**Geeking搜索引擎开发**

目录

[项目模块划分&功能实现 1](#_Toc402040740)

[ **GeekSearch（dynamic web project）** 1](#_Toc402040741)

[ **spider** 1](#_Toc402040742)

[ **indexer** 2](#_Toc402040743)

[ **queryer** 2](#_Toc402040744)

[数据结构设计 3](#_Toc402040745)

[ **原始网页库** 3](#_Toc402040746)

[ **网页信息表** 3](#_Toc402040747)

[ **文档索引** 4](#_Toc402040748)

[ **倒排索引** 4](#_Toc402040749)

[ **数据处理流程** 4](#_Toc402040750)

[ **Tips** 5](#_Toc402040751)

[命名规则 5](#_Toc402040752)

[ **package：** 5](#_Toc402040753)

[ **class：** 5](#_Toc402040754)

[ **function/variable：** 5](#_Toc402040755)

# 项目模块划分&功能实现

整个项目分为两个工程，具体如下：

* **GeekSearch（dynamic web project）**
* **spider**
  + 页面抓取管理（geekspider.clawer）：
    - 设定抓取策略，访问、采集网页
    - 实现定期更新（e.g.网站添加了新的url或者原有url网页内容被修改过）
  + 页面信息过滤与抽取（geekspider.extracter）：
    - 选择抽取策略，提取关键信息，e.g.时间、标题、关键词、热度（评论数？）
    - 尽量提高抽取算法鲁棒性，而不是只针对固定几种网页模板
  + 页面信息数据库（com.geekspider.util [**公共方法，util->database**]）：
    - 抽取后的页面信息存储于webDB（基于mysql）
* **indexer**
  + 建立索引（geeksearch.indexer）
    - 根据webDB（Read-Only here），建立词典（第三方词典/开源工具）：dicDB
    - term->webIDList,location，建立倒排索引：indexDB
    - 词典&索引压缩
    - 先学习lucene，后期优化尝试多种索引建立的实现方法，e.g. MapReduce等
  + 查询(geeksearch.query)
    - 查询语句分析
      * 可利用好的开源中文分词器
        + 需对现有中文分词器进行比较，选出优秀的一个
        + 将其嵌入lucene框架
    - 搜索策略
      * 合并算法等
  + 结果处理（geeksearch.sort）
    - 对相似新闻进行聚类
      * 聚类算法
      * UI上如何显示
    - 排序：
      * 默认按照相关度排序
      * User可以自己定义排序标准，e.g.相关度(location)、热度(评论数)、时间等
    - 返回结果（）：
      * 传递给web端
* **queryer**
  + - 搜索页：
      * 可配置项，高级搜索（结果排序选项）等
      * 查询自动补齐
      * 传递搜索词到query中（转换为string）
    - 结果页（参考百度）：
      * 从query获得检索结果
      * Snippet生成
      * 结果预览（快照）
      * 相关搜索推荐（纠错）
  + 系统测试
    - 召回率
    - 精准度
    - 响应速度

# 数据结构设计

* **原始网页库**

方法一：

以html文件形式存储，**文件名为url**（将dot替换成**分号**）。

以分类目录方式存储：sina、sohu目录（或basketball/football）等

缺点：构建倒排索引时输入速度慢。

方法二：

按照分类文件存储。

同一类型的文件按照一定格式存储于同一个文件中。

E.g.

**原始网页库中的一条网页记录**

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx // 之前的记录

url:http://ast.nlsde.buaa.edu.cn/

date:Mon Apr 05 14:22:53 CST 2010

length:3981

<!DOCTYPE …… // 记录数据部分

<html> …… </html>

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx // 之后的记录

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

缺点：不能按照网页 URL 直接定位到所指向的网页。

需要建立网页库索引。不过网页库索引可同时存储网页标题、摘要、日期等信息；

法1要存储这些信息也需要建立网页信息数据表，这点上扯平。

* **网页信息表（PageIndex）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 属性名 | 文档ID | URL | 标题 | 摘要 | 日期 | 类型 | 关键字 |
| 对应名称 |  | Url | Title | Discription | Date | Type | Keywords |
| 实例 | 18 | [www.sina.com](http://www.sina.com)... | XxX | XXX | 2014-10-27 | 篮球 | X,X,X |
| 类型 | INT | varchar | varchar | text | DATE | varchar | varchar |
| 范围 |  | 100B | 50B | 64KB | 1111-01-01 TO 9999-12-31 | 30B | 30B |

### 数据库存储形式：

**PageIndex**表格，表名不区分大小写

SQL创建表格**PageIndex**：

工具: Navicat for MySQL

参数说明：属性名 + 类型 + （主键）+ （最大长度范围）

CREATE TABLE PageIndex

(

ID int primary key,

Url varchar(100),

Title varchar(50),

Digest text,

Date date,

Category varchar(30),

Offset bigint,

Keywords varchar(30)

)

文档ID：主键，（映射为URL，一一对应）。

URL：非常重要

标题：用于 结果显示/分层索引

摘要：用于 结果显示（如果动态摘要就忽略词项）/分层索引

日期：网页日期

类型：网页类型（~~具体根据什么分类有待商议~~，根据门户网站名分类），用于 法二存储网页/根据类型进入目录读取文件/分类搜索等

关键字：体育新闻网页上的关键字，用于 结果显示/结果排序

偏移：法二独有

原始文档路径：

法一：根据 类型+url 进入目录读取原始文档

法二：根据 类型+偏移进入文件读取原始文档

两种方法均需要先从配置文件读取目录的 绝对路径

* **文档索引（DocIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 文档ID | 词项IDs |
| 对应名称 | DocumentID | TermIDs |
| 实例 | 1 | 11#2#13… |
| 类型 | int | text |
| 范围 | 0 - 4294967295 | 64KB |

SQL创建表格**DocIndex**：

工具: Navicat for MySQL

参数说明：属性名 + 类型 + （主键）+ （最大长度范围）

CREATE TABLE DocIndex

(

DocumentID int primary key,

WordIDs text

)

(注：此处的词项会有重复项。构建过程中，生成 )

* **词项 - 词项ID 的映射表（TermsIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 词项ID | 词项 |
| 对应名称 | TermID | Term |
| 实例 | 11 | 极客 |
| 类型 | int | varchar |
| 范围 | 0 - 4294967295 | 20B |

SQL创建表格**TermsIndex**：

工具: Navicat for MySQL

参数说明：属性名 + 类型 + （主键）+ （最大长度范围）

CREATE TABLE TermsIndex

(

WordID int primary key,

Term varchar(20)

)

* **倒排索引（InvertedIndex）**

|  |  |  |
| --- | --- | --- |
| 属性名 | 词项ID | 文档IDs |
| 对应名称 | TermID | DocumentIDs |
| 实例 | 11 | 1;2;1:10#1;2;1:10#.... |
| 类型 | int | text |
| 范围 | 0 - 4294967295 | 64KB |

SQL创建表格**InvertedIndex**：

工具: Navicat for MySQL

参数说明：属性名 + 类型 + （主键）+ （最大长度范围）

CREATE TABLE InvertedIndex

(

WordID int primary key,

DocumentIDs text

)

注：

**文档ID**格式：

<DocID>;<TF>;<POS1>:<POS2>

其中**位置信息**取词项在文档索引中的index号

倒排索引存于数据库中，启动服务时再读入内存。

* **数据处理流程**

Spider部分：

1. 爬取原始html网页
2. 存储为**原始网页库**（采用~~法2~~ **法1**存储）
3. ~~同时生成网页信息表~~

Indexer部分：

1. 读取**原始网页库**，抽取关键信息（<title>等信息），生成**网页信息表**
2. 过滤**（**去除<script>等无用信息**）**再将html标签清除，提取正文（全部内容））
3. 对正文进行分词
4. 生成**文档索引**（同时生成**词项ID-词项映射表**）
5. 生成**倒排索引**（合并相同词项，统计TF，POS）
6. 网页信息表、正向索引、倒排索引，均需存储到数据库

Query部分：

1. （服务器启动时读入**倒排索引**）
2. 对查询词进行分词得到**词项ID集合**
3. 根据词项ID集合检索**倒排索引**，合并之后得到**文档ID集合**(包含TF)
4. 根据TF/POS等对**文档ID集合**进行排序
5. 根据**文档ID集合**从**网页信息表**中获取相关字段，显示于结果页面。
6. 根据网页信息表中的偏移等，从**原始网页库**读取相应网页实现快照功能

* **Tips**

1. 关于倒排索引的构建：

上述Indexer流程概括了倒排索引的构建，实际上可分为title倒排索引，摘要倒排索引，正文倒排索引。便于之后Query部分的排序。

1. 原始网页库存为少数几个文件在进行结果显示的时候效率会比较高，不必打开众多文件。
2. 每个数据结构都的存储要进行数据库操作，实际是读写磁盘处理，会降低索引构建速度。可考虑若数据集不大，可在query前先读入内存。

# 命名规则

* **package：**
  + 全部小写
  + com.geek.geeksearch.spider
  + …
* **class：**
  + 首字母大写，动宾结构/名词
  + com.geek.geeksearch.model：DocIndex.java
* **function/variable：**
  + 首字母小写
* 具体细节请参考**/references/JAVA语言编程规范(华为).pdf**