Soapy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。Soapy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。

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

Spider

(1)解析Downloader返回的响应（Response）

(2)产生爬取项（scraped item） pip install Scrapy

#验证是否安装成功

pip list

#输出如下

cffi (0.8.6)

cryptography (0.6.1)

cssselect (0.9.1)

lxml (3.4.1)

pip (1.5.6)

pycparser (2.10)

pyOpenSSL (0.14)

queuelib (1.2.2)

Scrapy (0.24.4)

setuptools (3.6)

six (1.8.0)

Twisted (14.0.2)

w3lib (1.10.0)

wsgiref (0.1.2)

zope.interface (4.1.1)

(3)产生额外的爬取请求（Request）

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.

比如说，MIT发表过一篇论文.[Bayesian regression and Bitcoin][.](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)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 data训练出的部分交易结果图，绿色点买入，红色点卖出。

所谓网络爬虫，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，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。

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。

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>

3.4.2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=

 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

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.

Ajax

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“提交一响应一等待一刷新”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。

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。

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。

通过JSON进行前后端交互的爬取技巧解决上面的问题。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bilibili视频的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：

这一节介绍一下通过JSON进行前后端交互的爬取技巧和一些我个人觉得挺有意思的反爬措施.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：

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）。   
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。

找到这个服务商，这就是为什么我们需要爬虫信息交流的平台。

笔者有意建立这样的平台，不管是微博、微信公众号、文章，还是政府门户的信息公开，或者门户网站、新闻社区等媒体，或者其他领域如金融的数据，都可以在平台上找到，也很容易实现供应和需求者的相互匹配。

然而，对于这其中的法律问题，特别是知识产权问题，不甚明了，除了robots协议，很多数据网站都会另外声明其数据使用规范，同时，如大众点评等网站提供的API很难满足分析的需要，就形成了数据被独占而其价值得不到有效开发的不合理现象。

大数据时代，对大数据的分析应当成为一个行业，数据拥有者应该开放数据的分析接口，让数据的价值释放，而爬虫开发者，很多时候是数据分析者（最起码是个数据清洗和筛选者）。他们蒙上了一层神秘面纱，带着一丝黑客气息，法律应当给他们更大的生存空间，让这个有价值的行业创造更大的价值。

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。

网络爬虫现状：

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。

国内外研究现状

当前互联网发展迅猛，对大数据的利用也越来越多，对大数据处理的要求也

越来越高，由此产生了云计算。因为云计算机提出时间还很早，很多企业或用户

都对其不甚了解，所以对它也没有具体且统一的定义。然而，根据一些权威机构

或企业对云计算的介绍，还是能够得出云计算的主要特征。如Wikipedia定义云计

算的概念是“基于网格计算的新型标志，利用公共的计算等方法实现资源共享。它

不同于利用本机或个人计算机的应用程序来处理事务”;News Blog则将云计算定

义为“把硬、软件交由互联网提供商的一种新技术”;Forrester把云计算称为某种非

常经典的具有很强创新性的技术。

一直到现在，Microsoft、亚马逊、谷歌、EMC, IBM以及惠普等公司都己经

发表声明，将着重建立基于企业级的云计算架构，其中成功的例子也不少。

对于基于企业级的云计算案例，最受关注的就是2009年谷歌与IBM一起打造

的“云计算”计划[f6l，参与这一计划的不仅仅有大型企业，还有像斯坦福大学、加

利福利亚大学、华盛顿大学以及中国的清华大学等等著名的教育教学机构。这些

学校或企业通过参与这一计划，获得了很多的好处，比如学校能够借助大企业丰

富的资源来研究创新性强的应用;而企业则能够借助学校丰富的科研人才来研究

云计算，以促进其发展。

在谷歌与IBM进行“云计算”计划的同时，亚马逊也开发出了它自己的云计算

架构EC2 }'}，这一架构系统具有很大的开放性，使用者可以利用它对大数据进行

计算。这样不仅大大提高数据计算的效率，而且还省去了自己搭建计算平台的麻

烦。

1. 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

1. 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，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。

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 ).或者在一此文献中称为“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，l(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”( Hasb 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，即使更新最频萦的基础索引也有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个服务器，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。

**一、网站的树结构**

1.1、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。

1.2、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。

Cookies:

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cookes一起识别。

在Python处理Cookie，一般是通过cookielib模块和 urllib2模块的HTTPCookieProcessor处理器类一起使用。

cookielib模块：主要作用是提供用于存储cookie的对象

HTTPCookieProcessor处理器：主要作用是处理这些cookie对象，并构建handler对象。

cookielib 库：该模块主要的对象有CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

CookieJar：管理HTTP cookie值、存储HTTP请求生成的cookie、向传出的HTTP请求添加cookie的对象。整个cookie都存储在内存中，对CookieJar实例进行垃圾回收后cookie也将丢失。

FileCookieJar (filename,delayload=None,policy=None)：从CookieJar派生而来，用来创建FileCookieJar实例，检索cookie信息并将cookie存储到文件中。filename是存储cookie的文件名。delayload为True时支持延迟访问访问文件，即只有在需要时才读取文件或在文件中存储数据。

MozillaCookieJar (filename,delayload=None,policy=None)：从FileCookieJar派生而来，创建与Mozilla浏览器 cookies.txt兼容的FileCookieJar实例。

LWPCookieJar (filename,delayload=None,policy=None)：从FileCookieJar派生而来，创建与libwww-perl标准的 Set-Cookie3 文件格式兼容的FileCookieJar实例。

其实大多数情况下，我们只用CookieJar()，如果需要和本地文件交互，就用 MozillaCookjar() 或 LWPCookieJar()

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。

在此之前呢，我们必须先介绍一个opener与Cookielib的概念。

**1.Opener**

当你获取一个URL你使用一个opener(一个urllib2.OpenerDirector的实例)。在前面，我们都是使用的默认的opener，也就是urlopen。它是一个特殊的opener，可以理解成opener的一个特殊实例，传入的参数仅仅是url，data，timeout。

如果我们需要用到Cookie，只用这个opener是不能达到目的的，所以我们需要创建更一般的opener来实现对Cookie的设置。

**2.Cookielib**

cookielib模块的主要作用是提供可存储cookie的对象，以便于与urllib2模块配合使用来访问Internet资源。Cookielib模块非常强大，我们可以利用本模块的CookieJar类的对象来捕获cookie并在后续连接请求时重新发送，比如可以实现模拟登录功能。该模块主要的对象有CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。

它们的关系：CookieJar —-派生—->FileCookieJar —-派生—–>MozillaCookieJar和LWPCookieJar

**1）首先获取Cookie保存到变量**

首先，我们先利用CookieJar对象实现获取cookie的功能，存储到变量中，先然后保存Cookie到文件在，我们将cookie保存到了cookie这个变量中，如果我们想将cookie保存到文件中该怎么做呢？这时，我们就要用FileCookieJar这个对象了，在这里我们使用它的子类MozillaCookieJar来实现Cookie的保存

**2）保存Cookie到文件**

在上面的方法中，我们将cookie保存到了cookie这个变量中，如果我们想将cookie保存到文件中该怎么做呢？这时，我们就要用到

FileCookieJar这个对象了，在这里我们使用它的子类MozillaCookieJar来实现Cookie的保存

**3）从文件中获取Cookie并访问**

那么我们已经做到把Cookie保存到文件中了，如果以后想使用，可以利用下面的方法来读取cookie并访问网站，感受一下设想，如果我们的 cookie.txt 文件中保存的是某个人登录新浪微博的cookie，那么我们提取出这个cookie文件内容，就可以用以上方法模拟这个人的账号登录百度。

**4）再来利用cookie模拟网站登录**

利用cookie实现模拟登录，并将cookie信息保存到文本文件中，来感受一下cookie**。**关于最后save方法的两个参数在此说明一下：

官方解释如下：ignore\_discard: save even cookies set to be discarded.

ignore\_expires: save even cookies that have expiredThe file is overwritten if it already exists

由此可见，ignore\_discard的意思是即使cookies将被丢弃也将它保存下来，ignore\_expires的意思是如果在该文件中cookies已经存在，则覆盖原文件写入，在这里，我们将这两个全部设置为True。运行之后，cookies将被保存到cookie.txt文件中，我们查看一下内容。再来那么我们已经做到把Cookie保存到文件中了，如果以后想使用，可以利用下面的方法来读取cookie并访问微博。

以下程序的原理如下：

创建一个带有cookie的opener，在访问登录的URL时，将登录后的cookie保存下来，然后利用这个cookie来访问其他网址。在世界编写代码中我把15个微薄的账号

关于重写中间件的详细情况 请参考 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。

1 数据获取

在上一篇文章里，我提了一下

爬的方法是，从我开始，DFS 层级展开关注的人。技术栈大概就是用 redis 做队列和记录，gevent 协程，mongodb 做数据库， 做底层爬虫。一共 10 个 worker，再加 1 个 db writer 10 秒一次 commit，还有一个 socket server 管 IPC。

我爬的东西很少，就是用户的基础信息，在后面会介绍的。

爬了多少天我忘了，因为写了暂停和继续功能，我基本是开一下关一下的，不过速度确实不快。我记得有个人 PHP 写的一天就能 100w 呢，最好的语言就是6！

好啦，正式开始吧，介绍一下数据先：

因为还没爬完，所以数据量比较小，2528099 个用户

记录的属性有：用户 ID，用户名，签名，性别，答题数，问题数，收到赞数，收到感谢数，关注了多少人，被多少人关注，学校，专业，行业，公司，职位

没有其他和答案，问题，话题，文章，评论，用户关注关系有关的数据

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。

2. 1. 2哈希表

MDS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，MDS ( )函数产生的值很大，为2}zs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MDS()函数的值进行模运算映射到哈希表中。其公式可设为:

MDS (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MDS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的U}的Nvs集合判断是否已抓取过.如没有被抓取.则放入“未爬行的U}数据库”中.否则放入“已爬行的URI库”

用的是scrapy+redis分布式架构（我这里是单机不用分布式，但是想变为分布式爬虫只变机器的部署就行），任务调度用的scrapy\_redis模块大家应该知道scrapy是默认开启了去重的，用了scrapy\_redis后去重队列放在redis里面。

1、首先看调度器，即scrapy\_redis模块下的scheduler.py文件，可以看到enqueue\_request()方法和next\_request()方法就是种子入队列和出队列的地方，self.queue指的是我们在setting.py里面设定的SCHEDULER\_QUEUE\_CLASS值，常用的是'scrapy\_redis.queue.SpiderPriorityQueue'。

2可以看到，上面用到了Redis的zset数据结构（它可以给种子加优先级），在进Redis之前用\_encode\_request()方法将Request对象转成字符串，\_encode\_request()和\_decode\_request是Base类下面的两个方法：

这里先将Request对象转成一个字典，再将字典序列化成一个字符串

3、了解完scrapy\_redis默认的种子处理方式，现在针对自己的项目作一些调整。我的是一个全网爬虫，每个种子需要记录的信息主要有两个：url和callback函数名。此时我们选择不用序列化，直接用简单粗暴的方式，将callback函数名和url拼接成一条字符串作为一条种子，这样种子的长度至少会减少一半。另外我们的种子并不需要设优先级，所以也不用zset了，改用Redis的list。以下是我新建的SpiderSimpleQueue类，加在queue.py中。如果在settings.py里将SCHEDULER\_QUEUE\_CLASS值设置成'scrapy\_redis.queue.SpiderSimpleQueue'即可使用我这种野蛮粗暴的种子。

4、另外需要提醒的是，如果scrapy中加了中间件process\_request()，当yield一个Request对象的时候，scrapy\_redis会直接将它丢进Redis种子队列，未执行process\_requset()；需要一个Request对象的时候，scrapy\_redis会从Redis队列中取出种子，此时才会处理process\_request()方法，接着去抓取网页。

所以并不需要担心process\_request()里面添加的Cookie在Redis中放太久会失效，因为进Redis的时候它压根都还没执行process\_request()。事实上Request对象序列化的时候带上的字段很多都是没用的默认字段，很多爬虫都可以用 “callback+url” 的方式来优化种子。

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

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。在此，特向宋老师表示最衷心的谢谢!

本课题来自于实验室的“网络内容监控系统一WCMS ( Web Content Monitoring

S州em )”项目，属于WCMS系统中的网络爬虫部分。本文着眼于中小规模，力

求实现一个能灵活部署和配置的分布式网络爬虫系统一DWCS ( Distributed Web

Crawler System ) o DWCS系统的开发目标如下:

可移植与跨平台性。现在会用到网络爬虫的情况有很多，所以DWCS系统的

设计目标之一就是良好的可移植性与跨平台性。使DWCS系统的部署更加灵活，

能够适用于多种操作系统平台。

高效性。在有限的资源与带宽下，以尽量小的资源消耗，获取尽量多的数据。

轻量级。只保留最基本和必要的功能，减小系统体积，简化系统的配置与运

行步骤。

由于对网络爬虫领域了解有限，所以不可能在一开始就预见到以后将会遇到的

问题和困难，因此需要将DWCS系统划分成数个功能模块进行开发。在遇到问题

的时候逐一解决，并在必要的时候对系统进行重构，以适应新的环境和需求。

本文的主要工作是设计并实现一个基于微博的数据挖掘可视化系统。系统包括三

部分:数据采集子系统、数据挖掘子系统和数据可视化子系统。

在数据采集子系统中，本文比较并分析了各种微博数据获取方案，最终选择采用

基于模拟登录的网页爬虫，以实现全面高效的微博信息采集。

在数据挖掘子系统中，设计并实现了微博数据挖掘的各功能模块，并重点研究了

基于机器学习的微博文本情感分析。采用多种特征抽取方法和多种机器学习算法训练

分类模型，得到的分类模型正确率达到85%, AUC值达到0. 94 0

在数据可视化子系统中，通过浏览器呈现微博数据挖掘的结果。系统遵循MVC

C Model-View-Controller)原则，前端采用HTML, CSS和JavaScript实现，并利用

Echarts框架实现柱状图、折线图、饼图、地图、词云等多种可视化分析，后台控制层

采用PHP实现，数据库采用MySQLo

1.3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。

1.4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。

1.5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.多线程并发实现。根据爬取目标的网页设计，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。

4.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。

北京邮电人学计算机科学与技术硕士论文

第六章总结与展望

6. 1论文工作总结

随着互联网络的迅速发展，WWW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

平衡等。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。

本文变针对对于现有主题网络爬虫的研究，和自己实践经验的探索，对于可

能存在的问题和改进方式进行了详细的解释。同时也制定相应的实验平台，根据

修改后的算法，依据一定的实验数据加以说明。主要的创新点是针对现在主题网

络爬虫系统中普遍存在的隧道现象提出了相应的算法改进并应用于Fish-Search

以及Shark-Search算法之中，另外还对于主题爬虫架构中的主题网页分析模块进

行了适当了修改，目的是提高爬虫的爬取效率。

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。

6.2问题与展望

本文虽然对于主题网络爬虫的搜索策略以及关键的架构方面做出了适当的

改进，不过实际网络爬虫应用中还是有一些需要注意的问题。

一方面，在进行主题爬虫相关度计算时，运用的一般都是以往前人的经验，

所以对于阑值的如何选择，也没有实现如何才‘能做到最好。闭值设定过高或者过

小难免会影响实验结果的分析，这也是需要进一步研究的目标。

另一方面，由于网络资源中的网页动态增长，而本文只是通过静态的网络资

源进行的实验，对于如何获得实时性，真正满足用户的需求，并没有特别的研究，

这也是下一步主要的工作所在。

社交网络的出现和兴起正在不知不觉中改变着人们的生活习惯和社交行为以

及获取信息的方式。微博作为社交网络的主要平台之一，已经引起了人们越来越多

的关注和重视。而处在大数据时代中，数据的价值无法估量。通过对海量数据进行

分析，我们可以从中提取出未被发现的、具有研究和商业价值的规律和内容。由于

微博在信息传递方面的巨大优势以及其拥有的广泛而又多元化的用户群体，因此

使得微博相关数据的价值更为突出和重要。

对此，为了探究影响微博信息传递的主要因素，总结影响微博传播的重要因子，

以此对用户行为进行预测，从而更好的探究关于微博用户行为的规律。

在利用了网络爬虫获取了不同种类微博数据的基础上，本文提取了关于微博

和用户的13项基本特征，并利用机器学习算法对这些特征进行训练，得出了预测

模型，并找到了对微博转发影响较大的基本特征，首次对特征组合的影响进行讨论。

并探究不同算法对具有不同特点的数据有哪些影响，找出了最优算法。

同时，针对微博情感分类预测方面的问题，本文提出了基于分类器混合模型的

情感分析方法，并与文献[[43]进行了比较，最后的实验结果也证明了该方法在处理

微博内容情感分析方面具有较高的性能是预测效果。

在接下来的工作中，要对网络爬虫系统进行进一步优化，提高数据爬取效率;

探究不同机器学习算法对不同特征的数据最适应的算法，进一步优化实验结果，提

高预测准确率。并尝试用不同的分类算法对分类器的混合模型进行优化改进，以希

望得到更高的预测效果。

在本文的研究基础上，可以在以下三个方面进一步研究:

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。

摘要：

基于Soapy框架的网络爬虫实现与数据抓取分析

随着信息时代的发展和编程技术的普及，搜索引擎成为了人们日常生活中的必须

品。搜索引擎大多使用爬虫技术作为核心模块，通过关键词返回用户查询的结果。但是

网络信息呈现爆炸式的增长，使得信息的查找和定位也变得困难。

针对上述问题，本文以Python和Soapy环境为基础，以“新浪微博”为爬取对象，

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S crapy框架的网络爬虫，完成数据抓取等目标。

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和Robot协议。

其次，通过使用基于Python语言开发的S crapy开源爬虫框架来进行爬虫开发，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S crapy

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。

采用了更换Cookie和user-agent欺骗来突破站点限制。而URL去重和多线程并发的问

题，则采用并分析S crapy自带的解决方案。

最后对爬虫进行测试并展示成果，思考存在的问题和改进的可能。

关键词:

爬虫，Soapy, URL去重，Python, Cookie

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.

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.