现状：

随着智能手机在人们日常生活中的渗透，越来越多的人信息来源均来自互联网，而互联网上的信息量也以几何级增长着。如何在浩如烟海的内容中找到所需信息，变得非常关键。而搜索引擎也应用而生，而且必不可缺。另一方面，网页是网站设计者提供信息让用户阅读的唯一方式，是信息的载体。一个网页就是一定量信息的集合，一个网站是网页的集合，是信息的更大的集合。所以对于向访问者提供信息的网站来说，要解决好两个问题，一是信息在哪里，二是信息是什么。当今的搜索引擎根据日常生活中的需求和使用格局的差异，将搜索引擎分为了三类：Directory Search Engine, Robot Search Engine, Meta Search Engine。

目的和意义：

在本系统的模块运行后，对网络上各式各样的信息进行抓取，并存入数据库，在数据库里分词创建索引表，可以为个人用户提供更为方便快捷的信息查询服务，让用户更快的找到其所需要的信息。不仅如此，本系统通过对信息的关联性结合，还可以为用户搜索到所需信息后进一步提供尽可能多的信息拓展。

本系统通过对数据的分类分词，并且关键字索引，为个人用户在互联上起到了导航作用。

由于存储数据的可查性，还可以为企业提供信息推送服务。

**参考：**

**<https://segmentfault.com/a/1190000014333162>**

**<https://www.jasondavies.com/bloomfilter/>**

**https://www.cnblogs.com/cpselvis/p/6265825.html**

1. **引言**
   1. 系统研究的现状
   2. 系统设计的目的和意义
2. **需求分析**
   1. 设计需求
   2. 性能需求
   3. 运行环境
3. **关键技术**
   1. Scrapy-redis分布式爬取数据
   2. 反爬虫的技术实现
   3. Elasticsearch 的使用
   4. Django 框架的使用
4. **系统的设计与实现**
   1. 开发环境的介绍
   2. 系统模块的设计
   3. 页面UI设计
   4. 系统测试及其测试结果
5. **总结**
6. **致谢**

2.1设计需求

2.1.1 前端UI

简洁的页面，能够让任何用户快速上手，明白如何使用。

2.1.2数据准备

对相关网站进行数据的抓取

2.1.3 数据处理

对数据进行分词建表，构建起索引表

2.1.4 功能模块

完成查询字符串的获取，并返回相应数据

2.2性能需求

能够将用户输入信息进行实时的捕获，并且可以进行补全，纠错。

2.3 运行环境

硬件运行环境：

软件运行环境：

系统代码语言： Python

IDE： Pycharm

数据库：MySQL, Redis, Elasticsearch

虚拟环境：Vrtualenv

Python标准库：

相关框架运行环境：

爬虫框架:

Scrapy

Scrapy-redis

web框架：

Django

Python标准库：

3.1 **Scrapy-redis分布式爬取数据**

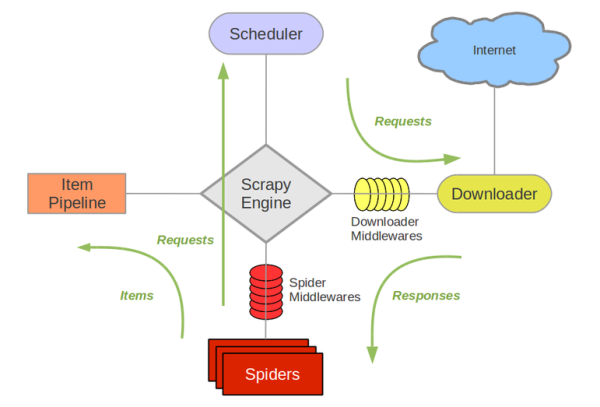
3.1.1. Scrapy

1.简介：

基于Python的Scrapy框架，它是一个为了爬取网站数据，并且能够提取结构性数据而编写的应用框架。模块化的设计，使它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。

最初，我使用python requests库去访问网页，然后再对response进行数据提取。当使用框架后，框架会帮忙处理一部分事情，我们只需要关注网络爬取的某几个部分，比如下载模块就不需要我们自己去实现，我们只需要更多的关注在数据提取上。

2.Scrapy框架的架构图：



Scrapy 组件：

**1、Scrapy Engine** (引擎)：负责管理整个程序系统，同时负责各种事件的触发，包括Spiders、Item Pipeline、Downloader、Scheduler中间的通讯、信号、数据传递等等。

**2、Scheduler** (调度器)： 接收爬虫模块(Spider)发送的request请求，并进行整理排序，去重放入队列中；取出队列中的request交给下载模块(Downloader)进行请求响应。

**3、Downloader** (下载器)：负责将从引擎模块(Scrapy Engine)发送的所有Requests请求进行网络请求，并将获取到的Responses交给引擎模块(Scrapy Engine)。

**4、Spiders** (爬虫)：负责处理所有从引擎模块(Scrapy Engine)发送过来的Responses，从这些Responses中提取数据，提取出Item字段所需要的数据和需要继续爬取数据的URL，并将Item数据和URL提交给引擎。

**5、Item Pipeline** (项目管道)：负责处理爬虫模块从网页中抽取的数据，它的主要任务是清洗、验证和存储数据。

**6、Downloader Middlewares** (下载器中间件)：介于引擎模块和下载器之间的钩子框架，主要是处理下载器与引擎之间的请求和响应。是一个可以自定以扩展的下载功能组件。

**7、Spider Middlewares** (爬虫中间件)：介于引擎模块和爬虫模块之间的钩子框架，主要任务是处理引擎与爬虫之间的请求输出和响应输入(从Spiders出去的Requests和进入Spiders的Responses)。

**9、Scheduler Middlewares** (调度中间件)：介于引擎模块和调度之间的钩子框架，从引擎模块发送到调度器的请求和响应。

1. Scrapy框架数据流程：

引擎(Scrapy Engine)打开一个网站(open a domain)，找到处理该网站的Spiders并且向该spider请求第一个要爬取的URL(start\_url)。

引擎(Scrapy Engine)从爬虫模块(Spiders)中获取到第一个要爬取的URL，并将其通过调度器中间件发送给调度器(Scheduler)，调度器将request进行排序入队。

引擎(Scrapy Engine)向调度器(Scheduler)请求处理好的request，并将处理好的request通过下载中间件(请求request方向)交给下载器(Downloader)。

下载器(Downloader)根据下载中间件的设置下载request，并将下载好的response通过下载中间件(返回response方向)交给引擎(Scrapy Engine)。

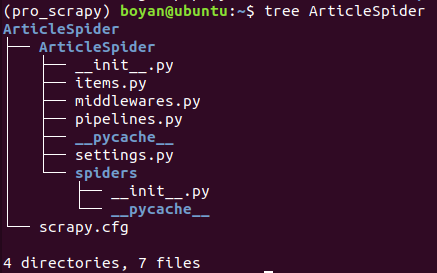
引擎(Scrapy Engine)将从下载器接收到的response通过Spiders中间件(输入方向)进行处理后交给爬虫模块(Spiders)。

爬虫模块(Spiders)将处理response，提取出Item以及需要继续爬取数据的request，将Item和request通过爬虫中间件(输出方向)交给引擎(Scrapy Engine)。

引擎(Scrapy Engine)将Item字段数据通过管道中间件交给管道模块(Item Pipeline)，并且将request通过调度器中间件交给调度器(Scheduler)。

重复执行(从第二步开始)直到调度器(Scheduler)中没有request，引擎(Scrapy Engine)关闭该网站(domian)。

1. Scrapy 目录结构图



其中：

scrapy.cfg：项目的配置文件  
 ArticleSpider/：项目的Python模块，将会从这里引用代码  
 ArticleSpider/items.py：项目的字段定义文件

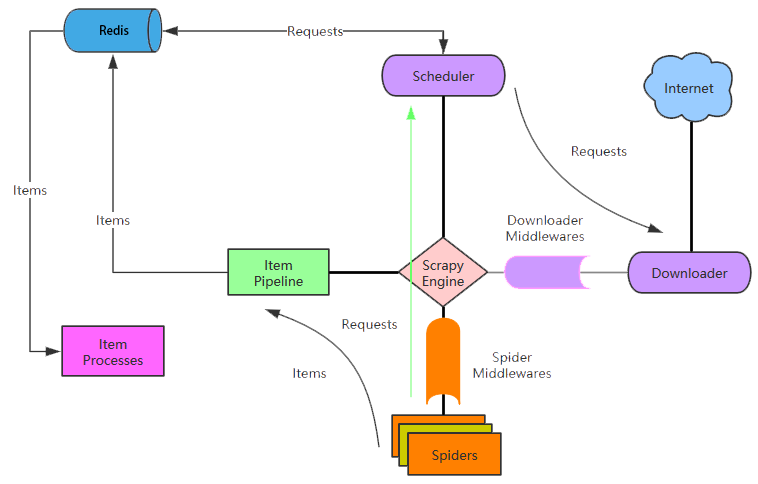
ArticleSpider/middlewares.py： 自定义中间件文件  
 ArticleSpider/pipelines.py：项目的管道文件  
 ArticleSpider/settings.py：项目的设置文件  
 ArticleSpider/spiders/：存储爬虫代码目录

3.1.2 Scrapy-redis

1、简介：

scrapy是基于python的一个很出名的爬虫框架。Scrapy虽然能做很多事情，包括数据挖掘，信息处理或者存储数据等一系列的应用，但是要做到大规模，分布式应用则捉襟见肘。所以有人提出能否在原本scrapy的队列调度的基础上，将request分离出来，从redis中读取request，这样多个客户端就可以同时从一个redis读取request并进行爬取，从而就实现了分布式的爬虫系统，因此scrapy-redis出现了以便于满足这种需求。

1. scrapy-redis 架构图



1. Scrapy-redis组件

Scrapy-redis在Scrapy框架的基础上增加了redis，基于redis的特性拓展了四个组件分别为：Scheduler, Duplication Filter, Item Pipeline, Base Spider

Scheduler: 该组件将Scrapy queue换成redis队列，将待爬取的request队列按照优先级建立一个字典结构，然后根据不同的request中的优先级，来决定该入那个队列，出列时按照优先级较小的优先出列。

Duplication Filter: 该组件对于request进行去重操作，scrapy-redis不再使用原scrapy中集合去重的方式，而是通过redis的set数据结构的不重复特性，实现了DuplicationFilter去重。

Item Pipeline: 引擎将爬取到的Item交给Item Pipeline模块，scrapy-redis的Item Pipeline模块会将爬取到的Item存入redis中的 items queue中。从而可以很方便的根据key值从items queue中取到item，从而实现items processes集群。

Base Spider: 重写的RedisSpider不再是使用Scrapy的原始Spider 类，而是继承自Spider类和RedisMixin类，RedisMixin是用于从redis 读取URL的类。 在生成一个继于RedisSpider的Spider时，调用 setup\_redis函数以连接到redis数据库，然后设置signals(信号)。 一 个 是spider空闲时的信号，会调用spider\_idle函数，而这个函数又 会去调 用schedule\_next\_request函数确保spider始终处于活动状态， 并抛出 DontCloseSpider异常。一个是捕获item时的信号，会调用 item\_scraped 函数，此函数调用schedule\_next\_request函数以获取 下一个请求 (request)。

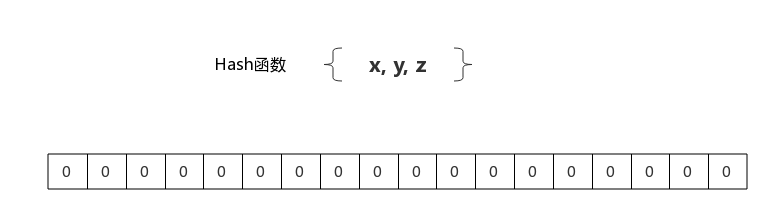
1. scrapy-redis工作流程

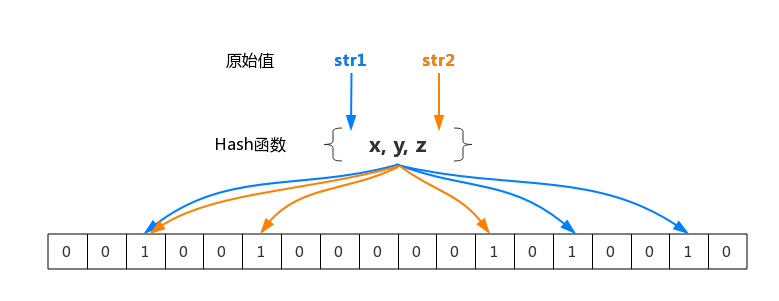
每次spider发出一个request(请求)时，scrapy引擎都会向与spider相对应的调度(scheduler)对象提交请求以进行调度。调度对象访问redis以评估请求。如果不重复，就将其添加到redis。在调度队列中当满足调度条件时，调度对象从redis调度队列接收请求并将其发送到spider进行爬取。当spider抓取所有临时可用的URL之后，调度程序会发现与spider相对应的Redis调度队列为空，于是触发信号spider\_idle，在spider收到此信号后，直接连接redis以读取strart\_url池，获取新的一批处理URL条目并再次重复上述任务。

3.1.3 Bloom Filter(布隆过滤器)

1、简介： 本质上Bloom Filter是一种数据结构，比较巧妙的概率型数据结构（probabilistic data structure），特点是高效地插入和查询，可以用来告诉你 “某样东西一定不存在或者可能存在”。

2、实现原理：





Bloom Filter是一个bit向量或者说是bit数组，初始化每个位都设置为0。

我们通过使用多个不同的哈希函数生成多个哈希值，并对每个哈希值映射到Bloom Filter中去，每个哈希值所指向的bit位置值变为1。

添加(add)操作：将要添加的元素通过k个哈希函数映射出k个值，对应Bloom Filter上的k个位置，将这k个位置设置为1。

查询(Is Exists)操作：将要查询的元素通过k个哈希函数映射出k个值，对应Bloom Filter上的k个位置，如果k个位置上有一个为0，则肯定不存在集合中，如果k个位置全部为1，则有可能在几何中。

3.2**反爬虫的技术**

3.2.1 通过headers字段来反爬

1.通过headers中的User-Agent字段来反爬,使用函数完成User-Agent池,随机生成User-Agent

import random

def get\_ua():

first\_num = random.randint(55, 62)

third\_num = random.randint(0, 3200)

fourth\_num = random.randint(0, 140)

os\_type = [

'(Windows NT 6.1; WOW64)', '(Windows NT 10.0; WOW64)', '(X11; Linux x86\_64)',

'(Macintosh; Intel Mac OS X 10\_12\_6)'

]

chrome\_version = 'Chrome/{}.0.{}.{}'.format(first\_num, third\_num, fourth\_num)

ua = ' '.join(['Mozilla/5.0', random.choice(os\_type), 'AppleWebKit/537.36',

'(KHTML, like Gecko)', chrome\_version, 'Safari/537.36']

)

return ua

1. 通过referer字段或者其他字段来反爬
2. 通过Cookie来反爬

不需要登陆,每次请求带上上一次返回的cookie

需要登陆, 准备多个账号通过程序登陆网站,

并收集Cookie,组成cookie池

3.2.2 通过js实现跳转来反爬

1. 通过js实现跳转来反爬

在请求目标网站的时候，我们看到的似乎就请求了一个网站，然而实际上在成功请求目标网站之前，中间可能有通过js实现的跳转，我们肉眼不可见，这个时候可以通过点击perserve log按钮实现观察页面跳转情况

在这些请求中，如果请求数量很多，一般来讲，只有那些response中带cookie字段的请求是有用的，意味着通过这个请求，对方服务器有设置cookie到本地

2.通过js生成了请求参数

3.通过js实现了数据的加密

以上两种可以通过分析js加密方式,然后运用Python的第三方库运行js获取response,或者使用Selenium+PhantomJS模拟浏览器操作绕过js但该爬虫方式爬行速度极其缓慢

3.2.3 通过IP地址来进行反爬

同一个ip大量请求了对方服务器，有更大的可能性会被识别为爬虫，对应的通过购买高质量的ip的方式能够结局问题

3.3 **Elasticsearch**

3.3.1 简介

开源的 Elasticsearch 是目前全文搜索引擎的首选。

它可以快速地储存、搜索和分析海量数据。维基百科、Stack Overflow、Github 都采用它。

Elasticsearch 的底层是开源库 Lucene。但是，你没法直接用 Lucene，必须自己写代码去调用它的接口。Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。

Elasticsearch 是用Java实现的。

搜索引擎在对数据构建索引时，需要进行分词处理。分词是指将一句话拆解成多个单字或词，这些字或词便是这句话的关键词。

Elasticsearch 不支持对中文进行分词建立索引，需要配合扩展elasticsearch-analysis-ik来实现中文分词处理。

3.4 **Django 框架**

Django的主要目的是简便、快速的开发数据库驱动的网站。它强调代码复用，多个组件可以很方便的以"插件"形式服务于整个框架，Django有许多功能强大的第三方插件，你甚至可以很方便的开发出自己的工具包。这使得Django具有很强的可扩展性。它还强调快速开发和DRY(DoNotRepeatYourself)原则。

重量级框架

对比Flask框架，Django原生提供了众多的功能组件，让开发更简便快速。

提供项目工程管理的自动化脚本工具

数据库ORM支持（对象关系映射，英语：Object Relational Mapping）

模板

表单

Admin管理站点

文件管理

认证权限

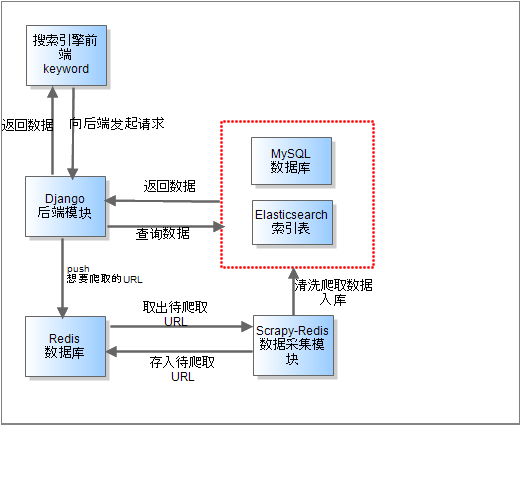
session机制

缓存

4.1开发环境的介绍

4.2系统模块的设计

4.2.1 设计流程图



本系统共有三个模块，分别为：爬虫模块，数据库模块，web框架模块。

4.2.2 模块详解

1、爬虫模块

(1)介绍

本模块完成了对站点进行分布式爬取数据，并将数据清洗，存储到数据库中。

在爬取数据阶段，通过使用基于Scrapy框架的Scrapy-redis进行分布式爬取。

在设计Scrapy-redis时

(2)scrapy-redis的配置

(3)中间件处理反爬虫机制

(4)针对request去重占用大内存，提出使用bloom filter

(5)关于Scrapy-redis分布式策略

2、数据库模块

(1)介绍

(2)爬取数据的存储

(3)docker 运行redis数据库

(4)docker 运行elasticsearch数据库

3、web框架模块

(1)介绍

1. Django ORM数据库模型

(3)Django-haystack 对接elasticsearch数据库

(4)返回数据接口

4.3页面UI设计

1. 首页(index)
2. 结果页 (result)

4.4系统测试及其测试结果

1. 爬虫模块测试以及测试结果
2. web返回数据测试以及测试结果