目录

[一、 爬虫设计 1](#_Toc5147)

[1. 相关技术背景（背景技术） 1](#_Toc18163)

[2. 架构的现有实现方案（现有技术） 1](#_Toc17797)

[3. 本架构设计 5](#_Toc23089)

[二、 模块图 5](#_Toc10740)

[三、 详细设计 8](#_Toc5252)

# 爬虫设计

## 相关技术背景（背景技术）

**背景技术**

随着大数据时代的到来，数据的重要性日渐突出，大量数据的采集工作也越来越重要。目前，数据采集的途径主要分为两种：第一种是充分利用公司内部数据流来进行数据的采集，这种采集方式优势是数据量大，采集成本低，但只能采集数据流中的数据；第二种是充分利用互联网的开放性，在互联网中采集目标数据，常见技术是针对需要采集的数据来进行编码抓取指定数据。在本发明中，通过建立爬虫平台，并根据任务需求配置相应的配置文件，不同任务组合成不同的规则模板，执行器运行规则模板，最终实现网页爬取。

## 架构的现有实现方案（现有技术）

**现有技术的技术方案**

（1）搜索引擎类信息采集：像google、百度，这种平台能够全网采集Internet上的开放信息，采集数据范围广，更新频率快，采集数据量大，不过采集的信息多为模糊信息，虽然有利于信息检索，不过想要采集精准数据，不做特殊处理较为吃力。

（2）精确爬取类信息采集：此类信息采集可以通过两种方法：a.需要针对每类页面来定制编码抓取，优点是数据采集灵活，抓取信息比较精准，抓取方向可控。b.通过配置的方式实现多个领域的网页抓取和信息抽取。

**现有技术的缺点**

传统的搜索引擎搜索意图不明，结果精确度不高，数据非结构化，难以提供准确而干净的数据。

我们来看一下主流爬虫框架在GitHub上的活跃度：



**Nutch介绍：**

Nutch是一个开源的Java实现的搜索引擎。它提供了我们运行自己的搜索引擎所需的全部工具，包括全文搜索和网络爬虫。

尽管搜索是上网的基本要求，但是现有的搜索引擎的数目却在下降。并且这很有可能进一步演变成为一个公司垄断了几乎所有的网页搜索为其谋取商业利益。这显然不利于广大Internet用户。

Nutch 为我们提供了这样一个不同的选择。相对于那些商用的搜索引擎，Nutch作为开放源代码的搜索引擎将会更加透明，从而更值得大家信赖。现在所有主要的搜索 引擎都采用私有的排序算法, 而不会解释为什么一个网页会排在一个特定的位置。除此之外, 有的搜索引擎依照网站所付的费用, 而不是根据它们本身的价值进行排序。与它们不同，Nutch没有什么需要隐瞒，也没有动机去扭曲搜索的结果。Nutch将尽自己最大的努力为用户提供最好 的搜索结果。

优点：

Nutch支持分布式抓取，并有Hadoop支持，可以进行多机 分布抓取，存储和索引。另外很吸引人的一点在于，它提供了一种插件框架，使得其对各种网页内容的解析、各种数据的采集、查询、集群、过滤等功能能够方便的 进行扩展。正是由于有此框架，使得 Nutch的插件开发非常容易，第三方的插件也层出不穷，极大的增强了 Nutch的功能和声誉。

缺点：

对 于大多数用户来说，一般是想做一个精确数据爬取的爬虫“主题爬虫”。如果以此为标准，那么为搜索引擎而生的Nutch就有着天生缺点。Nutch的架构里大部分为搜索引擎而设计的，对精确爬取没有特别的考虑。也就是 说，用Nutch做主题爬虫，会浪费很多的时间在不必要的计算上。而且如果你试图通过对Nutch进行二次开发来扩展其定制能力，基本上就要破坏 Nutch的框架，反而不如自己写了。

**Pyspider介绍：**

Pyspider是一个国人编写的强大的网络爬虫系统并带有强大的WebUI。采用Python语言编写，分布式架构，支持多种数据库后端，强大的WebUI支持脚本编辑器，任务监视器，项目管理器以及结果查看器。

Pyspider 的主要功能包括，抓取、更新调度多站点的特定的页面；需要对页面进行结构化信息提取；灵活可扩展，稳定可监控。满足了绝大多数Python爬虫的需求 —— 定向抓取，结构化化解析。但是面对结构迥异的各种网站，单一的抓取模式并不一定能满足，灵活的抓取控制是必须的。为了达到这个目的，单纯的配置文件往往不 够灵活，于是，通过脚本去控制抓取成为了最后的选择。而去重调度，队列，抓取，异常处理，监控等功能作为框架，提供给抓取脚本，并保证灵活性。最后加上 web的编辑调试环境，以及web任务监控，即成为了最终的框架。

优点：

1.支持分布式部署。

2.完全可视化，对用户非常友好：WEB 界面编写调试脚本，起停脚本，监控执行状态，查看活动历史，获取结果产出。

3.简单，五分钟就能上手。脚本规则简单，开发效率高。

4.支持抓取JavaScript的页面。

总之，Pyspider非常强大，强大到更像一个产品而不是一个框架。

缺点：

1.URL去重使用数据库而不是布隆过滤器，亿级存储的db io将导致效率急剧降低。

2.使用上的人性化牺牲了灵活度，定制化能力降低。

**Scrapy介绍：**

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 使用 Twisted这个异步网络库来处理网络通讯，架构清晰，并且包含了各种中间件接口，可以灵活的完成各种需求。Scratch，是抓取的意思，这个 Python的爬虫框架叫Scrapy，大概也是这个意思吧。

优点：

1.极其灵活的定制化爬取。

2.社区人数多、文档完善。

3.URL去重采用布隆过滤器方案。

4.可以处理不完整的HTML，Scrapy已经提供了selectors（一个在lxml的基础上提供了更高级的接口），可以高效地处理不完整的HTML代码。

缺点：

1.不支持分布式部署。

2.原生不支持抓取JavaScript的页面。

3.全命令行操作，对用户不友好，需要一定学习周期。

## 本架构设计

### 模块图

Fetcher采集

+Proxy

1. Get
2. Post
3. Webdriver-selenium

ETL

tools

DB 落库

1. Mq
2. mongoDB
3. HBase

Parser解析

1. Jsoup
2. Regex
3. Xpath

JobTimer

根据Job周期启动Job

JobTimer(master)

JobService:

1. createJob
2. updateJob
3. deleteJob
4. selectJob

DoingJobQueue

CrawlerQueue

Request、taskName

JobControl

开多线程执行Task

TaskWork\_list

Init  
tsakName1

taskName2

FlyCompile

动态编译需要的task脚本

+cache

### 数据流图

根据需求业务逻辑和项目规范编写Job脚本

JobTimer(master)

根据Job周期启动Job，

把要启动的Job放入JobDoing池

并初始化CrawlerQueue

CrawlerQueue

Crawler这个对象会存request、

JobId、taskName，

可根据crawler.uuid去重

JobControl

去DoingJobQueue循环拿取Job并去对应的CrawlerQueue拿取Crawler多线程调用TaskWork

当判拿取的Job任务结束时把本Job从队列移除

DoingJobQueue

存放所有正在执行的Job

MySQL

DB

tools

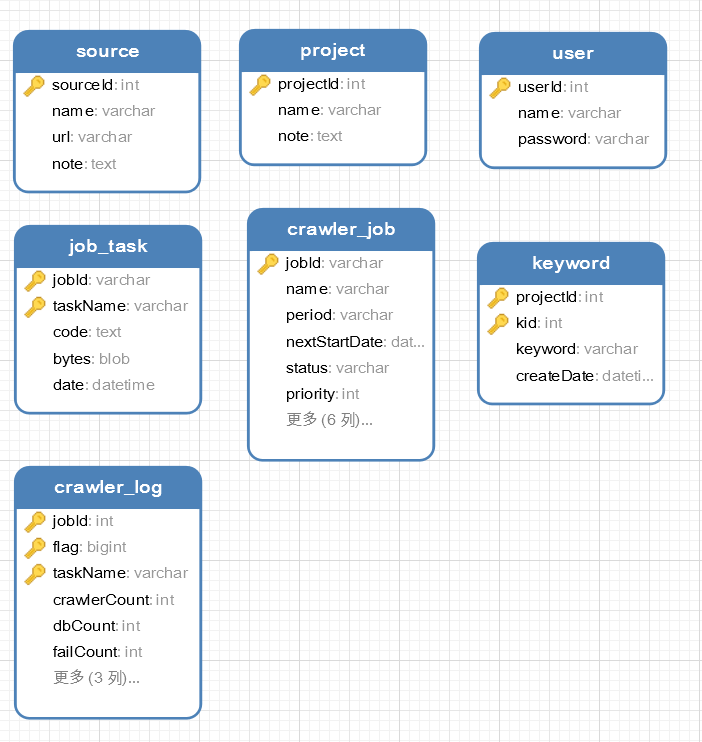
TaskWork\_list

各Job的脚本会调tools工具获取页面并解析页面和生成新task并落库数据

JobService

创建Job 调用动态编译模块把源码和class入库

### 数据库设计



# 详细设计

## 数据库详细设计

##crawler\_job表 存储job相关记录

CREATE TABLE `crawler\_job` (

`jobId` varchar(255) NOT NULL DEFAULT '' COMMENT 'job Id 唯一',

`name` varchar(255) DEFAULT NULL COMMENT 'job 名称',

`period` varchar(255) DEFAULT NULL COMMENT 'job 周期 毫秒单位',

`nextStartDate` datetime DEFAULT NULL COMMENT 'job 下一个周次执行时间',

`status` varchar(255) DEFAULT NULL COMMENT 'job 状态',

`priority` int(11) DEFAULT NULL COMMENT 'job 优先级',

`maxThread` int(11) DEFAULT NULL COMMENT 'job 最大使用线程数',

`note` text COMMENT 'job 备注',

`projectId` int(11) DEFAULT NULL COMMENT 'job 所属项目id ',

`sourceId` int(11) DEFAULT NULL COMMENT 'job 数据源id',

`createDate` datetime DEFAULT NULL COMMENT 'job创建时间',

PRIMARY KEY (`jobId`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

##job\_task表 存储job拆分的每一个class

CREATE TABLE `job\_task` (

`jobId` varchar(255) NOT NULL DEFAULT '' COMMENT 'jobId',

`taskName` varchar(77) NOT NULL COMMENT 'taskName',

`code` text COMMENT '脚本代码',

`bytes` mediumblob COMMENT '脚本二进制',

`date` datetime DEFAULT NULL COMMENT '创建时间',

PRIMARY KEY (`jobId`,`taskName`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

##crawler\_log 表 存储任务执行状态日志

CREATE TABLE `crawler\_log` (

`jobId` varchar(180) NOT NULL COMMENT 'jobId',

`flag` bigint(20) NOT NULL COMMENT 'job周期 启动的时间戳',

`taskName` varchar(150) NOT NULL COMMENT 'taskName',

`successCount` int(255) DEFAULT NULL COMMENT 'task 执行成功次数',

`failCount` int(255) DEFAULT NULL COMMENT 'task执行失败次数',

`toDoCount` int(255) DEFAULT NULL COMMENT 'task 剩余个数',

`createDate` datetime DEFAULT NULL COMMENT '日志创建时间',

`modifiedDate` datetime DEFAULT NULL COMMENT '日志更新时间',

PRIMARY KEY (`jobId`,`flag`,`taskName`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

##keyword表 存储关键词

CREATE TABLE `keyword` (

`projectId` int(11) NOT NULL COMMENT '项目id',

`kid` int(11) NOT NULL AUTO\_INCREMENT COMMENT '关键词id',

`keyword` varchar(255) DEFAULT NULL COMMENT '关键词',

`note` varchar(255) DEFAULT NULL COMMENT '关键词注释',

`createDate` datetime DEFAULT NULL COMMENT '关键词创建时间',

PRIMARY KEY (`projectId`,`kid`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

##另有project source user表作为crawler\_job表的补充标记

CREATE TABLE `project` (

`projectId` int(11) NOT NULL,

`name` varchar(255) DEFAULT NULL,

`note` text,

PRIMARY KEY (`projectId`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE `source` (

`sourceId` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) DEFAULT NULL,

`url` varchar(255) DEFAULT NULL,

`note` text,

PRIMARY KEY (`sourceId`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

CREATE TABLE `user` (

`userId` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(255) DEFAULT NULL,

`password` varchar(255) DEFAULT NULL,

PRIMARY KEY (`userId`)

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

## 容器模块

*/\*\*  
 \* 单机Crawler容器，负责存储多个job的海量crawler  
 \*/*public class LonelyCrawlerButler implements Butler<Job,Crawler> {

*/\*\*  
 \* 单机Job容器，负责存储运行的job信息  
 \*/*public class LonelyJobNotice implements Notice<Job,Job> {

*/\*\*  
 \* 单机Job计数器，负责存储运行的job统计信息  
 \*/*public class LonelyJobCounter implements JobCounter {

*/\*\*  
 \* 本机Script缓存容器 负责存储脚本缓存  
 \*/*@Service  
public class NativeScriptCache implements Cache<JobCrawler,Script>{

## JobTimer 调度模块

jobTimer是job调度模块，maseer节点 当有需要调度的job会放入 LonelyJobNotice 容器

和放入 LonelyCrawlerButler 中一个种子Crawler

*/\*\*  
 \* job调度 参考crawler\_job表启动相关job  
 \*/*@Service  
public class JobTimerThread implements Runnable{

## JobControl 模块

*/\*\*  
 \* 负责读取正在运行的job信息 并且拉去该job的crawler到本机的消费池中  
 \*/*@Service  
public class JobControl implements Runnable {

## TaskWorkControl 模块

*/\*\*  
 \* Created by dang on 17-5-10.  
 \* task 管理模块 负责taskWork的线程池 、crawler消费池 和消费完毕的后续处理  
 \*/*@Service  
public class TaskWorkControl {

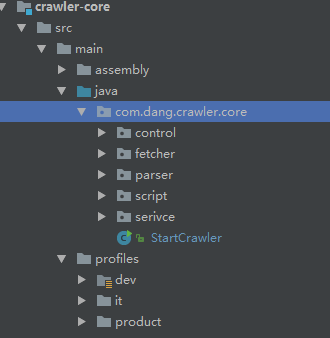
*/\*\*  
 \* 负责crawler的消费  
 \*/*public class TaskWork implements Runnable {

## Service 调度模块

*/\*\*  
 \* 负责job task 的创建工作  
 \*/*@Service  
public class JobTaskService {

*/\*\*  
 \* 关键词服务 对关键词的增删改查  
 \*/*@Service  
public class KeywordService {

# 代码目录



其中

control 存放任务调度执行引擎

Fetcher 存放关于页面采集类的代码

Parser 存放关于页面解析类的代码

Script 存放脚本需要用到的工具 和规范接口

Serivce 存放对外服务代码