

中国矿业大学计算机学院

2017级本科生课程报告

课程名称 信息内容安全

报告题目 基于分布式爬虫的文章搜索引擎

报告时间 2020.6.23

姓 名 袁孝健

任课教师 曹天杰

2019-2020(二)《信息内容安全》课程报告评分表

姓名 袁孝健 学号 06172151 班级 信息内容安全 2017-1 班

序号	毕业要求	课程教学目标	考查方式与考查点	占比	得分
1	2.3	目标 1: 掌握信息内容安全的基本概念、分类、原理和相关技术，能够根据课程基本知识对信息内容安全领域出现的问题进行归类、分析、并有初步分析和解决问题的能力。	通过课堂讲授和课堂研讨掌握信息内容安全概念和理论知识。	40%	
	3.2	目标 2: 掌握信息内容安全处理相关的理论、技术以及健全的评价体系，能够根据具体问题设计算法、设计算法、实现算法并能综合评价算法。			
2	4.3	目标 3: 掌握信息内容安全的基础知识，针对具体问题和要求选择正确的技术路线，通过在实验环境中进行仿真实验并能根据算法特点进行攻击测试和综合性能评价，得到具有参考价值的结论。	课程报告：实现有关信息内容安全的一个软件系统。分析和对比各项技术，选择相应的技术进行算法设计并在实验环境中进行仿真实验和性能评价，得到有效结论。	60%	
总分				100%	

评阅人：

2020 年 7 月 10 日

摘 要

随着大数据时代的到来,信息的获取与检索尤为重要,如何在海量的数据中快速准确获取到我们需要的内容显得十分重要。因此本项目为了更好的整合利用安全领域特有的社区资源优势,首先使用 Scrapy 爬虫框架结合 NoSQL 数据库 Redis 编写分布式爬虫,并对先知、安全客、嘶吼三个知名安全社区进行技术文章的爬取;然后选取 Elasticsearch 搭建搜索服务,同时提供了 RESTful web 接口;最后通过 Django 搭建可视化站点,供用户透明的对文章进行搜索。

关键词: 分布式爬虫; Scrapy; 搜索引擎; Redis

Abstract

With the advent of the era of big data, the acquisition and retrieval of information is particularly important. How to get the content we need quickly and accurately in massive data is very important. Therefore, in order to better integrate and utilize the unique community resource advantage in the security field, this project firstly uses the Scrapy crawler framework combined with NoSQL database Redis to write distributed crawler, and then crawl for technical articles to three well-known security communities: XianZhi, AnQuanKe and RoarTalk. Then select Elasticsearch to build the search service while providing a RESTful Web interface. Finally, the visual site is built through Django for users to search articles transparently.

Key words: distributed crawler; Scrapy; Search engines; Redis

目 录

摘 要.....	1
1 概述.....	6
2 技术选型.....	6
2.1 Scrapy-Redis 分布式爬虫.....	6
2.1.1 Redis.....	6
2.1.2 Scrapy.....	7
2.2 MySQL 数据存储.....	8
2.3 Django 搭建搜索网站.....	8
2.4 ElasticSearch 搜索引擎.....	9
2.4.1 Elasticsearch-RTF.....	9
2.4.2 Elasticsearch-head.....	10
2.4.3 Kibana.....	10
3 实现细节.....	10
3.1 处理反爬.....	10
3.1.1 更换随机 User-Agent.....	10
3.1.2 使用 IP 代理池.....	11
3.1.3 访问频率限制.....	12
3.1.4 Cookie 的禁用.....	13
3.1.5 验证码识别.....	13
3.2 爬取数据.....	14
3.2.1 先知社区.....	14
3.2.2 安全客.....	17
3.2.3 嘶吼.....	19
3.3 重构分布式爬虫.....	21
3.3.1 需要解决的问题.....	21
3.3.2 分布式的原理.....	22
3.3.3 分布式的实现.....	23
3.4 搜索引擎.....	24
3.4.1 倒排索引.....	24
3.4.2 排序评分.....	25
3.4.3 搜索提示.....	26
3.4.4 模糊搜索.....	27

3.5 网页搭建.....	28
3.5.1 爬虫统计数据.....	28
3.5.2 热门搜索.....	28
3.6 其他技术.....	29
3.6.1 URL 去重策略.....	29
3.6.2 Bloom Filter 使用.....	31
4 系统展示.....	34
4.1 分布式爬取.....	34
4.2 搜索网站首页.....	36
4.3 搜索提示展示.....	36
4.4 搜索结果展示.....	37

1 概述

爬虫的应用领域非常广泛，目前利用爬虫技术市面上已经存在了比较成熟的搜索引擎产品，如百度、谷歌，以及其他垂直领域搜索引擎，这些都是非直接目的；还有一些推荐引擎，如今日头条，可以定向给用户推荐相关新闻；爬虫还可以用来作为机器学习的数据样本。

本项目的主要目的是为了为了更好的整合利用安全领域特有的社区资源优势。首先使用 Scrapy 爬虫框架结合 NoSQL 数据库 Redis 编写分布式爬虫，并对先知、安全客、嘶吼三个知名安全社区进行技术文章的爬取；然后选取 ElasticSearch 搭建搜索服务，同时提供了 RESTfulweb 接口；最后通过 Django 搭建可视化站点，供用户透明的对文章进行搜索。

最终通过本项目可以更加透彻的理解爬虫的相关知识；在熟练运用 Python 语言的基础上，更加深入的掌握开源的爬虫框架 Scrapy，为后续其他与爬虫相关的业务奠定理论基础和数据基础；进一步理解分布式的概念，为大数据的相关研究和硬件条件奠定基础；熟练掌握 Python 搭建网站的框架 Django，深入理解基于 Lucene 的搜索服务器 ElasticSearch。

2 技术选型

2.1 Scrapy-Redis 分布式爬虫

2.1.1 Redis

Redis 是完全开源免费的，遵守 BSD 协议的，高性能的 key-value 数据库。Redis 与其他 key-value 缓存产品有以下三个特点：

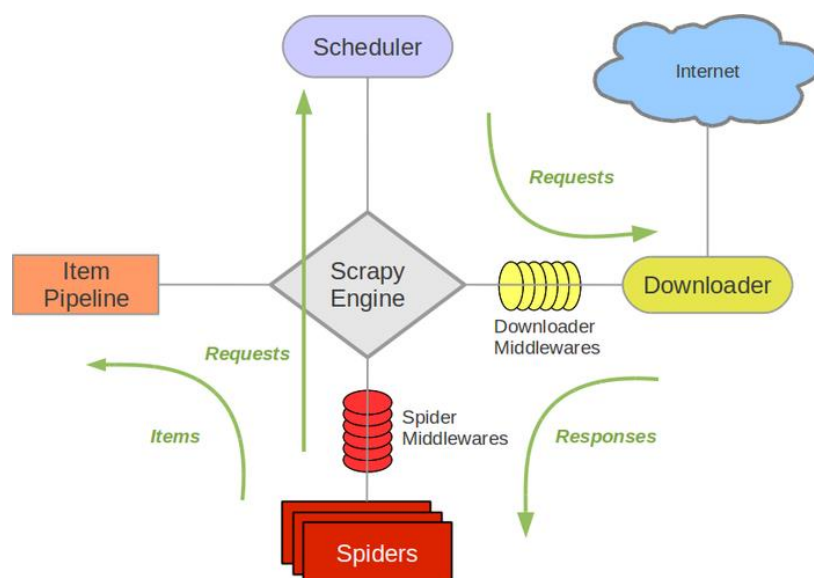
(1) Redis 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。这样可以防止数据的丢失，在实际生产应用中数据的完整性是必须保证的。

(2) Redis 不仅仅支持简单的 key-value 类型的数据，同时还提供 list, set, zset, hash 等数据结构的存储。这些功能更强大的数据存储方式极大地节约了存储空间，优化了查询的性能，大大提高了查询效率。存储的目的是为了后期更好的取出，Redis 很好地做到了这一点。

(3) Redis 支持数据的备份，即 master-slave 模式的数据备份。主从结构目前是大数据里面的主流结构，主从模式能保证数据的健壮性和高可用。当出现电脑宕机，硬盘损坏等重大自然原因时，主从模式能很好的保证存储的数据不丢失，随时恢复到可用状态。

2.1.2 Scrapy

Scrapy 的原理如下所示：



各个组件的解释如下：

(1)Scrapy Engine(引擎):负责 Spider、Item Pipeline、Downloader、Scheduler 中间的通讯,信号、数据传递等,相当于人的大脑中枢,机器的发动机等,具有显著的作用。

(2) Scheduler(调度器):负责接收引擎发送过来的 Request 请求,并按照一定的方式逻辑进行整理排列,入队,当引擎需要时,再交还给引擎。

(3)Downloader(下载器):负责下载 Scrapy Engine(引擎)发送的所有 Requests 请求,并将其获取到的 Responses 交还给 Scrapy Engine(引擎),由引擎交给 Spider 来处理。

(4) Spider(爬虫):负责处理所有的 Responses,从中分析提取数据,获取 Item 字段需要的数据,并将需要跟进的 URL 提交给引擎,再次进入 Scheduler(调度器)。

(5) Item Pipeline(管道):负责处理 Spider 中获取到的 Item,并进行后期处理(详细分析、过滤、存储等)。

(6) Downloader Middlewares(下载中间件):可以当作是一个可以自定义扩展下载功能的组件。

(7) Spider Middlewares(Spider 中间件):可以理解为是一个可以自定义扩展和操作引擎以及 Spider 中间通信的功能组件(例如进入 Spider 的 Responses;和从 Spider 出去的 Requests)。

整个 Scrapy 爬虫框架执行流程可以理解为:爬虫启动的时候就会从

start_urls 提取每一个 URL，然后封装成请求，交给 engine，engine 交给调度器入队列，调度器入队列去重处理后再交给下载器去下载，下载返回的响应文件交给 parse 方法来处理，parse 方法可以直接调用 XPATH 方法提取数据了。

Scrapy 是一个通用的爬虫框架，但是不支持分布式，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，具体的分布式 Scrapy-Redis 介绍见本文 3.3 章节。

2.2 MySQL 数据存储

MySQL 是一个关系型数据库管理系统，最流行的关系型数据库管理系统之一，在 WEB 应用方面，MySQL 是最好的 RDBMS 应用软件之一。MySQL 是一种关系型数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

正是基于 MySQL 体积小、速度快、成本低的特点，本项目使用它对爬取下来的数据进行长期稳定的存储，同时利用 Scrapy 还可以异步进行数据的存储。在本项目中，我们要爬取的数据至少包括：文章题目、文章内容、文章链接、文章标签、作者、发表时间、访问量、封面图等。

title	create_date	url	url_object_id	front_image_url	from
Radare2从入门到进阶(上)	2020-05-27 09:27:00	https://xz.aliyun.com/t/7780	f14bb6cd17e9df346a86373a4f7448c1	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
Cobalt Strike 内网渗透教程记录	2020-02-27 09:05:41	https://xz.aliyun.com/t/7265	01a284cd40ad98e0826675wdb3eb1d3	https://xzfile.aliyuncs.com/media/upload/avatars/21852_2f8c9ba40d9f669b.png	full/
如何写 Suricata 和 ELK 进行流量检测	2020-03-10 09:53:27	https://xz.aliyun.com/t/7375	69808ad15d499aab787d23c4a86e9b6	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
使用 Suricata 和 ELK 进行流量检测	2019-12-30 09:18:00	https://xz.aliyun.com/t/7007	1de6fa118115d545f6771a26ad44f95c11	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
GitHub 数据爬取工具 GSIIL	2020-02-26 09:38:00	https://xz.aliyun.com/t/7263	66e3f177dd65d5d4a2e0d0474c2aff	https://xzfile.aliyuncs.com/media/upload/avatars/1874_62d3761eb4fd92e631.png	full/
Stowaway - go 语言编写的多线程工具	2020-02-19 08:54:00	https://xz.aliyun.com/t/7229	43b62e608bd2c6b73c9eac74b4d0b7	https://xzfile.aliyuncs.com/media/upload/avatars/10995_36191b735a3f1cf36c.png	full/
通过 AST 来构造 pickle opcode	2020-01-23 09:22:00	https://xz.aliyun.com/t/7119	6e5d48b5c323a8a3fba558572b652b0	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
一款适合甲方构造 pickle 的工具	2019-12-30 09:24:31	https://xz.aliyun.com/t/7012	3e5d8f6778529b214eeb9345bd740e35	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
DIY 树莓派搭建 Kali Linux	2020-02-01 15:49:00	https://xz.aliyun.com/t/7133	8be2577b3fdd9f0b0d798f3b59d413	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
数字取证之哈希值与哈希表	2020-03-20 09:52:00	https://xz.aliyun.com/t/7359	e872becba7abe59092f25ac8037c91e	https://xzfile.aliyuncs.com/media/upload/avatars/A009_3714391f8498a84e41.png	full/
Empire 进阶研究	2020-01-06 09:39:00	https://xz.aliyun.com/t/7047	a53396b51a6400c0d4be836a5e1e765d	https://xzfile.aliyuncs.com/media/upload/avatars/16201_958bc0da2b80ad8e16.png	full/
JWT PyCrack - JWT 攻击脚本	2020-01-10 09:36:00	https://xz.aliyun.com/t/7071	218520a59407079e12a9e846f483d5	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
打造一个通用的登录验证框架	2020-02-06 09:12:30	https://xz.aliyun.com/t/7165	3dc243236529c14b663d1253bd88f7d	https://xzfile.aliyuncs.com/media/upload/avatars/12945_8ed1635e5ba6478227.png	full/
Go_Auto_Proxy - 自动设置代理 (Windows)	2020-02-06 09:12:30	https://xz.aliyun.com/t/7165	84da00599433c790584625ec1aee07b	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
venom@powershell 免杀技术分析	2019-11-02 10:24:00	https://xz.aliyun.com/t/6509	75fc38316737472ee1c24d7de01c8aa8	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
Android 智能终端系统的安全加固 (上)	2019-12-01 10:01:45	https://xz.aliyun.com/t/6852	4a2c3c3fab3c5e814857d01f75d4af7	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
薄雾个人渗透系统 7.0 版 - 正式发布	2019-10-10 09:07:54	https://xz.aliyun.com/t/6494	9ee59e134ec56b1f4ef357d4ec7e85d9	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
Hive - 开源实时数仓平台	2019-12-09 09:19:46	https://xz.aliyun.com/t/6889	1c5abdd32c196f8b3a9f6baaeb583b30	https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/
zerotool - 基于 angr 的 CTF pwn 自动化利用工具介绍	2020-02-18 09:11:39	https://xz.aliyun.com/t/7224	1d3c0da99658d800ad21070a38f766	https://xzfile.aliyuncs.com/media/upload/avatars/23116_186fc6701ea30e2f69.png	full/

front_image_url	front_image_path	author	view_count	follow_count	mark_count	tags	content
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	阿狸shuqi	5391	1	0	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/21852_2f8c9ba40d9f669b.png	full/141ac77688e6e27	ERFZE	7028	2	3	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	castiel	3921	2	3	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	此生已尽我	9634	4	9	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/1874_62d3761eb4fd92e631.png	full/5cfdedfd3b5aa37c	raul	7186	1	1	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/10995_36191b735a3f1cf36c.png	full/55954503d02e3f11	All ex	7809	1	2	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	ph4n***	15371	1	2	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	lv4n	7252	3	3	安全工具, 工具	<div id="topic_content" class="topic-conte
https://xzfile.aliyuncs.com/media/upload/avatars/default_avatar.png	full/abc451300b12bfc	99008***g	10312	1	0	安全工具, 工具	<div id="topic_content" class="topic-conte

2.3 Django 搭建搜索网站

Django 是一个开放源代码的 Web 应用框架，由 Python 开发的基于 MVC 构造的框架。在 Django 中，控制器接受用户输入的部分由框架自行处理，因此更加关注模型，模板和视图，即 MVT。

- 模型 (Model)，即数据存取层，处理与数据相关的所有事物：包括如何存取，如何验证有效性，数据之间的关系等。
- 视图 (View)，即表现层，处理与表现相关的逻辑，主要是显示的问题。
- 模板 (Template)，即业务逻辑层，主要职责是存取模型以及调取恰当模板的

相关逻辑。

控制器部分，由 Django 框架 URLconf 来实现，而 URLconf 机制恰恰又是使用正则表达式匹配 URL，然后调用合适的函数。因此只需要写很少量的代码，只需关注业务逻辑部分，大大提高了开发的效率。使用 Django 搭建搜索引擎的界面，简单便捷且界面交互效果良好，适应需求，无须成本。同时 Scrapy 与 Django 的代码模式有些类似，如 Django 中的 Model 与 Scrapy 中的 Item，Scrapy 也提供了相应的 DjangoItem 类，其从 Django 模型中获取字段(field)定义。

2.4 Elasticsearch 搜索引擎

ElasticSearch 是一个基于 Lucene 的实时的分布式搜索和分析引擎，设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用非常方便。基于 RESTful 接口。ElasticSearch 具有广泛的用户，如 DELL，GitHub，Wikipedia 等。ElasticSearch 和关系型数据库之间的对比如下所示：

Index（索引库）	Database（数据库）
Type（类型）	Table（表）
Document（文档）	Row（行）
Field（字段）	Column（列）

2.4.1 Elasticsearch-RTF

Elasticsearch-RTF 是针对中文的一个发行版，使用稳定的 Elasticsearch 版本，并且下载测试好对应的插件，如中文分词插件等，目的是可以下载下来就可以直接的使用。项目构建过程中选择的是 Elasticsearch-RTF 5.1.1 版本。安装后启动，效果如下所示，通常运行在 9200 端口：



```
{
  "name" : "0X3QpYz",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "qbPg54U_RLWQGeJMc7httg",
  "version" : {
    "number" : "5.1.1",
    "build_hash" : "5395e21",
    "build_date" : "2016-12-06T12:36:15.409Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

2.4.2 Elasticsearch-head

ElasticSearch-head 是一个 Web 前端插件，用于浏览 Elastic-Search 集群并与之进行交互，它可以作为 ElasticSearch 插件运行，一般首选这种方式，当然它也可以作为独立的 Web 应用程序运行。它的通用工具有三大操作：

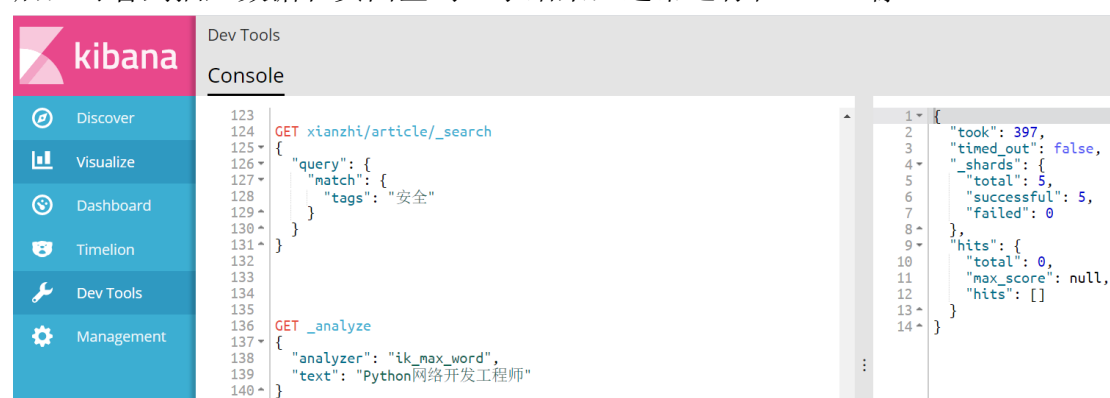
ClusterOverview，显示当前集群的拓扑，并允许执行索引和节点级别的操作；有几个搜索接口可以查询原生 Json 或表格格式的检索结果；显示集群状态的几个快速访问选项卡；一个允许任意调用 RESTful API 的输入部分。

运行结果如下，通常运行在 9100 端口：



2.4.3 Kibana

Kibana 是一个开源的分析与可视化平台，设计出来用于和 Elasticsearch 一起使用的。可以用 kibana 搜索、查看、交互存放在 Elasticsearch 索引里的数据，使用各种不同的图表、表格、地图等。kibana 能够很轻易地展示高级数据分析与可视化。Kibana 使理解大量数据变得容易。它简单、基于浏览器的接口能快速创建和分享实时展现 Elasticsearch 查询变化的动态仪表盘。Kibana 启动完成后，可看到插入数据和页面查询显示结果，通常运行在 5607 端口：



3 实现细节

3.1 处理反爬

3.1.1 更换随机 User-Agent

反爬措施：网站在处理反爬的过程中，很常见的一种方式就是通过检测 User-

agent 来拒绝非浏览器的访问。

作为应对，我们需要在每次发送请求时加上浏览器的 User-Agent 字段，可以在 Scrapy 的 middlewares.py 中自定义 RandomUserAgentMiddleware 类，并作为 Download Middleware 启用。Download Middleware 是引擎和下载器的中间件，每个 Request 在爬取之前都会调用其中开启的类，从而对 Request 进行一定的处理，在这里就是对每个请求加上随机的 User-Agent。

```
class RandomUserAgentMiddleware(object):
    # 随机更换user-agent
    def __init__(self, crawler):
        super(RandomUserAgentMiddleware, self).__init__()
        self.ua = UserAgent()
        self.ua_type = crawler.settings.get("RANDOM_UA_TYPE", "random")

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler)

    def process_request(self, request, spider):
        def get_ua():
            return getattr(self.ua, self.ua_type)

        request.headers.setdefault('User-Agent', get_ua())
```

关于如何获取到随机的 User-Agent，可以通过 fake_useragent 库获取，实际上这个库的作者维护了一个 User-Agent 的集合，使用方法如下，通过实例化 UserAgent 类，可以利用 random 随机取出一个 User-agent 以供使用：

```
>>> from fake_useragent import UserAgent
>>> ua = UserAgent()
>>> ua.random
'Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.17 Safari/537.36'
>>> ua.random
'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.36'
>>> ua.random
'Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1667.0 Safari/537.36'
>>> ua.random
'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36'
>>> ua.random
'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.36'
>>> ua.random
'Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.7.62 Version/11.01'
>>> ua.random
'Mozilla/5.0 (Windows NT 4.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2049.0 Safari/537.36'
```

3.1.2 使用 IP 代理池

反爬措施：如果同一个 IP 地址访问过于频繁时，就直接将访问的 IP 地址进行封锁，短期内进行访问。

对于这种反爬手段，可以维护一个 IP 代理池，当 IP 被封锁时进行 IP 的更换，或者通过代理隐藏自己的 IP。目前网上提供代理的网站服务有不少，通常是充值一定的费用，就可以使用它提供的代理 IP。由于本项目中对代理池的需求不大，因此仅是使用一个免费的代理网站进行演示，如下：

公告：本站所有代理IP地址均收集整理自国内公开互联网，本站不维护运营任何代理服务，请自行筛选。

国家	IP地址	端口	服务器地址	是否匿名	类型	速度	连接时间	存活时间	验证时间
	218.76.253.201	61408	湖南永州	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	621天	20-06-08 08:21
	120.192.75.82	808	山东	高匿	HTTP	<div><div></div></div>	<div><div></div></div>	1分钟	20-06-08 08:21
	58.56.149.198	53281	山东青岛	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	996天	20-06-08 08:21
	61.145.35.113	4216	广东江门	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	1分钟	20-06-08 08:00
	124.93.201.59	59618	辽宁大连	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	578天	20-06-08 08:00
	113.12.202.50	40498	广西南宁	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	563天	20-06-08 07:21
	114.99.54.65	8118	安徽安庆	高匿	HTTPS	<div><div></div></div>	<div><div></div></div>	264天	20-06-08 07:20
	114.98.24.221	4216	安徽芜湖	高匿	HTTP	<div><div></div></div>	<div><div></div></div>	13天	20-06-08 07:01

为了获取该网站上的代理 IP 地址，可以单独写一个小爬虫将该网站提供的 http 代理爬取到数据库 proxy_ip 中：

对象	proxy_ip @article_spider (MySQL5) -...				
开始事务	文本	筛选	排序	导入	导出
id	ip	port	speed	proxy_type	
1	120.192.75.82	808	0.157	HTTP	
2	114.98.24.221	4216	0.366	HTTP	
3	223.241.0.180	4216	3.167	HTTP	
4	223.241.6.69	4216	0.373	HTTP	
5	113.208.115.190	8118	0.166	HTTP	
6	61.145.48.163	9999	0.641	HTTP	
7	180.124.87.48	4216	0.148	HTTP	
8	61.145.48.149	9999	0.208	HTTP	
9	36.6.224.204	3000	0.459	HTTP	
10	112.14.47.6	52024	6.337	HTTP	

同样是以 Download Middleware 的方式，在 Request 爬取前，从数据库中随机取出一个代理，通过代理进行访问并爬取。

```
class RandomProxyMiddleware(object):
    # 动态设置ip代理
    def process_request(self, request, spider):
        get_ip = GetIP()
        request.meta["proxy"] = get_ip.get_random_ip()
```

3.1.3 访问频率限制

反爬措施：如果访问过快的话，会返回其他状态码/跳转等（302、429 等）来阻止你继续访问内容页面，如下在爬取嘶吼时候，若访问频率过快，则会出现如下页面：

429 | Too Many Requests

对于这一反爬的应对措施，主要有以下两种思路：

- 在 Scrapy 中可以直接设置每次请求之间的间隔来降低频率。
- 自定义 Middleware 在每次接收到的 Response 被跳转时，将爬虫暂停一段时间再继续爬取。

在本项目中，使用的第二种方法，实现代码如下，在遇到频率先之后将爬虫暂停 15 秒再继续，经测试，可以有效的处理嘶吼的该反爬措施：

```
class TooManyRequestsRetryMiddleware(RetryMiddleware):
    # 处理Too many requests,即暂停爬虫一段时间再继续
    def __init__(self, crawler):
        super(TooManyRequestsRetryMiddleware, self).__init__(crawler.settings)
        self.crawler = crawler

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler)

    def process_response(self, request, response, spider):
        if request.meta.get('dont_retry', False):
            return response
        elif response.status == 429:
            self.crawler.engine.pause()
            print("Too many requests, spider pause 15 seconds.")
            # If the rate limit is renewed in a minute, put 15 seconds, and so on.
            time.sleep(15)
            self.crawler.engine.unpause()
            reason = response_status_message(response.status)
            return self._retry(request, reason, spider) or response
        elif response.status in self.retry_http_codes:
            reason = response_status_message(response.status)
            return self._retry(request, reason, spider) or response
        return response
```

3.1.4 Cookie 的禁用

反爬措施：有些网站可以通过跟踪 Cookie 来识别是否是同一个客户端。如果同一个客户端在单位时间内的请求过于频繁，则判断为爬虫，对此客户端进行禁止。

这个处理反爬的方式比较简单，Scrapy 默认开启了 Cookie，而如果爬取过程中不需要登录的话，就可以在 setting.py 文件中将 COOKIES_ENABLED 设置为 False 即可。

```
)# Disable cookies (enabled by default)
COOKIES_ENABLED = False
```

3.1.5 验证码识别

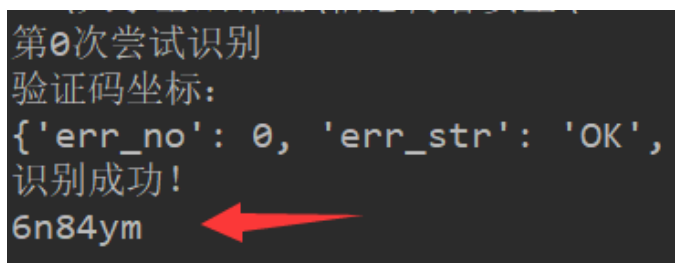
反爬措施：在进行登录等操作时，要求用户输入随机生成的验证码，验证码输入正确才能进行下一步的操作。

验证码可以说是当前最常见的一种反爬措施，几乎所有需要登录的地方都需要用户输入验证码来证明操作不是有机器人或者爬虫完成的。验证码的识别问题是一个较为复杂的问题，由于其并不是本项目中的研究重点，因此我们主要考虑

如何应用现有的验证码识别方式即可。当前比较常见的验证码识别方式有如下三种：

- 编码实现：有能力的情况下可以自己训练模型，通常可以使用谷歌公司提供的 tesseract-ocr 库，但是对于噪音较多的而验证码来说，识别率并不是很高。
- 在线打码：即使用某些在线平台提供的 api 接口，将爬虫获取的验证码图片上传至 api，该平台会在内部进行识别，并返回结果。

目前在线打码是使用最多的一种方式，使用起来快捷方便、可以识别多种验证码类型，且准确率较高，缺点就是通常来说需要收费，不过价格也不会太高。代码就是像平台 api 提供使用者的用户名、密码和验证码即可，下右图是对左图验证码的识别效果：



3.2 爬取数据

3.2.1 先知社区

对于先知社区的爬取，可以说是比较容易的，该网站逻辑架构简洁明了，首页中即有文章列表和文章分类 tab。通过进一步分析可知，每个分类的 tab 序号不同，通过/tab/1 的方式可获取不同种类的文章。



分页也是通过?page=2 参数进行分页，也比较的清楚：



因此我们的爬虫实现思路如下：

- (1) 遍历每一个 tab 页面。
- (2) 对页面中的每个文章信息进行爬取，包括文章链接。
- (3) 对步骤 2 中提取的每个文章页面再进行爬取，取得文章具体信息。
- (4) 当前页面爬取完后，判断是否存在“下一页”按钮来判断是否爬取完成。

在 Scrapy 中的 Spider 代码如下：

```
from scrapy.http import Request
from urllib import parse

from ..items import XianzhiArticleItem, ArticleItemLoader
from ..utils.common import get_md5
from scrapy_redis.spiders import RedisSpider

class XianzhiSpider(RedisSpider):
    name = 'xianzhi' # 启动的时候指定名称

    redis_key = 'xianzhi:start_urls'

    # allowed_domains = ['xz.aliyun.com']
    # start_urls = ['https://xz.aliyun.com/']

    def parse(self, response):
        tabs = [1, 4, 7, 9, 10, 13]
        for tab in tabs:
            tab_url = response.url + 'tab/{}'.format(tab)
            yield Request(url=tab_url, callback=self.tab_parse)

    # 爬取的每个 url 会进入这个函数，会返回 response
    def tab_parse(self, response):
```

```

# 解析列表页中的所有文章 url 并交给 scrapy 下载后并进行解析
post_nodes = response.css('table.table.topic-list tr')
for post_node in post_nodes:
    image_url = post_node.css('a.user-link
img::attr(src)').extract_first() # 图片地址
    post_url = post_node.css('a.topic-
title::attr(href)').extract_first() # 获取文章的地址
    yield Request(url=parse.urljoin(response.url, post_url),
meta={"front_image_url": image_url},
        callback=self.parse_detail)

# 提取下一页并交给 scrapy 下载
next_url = response.css('div.pagination.clearfix li:nth-child(3)
a::attr(href)').extract_first("")
if next_url and next_url.startswith('?'):
    yield Request(url=parse.urljoin(response.url, next_url),
callback=self.tab_parse)

def parse_detail(self, response):

    # 通过 item_loader 加载 item
    item_loader = ArticleItemLoader(item=XianzhiArticleItem(),
response=response)
    item_loader.add_value('front_image_url',
[response.meta.get('front_image_url', '')])
    item_loader.add_css('title', '.content-title::text') # 添加 css 选择
器

    item_loader.add_value('url', response.url)
    item_loader.add_value('url_object_id', get_md5(response.url))
    item_loader.add_css('create_date', '.info-left span:nth-
child(3)::text')
    item_loader.add_css('view_count', '.info-left span:nth-
child(5)::text')
    item_loader.add_css('author', '.info-left span:nth-child(1)::text')
    item_loader.add_css('follow_count', '#follow-count::text')
    item_loader.add_css('mark_count', '#mark-count::text')
    item_loader.add_css('content', '#topic_content')
    item_loader.add_css('tags', '.content-node a::text')
    article_item = item_loader.load_item()
    yield article_item # 传递到 pipelines

```


3.2.2 安全客

安全客的首页看起来比先知社区要复杂一些，并且是鼠标点击“加载更多”来分页的，爬取难度看似更大。但是通过分析翻页时的请求包，发现安全客也是通过请求 api 来获取文章的信息的，api 链接如下：

<https://api.anquanke.com/data/v1/posts?size=20&page=2>

我们通过观察此 api 返回的内容，不难推测到：url 中参数 size 指的是一页请求的文章数量，参数 page 则是表示当前页数。

```

x Headers Preview Response Initiator Timing Cookies
▼ {, ...}
  data: [{id: 206483, title: "RATicate攻击活动分析", category_name: "安全知识", category_slug: "knowledge", ...}, ...]
    0: {id: 206483, title: "RATicate攻击活动分析", category_name: "安全知识", category_slug: "knowledge", ...}
    1: {id: 206460, title: "Tide-Mars: 一款资产管理与威胁监测平台", category_name: "安全工具", category_slug: "tool", ...}
    2: {id: 206525, title: "啤酒评级应用Untappd竟然可以用来追踪军事人员的个人敏感信息", category_name: "安全资讯", category_slug: "news", ...}
    3: {id: 206442, title: "CVE-2020-3153: Cisco AnyConnect Installer本地提权漏洞分析及利用", category_name: "安全知识", ...}
    4: {id: 206517, title: "5月25日每日安全热点 - Darkhotel组织渗透隔离网络的Ramsay组件分析", category_name: "安全资讯", ...}
    5: {id: 206500, title: "5月24日每日安全热点 - ZOOM 端端加密介绍白皮书", category_name: "安全资讯", category_slug: "news", ...}
    6: {id: 206496, title: "5月23日每日安全热点 - 美国贸易管制黑名单新增33家", category_name: "安全资讯", category_slug: "news", ...}
    7: {id: 206487, title: "被美国列入实体清单，我们感到意外", category_name: "安全资讯", category_slug: "news", ...}
    8: {id: 206477, title: "美国贸易管制黑名单新增：奇虎360、烽火通信、云从科技、东方网力等33家", category_name: "安全资讯", ...}
    9: {id: 206471, title: "360网络安全周报第249期", category_name: "360网络安全周报", category_slug: "week", ...}
    10: {id: 206433, title: "招聘 | 看介里！爱奇艺安全有我等着你！", category_name: "招聘", category_slug: "job", ...}
    11: {id: 205987, title: "一起来学PHP代码审计（一）入门", category_name: "安全知识", category_slug: "knowledge", ...}
    12: {id: 206432, title: "2019年开源软件风险研究报告", category_name: "安全知识", category_slug: "knowledge", ...}
    13: {id: 206288, title: "Windows 10 19041版本的Infinity hook 原理", category_name: "安全知识", ...}
    14: {id: 205679, title: "2020网鼎杯朱雀组部分Web题wp", category_name: "安全知识", category_slug: "knowledge", ...}
    15: {id: 206001, title: "Java反序列化从入门到入土", category_name: "未分类", category_slug: "uncategorized", desc: "", ...}
    16: {id: 205978, title: "泛主机场景搭建安全体系", category_name: "安全知识", category_slug: "knowledge", ...}
    17: {id: 202639, title: "运用命令行、注册表编程、windowsAPI修改本地安全策略", category_name: "安全知识", ...}
    18: {id: 206295, title: "5月22日每日安全热点 - 美国银行：COVID-19贷款数据可能泄露", category_name: "安全资讯", ...}
    19: {id: 205982, title: "360发布《2020年Q1手机安全状况报告》，带你快速了解疫情下的网络安全趋势", category_name: "安全知识", ...}
  next: "https://api.anquanke.com/data/v1/posts?page=3&size=20"
  success: true

```

我们继续分析该 api 返回的 json 内容：

```

▼ 0: {id: 208671, title: "克隆版笔记共享网站窃取用户比特币；瑞幸董事会成员邮件曾被黑客攻破", category_name: "安全资讯", category_slug: "news", ...}
  author: {nickname: "安全客", user_url: "http://www.anquanke.com", id: 2, ...}
  category_name: "安全资讯"
  category_slug: "news"
  comment: 0
  cover: "https://p0.ssl.qhimg.com/t016da32a9fec0774a2.jpg"
  date: "2020-06-18 17:15:56"
  desc: "克隆版笔记共享网站Privnote.com窃取用户比特币；滴滴正式起诉“性侵犯视频”表演者及涉黄直播平台，网络色情灰色产业链曝光；瑞幸自曝造假前，董事会成员..."
  favorite_count: 1
  fee: ""
  id: 208671
  is_favorite: false
  like_count: 0
  liked: false
  origin_author: ""
  pv: 68103
  red: false
  source: ""
  status: "publish"
  subject: false
  tags: ["资讯充电站"]
  time_interval: "2天前"
  title: "克隆版笔记共享网站窃取用户比特币；瑞幸董事会成员邮件曾被黑客攻破"
  type: "origin"
  url: ""

```

除了没有直接给出文章的 url 以及文章内容之外，我们爬虫所需要的信息都可以直接获取，只需要简单的解析返回的 json 即可。

- 虽然没有给出文章 url 但是给出了 id，我们容易得到固定的文章 url 格式为：/post/id/206483，即我们可以通过 id 来得到文章 url。
- 而文章的内容，我们则需要先用 id 构造文章 url，然后异步发送请求来获取

content。

实现代码如下：

```
import json
from scrapy.http import Request
from ..items import AnquankeArticleItem, ArticleItemLoader
from ..utils.common import get_md5
from scrapy_redis.spiders import RedisSpider

class AnquankeSpider(RedisSpider):
    name = 'anquanke'
    redis_key = 'anquanke:start_urls'

    # allowed_domains = ['api.anquanke.com', 'anquanke.com']
    # start_urls = ['https://api.anquanke.com/data/v1/posts?size=500']

    def parse(self, response):
        res = json.loads(response.text)
        posts = res.get('data')
        for post in posts:
            # 剔除掉活动和招聘的文章
            if ('招聘' in post.get('tags')) or ('活动' in post.get('tags')):
                continue

            item_loader = ArticleItemLoader(item=AnquankeArticleItem(),
response=response)
            item_loader.add_value('title', post.get('title'))
            item_loader.add_value('create_date', post.get('date'))
            item_loader.add_value('front_image_url', post.get('cover'))
            item_loader.add_value('author',
post.get('author').get('nickname'))
            item_loader.add_value('view_count', post.get('pv'))
            item_loader.add_value('comment_count', post.get('comment'))
            item_loader.add_value('like_count', post.get('like_count'))
            item_loader.add_value('tags', post.get('tags'))
            yield
Request(url='https://www.anquanke.com/post/id/{}'.format(post.get('id')),
        meta={'article_item': item_loader},
callback=self.parse_content)
            # 获取下一页
            next_url = res.get('next')
            if next_url:
```

```

        yield Request(url=next_url, callback=self.parse)

    def parse_content(self, response):
        item_loader = response.meta.get('article_item', '')
        content = response.css('div.article-
content').extract_first().replace('data-original=', 'src=')
        item_loader.add_value('content', content)
        item_loader.add_value('url', response.url)
        item_loader.add_value('url_object_id', get_md5(response.url))

        article_item = item_loader.load_item()
        yield article_item

```

3.2.3 嘶吼

嘶吼的分页也是通过点击“加载更多”，但是分析后可知其分页方式实际上和先知的类似，即通过?page=2 进行分页。



但是这样有个问题是如何判断最后一页，在项目中我尝试了以下两种方法，最终使用了第二个方法：

- 首先考虑设置一个变量 page，每次翻页时加一，当页面的文章列表为空的时候判断为最后一页，但是 Scrapy 是异步框架，在快速爬取的过程中 page 的使用会混乱。
 - 于是考虑设定一个最大的 page 数，这里页数最多的是 web 安全分类(126 页)，所以我们就默认爬取的每个分类页数都设置为大于 126 的数（或者更大一些），然后再判断是否存在文章列表，存在就继续爬取，否则就转到下一页。
- 另外就是嘶吼的文章分类位于首页的“阅读”下拉栏中，但是实际上也已经在 html 中可以找到：



```
.....  
▼<div class="list_inav clearfix">  
  ▼<span>  
    <a href="https://www.4hou.com/category/web">web安全 </a>  
  </span>  
  ▼<span>  
    <a href="https://www.4hou.com/category/business">业务安全 </a>  
  </span> == $0  
  ▼<span>  
    <a href="https://www.4hou.com/category/binary">二进制安全</a>  
  </span>
```

实现代码如下：

```
from scrapy.http import Request  
from ..items import ArticleItemLoader, SihouArticleItem  
from ..utils.common import get_md5  
from scrapy_redis.spiders import RedisSpider  
  
class A4houSpider(RedisSpider):  
    name = 'sihou'  
    redis_key = 'sihou:start_urls'  
  
    # allowed_domains = ['4hou.com']  
    # start_urls = ['https://www.4hou.com']  
  
    # 爬取的每个 url 会进入这个函数，会返回 response  
    def parse(self, response):  
        # types = response.css('div.technology a::attr(href)').extract()  
        for type_url in response.css('div.technology  
a::attr(href)').extract(): # 获取各个分类标签  
            for i in range(1, 150):  
                page = "?page={}".format(i)  
                yield Request(url=type_url + page, callback=self.parse_post)  
  
    def parse_post(self, response):  
        # 解析列表页中的所有文章 url 并交给 scrapy 下载后并进行解析  
        if response.css('div.main-box') != []:  
            post_nodes = response.css('div.main-box')  
            for post_node in post_nodes:  
                image_url = post_node.css('div.new_img  
img::attr(src)').extract_first() # 图片地址  
                view_count = post_node.css('div.read  
span::text').extract_first().replace(',', '') # 浏览量  
                praise_count = post_node.css('div.Praise  
span::text').extract_first().replace(',', '') # 点赞数
```

```

        post_url = post_node.css('div.new_con
a::attr(href)').extract_first()
        yield Request(url=post_url,
                        meta={'front_image_url': image_url,
'view_count': view_count,
                        'praise_count': praise_count},
                        callback=self.parse_detail)

    def parse_detail(self, response):
        # 通过 item_loader 加载 item

        item_loader = ArticleItemLoader(item=SihouArticleItem(),
response=response)
        item_loader.add_value('front_image_url',
[response.meta.get('front_image_url', '')])
        item_loader.add_css('title', 'h1.art_title::text') # 添加 css 选择器
        item_loader.add_value('url', response.url)
        item_loader.add_value('url_object_id', get_md5(response.url))
        item_loader.add_css('create_date', 'div.art_time span:nth-
child(3)::text')
        item_loader.add_value('view_count', response.meta.get('view_count',
'0'))
        item_loader.add_css('author', 'span.sir::text')
        item_loader.add_value('praise_count',
response.meta.get('praise_count', '0'))
        item_loader.add_css('content', 'div.article_cen')
        item_loader.add_css('tags', 'span.lei::text')

        article_item = item_loader.load_item()
        yield article_item # 传递到 pipelines

```

3.3 重构分布式爬虫

3.3.1 需要解决的问题

原来的 Scrapy 是不支持分布式的, 我们知道 Scrapy 在进行爬虫爬取网站时, 会在内存中维护一个 Request 队列, 首先将 start_urls 放入该队列中, 然后在爬取的过程中, 不断的将 URL 插入该队列或从队列中取出。除此之外, Scrapy 还需要维护一个去重队列, 用来对爬取的 URL 进行去重操作。

所以, 若要将我们的 Scrapy 项目重构为分布式的项目, 主要需要解决的问题就是下面两个:

- Request 队列集中管理: scheduler 以队列形式存储在内存中, 而其他服务

器无法拿到当前服务器内存中的内容。

- scrapy 的去重队列也是放在内存中，如何对去重队列进行集中管理。

3.3.2 分布式的原理

实际上使用 Scrapy 进行分布式爬虫的编写并不是特别复杂，其原理及思路主要如下：

(1) 把自己的核心服务器称为 master，而把用于跑爬虫程序的机器称 slave。

(2) 采用 scrapy 框架抓取网页，我们需要首先给定它一些 start_urls，爬虫首先访问 start_urls 里面的 url，再根据我们的具体逻辑，对里面的元素、或者是其他的二级、三级页面进行抓取。

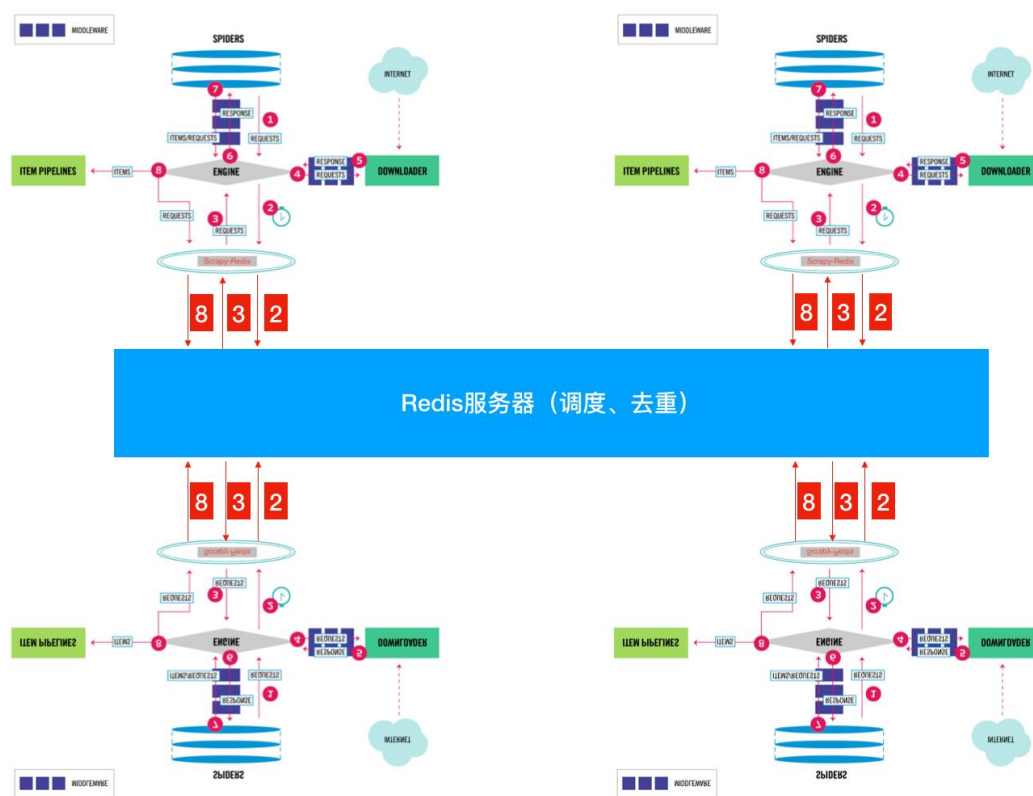
而要实现分布式，我们只需要在这个 start_urls 里面做文章就行了。

(3) 我们在 master 上搭建一个 redis 数据库（只用作 url 的存储，不关心爬取的具体数据），并对每一个需要爬取的网站类型，都开辟一个单独的列表字段。

(4) 这样尽管有多个 slave，我们都可以设置其 scrapy-redis 从 master 的 redis 服务器中获取 url。实际上除了 url，scrapy-redis 还会在 redis 服务器上存储去重队列，以及一个 items 数据库。

(5) 程序移植性强，把 slave 上的程序移植到另一台机器上运行非常的简单。

Scrapy-Redis 分布式爬虫的大致原理图如下：



3.3.3 分布式的实现

针对于本项目中，Scrapy-Redis 分布式爬虫的实现思路如下：

(1) 使用两台机器，一台是 win10(本机)，一台是(ubuntu)，分别在两台机器上部署 scrapy 来进行分布式抓取一个网站

(2) Win10 作为 Master，开启 redis 服务，同时也作为一个 slave 进行抓取；Ubuntu 作为一个 slave 来进行抓取，两台机器的爬虫均从 redis 服务端中获取 url。

(3) master 的爬虫运行时会把提取到的 url 封装成 request 放到 redis 中的数据库，并且从该数据库中不断的提取 request 后下载网页，再把网页的内容存放到 redis 的另一个 items 数据库中。

(4) slave 从 master 的 redis 中取出待抓取的 request，下载完网页之后就把网页的内容发送回 master 的 redis。

(5) 重复上面的 3 和 4，直到 master 的 redis 中的 request 数据库为空，再把 master 的 redis 中的 items 数据库写入到 Mysql 数据库中。

(6) Master 里的 reids 还有一个 dupefilter 数据库，用来存储抓取过的 url 的指纹（使用哈希函数将 url 运算后的结果），用来去重的。

具体代码实现，首先要重构的是 Scrapy 的 spider，即需要将原来爬虫类继承 Scrapy-redis 的 RedisSpider 类：

```
from scrapy_redis.spiders import RedisSpider

class MySpider(RedisSpider):
    name = 'myspider'

    def parse(self, response):
        # do stuff

        pass
```

除此之外，还需要再 setting.py 中进行一些分布式的设置，例如：

```
# Enables scheduling storing requests queue in redis.
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# Ensure all spiders share same duplicates filter through redis.
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# Store scraped item in redis for post-processing.
ITEM_PIPELINES = {
    'scrapy_redis.pipelines.RedisPipeline': 300
```



```
}
```

本项目的具体配置，可见源代码 `setting.py`。

3.4 搜索引擎

对于搜索引擎的使用，首先要明确以下两个问题：

(1) 我们对于搜索引擎需求：

- 高效
- 零配置、完全免费
- 能够简单通过 json 和 http 与搜索引擎交互
- 搜索服务器稳定
- 能够简单的将一台服务器扩展到上百

(2) 为什么不能用传统的关系数据库完成搜索功能

- 无法打分
- 无分布式
- 无法解析请求搜索
- 效率低
- 需要单独实现分词

3.4.1 倒排索引

倒排索引 (Inverted index)，也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。

以英文为例子，加上要索引的文本如下：

```
T_0="It is what it is"
T_1="what is it"
T_2="it is a banana"
```

若使用传统的正向索引，即索引如下：

```
Docs={
    0 => "It is what it is"
    1 => "what is it"
    2 => "it is a banana"
}
```

而使用倒排索引，索引如下：

```
"a": {2}
```


"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

另外，倒排索引待解决问题如下：

- 大小写转换问题，如 python 和 PYTHON 应该为一个词
- 词干抽取，looking 和 look 应该处理为一个词
- 分词
- 倒排索引文件过大，压缩编码

3.4.2 排序评分

使用 Elasticsearch 时，对于查询出的文档无疑会有文档相似度之别。而理想的排序是和查询条件相关性越高排序越靠前，而这个排序的依据就是 `_score`。Elasticsearch 使用布尔模型查找匹配文档，并用一个名为实用评分函数的公式来计算相关度。这个公式借鉴了词频/逆向文档频率(TF-IDF) 和 向量空间模型，同时也加入了一些现代的新特性，如协调因子、字段长度归一化以及词或查询语句权重提升，公式如下：

$$\text{score}(q,d) = \text{queryNorm}(q) \cdot \text{coord}(q,d) \cdot \sum (\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d)) (t \text{ in } q)$$

① `score(q,d)` 是文档 `d` 与查询 `q` 的相关度评分。

② `queryNorm(q)` 是 [查询归一化因子](#) (新)。

③ `coord(q,d)` 是 [协调因子](#) (新)。

④ 查询 `q` 中每个词 `t` 对于文档 `d` 的权重和。

⑤ `tf(t in d)` 是词 `t` 在文档 `d` 中的 [词频](#)。

⑥ `idf(t)` 是词 `t` 的 [逆向文档频率](#)。

⑦ `t.getBoost()` 是查询中使用的 [boost](#) (新)。

⑧ `norm(t,d)` 是 [字段长度归一值](#)，与 [索引时字段层 boost](#) (如果存在) 的和 (新)。

对上述公式中的部分概念进行如下解释：

(1) 词频 (Term frequency)

语在文档中出现的频度是多少，频度越高，权重越大。一个 5 次提到同一词语的字段比一个只有 1 次提到的更相关。词频的计算方式如下：

$$\text{tf}(t \text{ in } d) = \sqrt{\text{frequency}}$$

词语 `t` 在文件 `d` 的词频 (`tf`) 是这个词语在文档中出现次数的平方根。

(2) 逆向文档频率 (Inverse document frequency)

词语在集合所有文档里出现的频次。频次越高，权重越低。常用词如 `and` 或 `the` 对于相关度贡献非常低，因为他们在多数文档中都会出现，一些不常见词语如 `elastic` 或 `lucene` 可以帮助我们快速缩小范围找到感兴趣的文档。逆向文档

频率的计算公式如下：

$$\text{idf}(t) = 1 + \log(\text{numDocs} / (\text{docFreq} + 1))$$

词语 t 的逆向文档频率 (Inverse document frequency) 是：索引中文档数量除以所有包含该词语文档数量后的对数值。

(3) 字段长度正则值 (Field-length norm)

字段的长度是多少，字段越短，字段的权重越高。如果词语出现在类似标题 title 这样的字段，要比它出现在内容 body 这样的字段中的相关度更高。字段长度的正则值公式如下：

$$\text{norm}(d) = 1 / \sqrt{\text{numTerms}}$$

字段长度正则值是字段中词语数平方根的倒数。

(4) 查询正则因子 (Query Normalization Factor)

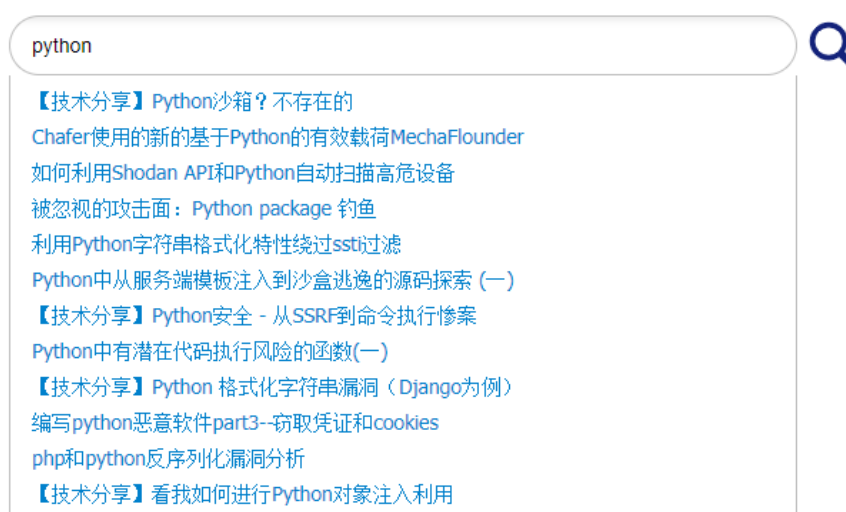
查询正则因子 (queryNorm) 试图将查询正则化，这样就能比较两个不同查询结果。尽管查询正则值的目的是为了使得查询结果之间能够相互比较，但是它并不十分有效，因为相关度分数 $_score$ 的目的是为了将当前查询的结果进行排序，比较不同查询结果的相关度分数没有太大意义。

(5) 查询协调 (Query Coordination)

协调因子 (coord) 可以为那些查询词语包含度高的文档提供“奖励”，文档里出现的查询词语越多，它越有机会成为一个好的匹配结果。

3.4.3 搜索提示

在本项目中为了更加模拟成熟搜索引擎的模式，并且方便用户的查询，加入了搜索提示的功能，所要达到的效果如下，即当用户输入搜索关键词后，搜索框下方会实时显示根据此关键词所相关联的结果。



要实现此功能，需要利用到 Elasticsearch 中的 suggest 用法：

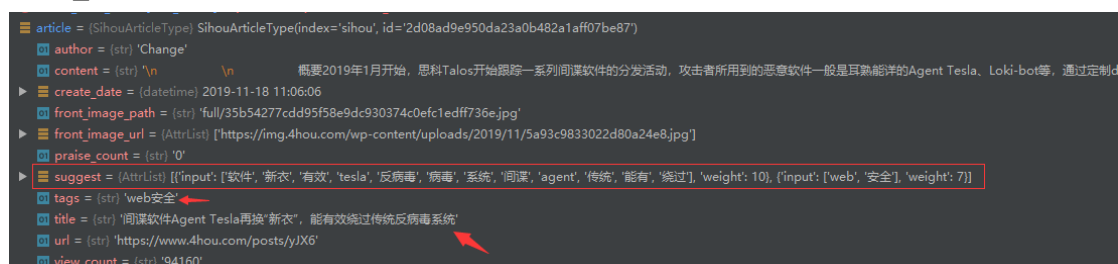
- 首先需要在每个 mapping 中增加一个 suggest 字段类型为 Completion。
- 在 items.py 中为每个类增加 suggest 字段，创建一个函数 (gen_suggests) 来生成 suggest 的数组 (可以使用 _analyze 接口自动分词获得相应的关键词数组)，并且可以对搜索字段设置权重 weight。

gen_suggests 函数的实现代码如下：

```
def gen_suggests(index, info_tuple, es):
    # 根据字符串生成搜索建议数组
    used_words = set() # 去重
    suggests = []
    for text, weight in info_tuple:
        if text:
            # 调用es的analyze接口分析字符串,进行分词
            words = es.indices.analyze(index=index, analyzer='ik_max_word', params={'filter': ['lowercase']}, body=text)
            analyzed_words = set([r['token'] for r in words['tokens'] if len(r['token']) > 1]) # 过滤掉只有一个词的
            new_words = analyzed_words - used_words # 已经存在的词过滤掉
        else:
            new_words = set()

        if new_words:
            suggests.append({'input': list(new_words), 'weight': weight})
    return suggests
```

使用 _analyze 接口自动分词调试结果如下：



```
article = (SihouArticleType) SihouArticleType(index='sihou', id='2d08ad9e950da23a0b482a1aff07be87')
author = (str) 'Change'
content = (str) '\n\n    概要2019年1月开始，思科Talos开始跟踪一系列间谍软件的分发活动，攻击者所使用的恶意软件一般是耳熟能详的Agent Tesla、Loki-bot等，通过定制d
create_date = (datetime) 2019-11-18 11:06:06
front_image_path = (str) 'full/35b54277cdd95f58e9dc930374c0efc1edff736e.jpg'
front_image_url = (AttrList) ['https://img.4hou.com/wp-content/uploads/2019/11/5a93c9833022d80a24e8.jpg']
praise_count = (str) '0'
suggest = (AttrList) [{'input': ['软件', '新衣', '有效', 'tesla', '反病毒', '病毒', '系统', '间谍', 'agent', '传统', '能看', '绕过'], 'weight': 10}, {'input': ['web', '安全'], 'weight': 7}]
tags = (str) 'web安全'
title = (str) '间谍软件Agent Tesla再换“新衣”，能有效绕过传统反病毒系统’
url = (str) 'https://www.4hou.com/posts/yjX6'
view_count = (str) '94160'
```

3. 4. 4 模糊搜索

模糊搜索即根据用户输入的搜索关键词，返回与其最相关的搜索内容。模糊搜索的实现主要是根据编辑距离来判断。

编辑距离是一种字符串之间相似程度的计算方法，即两个字符串之间的编辑距离等于使一个字符串变成另一个字符串而需要进行插入、删除、替换、相邻字符串交换位置四种操作的最少次数。例如，Linux 与 linux 的编辑距离是 1，它们之间的变换只需要进行 1 步操作。关于编辑距离的求法，普遍采用的是动态规划。

在 Elasticsearch 中需要设置：

- Fuzziness: 表示最小编辑距离，小于最小编辑距离的可以模糊搜索到，可设置为 AUTO。
- prefix_length: 前边不参与模糊查询的词长度为多少。例如在查询 football 时，若该参数设为 3，则 foatball 将不能匹配到。

3.5 网页搭建

由于 Web 开发的知识，并不是本课程设计的主要考察点，且报告长度有限，因此具体的网站开发流程将不在报告中体现。本节内容将对网站开发中所涉及到的爬虫、分布式及搜索服务设计的问题进行阐述。

3.5.1 爬虫统计数据

在网站中，我们预计在侧边加上分布式爬虫对各个网站所爬取的文章数量，效果如下：

网站

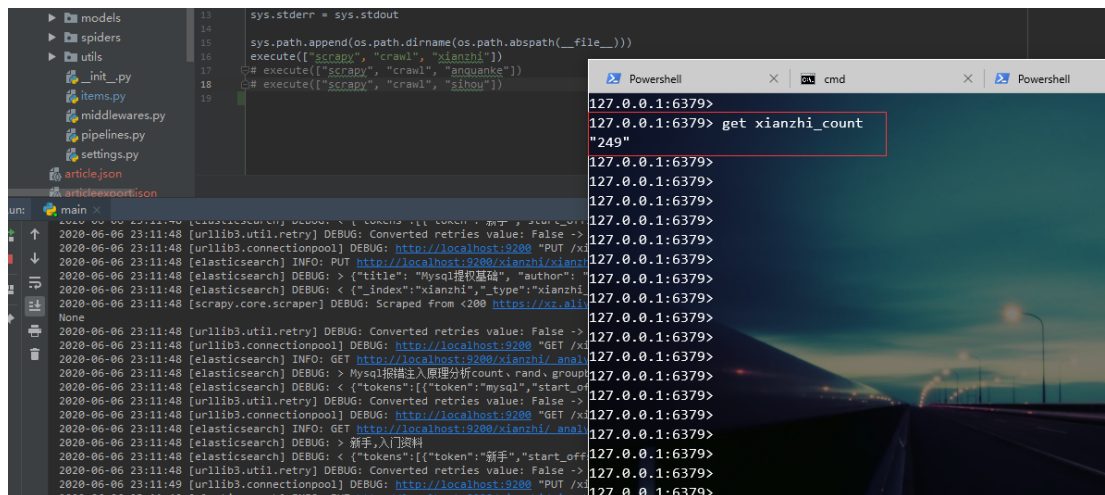
先知社区 (4960)

安全客 (10088)

嘶吼 (1865)

传统的思路是每次显示时，进行一次数据数量的查询，但是这样显然会降低网站的加载速率。本项目中采取一种更有效率的方式，即利用 Redis 实现。具体来说：每个爬虫在 redis 中维护一个全局变量，每次存储一个数据时就在 redis 中对一个全局变量加一。最后显示的时候，从 redis 中读取该变量即可获得统计数据，这样就避免了每次都去检索。

演示效果如下：



3.5.2 热门搜索

本项目的文章搜索网站中另一个比较重要的功能即为“热门搜索”，即将用户搜索次数最多的关键词列举出来。这实际上是一个 Top n 问题，利用 Redis 也是可以非常简单高效的实现。

具体实现方法为在 Redis 中创建一个可排序列表，每次当用户查询一个关键

词时，就将其加到列表中，同时加上相应的分数，最后取得时候按分数排序即可取出热门搜索。

另外，还有“我的搜索”功能，此功能实现较为简单，直接利用前端 Javascript 即可实现。

效果图如下：



3.6 其他技术

3.6.1 URL 去重策略

在分布式爬虫系统中面临的各种复杂的挑战中，除了分布式任务的调度、分布式系统中负载均衡等，分布式系统中 URL 去重更是非常重要的一点。分布式系统中 URL 去重算法影响着系统的效率，面对上百亿的 URL 设计一个优秀的去重算法能提高整个系统的性能。

目前的爬虫系统对于 URL 的去重策略主要有以下几种：

- 将访问过的 URL 保存到数据库中；
- 将访问过的 URL 保存到 set 中，只需要 $O(1)$ 的代价就可以查询 URL；
- URL 经过 md5 等方法哈希后保存到 set 中 (scrapy-redis 默认)
- 用 bitmap 方法将访问过的 URL 通过 hash 函数映射到某一位。
- bloomfilter 方法对 bitmap 进行改进，多重 hash 函数降低冲突。

Scrapy 框架使用的默认去重策略是上述的第三种方式，即使用 sha1 算法，对每一个 request 对象加密，生成 40 为十六进制数，并且 scrapy 的去重默认会保存到内存中，如果任务重启，会导致内存中所有去重队列消失。

而我们在将项目重构为分布式爬虫系统后，Scrapy-Redis 重写了 Scrapy 的

调度器和去重队列，所以需要在 settings 中修改如下两列：

```
# Enables scheduling storing requests queue in redis.
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
# Ensure all spiders share same duplicates filter through redis.
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
```

我们可以稍微分析一下 Scrapy-Redis 的去重重要代码：

```
def request_seen(self, request):
    """Returns True if request was already seen.
    Parameters
    -----
    request : scrapy.http.Request
    Returns
    -----
    bool
    """
    fp = self.request_fingerprint(request)
    # This returns the number of values added, zero if already exists.
    added = self.server.sadd(self.key, fp)
    return added == 0

def request_fingerprint(self, request):
    """Returns a fingerprint for a given request.
    Parameters
    -----
    request : scrapy.http.Request
    Returns
    -----
    str
    """
    return request_fingerprint(request)
```

首先拿到 scrapy.http.Request 会先调用 self.request_fingerprint 去计算，也就是 scrapy 的 sha1 算法去加密，然后会向 redis 中添加该指纹。该函数的作用是：计算该请求指纹，添加到 redis 的去重队列，如果已经存在该指纹，返回 True。

我们可以看到，只要有在 settings 中添加 DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"，就会在 redis 中新加一列去重队列，这样左右的优缺点如下：

- 优点：将内存中的去重队列序列化到 redis 中，及时爬虫重启或者关闭，也可以再次使用，你可以使用 SCHEDULER_PERSIST 来调整缓存

- 缺点：如果你需要去重的指纹过大，redis 占用空间过大。8GB=8589934592Bytes,平均一个去重指纹 40Bytes,约可以存储 214,748,000 个(2 亿)。所以在做关系网络爬虫中，序列化到 redis 中可能并不是很好，保存在内存中也不好，所以就产生了布隆过滤器。

3.6.2 Bloom Filter 使用

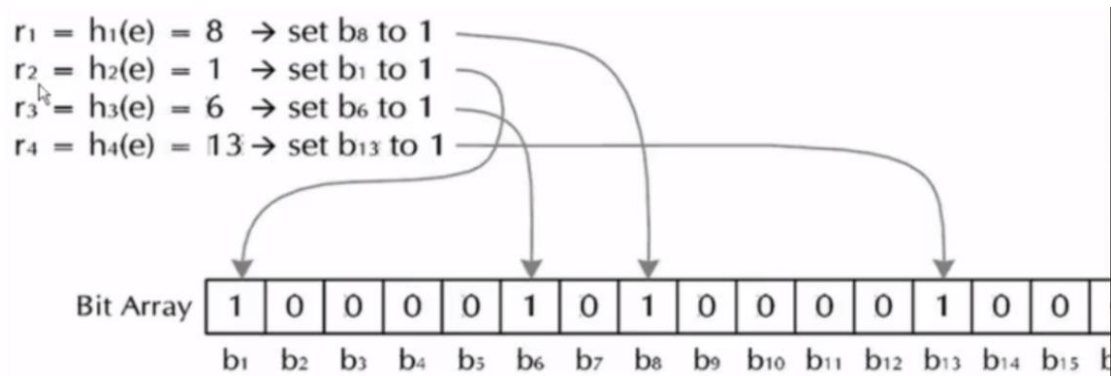
上一小节的最后提到了布隆过滤器 (Bloom Filter)，这是一种目前来看效率较高、对内存消耗也较少的 URL 去重策略。它的算法原创建一个 m 位的 BitSet，先将所有位初始化为 0，然后选择 k 个不同的哈希函数。第 i 个哈希函数对字符串 str 哈希的结果记为 $h(i, str)$ ，且 $h(i, str)$ 的范围是 $0 \sim m-1$ 。

Bloom Filter 的操作主要如下两种：

(1) Bloom Filter 加入字符串

对于字符串 str ，分别计算 $h(1, str)$ ， $h(2, str) \cdots (k, str)$ 。然后将 BitSet 的第 $h(1, str)$ 、 $h(2, str) \cdots (k, str)$ 位设为 1，这样字符串 str 映射到了 BitSet 中的 k 个二进制位了。

原理图如下：



(2) 检测字符串是否存在

对于字符串 str ，分别计算 $h(1, str)$ ， $h(2, str) \cdots (k, str)$ 。然后检查 BitSet 的第 $h(1, str)$ 、 $h(2, str) \cdots h(k, str)$ 位是否为 1，若其中任何一位不为 1 则可以判定 str 一定没有被记录过。若全部位都是 1，则“认为”字符串 str 存在。

若一个字符串对应的 Bit 不全为 1，则可以肯定该字符串一定没有被 Bloom Filter 记录过，因为字符串被记录过，其对应的二进制位肯定全部被设为 1 了。

但是若一个字符串对应的 Bit 全为 1，实际上是不能 100% 的肯定该字符串被 Bloom Filter 记录过的，因为有可能该字符串的所有位都刚好是被其他字符串所对应。这种将该字符串划分错的情况，称为 False Positive。

Bloom Filter 的具体实现代码如下：

```
import mmh3
```

```

import redis
import math
import time

class PyBloomFilter():
    # 内置 100 个随机种子
    SEEDS = [543, 460, 171, 876, 796, 607, 650, 81, 837, 545, 591, 946, 846,
             521, 913, 636, 878, 735, 414, 372,
             344, 324, 223, 180, 327, 891, 798, 933, 493, 293, 836, 10, 6,
             544, 924, 849, 438, 41, 862, 648, 338,
             465, 562, 693, 979, 52, 763, 103, 387, 374, 349, 94, 384, 680,
             574, 480, 307, 580, 71, 535, 300, 53,
             481, 519, 644, 219, 686, 236, 424, 326, 244, 212, 909, 202,
             951, 56, 812, 901, 926, 250, 507, 739, 371,
             63, 584, 154, 7, 284, 617, 332, 472, 140, 605, 262, 355, 526,
             647, 923, 199, 518]

    # capacity 是预先估计要去重的数量
    # error_rate 表示错误率
    # conn 表示 redis 的连接客户端
    # key 表示在 redis 中的键的名字前缀
    def __init__(self, capacity=1000000000, error_rate=0.00000001,
                 conn=None, key='BloomFilter'):
        self.m = math.ceil(capacity * math.log2(math.e) * math.log2(1 /
error_rate)) # 需要的总 bit 位数
        self.k = math.ceil(math.log1p(2) * self.m / capacity) # 需要最少的
hash 次数
        self.mem = math.ceil(self.m / 8 / 1024 / 1024) # 需要的多少 M 内存
        self.blocknum = math.ceil(self.mem / 512) # 需要多少个 512M 的内存
块,value 的第一个字符必须是 ascii 码,所有最多有 256 个内存块
        self.seeds = self.SEEDS[0:self.k]
        self.key = key
        self.N = 2 ** 31 - 1
        self.redis = conn

    def add(self, value):
        name = self.key + "_" + str(ord(value[0]) % self.blocknum)
        hashes = self.get_hashes(value)
        for hash in hashes:
            self.redis.setbit(name, hash, 1)

    def is_exist(self, value):

```



```

        name = self.key + "_" + str(ord(value[0]) % self.blocknum)
        hashes = self.get_hashes(value)
        exist = True
        for hash in hashes:
            exist = exist & self.redis.getbit(name, hash)
        return exist

    def get_hashes(self, value):
        hashes = list()
        for seed in self.seeds:
            hash = mmh3.hash(value, seed)
            if hash >= 0:
                hashes.append(hash)
            else:
                hashes.append(self.N - hash)
        return hashes

pool = redis.ConnectionPool(host='127.0.0.1', port=6379, db=0,
password='yuan123')
conn = redis.StrictRedis(connection_pool=pool)

if __name__ == '__main__':
    start = time.time()
    bf = PyBloomFilter(conn=conn)
    bf.add('www.baidu.com')
    bf.add('www.google.com')
    print(bf.is_exist('www.zhihu.com'))
    print(bf.is_exist('www.baidu.com'))
    end = time.time()
    print(end - start)

```

上述测试代码运行结果如下，结果中 0 代表未重复，1 代表已重复，从时间来看效率还是比较高的：

```

F:\大学基础课程\信息内容安全\code\venv\Scripts\python.exe F:/
0
1
0.022443771362304688

```

4 系统展示

4.1 分布式爬取

本项目的分布式演示用到了本机（Windows 10）同时作为 master 和 slave，另有一台 Vmware 虚拟机（Ubuntu）作为 slave。

首先分别在两个系统上运行 main.py 文件，使爬虫运行，但是由于此使 Redis 中并没有创建 start_urls，所以两个系统下的 slave 爬虫均处于等待状态：

① Windows 10:

```
2020-06-22 01:56:33 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2020-06-22 01:56:34 [scrapy.middleware] INFO: Enabled item pipelines:
['ArticleSpider.pipelines.ArticleImagePipeline',
 'ArticleSpider.pipelines.MysqlTwistedPipeline',
 'ArticleSpider.pipelines.ElasticsearchPipeline',
 'scrapy_redis.pipelines.RedisPipeline']
2020-06-22 01:56:34 [scrapy.core.engine] INFO: Spider opened
2020-06-22 01:56:34 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items
(at 0 items/min)
2020-06-22 01:56:34 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6023
```

② Ubuntu:

```
'DUPEFILTER_CLASS': 'scrapy_redis.dupefilter.RFPDupeFilter',
'NEWSPIDER_MODULE': 'ArticleSpider.spiders',
'RETRY_HTTP_CODES': [429],
'SCHEDULER': 'scrapy_redis.scheduler.Scheduler',
'SPIDER_MODULES': ['ArticleSpider.spiders']}
2020-06-22 02:17:52 [scrapy.extensions.telnet] INFO: Telnet Password: a096408cb8646a95
2020-06-22 02:17:52 [scrapy.middleware] INFO: Enabled extensions:
['scrapy.extensions.corestats.CoreStats',
 'scrapy.extensions.telnet.TelnetConsole',
 'scrapy.extensions.memusage.MemoryUsage',
 'scrapy.extensions.logstats.LogStats']
2020-06-22 02:17:52 [xianzhi] INFO: Reading start URLs from redis key 'xianzhi:start_urls' (batch size: 50, encoding: utf-8)
2020-06-22 02:17:52 [scrapy.middleware] INFO: Enabled downloader middlewares:
['scrapy.downloadermiddlewares.httputauth.HttpAuthMiddleware',
 'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',
 'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',
 'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',
 'ArticleSpider.middlewares.TooManyRequestsRetryMiddleware',
 'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',
 'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',
 'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',
 'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',
 'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',
 'scrapy.downloadermiddlewares.stats.DownloaderStats']
2020-06-22 02:17:52 [scrapy.middleware] INFO: Enabled spider middlewares:
['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',
 'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',
 'scrapy.spidermiddlewares.referer.RefererMiddleware',
 'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',
 'scrapy.spidermiddlewares.depth.DepthMiddleware']
2020-06-22 02:17:52 [scrapy.middleware] INFO: Enabled item pipelines:
['ArticleSpider.pipelines.ArticleImagePipeline',
 'ArticleSpider.pipelines.MysqlTwistedPipeline',
 'ArticleSpider.pipelines.ElasticsearchPipeline',
 'scrapy_redis.pipelines.RedisPipeline']
2020-06-22 02:17:52 [scrapy.core.engine] INFO: Spider opened
2020-06-22 02:17:52 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2020-06-22 02:17:52 [scrapy.extensions.telnet] INFO: Telnet console listening on 127.0.0.1:6023
```

要启动爬虫，首先需要建立一个 Redis 客户端并连接对应的 Redis 服务端，然后使用如下命令加入将带爬取的首页 url 加入 start_urls 中：

```
lpush xianzhi:start_urls https://xz.aliyun.com/
```

Redis 命令如下：

```
P5 F:\大学基础课程\信息内容安全\code\ArticleSpider> redis-cli.exe -h 127.0.0.1 -a yuan123
Warning: Using a password with '-a' or '-u' option on the command line interface may not be safe.
127.0.0.1:6379> lpush xianzhi:start_urls https://xz.aliyun.com/
(integer) 1
127.0.0.1:6379> |
```

执行完上述命令后，可以看到 slaves 上的爬虫开始进行分布式爬取：

① Windows 10

```
2020-06-22 02:24:09 [elasticsearch] DEBUG: < {"tokens":[{"token":"结合","start_offset":0,"end_offset":2,"type":"CN_WORD","position":0}, {"token":"漏洞","start_offset":2,"end_offset":4,"type":"CN_WORD","position":1}, {"token":"漏","start_offset":2,"end_offset":3,"type":"CN_WORD","position":2}, {"token":"洞","start_offset":3,"end_offset":4,"type":"CN_WORD","position":3}, {"token":"ssrf-lab","start_offset":5,"end_offset":13,"type":"LETTER","position":4}, {"token":"ssrf","start_offset":5,"end_offset":9,"type":"ENGLISH","position":5}, {"token":"lab","start_offset":10,"end_offset":13,"type":"ENGLISH","position":6}, {"token":"学习","start_offset":13,"end_offset":15,"type":"CN_WORD","position":7}, {"token":"ssrf","start_offset":15,"end_offset":19,"type":"ENGLISH","position":8}, {"token":"漏洞","start_offset":19,"end_offset":21,"type":"CN_WORD","position":9}, {"token":"漏","start_offset":19,"end_offset":20,"type":"CN_WORD","position":10}, {"token":"洞","start_offset":20,"end_offset":21,"type":"CN_WORD","position":11}]}
2020-06-22 02:24:09 [urllib3.util.retry] DEBUG: Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
2020-06-22 02:24:09 [urllib3.connectionpool] DEBUG: http://localhost:9200 "GET /xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word HTTP/1.1" 200 334
2020-06-22 02:24:09 [elasticsearch] INFO: GET http://localhost:9200/xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word [status:200 request:0.002s]
2020-06-22 02:24:09 [elasticsearch] DEBUG: > 安全技术,WEB安全
2020-06-22 02:24:09 [elasticsearch] DEBUG: < {"tokens":[{"token":"安全","start_offset":0,"end_offset":2,"type":"CN_WORD","position":0}, {"token":"技术","start_offset":2,"end_offset":4,"type":"CN_WORD","position":1}, {"token":"web","start_offset":5,"end_offset":8,"type":"ENGLISH","position":2}, {"token":"安全","start_offset":8,"end_offset":10,"type":"CN_WORD","position":3}]}
2020-06-22 02:24:09 [urllib3.util.retry] DEBUG: Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
```

② Ubuntu

```
2020-06-22 02:22:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://xz.aliyun.com/t/2033>
None
2020-06-22 02:22:03 [scrapy.core.scrapers] DEBUG: Scraped from <200 https://xz.aliyun.com/t/2080>
None
2020-06-22 02:22:04 [urllib3.util.retry] DEBUG: Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
2020-06-22 02:22:04 [urllib3.connectionpool] DEBUG: http://localhost:9200 "GET /xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word HTTP/1.1" 200 1059
2020-06-22 02:22:04 [elasticsearch] INFO: GET http://localhost:9200/xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word [status:200 request:0.002s]
2020-06-22 02:22:04 [elasticsearch] DEBUG: > 都0202年了老嗨还在用的 - 各种姿势jsp webshell
2020-06-22 02:22:04 [elasticsearch] DEBUG: < {"tokens":[{"token":"都","start_offset":0,"end_offset":1,"type":"CN_CHAR","position":0}, {"token":"0202","start_offset":1,"end_offset":5,"type":"ARABIC","position":1}, {"token":"年","start_offset":5,"end_offset":6,"type":"COUNT","position":2}, {"token":"老","start_offset":7,"end_offset":8,"type":"CN_CHAR","position":3}, {"token":"嗨","start_offset":8,"end_offset":9,"type":"CN_WORD","position":4}, {"token":"还在","start_offset":9,"end_offset":11,"type":"CN_WORD","position":5}, {"token":"在用","start_offset":10,"end_offset":12,"type":"CN_WORD","position":6}, {"token":"各种","start_offset":16,"end_offset":18,"type":"CN_WORD","position":7}, {"token":"姿势","start_offset":18,"end_offset":19,"type":"CN_WORD","position":8}, {"token":"jsp","start_offset":20,"end_offset":23,"type":"ENGLISH","position":9}, {"token":"webshell","start_offset":24,"end_offset":32,"type":"ENGLISH","position":10}]}
2020-06-22 02:22:04 [urllib3.util.retry] DEBUG: Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
2020-06-22 02:22:04 [urllib3.connectionpool] DEBUG: http://localhost:9200 "GET /xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word HTTP/1.1" 200 334
2020-06-22 02:22:04 [elasticsearch] INFO: GET http://localhost:9200/xianzhi/_analyze?filter=%5B%27lowercase%27%5D&analyzer=ik_max_word [status:200 request:0.002s]
2020-06-22 02:22:04 [elasticsearch] DEBUG: > 安全技术,WEB安全
2020-06-22 02:22:04 [elasticsearch] DEBUG: < {"tokens":[{"token":"安全","start_offset":0,"end_offset":2,"type":"CN_WORD","position":0}, {"token":"技术","start_offset":2,"end_offset":4,"type":"CN_WORD","position":1}, {"token":"web","start_offset":5,"end_offset":8,"type":"ENGLISH","position":2}, {"token":"安全","start_offset":8,"end_offset":10,"type":"CN_WORD","position":3}]}
2020-06-22 02:22:04 [urllib3.util.retry] DEBUG: Converted retries value: False -> Retry(total=False, connect=None, read=None, redirect=0, status=None)
```

4.2 搜索网站首页

搜索网站的首页效果图展示如下，可以看到搜索框下面具有“热门搜索”和“我的搜索”列表，只需要在搜索框中输入关键词，然后点击搜索按钮即可进行搜索：



4.3 搜索提示展示

用户进行搜索，假设搜索框中输入关键词“python”，系统可根据该关键词进行搜索提示，效果展示如下：



4.4 搜索结果展示

点击搜索后，会跳转到搜索结果页面，页面左侧即位爬虫统计数据，页面中间为搜索到的文章信息（包括题目、作者、标签、评分、来源、发布时间、阅读量），页面右侧即为实时更新的“热门搜索”和“我的搜索”列表：

The screenshot shows the ISASearch search results page for the keyword "python". The page layout includes a header with the ISASearch logo and a search bar containing "python". Below the search bar, a summary bar indicates "找到约 2695 条结果(用时0.096194秒)，共约180页". The main content area displays a list of search results. The first result is titled "再谈Python RASP" by fzcxc3, with a score of 7.279782. The second result is titled "KCon 议题解读 | Python动态代码审计" by 先知技术社区, with a score of 6.963217. The third result is titled "【技术分享】python web 安全总结" by 360u621946739, with a score of 6.686404. The fourth result is titled "【技术分享】文件解压之过 Python中的代码执行" by 真鑫然的小哥, with a score of 6.532019. On the right side of the page, there is a sidebar with a "热门搜索" (Hot Search) section listing terms like ctf, python, web, java, and RCTF, and a "我的搜索" (My Search) section.

在搜索结果中还可以看到根据关键词所搜索道德结果数量、搜索用时以及页数：

This screenshot shows the same ISASearch search results page for "python", but with a focus on the pagination and search statistics. The summary bar at the top indicates "找到约 2695 条结果(用时0.096194秒)，共约180页". The search results list is partially visible, showing the first result "再谈Python RASP" by fzcxc3. The sidebar on the right is also visible, showing the "热门搜索" and "我的搜索" sections.

分页跳转则在页面的最底部：

This screenshot shows the bottom of the ISASearch search results page for "python". It features a pagination bar with the following elements: "上一页", "1", "2", "3", "4", "...", "179", "180", "下一页", and "显示第 1 条到 15 条记录，总共 2695 条". Above the pagination bar, the search results list is partially visible, showing the first result "再谈Python RASP" by fzcxc3.