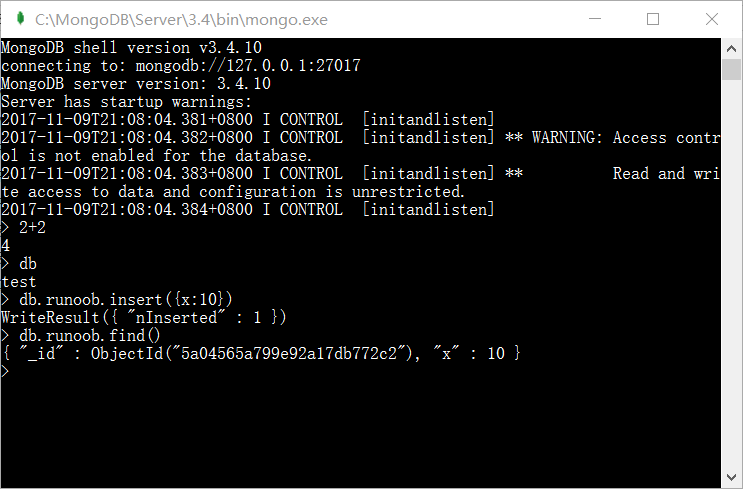
Mongodb启动成功：



Mongodb的服务：



Mongodb测试：



MongoDB自带的交互式Javascript shell，用来对MongoDB进行操作和管理的交互式环境。

当你进入mongoDB后台后，它默认会链接到test文档（数据库）：

测试案例

>2+2

4

> db.runoob.insert（{x：10}）

WriteResult（{“nInserted”：1}）

> db.runoob.find（）

{“\_id”：ObjectId（“5604ff74a274a611b0c990aa”），“x”：10}

**3.2.4Redis简介**

REmote DIctionary Server(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统，主要做内存中！Redis是一个键值存储系统。它支持存储的值类型相对更多，包括字符串（字符串），列表（链表），集合（集合），zset（有序集合 - 有序集合）和散列（哈希类型）。这些数据类型都支持压入/弹出，添加/删除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的支持各种不同方式的排序为了保证效率，数据都是缓存在内存中。区别的是redis的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主从。Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。Redis支持数据的备份，即master-slave模式的数据备份。

Redis 优势：

* 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
* 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Set数据类型操作。
* 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
* 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

**3.2.5Redis搭建**

下载的是Redis.0.504版本，我的电脑是win10 位，所以下载64位版本的，在运行中输入CMD，然后把目录指向解压的Redis目录。

安装测试过程中的error错误：

redis启动报错[9844] 02 Dec 13:57:40.682 # Creating Server TCP listening socket \*:6379: bind:Unknownerror

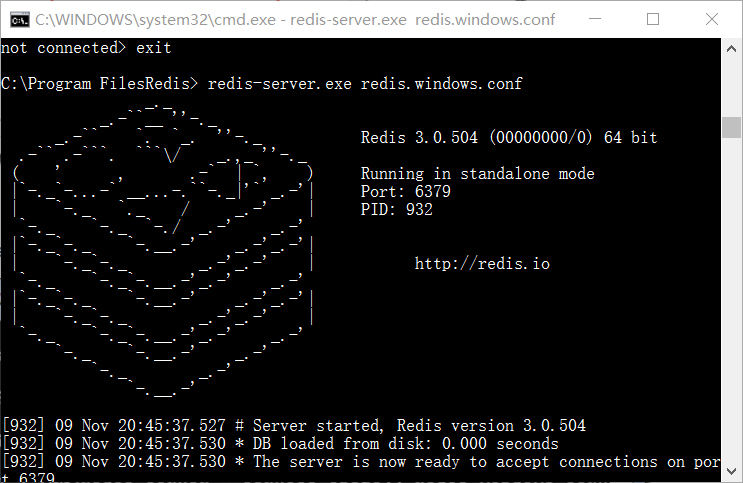
解决方案如下

按顺序输入如下命令就可以连接成功

1. redis-cli.exe  
2. shutdown  
3. exit  
4. redis-server.exe redis.windows.conf

启动命令：

redis-server.exe redis.windows.conf，出现下图显示表示启动成功了。



1.2对Redis-cli并进行测试

CMD下打开redis-cli.exe，输入如下命令的测试案例

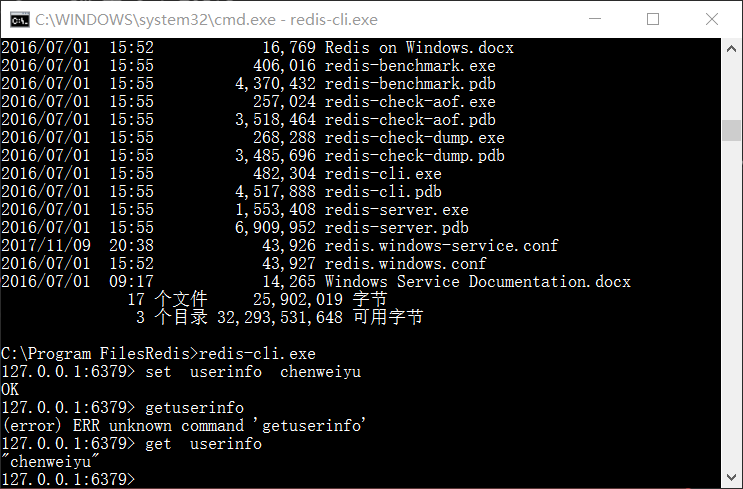
> Set userinfo chenweiyu

OK

>get userinfo

“chenweiyu”

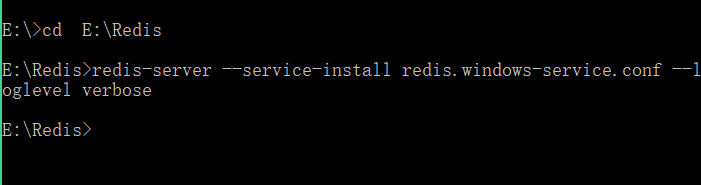
测试成功



windows安装redis服务

CMD的Redis目录下:

命令: s>redis-server --service-install redis.windows-service.conf --loglevel verbose



显示成功:



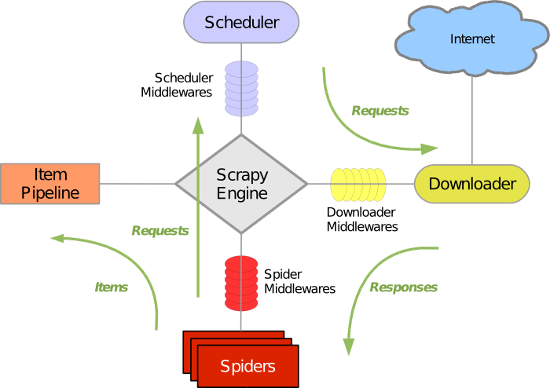
3.3Scrapy框架

**3.3.1Scrapy框架简介**

Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架。 可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。Scrapy是用Python开发的一个开源的Web爬虫框架，可用于快速抓取Web站点并从页面中高效提取结构化的数据。Scrapy可广泛应用于数据挖掘、监测和自动化测试等方面，提供了多种类型爬虫的基类，如BaseSpider,SitemapSpider等。

其最初是为了页面抓取 (更确切来说, 网络抓取 )所设计的， 也可以应用在获取API所返回的数据(例如 Amazon Associates Web Services ) 或者通用的网络爬虫。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。

Scrapy 使用了 Twisted异步网络库来处理网络通讯。整体架构大致如下：

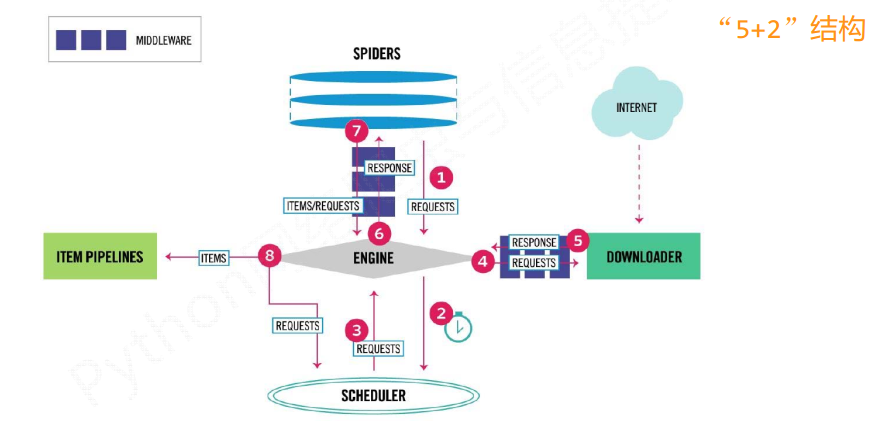


Scrapy框架组件：

如上图所示Scrapy框架包括了一下的：

* 引擎(Scrapy): 用来处理整个系统的数据流处理, 触发事务(框架核心)
* 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. 可以想像成一个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
* 下载器(Downloader): 用于下载网页内容, 并将网页内容返回给蜘蛛(Scrapy下载器是建立在twisted这个高效的异步模型上的)
* 爬虫(Spiders): 爬虫是主要干活的, 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
* 项目管道(Pipeline): 负责处理爬虫从网页中抽取的实体，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。当页面被爬虫解析后，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框架，主要是处理Scrapy引擎与下载器之间的请求及响应。
* 爬虫中间件(Spider Middlewares): 介于Scrapy引擎和爬虫之间的框架，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 Middewares): 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。

**3.3.2Scrapy框架解析**：

Scrapy框架是采用了5+2结构主要有三条主要的数据流路径

第一条路径：

1) Engine通过中间件获得了Spiders发送的请求, 这个请求叫做requests, 相当于是一个url

2) Engine再转发给scheduler, scheduler主要负责对爬取请求进行调度

第二条路径：

3) 从Scheduler获得下一个需要爬取的请求, 这是一个真实的请求

4) Engine获得这个请求之后, 通过中间件, 将请求给 Downloader模块

5) Downloader爬取请求中相关的网页, 并将爬取的内容封装成一个对象, 这个对象叫response(响应)

6) Engine再通过中间件将response返回给Spiders

第三条路径：

7) Spiders处理从Downloader获得的响应, 处理之后会产生两个数据类型, 一个是items(爬取项), 另一个是新的requests

8) Engine接受到这两数据之后, 将items发送给Item Pipelines, requests发送给Scheduler进行调度

整个框架的入口是Spiders, 出口是Item Pipelines其中Engine, Scheduler, 和Downloader是已经写好的, 不需要实现Spiders和Item Pipelines是需要编写的, 但是里面有既定的代码框架, 所以情况是要对某东西进行修改, 这种情况一般叫配置

**3.3.3Scrapy创建**

1. Scrapy运行流程大概如下：

首先，引擎从调度器中取出一个链接(URL)用于接下来的抓取；

引擎把URL封装成一个请求(Request)传给下载器，下载器把资源下载下来，并封装成应答包(Response)；

然后，爬虫解析Response；

若是解析出实体（Item）,则交给实体管道进行进一步的处理；

若是解析出的是链接（URL）,则把URL交给Scheduler等待抓取；

1. Scrapy Tutorial

安装命令在IDE(pycharm中):pip install Scrapy

在抓取之前要新建一个项目:

CMD下到相应的目录中去输入命令:

scrapy startproject Sina\_spider

就会创建如下的文件夹结构:

├── scrapy.cfg

└── Sina\_spider

   ├── \_\_init\_\_.py

   ├── items.py

   ├── pipelines.py

   ├── settings.py

   └── spiders

       └── \_\_init\_\_.py

文件主要是:

* scrapy.cfg: 项目配置文件
* Sina\_spider/: 项目python模块, 之后您将在此加入代码
* Sina\_spider/items.py: 项目items文件
* Sina\_spider/pipelines.py: 项目管道文件
* Sina\_spider/settings.py: 项目配置文件
* Sina\_spider/spiders: 放置spider的目录

**3.4Scrapy-redis架构**

**为什么使用Scrapy+redis架构:**

Scrapy，Python开发的一个快速,高层次的屏幕抓取和web抓取框架，用于抓取web站点并从页面中提取结构化的数据。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。

Scrapy框架已经可以完成很大的一部分爬虫工作了。但是如果遇到比较大规模的数据爬取，直接可以用上python的多线程/多进程，如果你拥有多台服务器，分布式爬取是最好的解决方式，也是最有效率的方法。

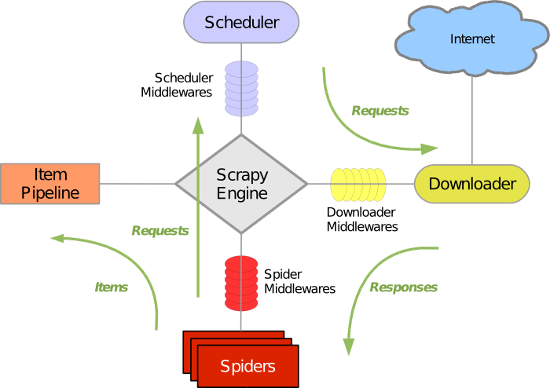
Scrapy-redis是基于redis的一个scrapy组件，scrapy-redis提供了维持待爬取url的去重以及储存requests的指纹验证。原理是：redis维持一个共同的url队列，各个不同机器上的爬虫程序获取到的url都保存在redis的url队列，各个爬虫都从redis的URL队列获取url，并把数据统一保存在同一个数据库里面。

**3.4.1Scrapy+Redis架构定义:**

Scrapy是一个比较好用的Python爬虫框架，你只需要编写几个组件就可以实现网页数据的爬取。但是当我们要爬取的页面非常多的时候，单个主机的处理能力就不能满足我们的需求了（无论是处理速度还是网络请求的并发数），这时候分布式爬虫的优势就显现出来。

而Scrapy-Redis则是一个基于Redis的Scrapy分布式组件。它利用Redis对用于爬取的请求(Requests)进行存储和调度(Schedule)，并对爬取产生的项目(items)存储以供后续处理使用。scrapy-redis重写了scrapy一些比较关键的代码，将scrapy变成一个可以在多个主机上同时运行的分布式爬虫。

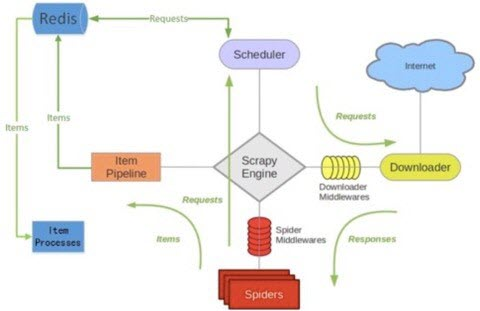
原生的Scrapy的架构是这样子的：



scrapy-redis的官方文档写的比较简洁，没有提及其运行原理，所以如果想全面的理解分布式爬虫的运行原理，还是得看scrapy-redis的源代码才行，不过scrapy-redis的源代码很少，也比较好懂，很快就能看完。

scrapy-redis工程的主体还是redis和scrapy两个库，工程本身实现的东西不是很多，这个工程就像胶水一样，把这两个插件粘结了起来。

加上了Scrapy-Redis之后的架构变成了：

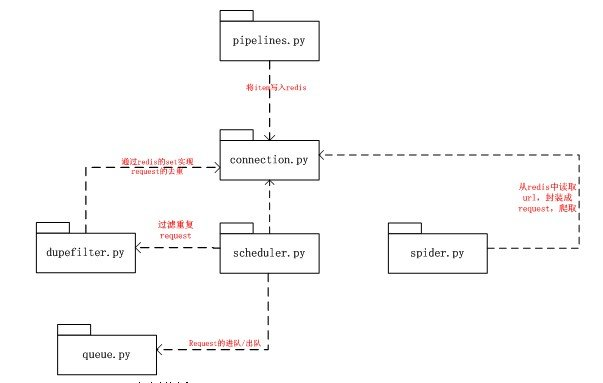


scrapy和scrapy-redis的主要区别在哪里？个人认为，scrapy和scrapy-redis不应该讨论区别。scrapy 是一个通用的爬虫框架，其功能比较完善，可以帮你迅速的写一个简单爬虫，并且跑起来。scrapy-redis是为了更方便地实现scrapy分布式爬取，而提供了一些以redis为基础的组件（注意，scrapy-redis只是一些组件，而不是一个完整的框架）。你可以这么认为，scrapy是一工厂，能够出产你要的spider。而scrapy-redis是其他厂商为了帮助scrapy工厂更好的实现某些功能而制造了一些设备，用于替换scrapy工厂的原设备。所以要想跑分布式，先让scrapy工厂搭建起来，再用scrapy-redis设备去更换scrapy的某些设备。那么这些scrapy-redis组件有什么突出特点呢？他们使用了redis数据库来替换scrapy原本使用的队列结构（deque），换了数据结构，那么相应的操作当然都要换啦，所以与队列相关的这些组件都做了更换

**Scrapy-Redis提供的组件:**

* connection.py 负责根据setting中配置实例化redis连接。被dupefilter和scheduler调用，总之涉及到redis存取的都要使用到这个模块。
* dupefilter.py 负责执行requst的去重，实现的很有技巧性，使用redis的set数据结构。但是注意scheduler并不使用其中用于在这个模块中实现的dupefilter键做request的调度，而是使用queue.py模块中实现的queue。当request不重复时，将其存入到queue中，调度时将其弹出。
* queue.py其作用如dupefilter.py所述，但是这里实现了三种方式的queue：FIFO的SpiderQueue，SpiderPriorityQueue，以及LIFI的SpiderStack。默认使用的是第二种
* picklecompat.py这里实现了loads和dumps两个函数，其实就是实现了一个serializer，因为redis数据库不能存储复杂对象（value部分只能是字符串，字符串列表，字符串集合和hash，key部分只能是字符串），所以我们存啥都要先串行化成文本才行。这里使用的就是python的pickle模块，一个兼容py2和py3的串行化工具。这个serializer主要用于一会的scheduler存reuqest对象，item pipeline的串行化默认用的就是json。
* pipelines.py这是是用来实现分布式处理(毕设这里是单机不做分布式)的作用。它将Item存储在redis中以实现分布式处理。另外可以发现，同样是编写pipelines，在这里的编码实现不同于文章中所分析的情况，由于在这里需要读取配置，所以就用到了from\_crawler()函数。
* cheduler.py 此扩展是对scrapy中自带的scheduler的替代（在settings的SCHEDULER变量中指出），正是利用此扩展实现crawler的分布式调度。其利用的数据结构来自于queue中实现的数据结构。
* spider.py 设计的这个spider从redis中读取要爬的url，然后执行爬取，若爬取过程中返回更多的url，那么继续进行直至所有的request完成。之后继续从redis中读取url，循环这个过程。

**3.4.2组件间关系:**



最后总结一下scrapy-redis的总体思路：这个工程通过重写scheduler和spider类，实现了调度、spider启动和redis的交互。实现新的dupefilter和queue类，达到了判重和调度容器和redis的交互，因为每个主机上的爬虫进程都访问同一个redis数据库，所以调度和判重都统一进行统一管理，达到了分布式爬虫的目的。

当spider被初始化时，同时会初始化一个对应的scheduler对象，这个调度器对象通过读取settings，配置好自己的调度容器queue和判重工具dupefilter。每当一个spider产出一个request的时候，scrapy内核会把这个reuqest递交给这个spider对应的scheduler对象进行调度，scheduler对象通过访问redis对request进行判重，如果不重复就把他添加进redis中的调度池。当调度条件满足时，scheduler对象就从redis的调度池中取出一个request发送给spider，让他爬取。当spider爬取的所有暂时可用url之后，scheduler发现这个spider对应的redis的调度池空了，于是触发信号spider\_idle，spider收到这个信号之后，直接连接redis读取strart url池，拿去新的一批url入口，然后再次重复上边的工作。

**4.1.1微博移动版Web的分析**

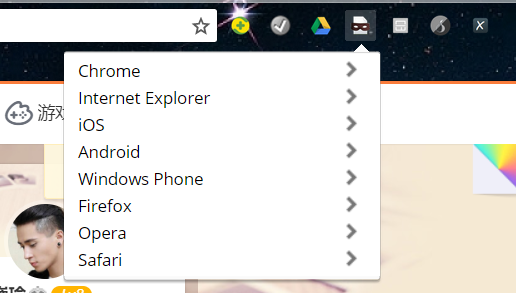
玩过爬虫的朋友应该都清楚，爬虫难度：www > m > wap （www是PC端，m和wap是移动端，现在的智能手机一般用的是m站，部分老手机用的还是wap），原因也很简单，现在的网站越来越多地使用AJAX加载，反爬虫机制也厉害。而像wap这种移动端网站限制比较小，网页结构也简单，我们获取、解析起来都简单很多，理论上速度也会快很多。所以如果允许的话我们尽量采用wap站抓取。



可能很多刚接触爬虫的朋友也想从wap爬取，但不知道怎么做。例如用PC端浏览器打开 http://weibo.com 在登录的时候会自动跳回m域名网站，甚至用requests打开网页时会返回403错误。   
这是因为网站服务器会根据你的浏览器表头判断你是从哪个平台发送的请求，识别到PC端的请求会给你作相应处理。所以我们只需要修改一下浏览器表头（User-Agent）即可。

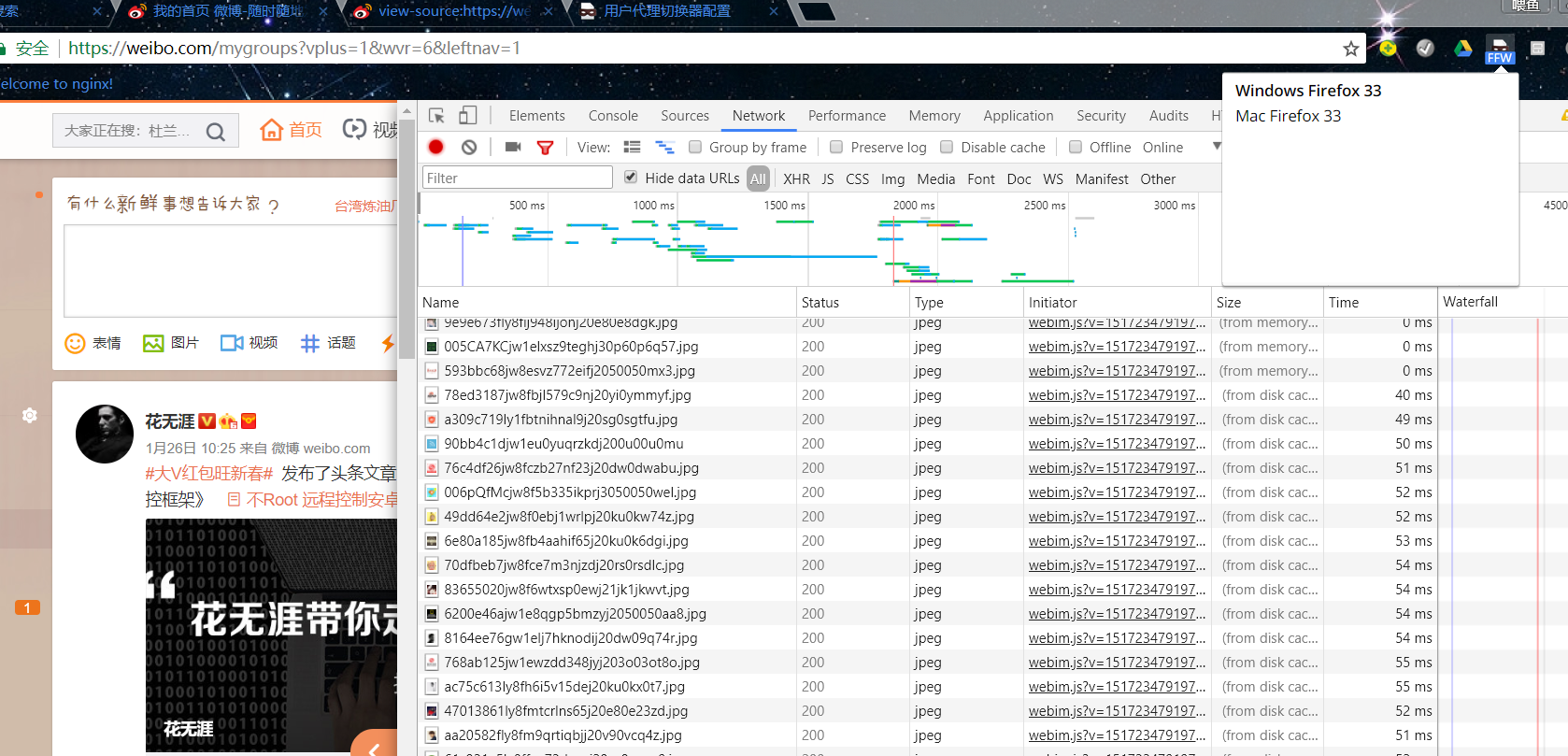
如果是爬虫程序，只需要带上旧版手机浏览器的User-Agent即可（例如：”Mozilla/5.0 (Linux; U; Android 2.3.6; en-us; Nexus S Build/GRK39F) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1”）。

然而我们只看程序返回的response内容并不舒爽，我们还想在PC端用浏览器模拟手机浏览器那样打开网页，怎么办？   
我们只需要把PC浏览器的User-Agent改成手机的User-Agent即可。   
例如我用的是Chrome50，修改浏览器的User-Agent有两种办法：一种是安装一个插件——User-agent Switcher，另一种是直接修改浏览器的表头（仅当前页面有效）。

User-agent Switcher插件：   


直接修改浏览器的User-Agent：

插件改为windows 火狐浏览器环境：



就新浪微博而言，打开一个微博用户的个人首页，wap站直接返回一个HTML文件，并不需要加载JS和CSS，而且格式、编码都很正常；而m站返回的内容格式比较混乱，用xpath解析不了（也有可能是我的程序有问题），而且使用的是Unicode编码格式。

**4.2.1User-Agent伪装**

User Agent中文名为用户代理，简称 UA，它是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。

那么，设置User Agent有什么用呢？

比如你在做移动APP开发时，有个网站在你手机端APP打开之后和直接在浏览器中打开之后看到的东西不一样，网页会根据UA判断你是app打开的还是直接浏览器打开的，如果没有找到事先商定的UA信息就认定这个链接是在浏览器打开的，如果查询到事先商定的UA就是用app打开的，两种情况可以让你看到不同的东西，做不同的操作，比如有些东西开发者只想让你使用app打开才能看见。

其实简单的说User-Agent就是客户端浏览器等应用程序使用的一种特殊的网络协议，在每次浏览器（邮件客户端/搜索引擎蜘蛛）进行 HTTP 请求时发送到服务器，服务器就知道了用户是使用什么浏览器（邮件客户端/搜索引擎蜘蛛）来访问的。既然是人为规定的协议，那么就是说不管什么浏览器，默认的UA都是可以更改的。有时候为了达到一些不可告人的目的，我就经常需要更改一下UA的设置，比如伪装一下浏览器类型，比如把浏览器伪装成手机上网。

**1．）浏览器的UA字串的标准格式**：

浏览器标识 (操作系统标识; 加密等级标识; 浏览器语言) 渲染引擎标识版本信息。但各个浏览器有所不同。

1、浏览器标识  
出于兼容及推广等目的，很多浏览器的标识相同，因此浏览器标识并不能说明浏览器的真实版本，真实版本信息在 UA 字串尾部可以找到。

2、操作系统标识



3、加密等级标识

N: 表示无安全加密  
I: 表示弱安全加密  
U: 表示强安全加密

4、浏览器语言  
在首选项 > 常规 > 语言中指定的语言

5、渲染引擎  
显示浏览器使用的主流渲染引擎有：Gecko、WebKit、KHTML、Presto、Trident、Tasman等，格式为：渲染引擎/版本信息

6、版本信息  
显示浏览器的真实版本信息，格式为：浏览器/版本信息

**2．）爬虫程序中的user-agent：**

**在user\_agent.py模块中：**

**import** random  
  
agents = [  
 **"Mozilla/5.0 (Linux; U; Android 2.3.6; en-us; Nexus S Build/GRK39F) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Avant Browser/1.2.789rel1 (http://www.avantbrowser.com)"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/532.5 (KHTML, like Gecko) Chrome/4.0.249.0 Safari/532.5"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US) AppleWebKit/532.9 (KHTML, like Gecko) Chrome/5.0.310.0 Safari/532.9"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/534.7 (KHTML, like Gecko) Chrome/7.0.514.0 Safari/534.7"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/534.14 (KHTML, like Gecko) Chrome/9.0.601.0 Safari/534.14"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.14 (KHTML, like Gecko) Chrome/10.0.601.0 Safari/534.14"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.20 (KHTML, like Gecko) Chrome/11.0.672.2 Safari/534.20"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.27 (KHTML, like Gecko) Chrome/12.0.712.0 Safari/534.27"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0.782.24 Safari/535.1"**,  
 **"Mozilla/5.0 (Windows NT 6.0) AppleWebKit/535.2 (KHTML, like Gecko) Chrome/15.0.874.120 Safari/535.2"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.36 Safari/535.7"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.0 x64; en-US; rv:1.9pre) Gecko/2008072421 Minefield/3.0.2pre"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.10) Gecko/2009042316 Firefox/3.0.10"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.0; en-GB; rv:1.9.0.11) Gecko/2009060215 Firefox/3.0.11 (.NET CLR 3.5.30729)"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 GTB5"**,  
 **"Mozilla/5.0 (Windows; U; Windows NT 5.1; tr; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR 3.5.30729; .NET4.0E)"**,  
 **"Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1"**,  
 **"Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1"**,  
 **"Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firefox/5.0"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0a2) Gecko/20110622 Firefox/6.0a2"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:7.0.1) Gecko/20100101 Firefox/7.0.1"**,  
 **"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0b4pre) Gecko/20100815 Minefield/4.0b4pre"**,  
 **"Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0 )"**,  
 **"Mozilla/4.0 (compatible; MSIE 5.5; Windows 98; Win 9x 4.90)"**,  
 **"Mozilla/5.0 (Windows; U; Windows XP) Gecko MultiZilla/1.6.1.0a"**,  
 **"Mozilla/2.02E (Win95; U)"**,  
 **"Mozilla/3.01Gold (Win95; I)"**,  
 **"Mozilla/4.8 [en] (Windows NT 5.1; U)"**,  
 **"Mozilla/5.0 (Windows; U; Win98; en-US; rv:1.4) Gecko Netscape/7.1 (ax)"**,  
 **"HTC\_Dream Mozilla/5.0 (Linux; U; Android 1.5; en-ca; Build/CUPCAKE) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (hp-tablet; Linux; hpwOS/3.0.2; U; de-DE) AppleWebKit/534.6 (KHTML, like Gecko) wOSBrowser/234.40.1 Safari/534.6 TouchPad/1.0"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; en-us; sdk Build/CUPCAKE) AppleWebkit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.1; en-us; Nexus One Build/ERD62) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; Nexus One Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; en-us; htc\_bahamas Build/CRB17) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.1-update1; de-de; HTC Desire 1.19.161.5 Build/ERE27) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; Sprint APA9292KT Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; de-ch; HTC Hero Build/CUPCAKE) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; ADR6300 Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.1; en-us; HTC Legend Build/cupcake) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; de-de; HTC Magic Build/PLAT-RC33) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1 FirePHP/0.3"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.6; en-us; HTC\_TATTOO\_A3288 Build/DRC79) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.0; en-us; dream) AppleWebKit/525.10 (KHTML, like Gecko) Version/3.0.4 Mobile Safari/523.12.2"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; en-us; T-Mobile G1 Build/CRB43) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari 525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.5; en-gb; T-Mobile\_G2\_Touch Build/CUPCAKE) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.0; en-us; Droid Build/ESD20) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; Droid Build/FRG22D) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.0; en-us; Milestone Build/ SHOLS\_U2\_01.03.1) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.0.1; de-de; Milestone Build/SHOLS\_U2\_01.14.0) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 3.0; en-us; Xoom Build/HRI39) AppleWebKit/525.10 (KHTML, like Gecko) Version/3.0.4 Mobile Safari/523.12.2"**,  
 **"Mozilla/5.0 (Linux; U; Android 0.5; en-us) AppleWebKit/522 (KHTML, like Gecko) Safari/419.3"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.1; en-gb; dream) AppleWebKit/525.10 (KHTML, like Gecko) Version/3.0.4 Mobile Safari/523.12.2"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.0; en-us; Droid Build/ESD20) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.1; en-us; Nexus One Build/ERD62) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; Sprint APA9292KT Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-us; ADR6300 Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 2.2; en-ca; GT-P1000M Build/FROYO) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 3.0.1; fr-fr; A500 Build/HRI66) AppleWebKit/534.13 (KHTML, like Gecko) Version/4.0 Safari/534.13"**,  
 **"Mozilla/5.0 (Linux; U; Android 3.0; en-us; Xoom Build/HRI39) AppleWebKit/525.10 (KHTML, like Gecko) Version/3.0.4 Mobile Safari/523.12.2"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.6; es-es; SonyEricssonX10i Build/R1FA016) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
 **"Mozilla/5.0 (Linux; U; Android 1.6; en-us; SonyEricssonX10i Build/R1AA056) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"**,  
]  
  
**print** random.choice(agents)

* + 1. **HTML页面解析:**

HTML的标记总是封装在由小于一号(<)和大于号(>)构成的一对尖括号

之中

1、单标记

某些标记称为“单标记”，因为它只需单独使用就能完整地表达意思，这类

标记的语法是:

<标记>

最常用的单标记是<P>，它表示一个段落(Paragraph的结束，并在段落

后面加一空行。

2、双标记

另一类标记称为“双标记”，它由“始标记”和“尾标记”两部分构成，必须成

对使用，其中始标记告诉Web浏览器从此处开始执行该标记所表示的功能，

而尾标记告诉Web浏览器在这里结束该功能。始标记前加一个斜杠(/)即成

为尾标记。这类标记的语法是:

<标记>内容</标记>

其中“内容”部分就是要被这对标记施加作用的部分。例如你想突出对某段

文字的显示，就将此段文字放在一对<EM> </EM>标记中:

<EM>text to emphasize</EM>

3、标记属性

许多单标记和双标记的始标记内可以包含一些属性，其语法是:

<标记属性1属性2属性3 ... >

各属性之间无先后次序，属性也可省略(即取默认值)，例如单标记<HR>

如下图微博的HTML页面:



4.页面链接的提取

页面链接的提取，只有当下载的文档是txt/html格式的页面时在有必要分

析。页面的类型可由该网页头信息分析得出，有些站点返回的应答信息格式不

完整，此时必须通过分析页面URL中的文件扩展名来判别类型。遇到带连接

的标志如<A> , <AREA> , <FRAME>等，就从标记结构的属性中找出目标

URL，并从成对的该标记之间提取正文作为该连接的说明文字(扩展原数

据)。这两个数据就代表了该链接。

一个页面连接提取的过程总结如下:

1判断文件的类型是否是txt/html如果不是则跳过，如果是继续分析。

2读取文件利用正则表达式匹配的方法，寻找标签<a href= >, <area href=>，<base href=>，<frame src=>，<img src=>，<body background=>，<appletcode=>等，记录下该URL 。

3将记录下来的URL按照预定的格式统一完整。以备进行下一步的URL分析。

**4.3.2正则表达式简介**

HTTP协议本质上就是传输HTML所以在写爬虫的时候会遇到许多HTML页面，我们需要从中找出我们需要的内容，如何检索出我们需要的内容是一个关键，在这里可能就需要用到正则表达式。通过正则表达式我们可以匹配出我们需要的内容，接下来就是对正则表达式在爬虫中的应用的介绍。

正则表达式是用来匹配字符串非常强大的工具，在其他编程语言中同样有正则表达式的概念，Python同样不例外，利用了正则表达式，我们想要从返回的页面内容提取出我们想要的内容就易如反掌了。

比如我们想抓取所有图片的 URL 链接，非常直接的做法就是用 findAll("img") 抓取所图片，对吗？但是，有个问题。除了那些明显“多余的”图片（比如， LOGO） 之外，式的网站里都有一些隐藏图片， 用于网页布局留白和元素对齐的空白图片，以及一些不易察觉到的图片标签。总之，你不能仅用商品图片来统计网页上所有的图片。而且网页的布局也可能会变化， 或者，因为某些原因，我们不想通过图片在网页中的位来查找标签。 那么当你想抓取随机分布在网站里的某个元素或数据时，就会出现问题。如，一些网页的最上面可能有一张商品图片，但是在另一些网页上没有。解决这类问题的办法， 就是直接定位那些标签来查找信息

**4.3.3正则表达式相关注解**

(1)数量词的贪婪模式与非贪婪模式:正则表达式通常用于在文本中查找匹配的字符串。Python里数量词默认是贪婪的（在少数语言里也可能是默认非贪婪），总是尝试匹配尽可能多的字符；非贪婪的则相反，总是尝试匹配尽可能少的字符。例如：正则表达式”ab\*”如果用于查”abbbc”，将找到”abbb”。而如果使用非贪婪的数量词”ab\*?”，将找到”a”。

注：我们一般使用非贪婪模式来提取。

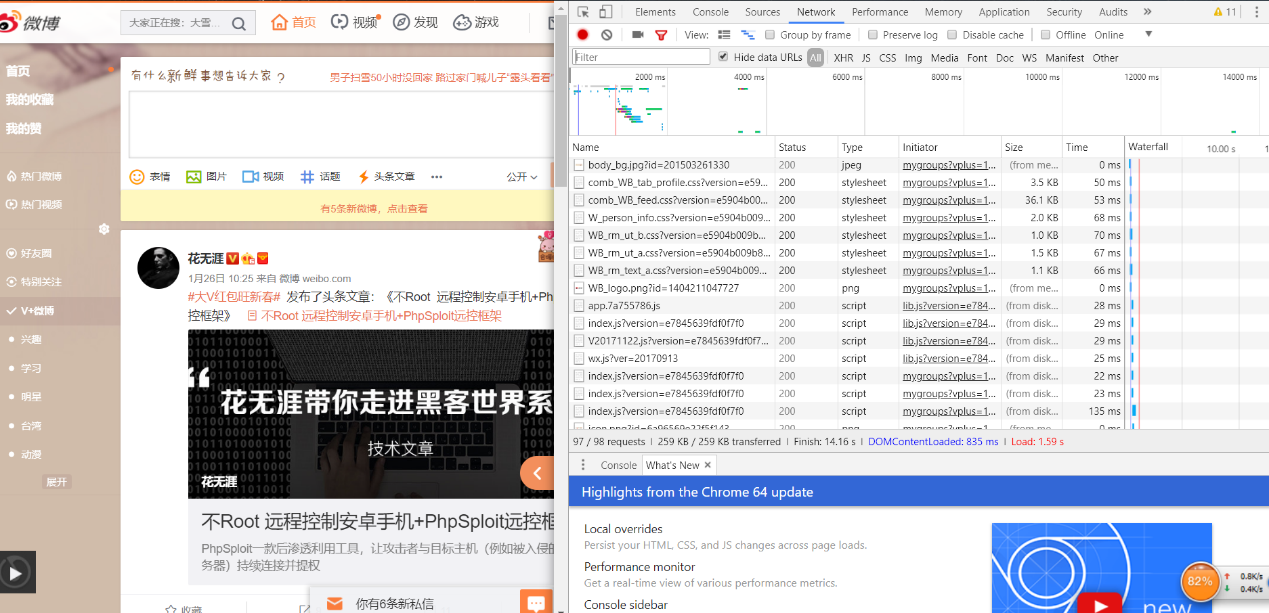
（2）反斜杠问题:与大多数编程语言相同，正则表达式里使用”\”作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符”\”，那么使用编程语言表示的正则表达式里将需要4个反斜杠”\\\\”：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r”\\”表示。同样，匹配一个数字的”\\d”可以写成r”\d”。有了原生字符串，妈妈也不用担心是不是漏写了反斜杠，写出来的表达式也更直观勒。这是Python的re库正则表达式的一些语法:



**4.3.4Chrome编写正则表达式:**



用Chrome抓包:



由于版面原因这里只写出爬取信息的代码:

**def** parse\_tweets(self, response):  
 *""" 抓取微博数据 """* selector = Selector(response)  
 ID = re.findall(**'(\d+)/profile'**, response.url)[0]  
 divs = selector.xpath(**'body/div[@class="c" and @id]'**)  
 **for** div **in** divs:  
 **try**:  
 tweetsItems = TweetsItem()  
 id = div.xpath(**'@id'**).extract\_first() *# 微博ID* content = div.xpath(**'div/span[@class="ctt"]//text()'**).extract() *# 微博内容* cooridinates = div.xpath(**'div/a/@href'**).extract() *# 定位坐标* like = re.findall(**'赞\[(\d+)\]'**.decode(**'utf8'**), div.extract()) *# 点赞数* transfer = re.findall(**'转发\[(\d+)\]'**.decode(**'utf8'**), div.extract()) *# 转载数* comment = re.findall(**'评论\[(\d+)\]'**.decode(**'utf8'**), div.extract()) *# 评论数* others = div.xpath(**'div/span[@class="ct"]/text()'**).extract() *# 求时间和使用工具（手机或平台）* tweetsItems[**"\_id"**] = ID + **"-"** + id  
 tweetsItems[**"ID"**] = ID  
 **if** content:  
 tweetsItems[**"Content"**] = **" "**.join(content).strip(**'[位置]'**.decode(**'utf8'**)) *# 去掉最后的"[位置]"* **if** cooridinates:  
 cooridinates = re.findall(**'center=([\d.,]+)'**, cooridinates[0])  
 **if** cooridinates:  
 tweetsItems[**"Co\_oridinates"**] = cooridinates[0]  
 **if** like:  
 tweetsItems[**"Like"**] = int(like[0])  
 **if** transfer:  
 tweetsItems[**"Transfer"**] = int(transfer[0])  
 **if** comment:  
 tweetsItems[**"Comment"**] = int(comment[0])  
 **if** others:  
 others = others[0].split(**'来自'**.decode(**'utf8'**))  
 tweetsItems[**"PubTime"**] = others[0].replace(**u"\xa0"**, **""**)  
 **if** len(others) == 2:  
 tweetsItems[**"Tools"**] = others[1].replace(**u"\xa0"**, **""**)  
 **yield** tweetsItems  
 **except** Exception, e:  
 **pass** url\_next = selector.xpath(**'body/div[@class="pa" and @id="pagelist"]/form/div/a[text()="下页"]/@href'**.decode(**'utf8'**)).extract()  
 **if** url\_next:  
 **yield** Request(url=self.host + url\_next[0], callback=self.parse\_tweets, dont\_filter=True)

**4.4.1去重原理**

“去重”是日常工作中会经常用到的一项技能，在爬虫领域更是常用，并且规模一般都比较大。去重需要考虑两个点：去重的数据量、去重速度。为了保持较快的去重速度，一般选择在内存中进行去重。

简单来说就是通过Hash函数存储网络爬虫的遍历轨迹，并规定某一Web页被遍历过，则在哈希表中的相应槽位填充1，否则填充0。也就是说，此类方式用1或。的值表示Web页是否被爬虫抓取过。在具体实现过程中，哈希函数起到至关重要的作用，目前一般使用MD5 ( )函数，将网页文件的地址即URL字符串转换为128位散列值。

MD5就是将任意长度的消息转换成 128位固定长度的消息摘要的函数，显然，MD5 ( )函数产生的值很大，为2个不同的数，需要的内存空间巨大。因此，在实际处理

中还要将MD5()函数的值进行模运算映射到哈希表中。其公式可设为:

MD5 (URL) MOD N

其中，URL为抓取的地址，N为存储哈希表的位长。通过该式的转换，可使输入的URL地址被映射到大小为N的哈希表的某个位上，以便确定其地址是否被抓取

过。

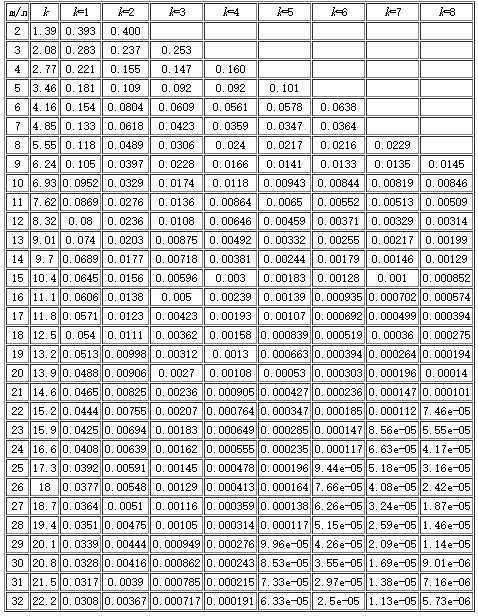
为了解决重复搜集网页的问题，可以定义两个数据库:“未爬行的U1u库”和“已爬行的URL库”。“未爬行的URL库”存储待访问队列的URL“已爬行的U1u库”存储已遍历过的URL。对于已访问过的、未访问过的URL利用MD5(URL)函数分别作MD5摘要，以获取其惟一标识，并建立两个集合。新解析出的URL，首先根据已经访问过URL集合判断是否已抓取过.如没有被抓取.则放入“未爬行的URL数据库”中.否则放入“已爬行的URI库”

**4.4.2 Redis去重**

* 数据量不大时，可以直接放在内存里面进行去重，例如python可以使用set()进行去重。
* 当去重数据需要持久化时可以使用redis的set数据结构。
* 当数据量再大一点时，可以用不同的加密算法先将长字符串压缩成 16/32/40 个字符，再使用上面两种方法去重；
* 当数据量达到亿（甚至十亿、百亿）数量级时，内存有限，必须用“位”来去重，才能够满足需求。Bloomfilter就是将去重对象映射到几个内存“位”，通过几个位的 0/1值来判断一个对象是否已经存在。
* 然而Bloomfilter运行在一台机器的内存上，不方便持久化（机器down掉就什么都没啦），也不方便分布式爬虫的统一去重。如果可以在Redis上申请内存进行Bloomfilter，以上两个问题就都能解决了。

本文即是用Python，基于Redis实现Bloomfilter去重。

1. Bloomfilter算法如何使用位去重，这个百度上有很多解释。简单点说就是有几个seeds，现在申请一段内存空间，一个seed可以和字符串哈希映射到这段内存上的一个位，几个位都为1即表示该字符串已经存在。插入的时候也是，将映射出的几个位都置为1。



1. 需要提醒一下的是Bloomfilter算法会有漏失概率，即不存在的字符串有一定概率被误判为已经存在。这个概率的大小与seeds的数量、申请的内存大小、去重对象的数量有关。下面有一张表，m表示内存大小（多少个位），n表示去重对象的数量，k表示seed的个数。例如我代码中申请了256M，即1<<31（m=2^31，约21.5亿），seed设置了7个。看k=7那一列，当漏失率为8.56e-05时，m/n值为23。所以n = 21.5/23 = 0.93(亿），表示漏失概率为8.56e-05时，256M内存可满足0.93亿条字符串的去重。同理当漏失率为0.000112时，256M内存可满足0.98亿条字符串的去重。
2. 基于Redis的Bloomfilter去重，其实就是利用了Redis的String数据结构，但Redis一个String最大只能512M，所以如果去重的数据量大，需要申请多个去重块（代码中blockNum即表示去重块的数量）。
3. 代码中使用了MD5加密压缩，将字符串压缩到了32个字符（也可用hashlib.sha1()压缩成40个字符）。它有两个作用，一是Bloomfilter对一个很长的字符串哈希映射的时候会出错，经常误判为已存在，压缩后就不再有这个问题；二是压缩后的字符为 0~f 共16中可能，我截取了前两个字符，再根据blockNum将字符串指定到不同的去重块进行去重。

反爬技术

我在设计，改进爬虫的时候感觉 难度最大的就是与反扒技术的博弈过程，反扒技术也是贯穿全文，在这里就不再冗余在用到的技术都相应的解释介绍比如伪装http的报文头修改User-agent爬取移动版Web,用多线程做的Cookies池,IP地址代理池,买微博的账号,破解JavaScript环境,打验证码等等!

4.5.1**企业中常见的反爬技术手段**

那些厂商为何确保自己的利益是怎么样进行反爬的.实话说，与对方的反爬措施做斗争可以称得上开发过程中为数不多的趣味了，同样也可以说是一种折磨了.，你会见到各种奇妙无比，匪夷所思，令人拍案叫绝的反爬手段。同时你还得绞尽脑汁的去绕过这些.在前面的章节中，我们提到爬虫是我们获取互联网信息的一种重要手段,那么有矛必有盾.一方面是千方百计想要爬取数据的你，另一方面是誓死捍卫自己数据的守护者.一场较量再所难免.......

回到正题，目前市面上的反爬措施很多，也有很多奇妙的，奇特的，恶毒的都有.我们从本质出发，目前市面的大概可以分成这么几个大类.

* 根据访问日志，流量，爬取行为以及各种相关算法来判定该行为是不是爬虫所为，从而触发黑名单或者蜜罐系统。例如最为经典的封禁IP和封禁账号.这一类用途最广，防范水平要看其判断策略写的如何.
* 利用ajax技术，通过js脚本实现动态渲染的后台传输类.比如说通过JSON实现前端与后端通信.这种有人也许会说这也能算反爬措施?但这种至少你读取网页源码时是得不到数据的，只能通过抓包等方式解决.这一类基本在需要交互的网站中运用的许多.抓包一般可以用burpsuite,fiddle这类软件进行抓包.
* 通过浏览器环境，在本地运算js，最终得到所需的数据的本地js类.因为现在浏览器都有内核从而实现了自带javascript环境.在本地环境中即时运算从而得到数据.从我的经验来看，这一类是很难对付的，而且即使有解决方案，但效率很低而且爬虫程序不容易写.
* 各种找不到形容词，只能用奇思妙想来形容的反爬类.一些通过图像混淆，页面无限循环，给爬虫投毒等等行为可以归入此类.

那么我们开始了解常用的几个反爬手段.首先是封禁类.我们经常会遇到爬取过度而导致IP被封或者是账号被封。你知道是怎么一回事么? 在这里，我们将详细的了解下我们在什么情况下会遭到此类惩罚。同时，一些常用的判断规则又是什么？一般来说，我们如果触发了反爬系统所设定的条件的话，就会引发封禁.该手段一般设定的标准有这几种：

* 是否携带有一个正常访问下所有的header头，cookies和正常的User-Agent.针对与这种情况，我们只要通过模仿浏览器发送的包，将我们的爬虫所发送的HTTP请求上都挂上和浏览器发送的请求一样的header头和cookie就行。此类标准由于技术难度低，且反爬简易，现在一般不采用此类.
* 根据行为是否表现的像一次人为的操作，诸如不分昼夜的机械化的发送请求，每次只读取特定内容，相隔固定的秒数分秒不差的，都可以认为是一个robot在操作.该类的话，需要我们模拟出一个正常的人类读取网页时的行为，比如使用乱序时间，将爬虫每轮间隔的爬取时间使用随机时间等待这些方法，这么做的话，比较不容易触发规则.
* 在网页源代码中设置一些正常访问看不到的标签，然而爬虫会触发到的标签，从而达到封禁的效果。如果在试验过程中发现这些措施的话，可以通过精准定位的方式（后面所讲的Xpath,selector）来避开那些有毒的代码块.

封禁的标准千千万万，阈值也是每个网站都不同.我们应当在正式爬取之前，先写几个测试用意的爬虫，用来试探该网站的反爬策略，从而决定我们的反反爬措施. 当然还有最终的解决办法，那么就是看封禁的是什么。大多数情况下，封禁的是IP地址，那么这时候我们可以去通过代理IP来无视掉封禁IP的情况.代理IP也就是代理网络用户去取得网络信息.我们如果通过代理IP来获取信息的话，那么就可以无视掉依靠IP封禁的反爬策略.一般基本上主流的HTTP库都提供代理功能.但有利必有弊，通过代理IP进行中转固然可以无视封禁，但代理节点的稳定性和速度就直接影响到整体爬虫的效率.如果该代理节点时常掉线，延时居高不下，那么从最终效果上来看就和不挂代理依靠低速运行来躲避阈值的爬虫一样.如何维持一个延迟低，可用率高的代理IP池不在本文叙述范围内，请自行寻找资料。

总而言之，封禁类中的如何判断是否是爬虫是反爬系统的核心关键所在，确定了是爬虫之后，那么就不光是封禁，可能会有各种手段惩罚或者戏弄你.比如后面讲的投毒，无限循环，伪装404页面等待......

这里只介绍怎么打验证码对JavaScript反爬的技巧与json的使用破解AJAX两个小技巧，其他部分因为版面等原因都在论文其他部分讲解这里就不再赘言。

**4.5.2 对抗AJAX技术**

利用Web开发应用程序因其易于部署、节省成本的特点而逐渐成为技术的主流，浏览器将是未来唯一需要的客户端。J2EE Web技术己经从一个静态的内容提供环境迅速发展为一个在Intranet和Internet上运行的动态业务应用程序的靠平台。过去几年中在后端(后端机制，backend mechanics)投入了太多的时间，而在Web用户界面对用户交互和响应灵敏方面却投入不足。用户在强大的业务逻辑背后，还忍受着“提交-响应-等待-刷新”的同步运行机制，不管在页面呈现上的变化多么小，都需要耐心地等待服务器将整个页面重新发送给客户端。为了构建更为动态和响应更灵敏的Web应用程序，实现浏览器和服务器的异步并行处理，减轻服务器端负担，Jesse James Garrett提出了AJAX(Asynchronous JavaScript and XML)这一新的概念。AJAX是一种创建交互式网页应用的网页开发技术，由多种技术组合而成，包括:

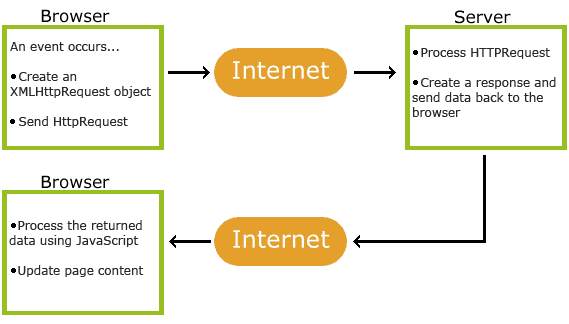
基于XHTML和CSS标准的表示;

使用Document Object Model进行动态显示和交互;

使用XMLHttpRequest对象与服务器进行异步通信;

使用JavaScript绑定一切。

与传统的Web开发不同，AJAX并不是以一种基于静态页面的方式来看待Web应用的。从AJAX的角度看来，Web应用应由少量的页面组成，其中每个页面是一个更小型的AJAX应用，每个页面包括有一些使用JavaScript开发的AJAX组件。这些组件使用XMLHttpRequest对象以异步的方式与服务器通信，从服务器获取需要的数据后使用DOM API来更新页面内容。



如图，AJAX的工作原理相当于在用户和服务器之间加了一个中间层一一AJAX引擎，使用户操作与服务器响应异步化一一并不是所有的用户请求都提交给服务器。一些数据验证和处理由AJAX来做而不必交给服务器处理，只有确定需要从服务器读取新数据时再由AJAX引擎向服务器提交请求。在使用AJAX引擎后，用户从感觉上几乎所有的操作都会很快响应，并且没有页面重载(白屏)的AJAX引擎实际上是一个比较复杂的JavaScript应用程序，用来处理用户请求然后根据需要动态读写服务器和更改DOM内容。以前为了使网页能无缝化重构，

也就是在页面己经下载完毕后改变页面内容，开发人员一直通过JavaScript和DOM来实现。但是要使网页真正动态起来，不仅需要内部的互动，还需要从外部获取数据，在XMLHttpRequest对象出现以前是让用户来输入数据并通过DOM来改变网页内容的，但现在 XMLHttpRequest对象可以在不重新载入页面的情况下读写服务器上的数据，使用户的输入达到最少。AJAX引擎允许用户与应用软件之间的交互过程异步进行，独立于用户与网络服务器间的交流。所以就算当请求和响应在引擎和服务器之间被来回传送，用户还是能与应用程序交互。从而可以用JavaScript调用AJAX引擎来代替产生一个HTTP的用户动作，同时引擎本身能够立即处理不需要请求服务器的动作，就像桌面应用程序处理一样，例如在内存中编辑数据，验证数据等。

AJAX提升了用户体验，但同时给网络爬虫的抓取带来了极大的挑战，因为很

多页面内容都是通过异步向服务器发请求后得到，而传统的爬虫仅仅分析静态

HTML页面里的超链接，显然，在利用AJAX技术的网站中使用传统爬虫抓取网页是

不够的，抓取的信息很不完善，这将导致搜索引擎查询结果准确率的下降，因此

有必要设计新的爬虫，能够支持AJAX，并且截获异步调用返回的结果，以及分析

由此结果生成的新内容，尽可能多的抓取页面信息。

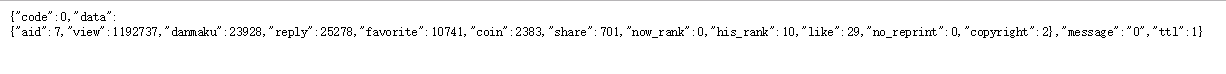
**4.5.3JSON格式爬取**

JSON：JavaScript 对象表示法（JavaScript Object Notation），JSON 是存储和交换文本信息的语法，类似 XML。JSON 比 XML 更小、更快，更易解析**。**JSON 是轻量级的文本数据交换格式同时 JSON 具有自我描述性，更易理解 JSON 使用 JavaScript 语法来描述数据对象，但是 JSON 仍然独立于语言和平台，JSON解析器和 JSON 库支持许多不同的编程语言。

由于AJAX提升了用户体验，但同时给网络爬虫的抓取带来了极大的挑战，因为很

多页面内容都是通过异步向服务器发请求后得到，而传统的爬虫仅仅分析静态HTML页面里的超链接，使用传统爬虫抓取网页是不够的，抓取的信息很不完善，这将导致搜索引擎查询结果准确率的下降，因此有必要设计新的爬虫，能够支持AJAX。

通过JSON进行前后端交互的爬取技巧解决上面AJAX的问题.JSON交互的话，基本就是通过ajax动态加载技术将后台传回的数据通过js渲染的方式放进网页中.由服务端发送一个ajax请求，然后根据后台配置返回相应的JSON数据来进行动态渲染.如果网站是这么运行的，那么页面就无需重新加载整个网页就可以对网页的某部分进行更新。绝大多数都是与后台搭建的服务端API所进行交互的.大概使用这种方式的你都会找到一个界面类似于下图这样的



.这图是微博的api接口，也就是后台与前端进行交互的地方.该技术其实严格意义上不算反爬措辞，相反，由于JSON的便利性，解析JSON要比解析页面文件要来的方便和快速的多.但至少对于才入门只会解析页面的新手来说，还是有一些难度的.

面对这种技术的话，我们应对的方式也很简单，那么就是对网页进行抓包，探知该与此网页进行数据交换的后台服务的地址，然后我们伪造和网页相同的数据包，跳过网页直接请求后台服务，得到所需的数据. 不过这类后台服务一般设有封禁类策略，在请求的时候，要特别注意阀值。目前使用的抓包工具一般是burpsuite,fidder或者Chrome和Firefox的开发人员工具.有关于JSON解析的话，我们可以通过[json](http://www.json.cn/)解析服务完成在线解析，.你通过这一步就可以把上图中的json解析成这样：



**4.5.4验证码处理**

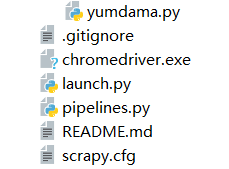
这一小节重点介绍两个利器phantomjs和selenium.通过利用这些工具可以破解本地JS环境的反爬措施.

本地JS环境是个杀器，可以通杀没有本地环境的任意爬虫.其原理是因为浏览器自带有引擎，引擎中包含有js运行环境.所有js脚本都可以在浏览器上直接运行，这一切都是发生在本地浏览器环境内的，没有上述的与后台交互的行为，所有我们不能通过分析抓包得到的报文来绕过这类反爬措施。 所以，这时候我们就得用模拟浏览器内核来绕过这个限制.现在业内一般有两种方式.无图形化的phantomjs和有图形化的selenium.分别属于无头浏览器与webdriver两个阵营.下面我们就分为介绍一下:

* 无头浏览器，具体来说是一个除了图形化界面外其余都与普通浏览器毫无区别的工具。运用无头浏览器，我们可以实现在爬虫运行过程中让js脚本运行在headlessa浏览器中，从而模拟出与一般浏览器一样的页面。然后我们就能如同一般情况下对该网页进行解析来提取数据.我们用目前比较热门的[phantomjs](http://phantomjs.org/)浏览器来举例，phantomjs是一个采用的是Webkit内核与目前的Safari，Chrome等浏览器兼容性十分好的无头浏览器。可以用来模拟所有浏览器操作，被广泛运用于自动化测试和web爬虫中.
* webdriver，此类一般用作自动化测试较多，用于爬虫的一般多用作测试环境.因为无头浏览器没有GUI图形界面，爬取数据还好说，看有无数据返回亦或者数据返回的对错，但如果是要通过点击或者其他行为才能获取到数据的页面，那么没有GUI界面的就无法即使的直到运行的对错，这时候就需要在类似[selenium](http://www.seleniumhq.org/)这种webdriver上进行自动化操作，来模拟无头浏览器上的操作。在爬虫这方面，[selenium](http://www.seleniumhq.org/)等于一个有界面的无头浏览器.

不过如果使用了这种方式来绕过的话，运行性能上会受到严重影响. 因为首先模拟浏览器就很费性能，然后还有本地运算JS更是消耗资源的大户.所以比起其他方法来说的话，这方面毫无疑问的会慢很多.如果存在其他方法的话，请尽量不要用此方法.

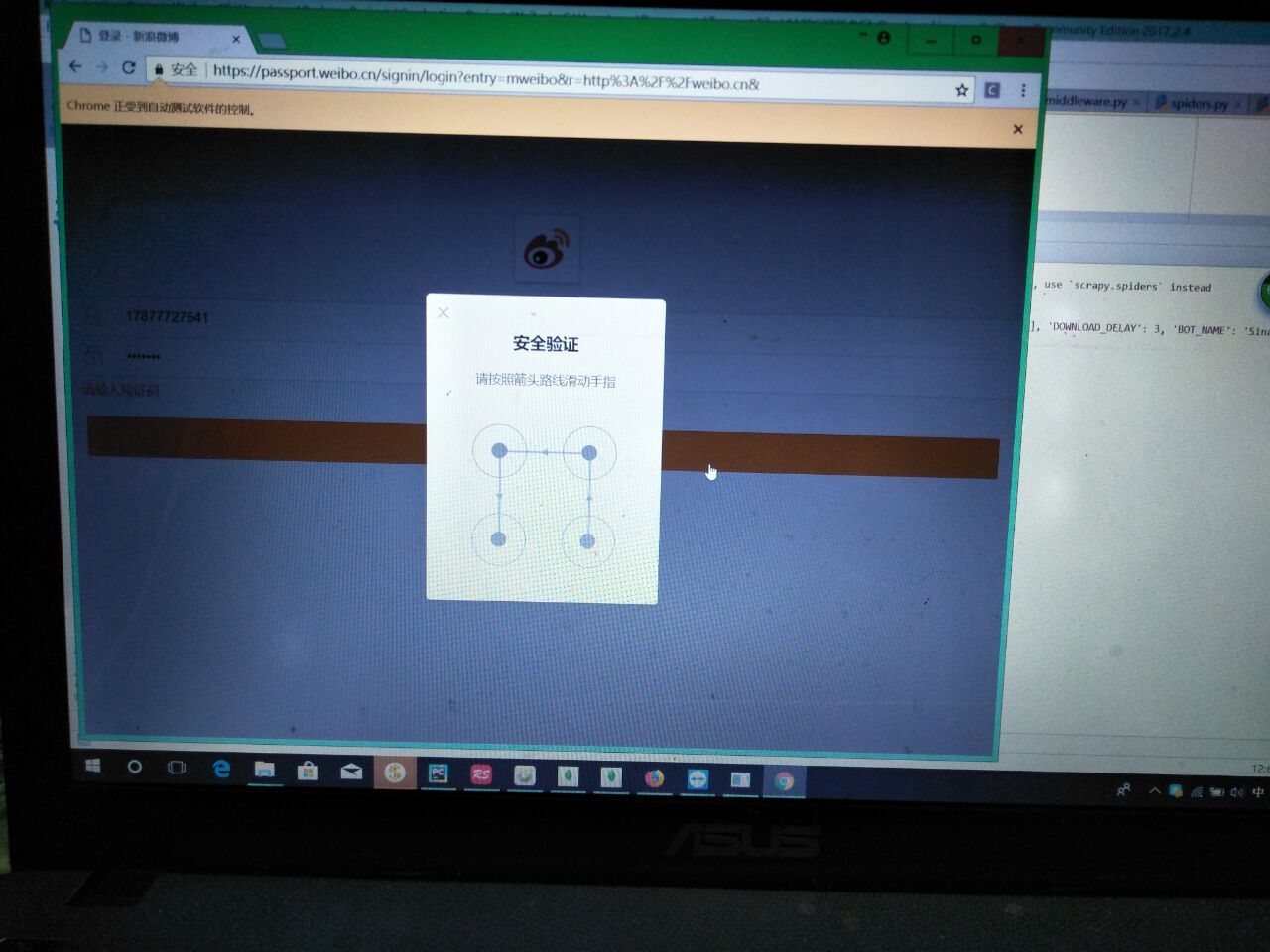
下载PhantomJS.exe，放在python的安装路径。



常用的验证码处理主要有两种：手动输入和打码平台自动填写（手动输入配置简单，打码平台输入适合大规模抓取）还有图形解锁验证码破解。但是！后两者不管是云打码还是图像识别出错率很高，云打码还不支持移动版微博拖动，同时图像识别又太难对于现在的我来说使用，反正没有几个，并且不是搞得多高大上就越好所以用最笨的方法是最见效的同时也是成功率100%的！

手动方式流程：   
1、下载PhantomJS.exe，放在python的安装路径（适合Windows系统，Linux请找百度）。   
2、运行launch.py启动爬虫，中途会要求输入验证码，查看项目路径下新生成的aa.png，输入验证码 回车，即可。

下图为手动打验证码



顺利成功。

**4.6.1 Cookies池简介**

Cookies是因为HTTP协议是无状态的，即服务器不知道用户上一次做了什么，这严重阻碍了交互式Web应用程序的实现。在典型的网上购物场景中，用户浏览了几个页面，买了一盒饼干和两饮料。最后结帐时，由于HTTP的无状态性，不通过额外的手段，服务器并不知道用户到底买了什么。 所以Cookie就是用来绕开HTTP的无状态性的“额外手段”之一。服务器可以设置或读取Cookies中包含信息，借此维护用户跟服务器会话中的状态。

从cookies的定义可以看出，cookies也是可以作为一个验证用户身份的工具，所以可以通过cookies来区别机器和人，所以有一种反爬的策略，就是通过cookies，微博的反爬虫机制就是基于cookies，所以同一个cookies可以重复请求，而同一个IP不带cookies却是不能重复请求，会封IP，但是微博做的还是不够，就是只需要一个cookies，但是微博的cookies所保存的时间短一点，就需要一个cookies池了，定期加入cookies，这样爬取的难度就会增大很多，再厉害一点，就是IP和cookies一起识别。

微博是这AJAX一类的网站，这一类网站的特点和上一种有点类似，但是为什么要分开来说呢？因为上一种情形一般会和源码一起结合出现，而Ajax网站，需要用户构造表单，向服务器要求返回想要的内容，而且这一类网站，一般是几乎所有信息（除了导航栏之类的通用信息）外，都是通过json, xml返回的，所以只需要从这个json包中便能获取所有数据。

**4.6.2构建Cookies池**

Cookie，指某些网站为了辨别用户身份、进行session跟踪而储存在用户本地终端上的数据（通常经过加密）比如说有些网站需要登录后才能访问某个页面，在登录之前，你想抓取某个页面内容是不允许的。那么我们可以利用Urllib2库保存我们登录的Cookie，然后再抓取其他页面就达到目的了关于重写中间件的详细

情况看官方文档：

http://scrapychs.readthedocs.io/zh\_CN/latest/topics/downloadermiddleware.html#scrapy.contrib.downloadermiddleware.DownloaderMiddleware

在项目中新建一个middlewares.py的文件（如果你使用的新版本的Scrapy，在新建的时候会有这么一个文件，直接用就好了）首先导User AgentMiddleware 毕竟我们要重写它啊第一行：定义了一个类UserAgentmiddleware继承自UserAgentMiddleware

第二行：定义了函数process\_request(request, spider)为什么定义这个函数，因为Scrapy每一个request通过中间件都会调用这个方法。下面就需要登陆了。这次我们不用上一篇博文的FromRequest来实现登陆了。我们来使用Cookie登陆。这样的话我们需要重写Cookie中间件！分布式爬虫啊！你不能手动的给每个Spider写一个Cookie吧。而且你还不会知道这个Cookie到底有没有失效。所以我们需要维护一个Cookie池(这个cookie池用redis)。

好！来理一理思路，维护一个Cookie池最基本需要具备些什么功能呢？

* 获取Cookie
* 更新Cookie
* 删除Cookie
* 判断Cookie是否可用进行相对应的操作（比如重试）

好，我们先做前三个对Cookie进行操作。首先我们在项目中新建一个cookies.py的文件用来写我们需要对Cookie进行的操作

首先我们把登陆用的账号密码 以Key:value的形式存入redis数据库。不推荐使用db0（这是Scrapy-redis默认使用的，账号密码单独使用一个db进行存储。）解决第一个问题：获取Cookie：使用requests模块提交表单登陆获得Cookie，返回一个通过Json序列化后的Cookie（如果不序列化，存入Redis后会变成Plain Text格式的，后面取出来Cookie就没法用啦。）第二个问题：将Cookie写入Redis数据库（分布式呀，当然得要其它Spider也能使用这个Cookie了）使用我们上面建立的redis链接获取redis db2中的所有Key(我们设置为账号的哦！)，再从redis中获取所有的Value(我设成了密码哦！)判断这个spider和账号的Cookie是否存在，不存在 则调用get\_cookie函数传入从redis中获取到的账号密码的cookie；保存进redis，Key为spider名字和账号密码，value为cookie。

以下是middlewares.py代码部分：

# encoding=utf-8

import random

from cookies import cookies

from user\_agents import agents

# from getproxies import proxies

import json

class UserAgentMiddleware(object):

""" 换User-Agent """

def process\_request(self, request, spider):

agent = random.choice(agents)

request.headers.setdefault("User-Agent", agent)

# request.headers["User-Agent"] = agent

class CookiesMiddleware(object):

""" 换Cookie """

def process\_request(self, request, spider):

cookie = json.loads(random.choice(cookies))

# print cookie

request.cookies = cookie

以下是cookies.py代码部分：

*import* base64  
*import* requests  
*import* sys  
*import* time  
*from* selenium *import* webdriver  
*from* selenium.webdriver.common.desired\_capabilities *import* DesiredCapabilities  
*from* selenium.webdriver *import* ActionChains  
*from* selenium.webdriver.common.by *import* By  
*from* selenium.webdriver.support.ui *import* WebDriverWait  
*from* selenium.webdriver.support *import* expected\_conditions *as* EC  
*import* logging  
*from* yumdama *import* identify  
*import* json  
  
reload(sys)  
sys.setdefaultencoding('utf-8')  
IDENTIFY = 1 # 验证码输入方式: 1:看截图aa.png，手动输入 2:云打码  
COOKIE\_GETWAY =2 # 0 从https://login.sina.com.cn/sso/login.php?client=ssologin.js(v1.4.18) 获取cookie # 1 从https://weibo.cn/login/获取Cookie # 2 使用chromedriver获取  
dcap = dict(DesiredCapabilities.PHANTOMJS) # PhantomJS需要使用老版手机的user-agent，不然验证码会无法通过  
dcap["phantomjs.page.settings.userAgent"] = (  
 "Mozilla/5.0 (Linux; U; Android 2.3.6; en-us; Nexus S Build/GRK39F) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"  
)  
logger = logging.getLogger(\_\_name\_\_)  
logging.getLogger("selenium").setLevel(logging.WARNING) # 将selenium的日志级别设成WARNING，太烦人  
  
myWeiBo = [  
{'no':'15874173914','psw':'bbx8514'},  
{'no':'15774109579','psw':'bbx4768'},  
{'no':'17877727541','psw':'bbx3464'},  
{'no':'15898562769','psw':'bbx8712'},  
{'no':'18407320608','psw':'bbx2145'},  
{'no':'15973424313','psw':'bbx8431'},  
{'no':'13762898341','psw':'bbx3186'},  
{'no':'18374112533','psw':'bbx9829'},  
{'no':'15274883774','psw':'bbx8748'},  
{'no':'13873384591','psw':'bbx7247'},  
{'no':'13974708834','psw':'bbx2579'},  
{'no':'18474777738','psw':'bbx3957'},  
{'no':'18397779843','psw':'bbx4491'},  
{'no':'15197752390','psw':'bbx1831'},  
{'no':'15273563186','psw':'bbx9756'},

*def* getCookie(*account*, *password*):  
 *if* COOKIE\_GETWAY == 0:  
 *return* get\_cookie\_from\_login\_sina\_com\_cn(*account*, *password*)  
 *elif* COOKIE\_GETWAY == 1:  
 *return* get\_cookie\_from\_weibo\_cn(*account*, *password*)  
 *elif* COOKIE\_GETWAY == 2:  
 *return* get\_cookie\_from\_weibo(*account*, *password*)  
 *else*:  
 logger.error("COOKIE\_GETWAY Error!")  
  
  
*def* get\_cookie\_from\_login\_sina\_com\_cn(*account*, *password*):  
 """ 获取一个账号的Cookie """  
 loginURL = "https://login.sina.com.cn/sso/login.php?client=ssologin.js(v1.4.18)"  
 username = base64.b64encode(*account*.encode("utf-8")).decode("utf-8")  
 postData = {  
 "entry": "sso",  
 "gateway": "1",  
 "from": "null",  
 "savestate": "30",  
 "useticket": "0",  
 "pagerefer": "",  
 "vsnf": "1",  
 "su": username,  
 "service": "sso",  
 "sp": *password*,  
 "sr": "1440\*900",  
 "encoding": "UTF-8",  
 "cdult": "3",  
 "domain": "sina.com.cn",  
 "prelt": "0",  
 "returntype": "TEXT",  
 }  
 session = requests.Session()  
 r = session.post(loginURL, data=postData)  
 jsonStr = r.content.decode("gbk")  
 info = json.loads(jsonStr)  
 *if* info["retcode"] == "0":  
 logger.warning("Get Cookie Success!( Account:%s )" % *account*)  
 cookie = session.cookies.get\_dict()  
 *return* json.dumps(cookie)  
 *else*:  
 logger.warning("Failed!( Reason:%s )" % info["reason"])  
 *return* ""  
  
  
*def* get\_cookie\_from\_weibo\_cn(*account*, *password*):  
 """ 获取一个账号的Cookie """  
 *try*:  
 browser = webdriver.PhantomJS(executable\_path='E:\\phantomjs\\bin\\phantomjs.exe',desired\_capabilities=dcap)  
 browser.get("https://weibo.cn/login/")  
 time.sleep(1)  
  
 failure = 0  
 *while* "微博" *in* browser.title *and* failure < 5:  
 failure += 1  
 browser.save\_screenshot("aa.png")  
 username = browser.find\_element\_by\_name("mobile")  
 username.clear()  
 username.send\_keys(*account*)  
  
 psd = browser.find\_element\_by\_xpath('//input[@type="password"]')  
 psd.clear()  
 psd.send\_keys(*password*)  
 *try*:  
 code = browser.find\_element\_by\_name("code")  
 code.clear()  
 *if* IDENTIFY == 1:  
 code\_txt = raw\_input("请查看路径下新生成的aa.png，然后输入验证码:") # 手动输入验证码  
 *else*:  
 *from* PIL *import* Image  
 img = browser.find\_element\_by\_xpath('//form[@method="post"]/div/img[@alt="请打开图片显示"]')  
 x = img.location["x"]  
 y = img.location["y"]  
 im = Image.open("aa.png")  
 im.crop((x, y, 100 + x, y + 22)).save("ab.png") # 剪切出验证码  
 code\_txt = identify() # 验证码打码平台识别  
 code.send\_keys(code\_txt)  
 *except* Exception, e:  
 *pass* commit = browser.find\_element\_by\_name("submit")  
 commit.click()  
 time.sleep(3)  
 *if* "我的首页" *not in* browser.title:  
 time.sleep(4)  
 *if* '未激活微博' *in* browser.page\_source:  
 *print* '账号未开通微博'  
 *return* {}  
  
 cookie = {}  
 *if* "我的首页" *in* browser.title:  
 *for* elem *in* browser.get\_cookies():  
 cookie[elem["name"]] = elem["value"]  
 logger.warning("Get Cookie Success!( Account:%s )" % *account*)  
 *return* json.dumps(cookie)  
 *except* Exception, e:  
 logger.warning("Failed %s!" % *account*)  
 logger.warning(e)  
 *return* ""  
 *finally*:  
 *try*:  
 browser.quit()  
 *except* Exception, e:  
 *pass  
  
  
def* get\_cookie\_from\_weibo(*username*, *password*):  
 driver = webdriver.Chrome()  
 driver.get('https://weibo.cn//login/')  
 time.sleep(10)  
 *assert* u"微博" *in* driver.title  
 login\_link = driver.find\_element\_by\_link\_text(u'登录')  
 ActionChains(driver).move\_to\_element(login\_link).click().perform()  
 login\_name = WebDriverWait(driver, 10).until(  
 EC.visibility\_of\_element\_located((By.ID, "loginName"))  
 )  
 login\_password = driver.find\_element\_by\_id("loginPassword")  
 login\_name.send\_keys(*username*)  
 login\_password.send\_keys(*password*)  
 login\_button = driver.find\_element\_by\_id("loginAction")  
 login\_button.click() #自动按下登陆  
 WebDriverWait(driver, 30).until(EC.title\_is(u"我的首页")) #等待过了验证到首页  
 cookie = driver.get\_cookies()  
 driver.close()  
 *return* json.dumps(cookie)  
  
  
*def* getCookies(*weibo*):  
 """ 获取Cookies """  
 cookies = []  
 *for* elem *in weibo*:  
 account = elem['no']  
 password = elem['psw']  
 cookie = getCookie(account, password)  
 # logger.warning(type(cookie))  
 *if* cookie != None:  
 cookies.append(cookie)  
  
 *return* cookies  
  
  
cookies = getCookies(myWeiBo)  
logger.warning("Get Cookies Finish!( Num:%d)" % len(cookies))

本章将就实现的网络爬虫进行测试运行，并展示爬取的数据。

**5.1测试环境**

开发语言：python2.7

需要的Python模块：scrapy、requests、lxml、pywin32、pymongo、json、httplib、base64、redis

开发环境：8G内存、i5-4200H处理器、三星SSD 850 EVO 120G

操作系统：Windows10专业版系统、操作系统安装在三星SSD上

网络带宽：联通15M带宽

数据库：Redis、Mongodb

IDE：Pycharm

绘图工具：Visio2013

统计工具：Execl2013

浏览器：Google Chrome版本 64.0.3282.119（正式版本） （64 位）

数据库可视化工具：mongobooster、RedisStudio-en-0.1.5

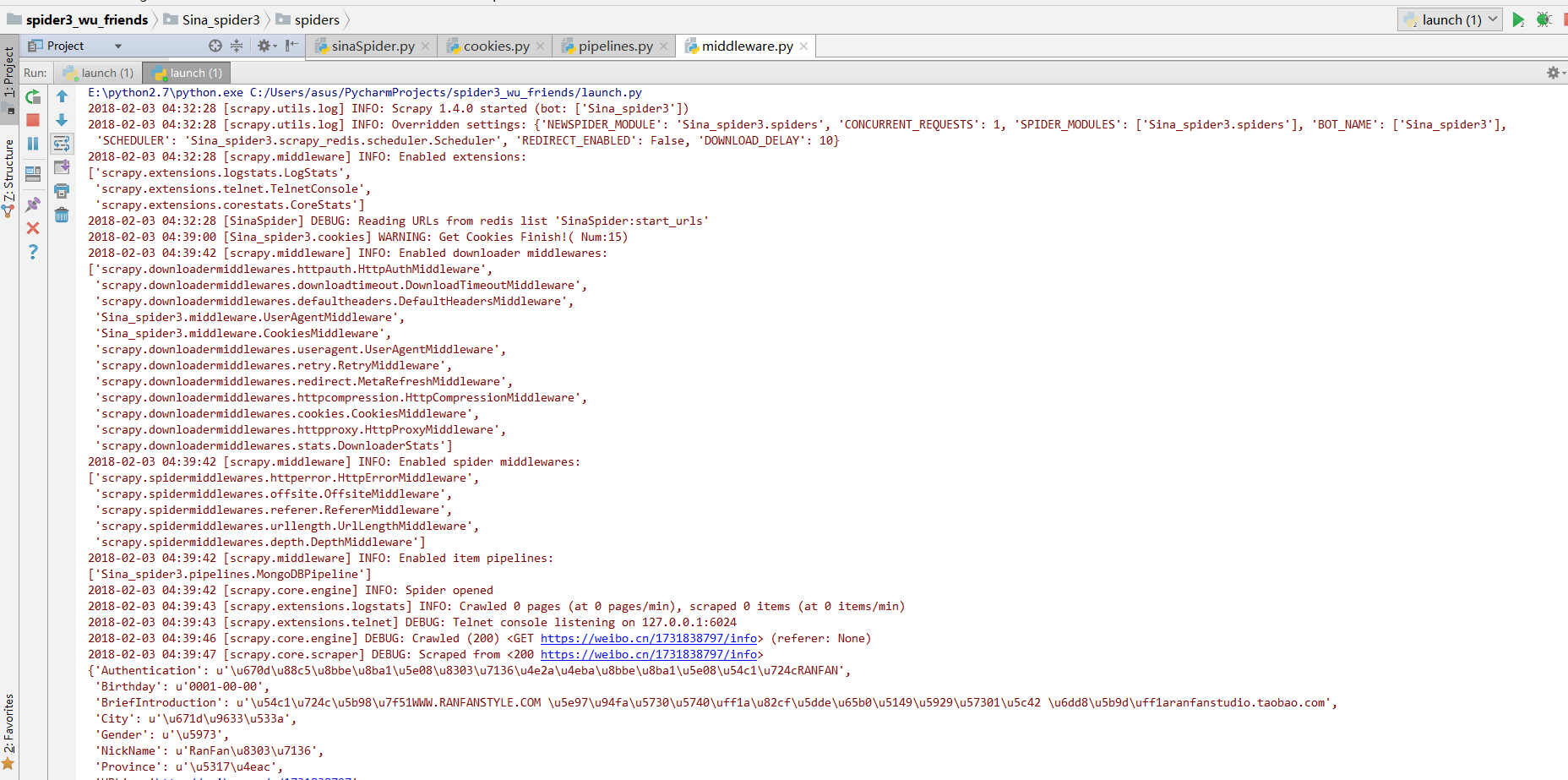
爬虫框架： Scrapy，使用 Scrapy+Redis 、Redis

**5.2运行状态及测试**

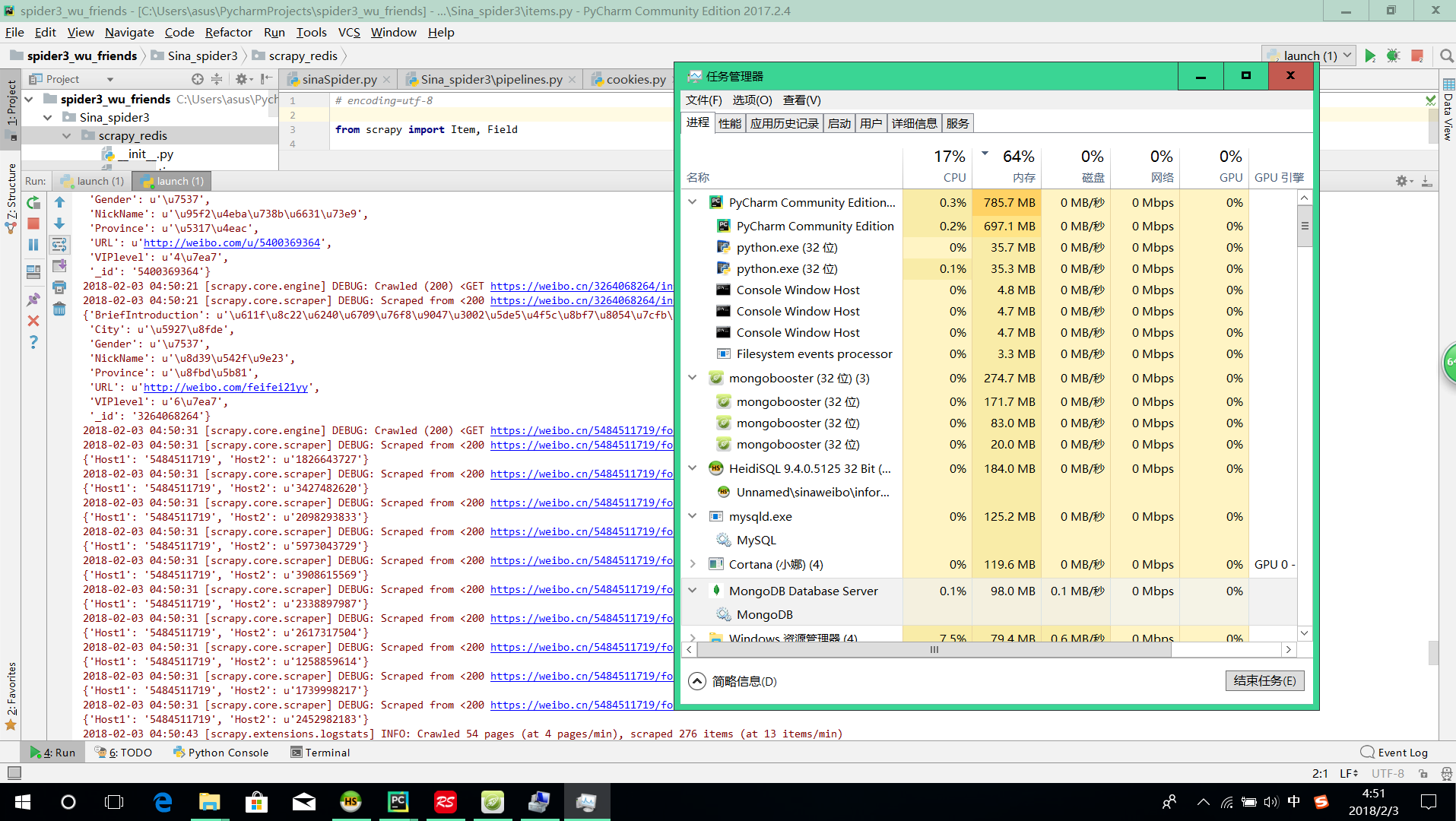
经测试，爬虫抓取微博的速度可以达到12个小时爬取400W条数据，具体要视网络情况。需要定期更换Cookies中的账号信息。建议大量预存。比如我在淘宝网上买了25个账号作为预存在Cookies上。

MongoDB性能良好。足以处理该量级的数据。下图为爬虫运行时的控制台

截图:



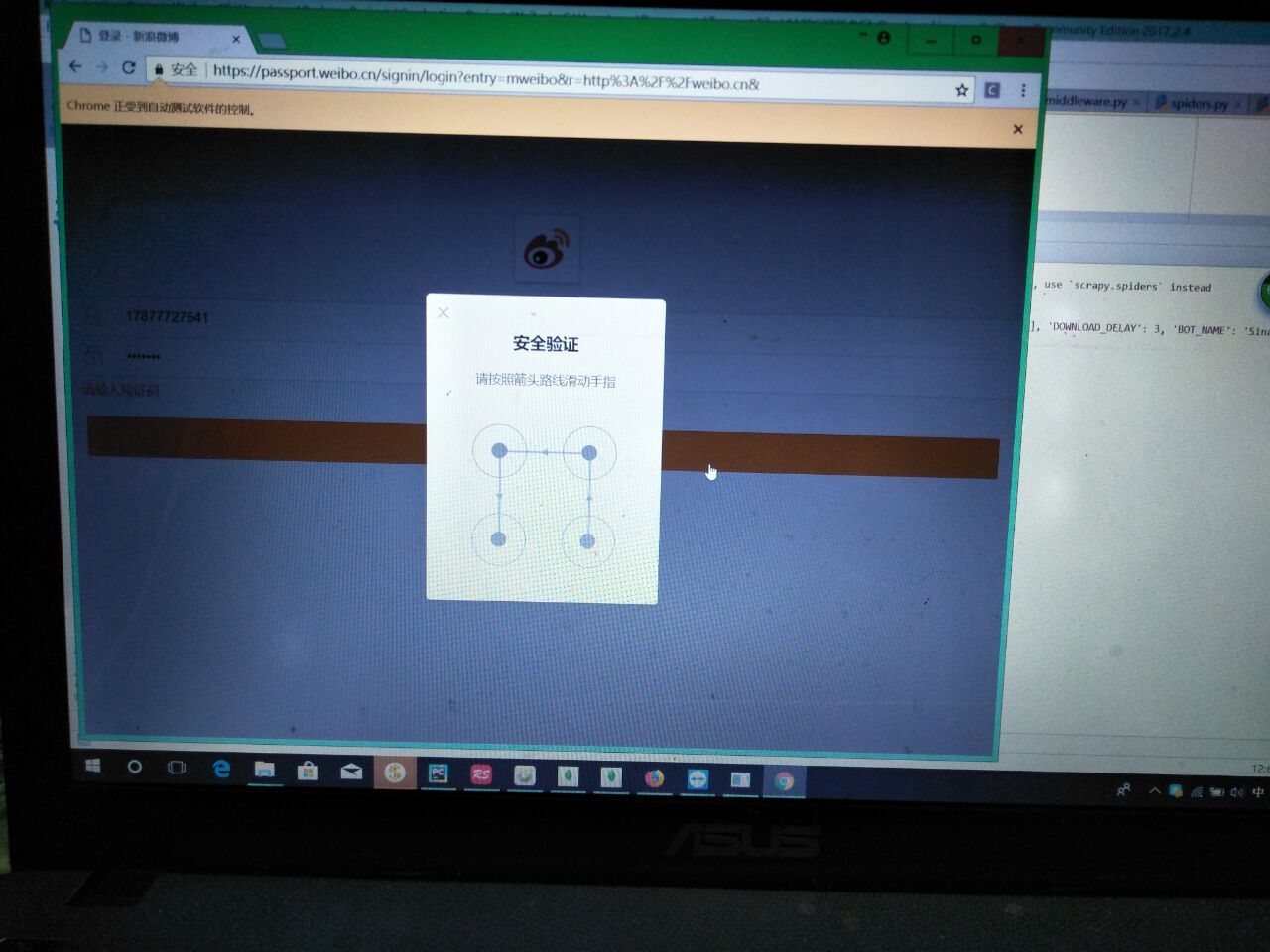
运行成功爬虫正在爬取



爬虫系统运行中各个状态

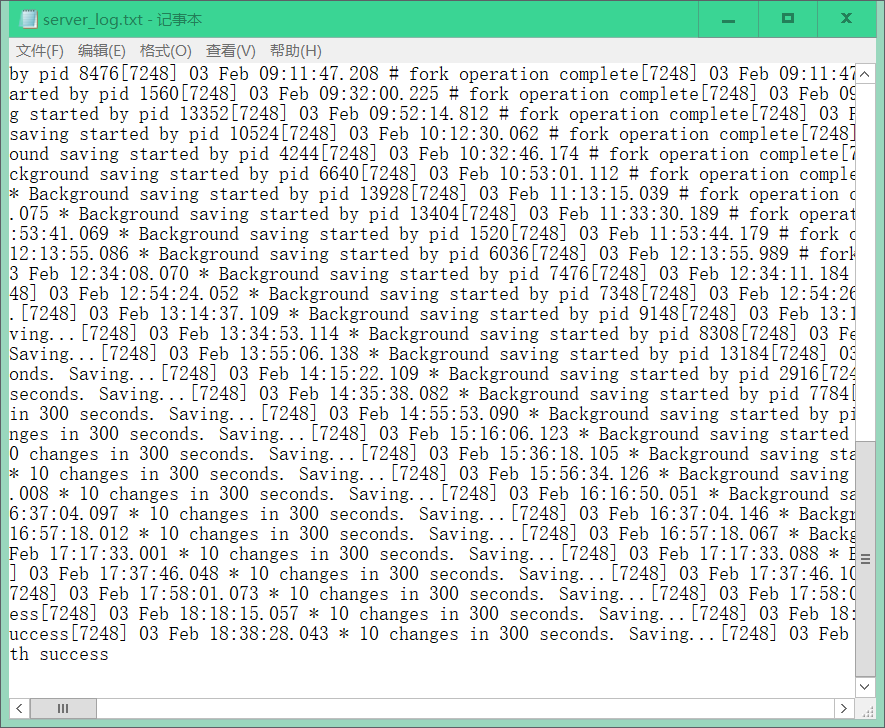
**5.3测试过程其他组件过程展示**

运行程序，登录移动版web微博，手动打码过程。



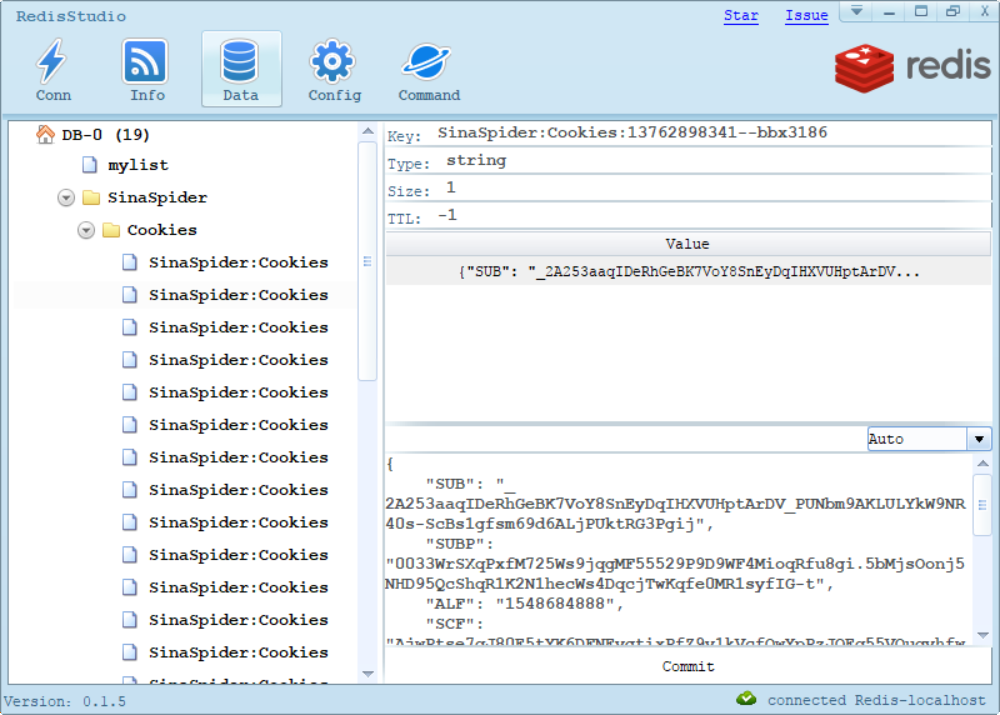
手动打码

以下是爬虫爬取的过程中Redis服务器的日志server\_log状态。



Redis的服务器日志

Redis在运行中

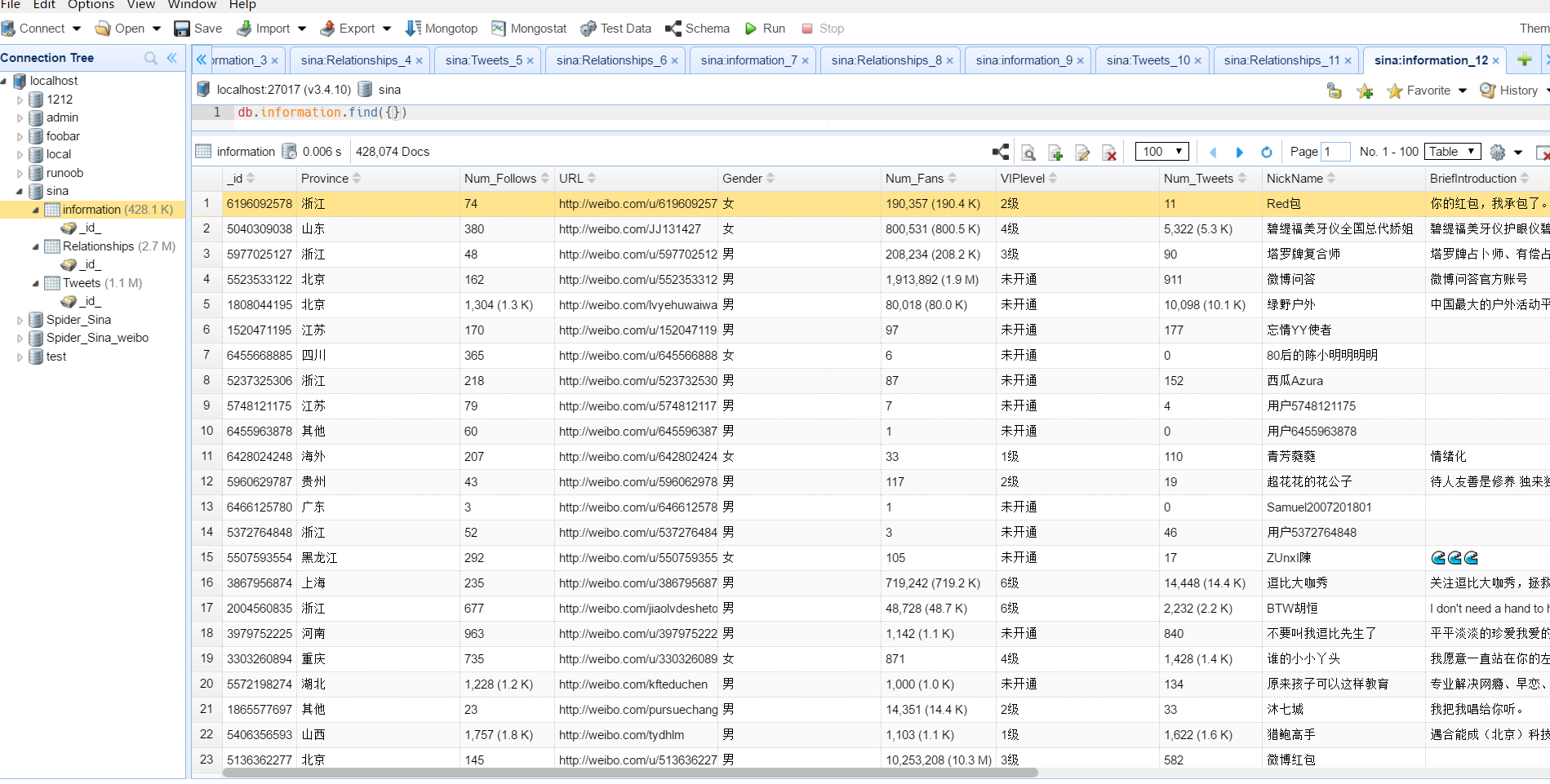


Redis中存储的Cookies

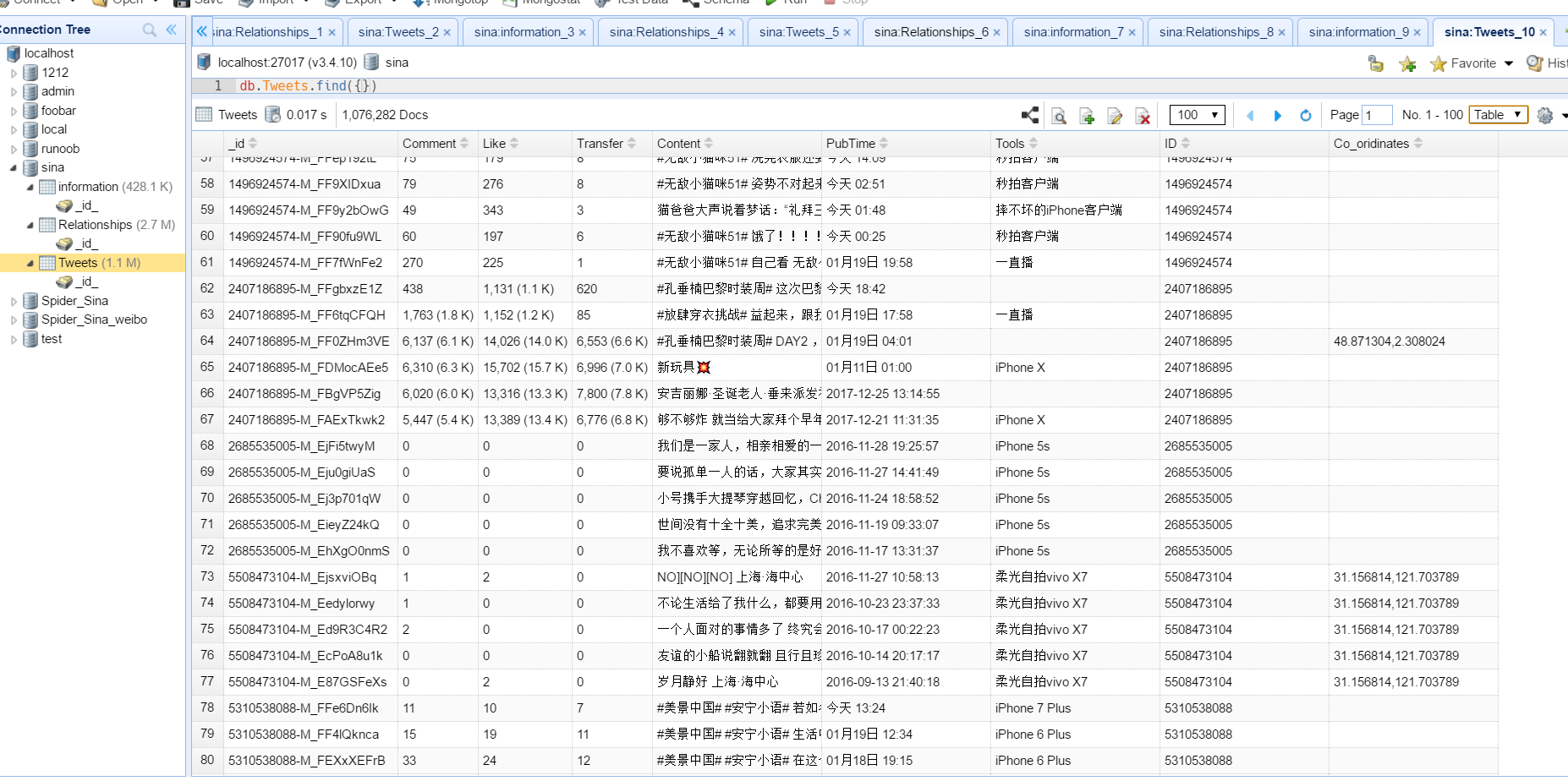
**5.4数据抓取结果展示与分析**

在程序运行过后得到名为”sina”的数据库，该数据库下保存有三张表，分别为

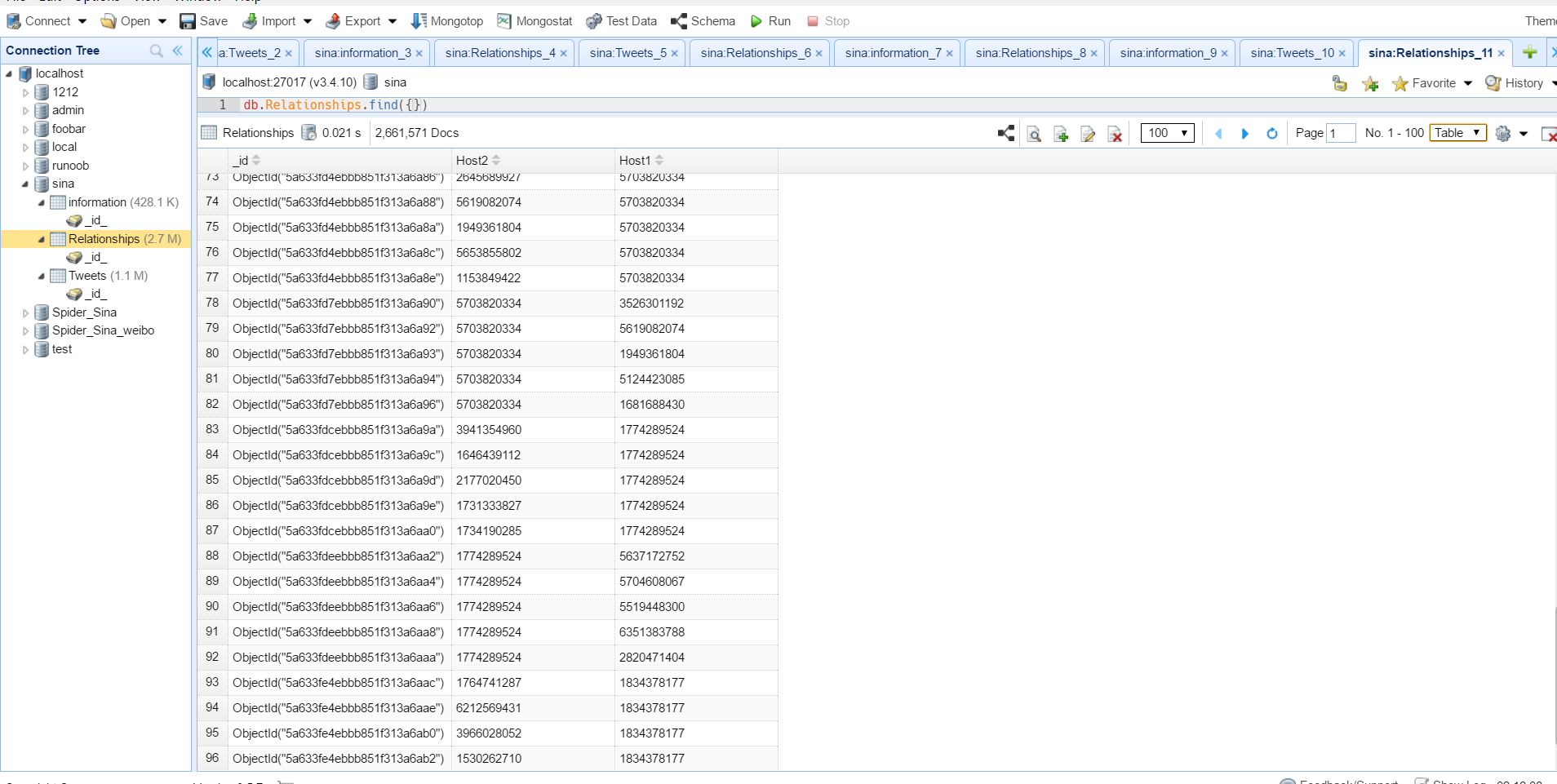
Information 、Tweets 、Relationships。实现存储了用户个人信息Information，用户发表的博客Tweets,微博用户关系表Relationships。



Information表



Tweets表



Relationships 表

**6.1.1数据库概念设计——E-R模型**

概念模型是对信息世界建模，其表示方法有很多，其中最为常用的是实体－联系模型，即E－R模型，它是用E－R图来描述现实世界的概念模型。下列E-R图清楚简便的描述出了本系统实体与其属性之间的关系。



**用户关系表(relationships)**



用户发表的微博(Tweets)



**用户个人信息(information)**



数据库概念模型E-R图

**6.1.2数据库逻辑设计**

下面将E-R图转换为关系模型。

用户发表的微博（\_id ，ID ，Content ，PubTime ，Co\_oridinates ，Tools ，Like ，Comment ，Transfer ）

用户关系表（\_id，Host2，Host1）

用户个人信息（\_id，NickName，Gender，Province，City，BriefIntroduction，Birthday，Num\_Tweets:，Num\_Follows，Num\_Fans，SexOrientation，Sentiment，VIPlevel，Authentication，URL）

**6.1.3数据表设计**

本爬虫系统数据库中总共设计了3张数据表，分别为具体结构设计如下：

**用户关系表(relationships)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **字段名称** | **类型** | **是否为空** | **主键** | **说明** |
| **\_id** | **Objectld** | **Not NULL** | **是** | **用户ID** |
| **Host1** | **String** | **NULL** | **否** | **粉丝** |
| **Host2** | **String** | **NULL** | **否** | **被关注的人** |

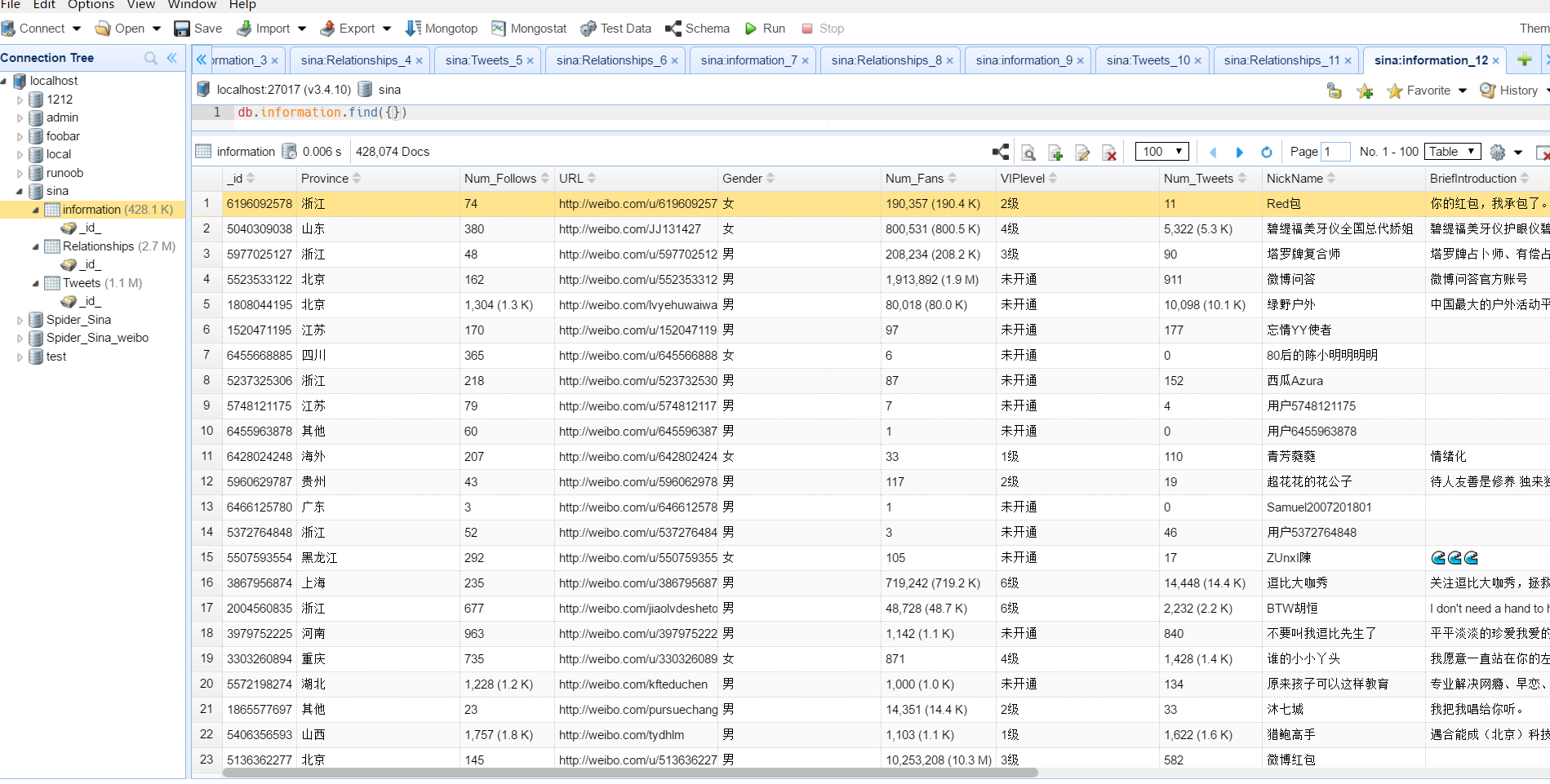
**用户个人信息(information)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 类型 | 是否为空 | 主键 | 说明 |
| \_id | **String** | **Not NULL** | **是** | **用户ID** |
| NickName | **String** | **NULL** | **否** | **昵称** |
| Gender | **String** | **NULL** | **否** | **性别** |
| Province | **String** | **NULL** | **否** | **所在省** |
| City | **String** | **NULL** | **否** | **城市** |
| BriefIntroduction | **String** | **NULL** | **否** | **简介** |
| Birthday | **Date** | **NULL** | **否** | **生日** |
| Num\_Tweets | **Int32** | **NULL** | **否** | **微博数** |
| Num\_Follows | **Int32** | **NULL** | **否** | **关注数** |
| Num\_Fans | **Int32** | **NULL** | **否** | **粉丝数** |
| SexOrientation | **String** | **NULL** | **否** | **性取向** |
| Sentiment | **String** | **NULL** | **否** | **感情状况** |
| VIPlevel | **String** | **NULL** | **否** | **会员等级** |
| Authentication | **String** | **NULL** | **否** | **认证** |
| URL | **String** | **Not NULL** | **否** | **首页链接** |

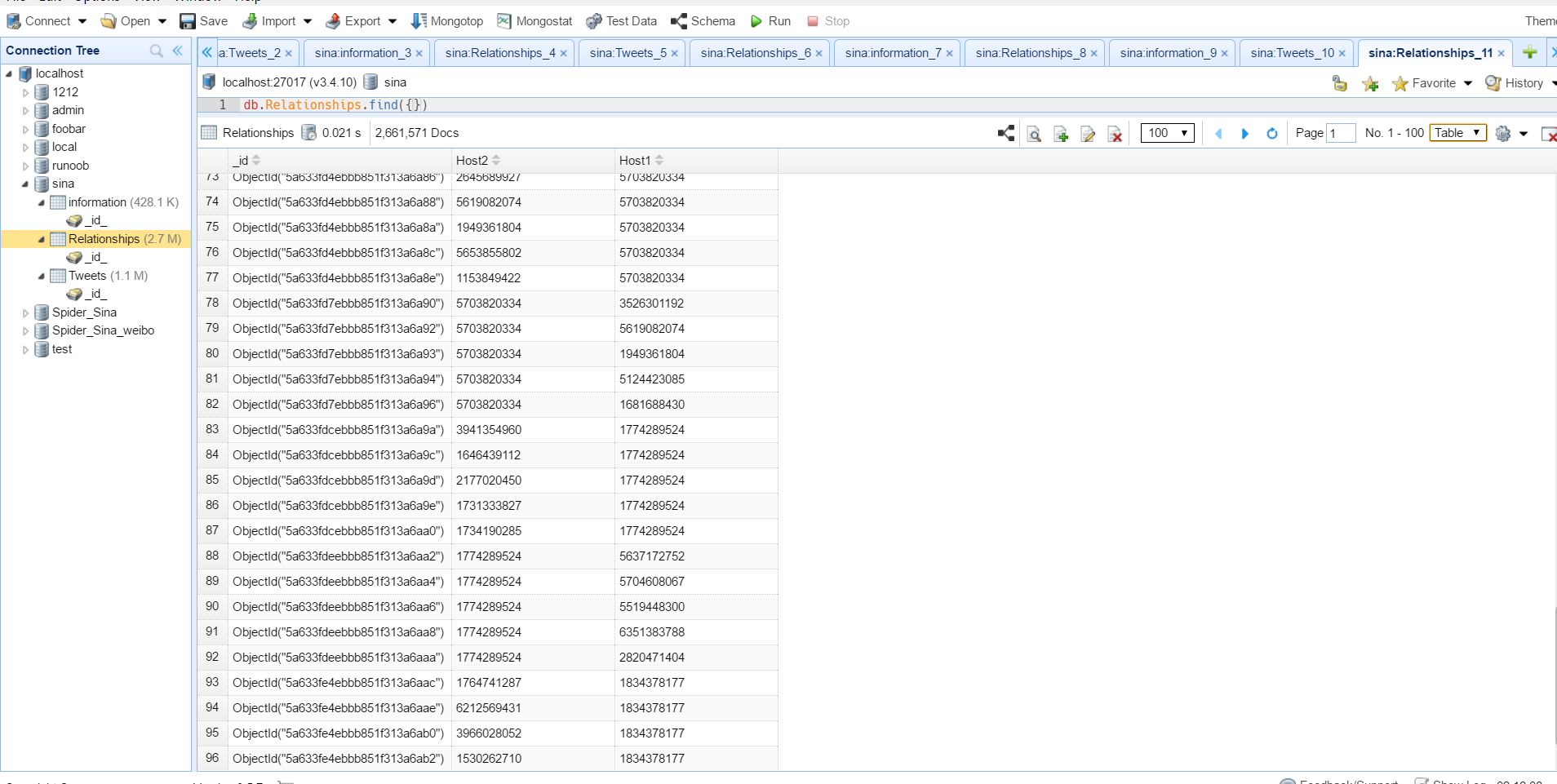
**用户发表的微博(Tweets)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 类型 | 是否为空 | 主键 | 说明 |
| \_id | **String** | **Not NULL** | **是** | **用户ID-微博ID** |
| ID | **String** | **NULL** | **否** | **用户ID** |
| Content | **String** | **NULL** | **否** | **微博内容** |
| PubTime | **Date** | **NULL** | **否** | **发表时间** |
| Co\_oridinates | **String** | **NULL** | **否** | **定位坐标** |
| Tools | **String** | **NULL** | **否** | **发表工具/平台** |
| Like | **Int32** | **NULL** | **否** | **点赞数** |
| Comment | **Int32** | **NULL** | **否** | **评论数** |
| Transfer | **Int32** | **NULL** | **否** | **转载数** |

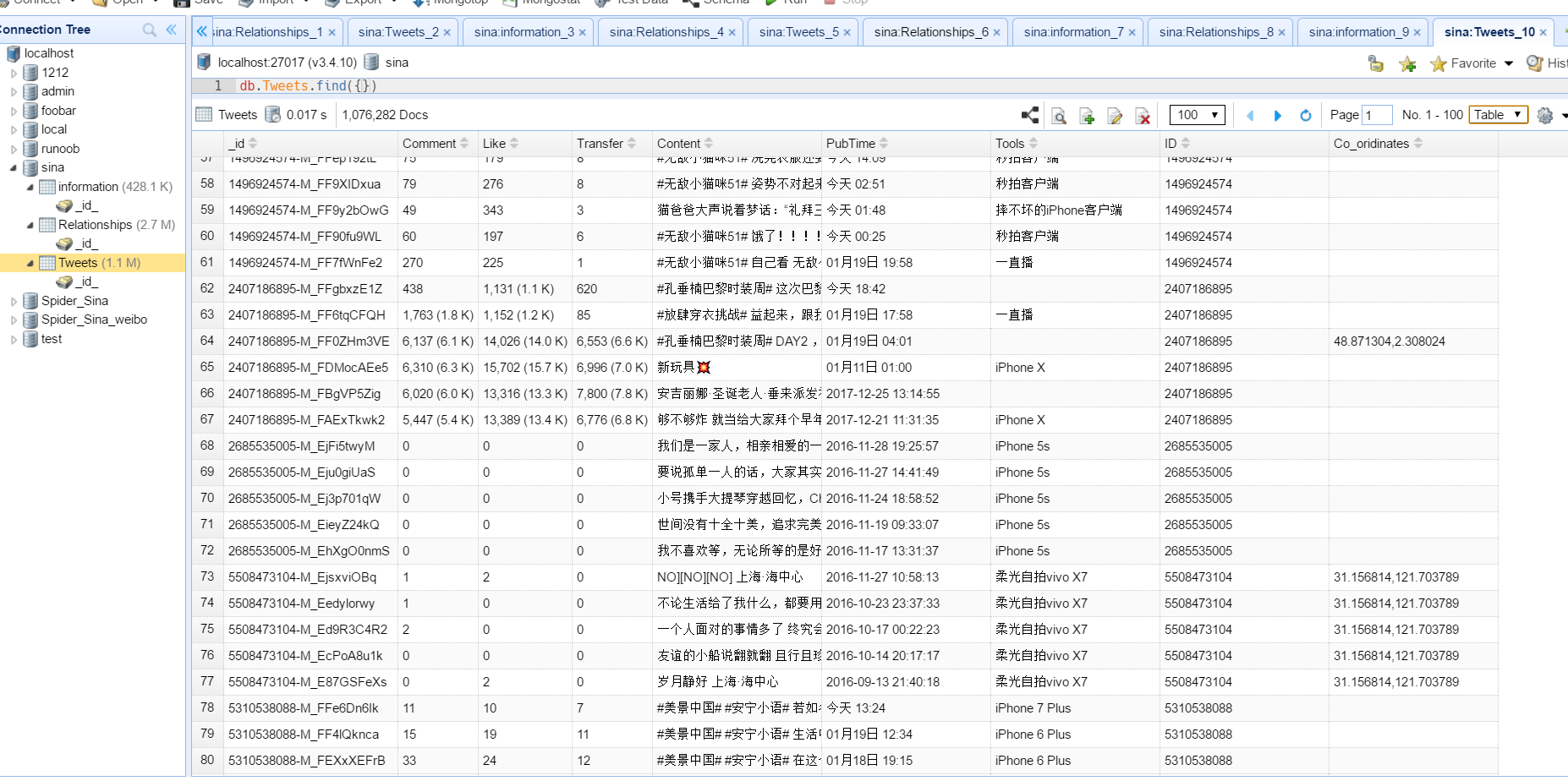
下图为information表:



下图为relationships表:



下图为Tweets表:



**6.2.1数据分析**

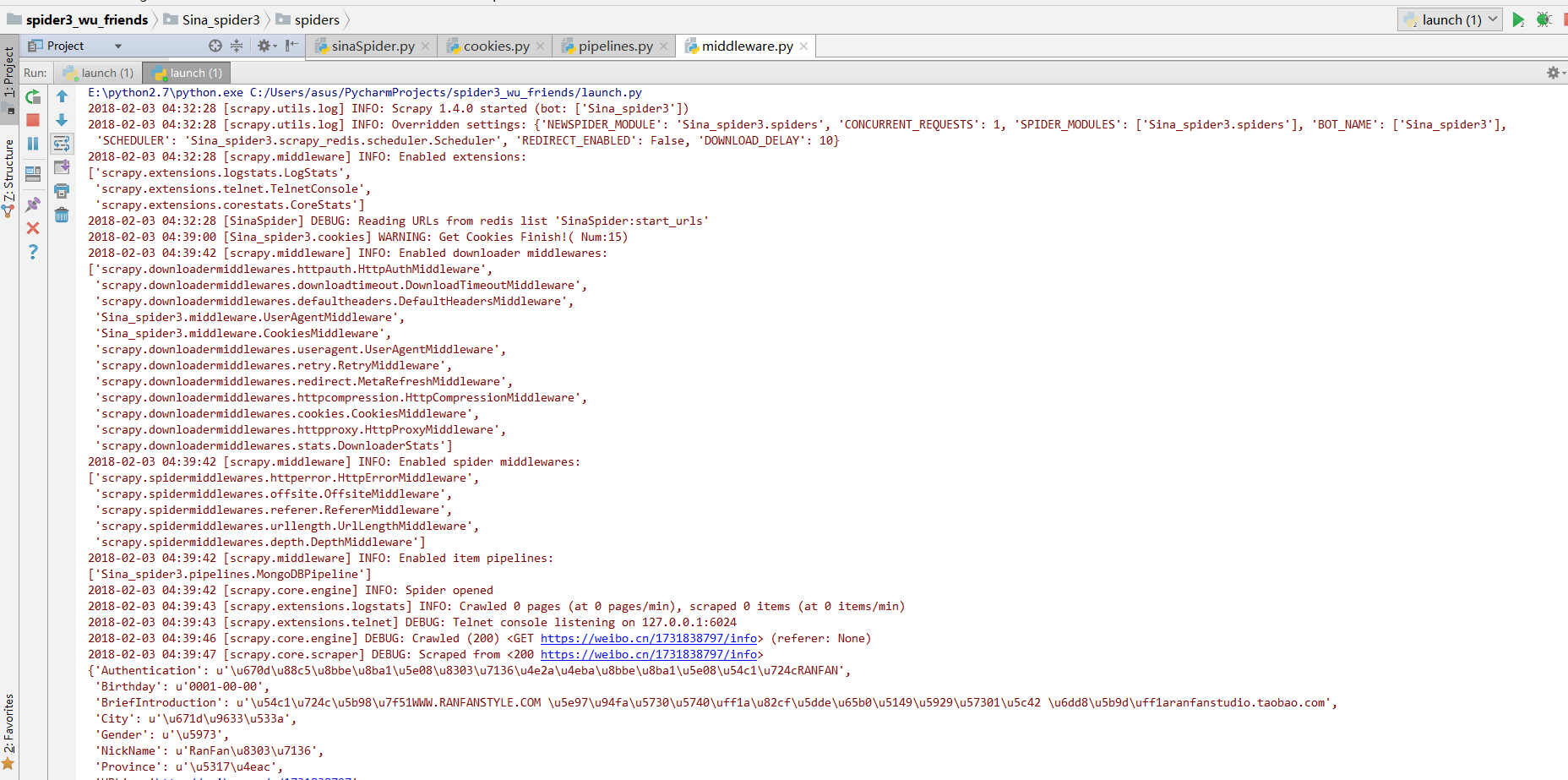
本论文是用Scrapy-Redis架构上的爬虫所爬取的数据，在Nosql类型的Mongodb数据库中的做简单的数据分析。爬了2天，因为写了暂停和继续功能，我基本是开一下关一下的，不过速度确实还不够快不快。我记得有个人 PHP 写的一天就能 1300w ！呢好啦，正式开始吧，上一章介绍了数据的E-R模型以及设计等等这里就不再啰嗦就简单介绍一下数据先：

用户发表的微博（\_id ，ID ，Content ，PubTime ，Co\_oridinates ，Tools ，Like ，Comment ，Transfer ）

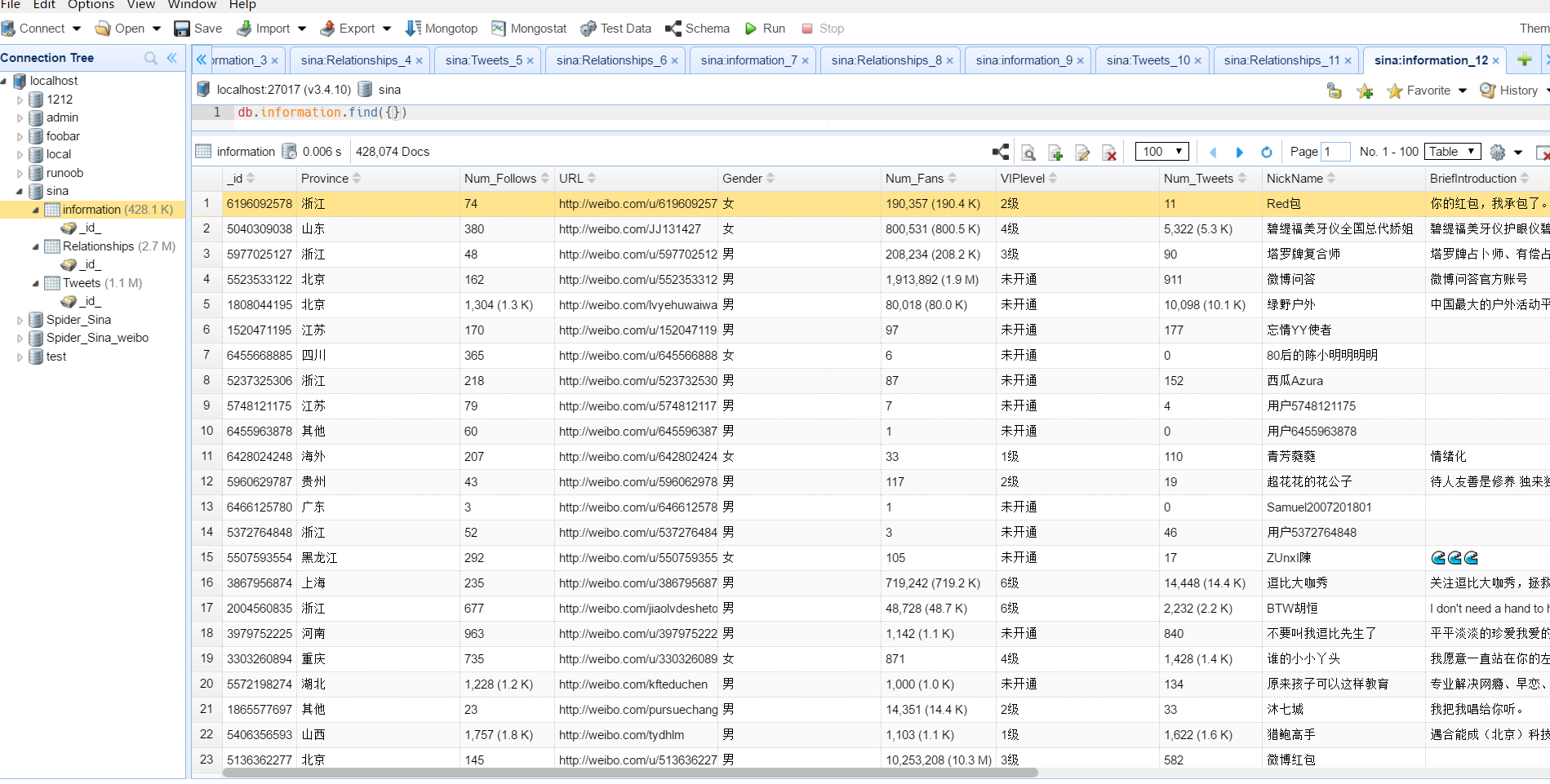
用户关系表（\_id，Host2，Host1）

用户个人信息（\_id，NickName，Gender，Province，City，BriefIntroduction，Birthday，Num\_Tweets:，Num\_Follows，Num\_Fans，SexOrientation，Sentiment，VIPlevel，Authentication，URL）

爬虫运行成功爬取数据中

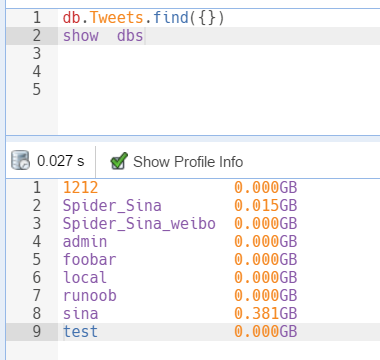


爬虫爬取的在mongodb中的数据



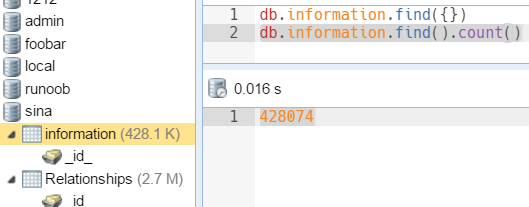
在Mongodb的可视化工具MongoBooster中写SQL语句做简单的数据分析：

>show dbs



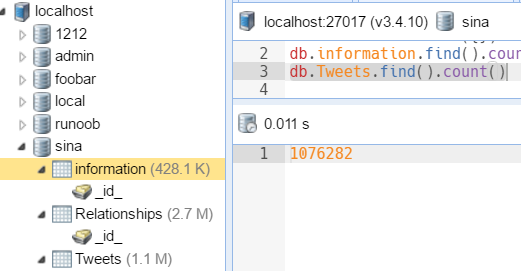
**1.统计各个表的数据总共多少条:**

>db.information.find().count()



用时0.016s information表一共428074条数据

> db.Tweets.find().count()

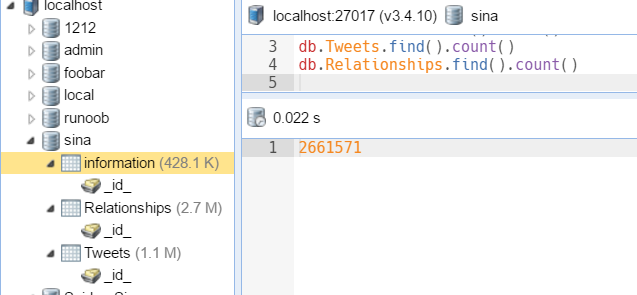


用时0.011s Tweets表一共1076282条数据

>db.Relationships.find().count()

用时0.022s Relationships表一共2661571条数据

所以我们爬虫爬取的数据有4165927条数据！

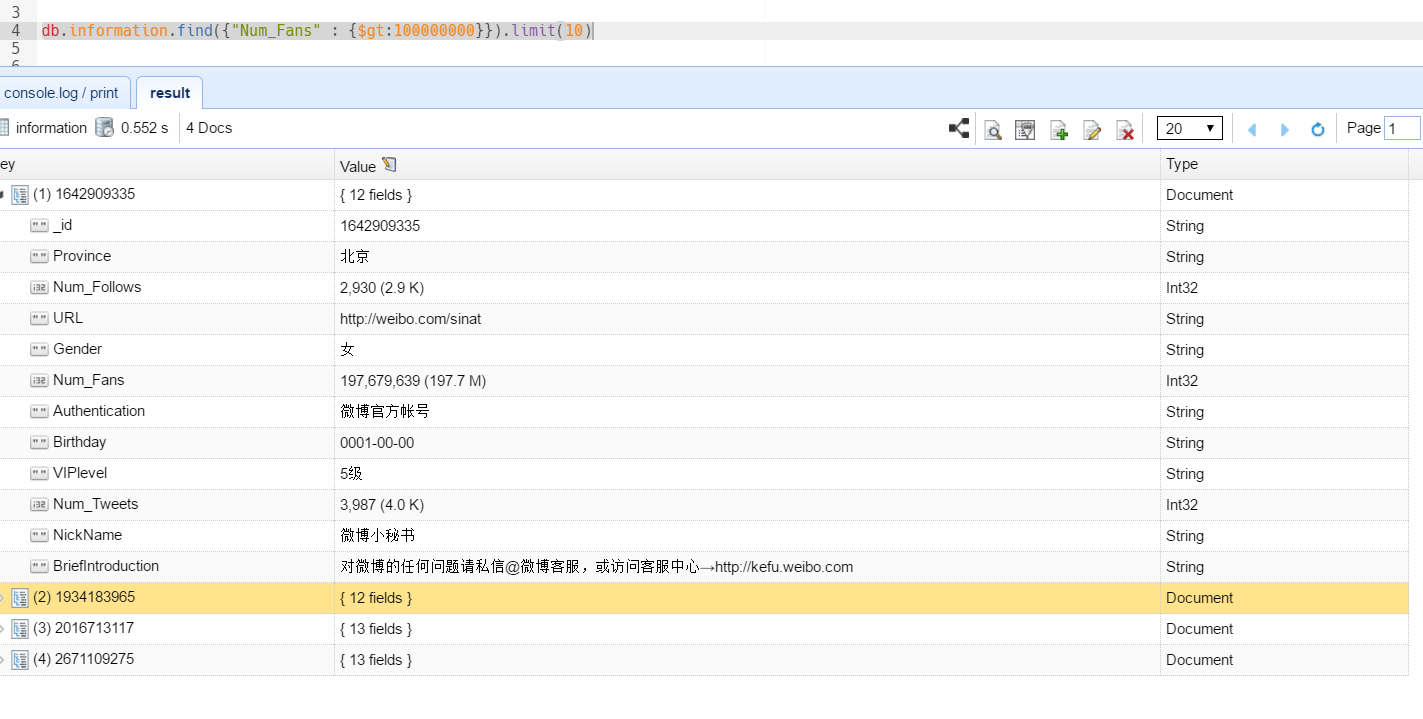


**2.查询粉丝数量大于一亿：**

这个最简单也最老套了吧，相信有一大堆人做过。

SQL 语句：

>db.information.find({"Num\_Fans" : {$gt:100000000}}).limit(10)



>db.information.find({"Num\_Fans" : {$gt:100000000}}).count()

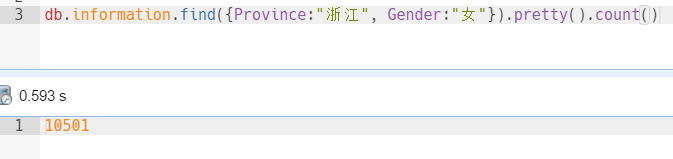
Result:1.027s 4

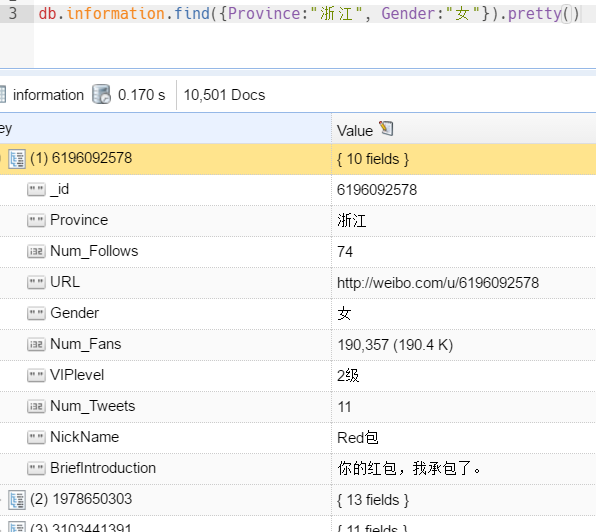
用时1.027s 总共4人

3.**查询浙江女用户情况：**

>db.information.find({Province:"浙江", Gender:"女"}).pretty()

>db.information.find({Province:"浙江", Gender:"女"}).pretty().count()





4.**性取向情况：**

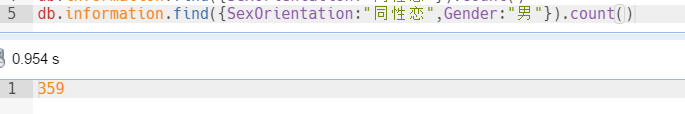
>db.information.find({SexOrientation:"同性恋"})

> db.information.find({SexOrientation:"同性恋"}).count()

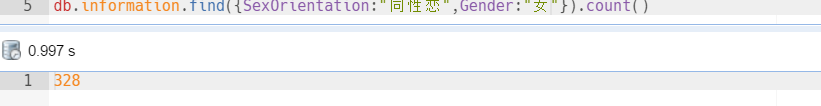
用时：0.635s 总共：687

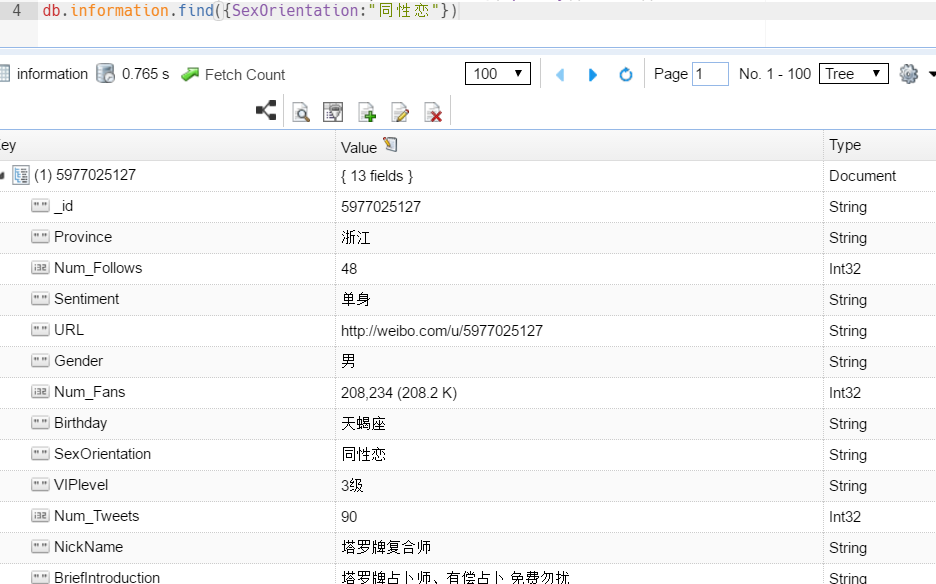
男同性恋情况：

>db.information.find({SexOrientation:"同性恋",Gender:"男"}).count()



女同性恋情况：





1.6**数据的导入导出：**

MongoDB的导出功能

mongodb提供了导入和导出的功能，分别是MongoDB下载目录下的mongoexport.exe和mongoimport.exe文件 ，具体的导出命令格式如下：

mongoexport -h dbhost -d dbname -c collectionName -o output

参数说明:

-h 数据库地址

-d 指明使用的库

-c 指明要导出的集合

-o 指明要导出的文件名（文件支持多种格式，如txt,wps,xls等）

在管理员权限CMD进入shell下：

mongoexport –host 127.0.0.1 -d sina -c Tweets -o Tweets.csv

MongoDB的导入功能

导入数据可以使用命令：

mongoimport -h dbhost -d dbname -c collectionname input

参数说明:

-h 数据库地址

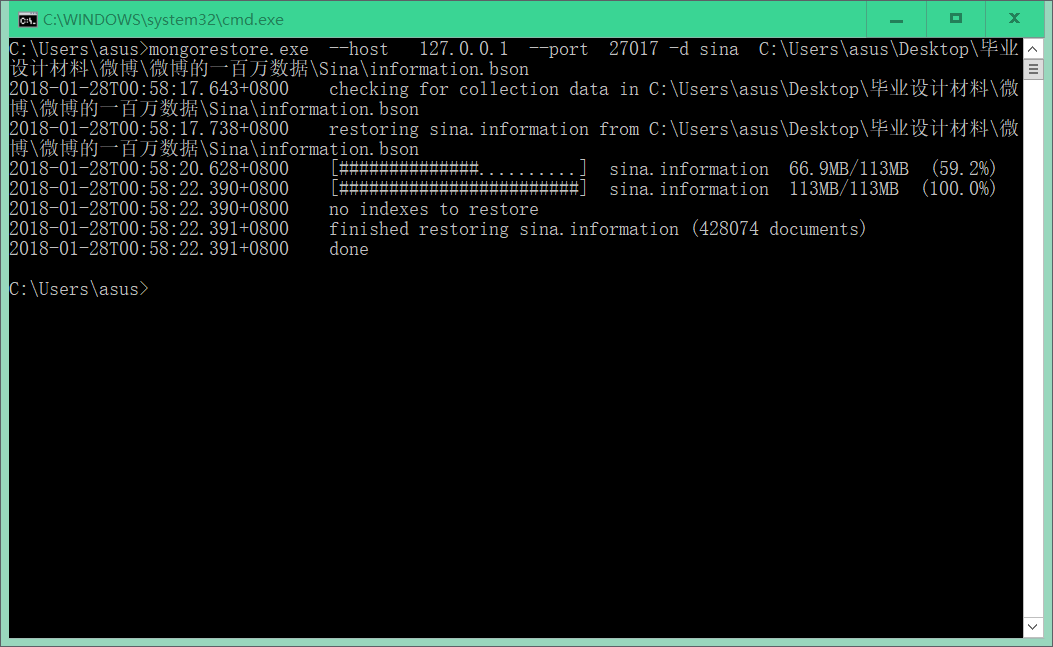
-d 指明使用的库

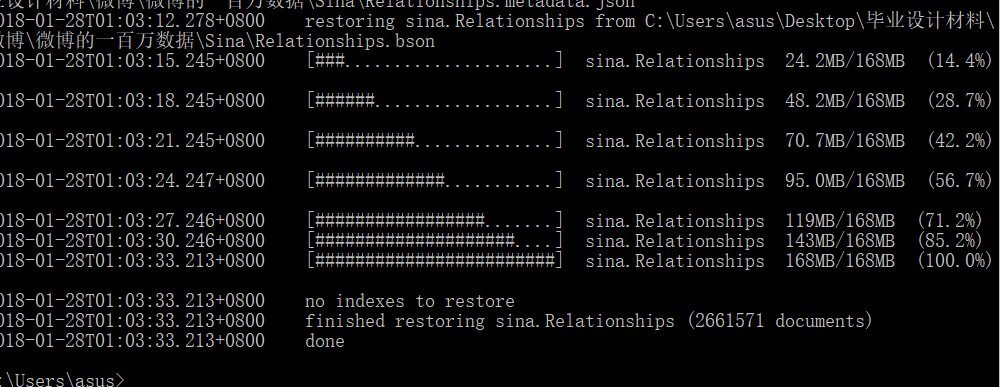
-c 指明要导入的集合（mongodb本身支持隐式创建，故事先无需创建集合）

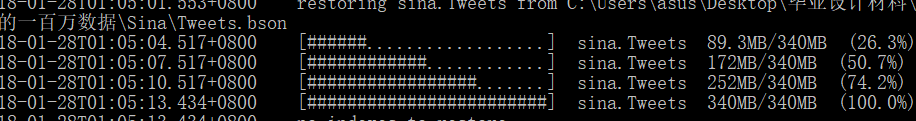
input 文件的地址

在管理员权限CMD进入shell下：

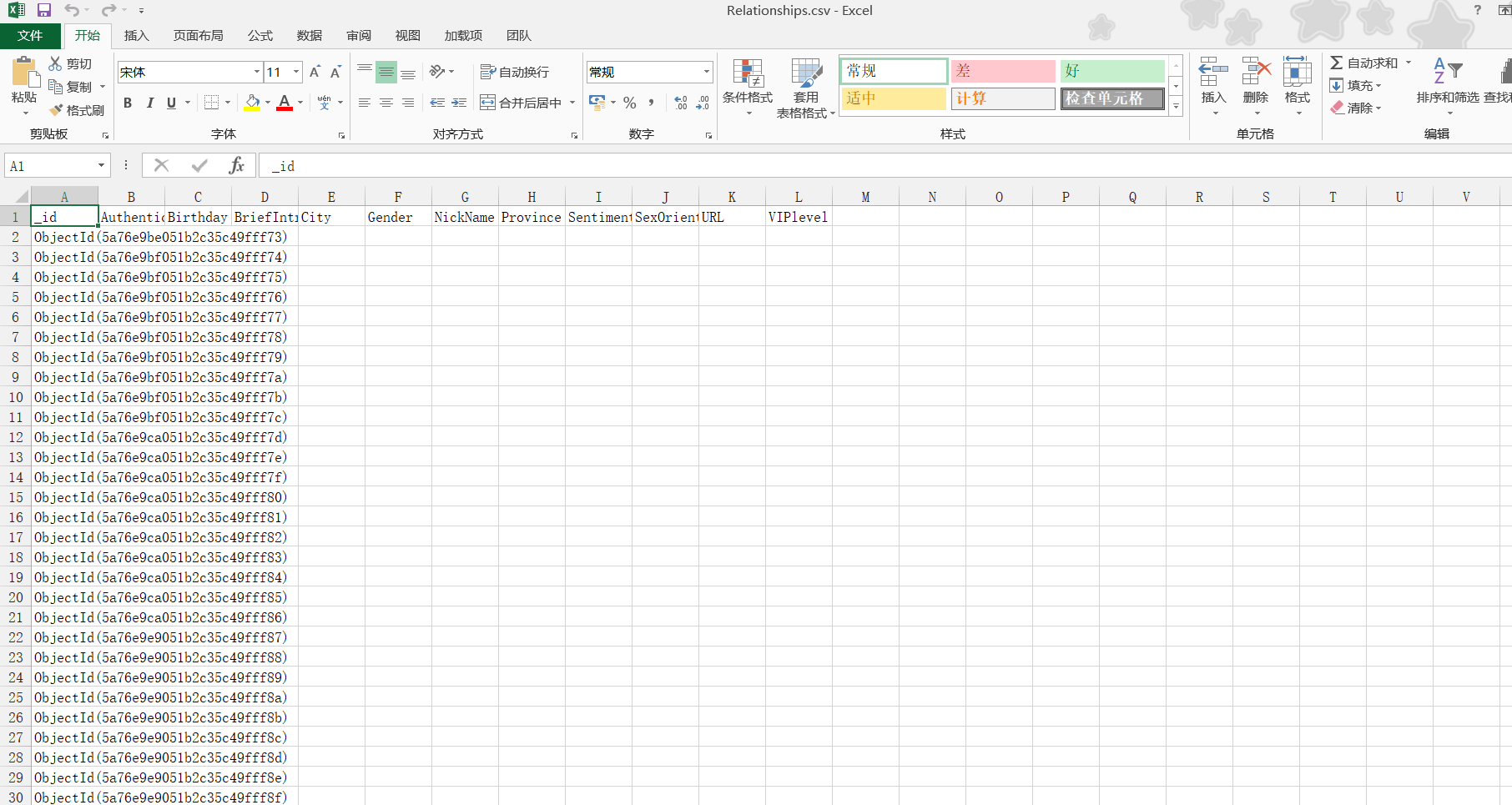
比如：Mongoimport --host 127.0.0.1 –port 27017 -d sina -c C:\user\asus\Desktop







用Excel就能打开cvs文件：



**7. 1毕业设计工作总结**

随着互联网络的迅速发展，Internet上网络信息的爆炸式增长，对于网络信息的搜索和需求显得至关重要。起初通用网络爬虫被运用到Internet的各个领域，因为它具备整合信息能力，能够实现用户的全面搜索。然而，尽管如此，通用爬虫的弊端也越来越明显，例如内容陈旧、查全率查准率偏低、信息冗余、分布不等。无法满足对于特定用户和人群的搜索需求，因此在这种背景下，主题网络爬虫为基础的主题搜索引擎诞生。其运行效率的高低直接关系到下一代搜索引擎的性能。

1. 设计并实现了微博的爬虫系统。通过模拟登录的方法解决身份认证，解决验证码问题，伪装成为移动端环境爬取微博移动端web等问题，爬虫采用广度优先搜索的思想。并结合网络爬虫、BeautifulSoup、Scrapy框架、正则表达式和多线程并发、Cookie池等技术实现多种用户信息和微博信息的采集。采用引入Cookie池输入多个微博用户账号代理的方法解决了微博服务器限制请求过频的问题，使用了内存非关系型Redis数据库作为多线程的队列，同时用Redis实现爬虫程序能中断的功能，并利用非关系型数据库MongoDB存储数据，实现了微博数据的持续采集与存储，为后续研究提供了数据支持。
2. 在数据库MongoDB中事先在爬虫的设计过程中对微博用户世界机型抽象建立数据库模型，并画出相应的E-R图与三张数据库表。它们分别是用户关系表，用户个人信息表，用户发表的微博表；然后根据微博的网页特性在爬虫系统各相应模块编写正则表达式。
3. 根据饰演的结果，对爬取道德400W条数据进行简单的统计分析，得出一些小结论达到实验的目的。

根据实验结果和相应的统计分析，显示了达到了预期的实验目标，成功地改

进了爬虫的工作方式和工作效率，取得了一定的创新成果。使得系统在采用该算

法时可以更好地过滤掉过程中与主题无关的网页，提高了效率。即便不完全遍历

整个互联网也可以较为准确的抓取主题页面。

**7.2不足**

在本文的研究上存在一些不足，以下分别列举出来：

1. 虽然本毕设采用了Scrapy-Redis架构这个实际上是分布式架构但是由于现实软硬件网络环境等的限制没有采用分布式，只是采取了单机上的多线程高并发式的爬取，在实际中是无法达到工业级别要求的只能算是模拟仿真的小demo。
2. 硬件上是采用我个人笔记本电脑，网络环境是采用的普通家庭的网络环境。
3. 由于软硬件网络环境等的限制只能爬取微博用户的400W条数据，数据量不够太少了。
4. 操作系统平台采取的Windows10专业版没有采用Linux网络操作系统，稳定性，鲁棒性等都不足。
5. 由于资金限制，不能部署到云端跑程序。

**7.3展望**

在本文的研究基础上，可以在以下五个方面进一步研究:

第一，本文的数据存储采用的是非关系型数据库MongoDB要把MongoDB分片在不同的数据库集群中,期间做好调度，同时为了更好地适应大数据分析，下一步考虑采用Scrapy-Redis架构搭建分布式爬虫系统，分布式文件系统更适用于大规模的数据处理,更能适应企业对公司业务的需求。实际工业水平上采用P2P分布式爬虫才能真正的达到企业对大型网站数据获取的大数据业务，比如百度搜索引擎就是在就在全国的50多个分布式集群。

第二，本文的研究由于硬件环境的限制是在自己笔记本电脑上进行的,只能做模拟仿真爬取一下,模拟形式下在实际工业生产过程中是不合用的,在工业级别中要求爬虫系统应该布置在集群上或者说至少是服务器哪怕是单台服务器上,相应的网速要更快,硬盘容量要更大达到TB、PB级别上，能是SSD（固态硬盘）更好。

第三，本文的研究是为了毕设的方便所以采用的操作系统平台是Windows10专业版，在实际工业中应该采取Linux操作系统，把程序放到Linux服务器上去跑，比如Redhat,Centos7等等，因为linux 稳定性、安全性、能全年365天跑程序都没有问题很适合爬虫。

第四，本文的研究能进一步改进，比如为了节省开支，去阿里云、腾讯云去购买VPS云服务器，数据库也采用云数据库等等。

第五，本文采用网络爬虫采集数据，由于微博网站的更新改版，带来了较高的维

护成本，今后可继续优化改进系统，降低爬虫系统的维护成本。

致 谢

时光消逝，岁月如梭，转眼间我的本科四年求学生涯即将结束。大学人生最美好的四年即将结束我也将要踏入社会!从高考,填报志愿,调剂到网络工程专业,说真的我以前从没回来没有想过我会往计算机行业发展,也没想过我会去东北这么远的地方读书,刚开始很懊悔也不了解这个我将来要从事的行业,后来我尝试着学习计算机渐渐地我热爱上这个行业这个学校如果在给我再来一次还会这样选择,包括我大学的四年我过的很充实。在这四年的光阴中，我感谢，使我成长为一名能够适应社会要求的青年值此毕业之际，我谨向所有关心、帮助过我的人们表示最诚挚地感谢与最美好地祝愿。

首先对于我的导师宋邵忠老师表示衷心的感谢!本论文是在宋老师的悉心的指导下才能顺利完成，从论文最初的选题到论文的撰写，老师都倾注了极大的热血给予各种帮助。宋老师严谨的学风、渊博的学识和平易近人的品德使我受益颇多。认识老师两年以来老师在我学习期间,宋老师不仅授我以文，而且在生活上和思想,以后出社会发展在方向上也给了我很大的关怀和帮助，在此向宋老师致以崇高的敬意和衷心的感谢,祝老师身体健康，万事如意。

感谢我的父母，虽然他们并不是特别了解我现在主要的工作内容，但正是他

们的养育与培养造就了现在的我。远广西跑到了东北长春求学以来对他们的了解与关心就少了很多，虽然时常拌嘴但他们永远都毫无保留的支持着我，感谢他们!

感谢学校网络工程所提供的良好环境，感谢张毅老师、胡奇老师、四年里的在学业和发展方向的关心和指导，感谢在贾秀丽老师带领下数学建模国赛与省赛分别取得了二等三等奖，感谢辛刚

感谢我的科协的同学，他们是张松学长、高于田学长、杨泽毅、徐迟、唐浩明、熊朝旭、申羽燕。他们是我的良师益友，正是有了他们的陪伴，我的大部分本科时光变得非富多彩而意义非凡。非常的能与他们一起工作、生活、进步，希望他们在毕业后都能从事自己理想的工作!

感谢我的室友李迪、伊雷、张坤、他们的陪伴让我感到有家人的温暖，他们都很优秀，在我的本科期间给我很多激励与帮助，感谢他们的陪伴，希望他们都能从事自己理想的工作!

最后，感谢评审老师在百忙之中抽出宝贵的时间帮我评审论文，感谢所有论文评审专家的无私付出!

[1]李勇，韩亮.主题搜索引擎中网络爬虫的搜索策略研究[J].计算机与数字工程，2008, 228(10): 50-53.

[2]罗刚，王振东.自己动手写网络爬虫[M].北京:清华大学出版社，2010.

[3](美)Miguel Grinberg. Flask Web开发[M].安道，译.北京:人民邮电出版社，2015.

[4] Magnus Lie Hetland. Python基础教程(第二版)[M].司维，曾军岚，谭颖华，等，译.北京:人民邮电出版社，2010.

[5」叶允明，于水，马范援，等.分布式Web Crawler的研究:结构、算法和策略[J].电子学报，2002, 30(121:2008-2011.

[6] CSDN博客 https://www.cnblogs.com/M-LittleBird/p/5902850.html

[7] CSDN博客 http: //www.ecma-international.org

[8] http://blog.csdn.net/chenweiyu11962/article/details/78627589

[9]Python爬虫开发与项目实践 范传辉 机械工业出版,2017年6月

[10]数学之美(第二版)作者吴军博士 人民邮电出版社 2014年

[11]TCP/IP详解 卷1：协议（第2版） （美）Kevin R. Fall，W. Richard Stevens 著 /2012-05-01 /机械工业出版社

[12]TCP/IP详解 卷3：TCP事务协议、HTTP、NNTP和UNIX域协议 [美]史蒂文斯（Stevens,W.R.） 著；胡谷雨等 译 /2000-09-01 /机械工业出版社

[13]计算机网络:自顶向下方法（原书第6版）（美）库罗斯，（美）罗斯 著，陈鸣 译 /2014-10-01 /机械工业出版社

[14]MongoDB权威指南（第2版）作者:(美)霍多罗夫 2014年01月 人民邮电出版社

[15]陈竹敏.面向垂直搜索引擎的主题爬虫技术研究[ D][博士论文].山东大学，2008: 10.

[16] Kleinberg J. Authoritative sources in a hyperlinked environment[J].Journal of the ACM ，1999, 46 (5). 604-632

[ 17]杨贞，等.基于本体的主题爬虫的设计与实现[J].科技情报开发与经济，2008, (2): 73-75.

[ 18]姚树宇，赵少东.一种使用分布式技术的搜索引擎[ J].计算机应用与软件，2005, 22 (10 ) : 127-129.

[19]Python爬虫开发与项目实践 范传辉 机械工业出版社 2017年6月

[20]数学之美(第二版)作者吴军博士 人民邮电出版社 2014年

[21]TCP/IP详解 卷1：协议（第2版） （美）Kevin R. Fall，W. Richard Stevens 著 /2012-05-01 /机械工业出版社

[22]TCP/IP详解 卷3：TCP事务协议、HTTP、NNTP和UNIX域协议 [美]史蒂文斯（Stevens,W.R.） 著；胡谷雨等 译 /2000-09-01 /机械工业出版社

[23]计算机网络:自顶向下方法（原书第6版）（美）库罗斯，（美）罗斯 著，陈鸣 译 /2014-10-01 /机械工业出版社

[24]MongoDB权威指南（第2版）作者:(美)霍多罗夫 2014年01月 人民邮电出版社

[25]知乎论坛 https://zhuanlan.zhihu.com/p/24669128

[26]简书创作 <https://www.jianshu.com/p/d16076661d40>

[27]http://www.runoob.com/mongodb/mongodb-window-install.html

[28]零基础入门学习Python 2016年11月1日小甲鱼 (作者)

[29]利用Python进行数据分析 (O'Reilly精品图书系列) Kindle电子书Wes McKinney (作者),‎ 唐学韬 (译者)

[30]Python编程 从入门到实践 2018年1月26日[美]埃里克·马瑟斯 (作者),‎ 袁国忠 (译者)

[31]MongoDB大数据处理权威指南(第3版) 2017年3月1日戴维·豪斯 (David Hows) (作者),‎ Peter Membrey (作者),‎ Eelco Plugge (作者),‎ Tim Hawkins (作者),‎ 周连科 (译者)

[32]Redis开发与运维 2017年2月28日付磊 (作者),‎ 张益军 (作者)

[33]罗兵.支持AJAX的互联网搜索引擎爬虫设计与实现 浙江大学出版 , 2007

[34]徐远超,刘江华,刘丽珍,关永 基于Web的网络爬虫的设计与实现《微计算机信息》 2007，23（21）：119-121

[35]赵本本, 殷旭东,王伟,基于Scrapy的GitHub数据爬虫 出版《电子技术与软件工程》，2016（6）：199-202

[36]姜杉彪，黄凯林，卢昱江，张俊杰，曾志高，基于Python的专业网络爬虫的设计与实现 出版《企业科技与发展》，2016（8）：17-19

[37]李勇，韩亮 主题搜索引擎中网络爬虫的搜索策略研究 出版《计算机工程与科学》，2008，30（3）：4-6