Geeking 搜索引擎项目报告

团队名称：GeeK

团队成员：

陈新荃 2014E8018461048

高妍 201428013229077

林裕杰 201428017729020

肖卡飞 201428013329024

2014 年 12 月 14 日

目录

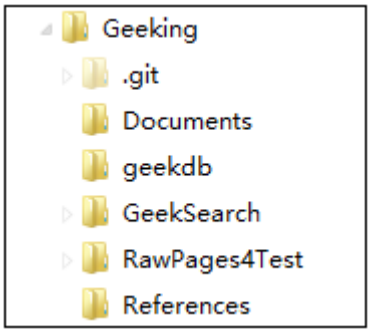
一．项目总体介绍

Geeking，是一款体育新闻搜索引擎，由 GeeK 团队开发。Geeking 能够爬取各大门户网站的体育新闻（目前支持搜狐、腾讯、网易、MSN），并建立成倒排索引，实现体育新闻搜索。其功能包括：相关度排序、相似新闻聚类、搜索词推荐、网页摘要、关键词高亮、网页快照等。

Geeking 项目由 Java 语言编写，总代码量约为 3750 行（包括 java、jsp 文件以及空行）。该项目的实现除了团队编写的核心内容之外，还采用了 htmlparser（网页过滤）、ansj_seg（ict.中文分词）等第三方工具辅助开发。

团队使用 GitHub 作为协同开发工具，目前项目版本为 0.2，总共发生了超过 200 次代码提交。项目地址为：https://github.com/UCAS-GeeK/Geeking。

项目目录结构如下：



Documents：团队开发过程中积累的开发文档，包括例会日志、协同开发教程、设计文档等等。

geekdb：团队开发过程中使用的小型数据库。

GeekSearch：项目工程，开发平台为 eclipse。

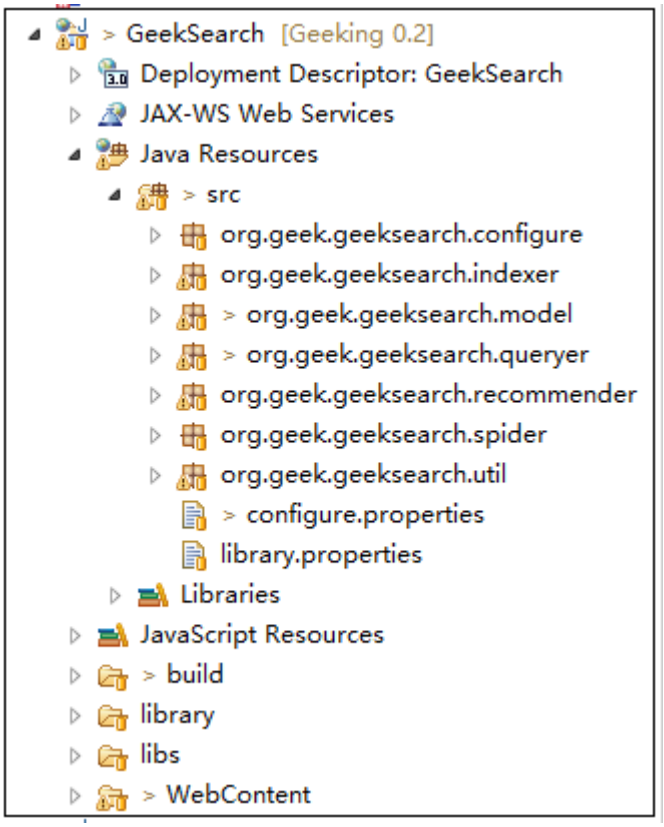
RawPages4Test：测试用的 html 网页。

References：参考文档，包括代码规范等。

本报告后续内容将根据 Geeking 项目工程 GeekSearch 来展开。

二．设计方案

GeekSearch 工程主要模块结构如下：



其中，主要的模块所实现的功能如下：

模块名	功能
configure	系统参数配置管理
model	数据结构
spider	爬虫
indexer	索引构建
queryer	搜索模块
recommender	检索词推荐
util	通用工具
webContent	前端显示
library	分词词典

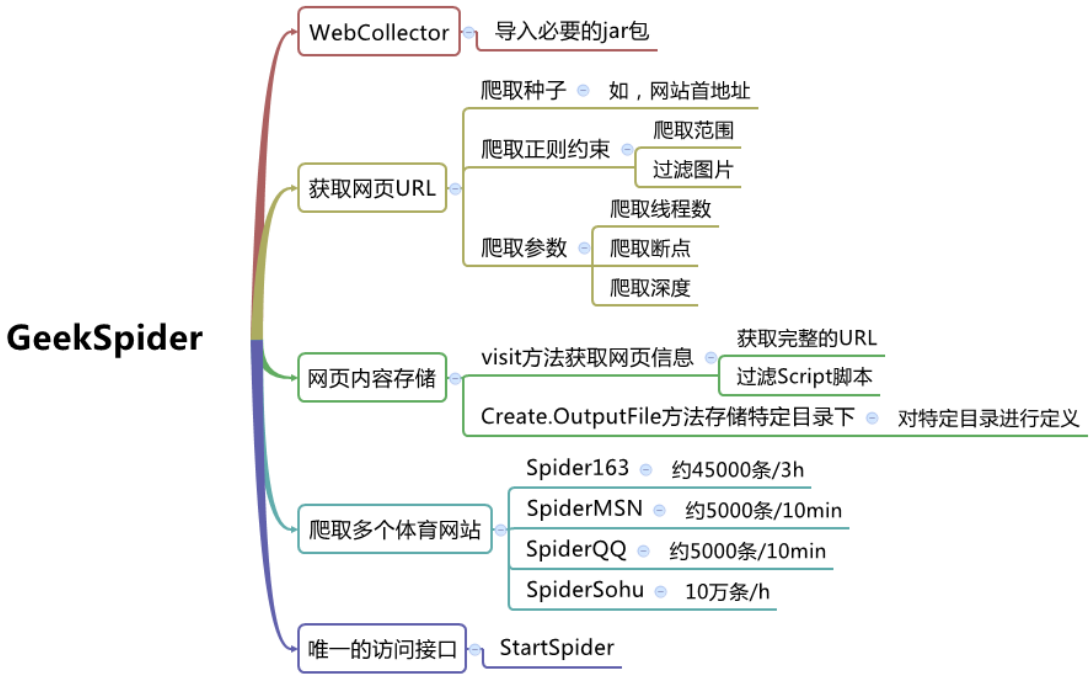
下面详细介绍各个模块的设计与实现。

1. 网页爬虫

1.1 爬取策略

本项目的 GeekSpider 是基于开源 JAVA 爬虫 WebCollector 实现而成。WebCollector 是一个无须配置、便于二次开发的 JAVA 爬虫框架（内核），它提供精简的 API，只需少量代码即可实现一个功能强大的爬虫。GeekSpider 在此内核上采用基于广度优先算法的多线程网页获取，并通过一定正则约束条件，剔除获取到本地的网页的图片信息，script 脚本信息等。并以网页的 URL 命名保存在本地（其中‘：’用‘$’代替，‘/’用‘#’代替，如：http$##sports.163.com#04#0918#23#10JMFUK400051CA0.html）。

1.2 具体实现



以爬取 Sports163 为例：

(1) 继承 WebCollector 中的 BreadthCrawler 父类，派生出 Spider163 类；

(2) 爬取网页信息；

i. 设定爬取种子入口，一般把爬虫的种子设为网站的首页；

ii. 定义正则约束条件，给定爬取约束范围并且过滤网页中的图片格式信息；

iii. 爬取参数的设定(断点爬取、爬取深度、爬取线程数等)；

iv. 自定义 visit 函数访问获取的 URL 页面信息，并通过自定义的 CreatHtml 类的 OutputFile 方法把网页信息保存在指定目录下，其中在保存本地的过程中剔除不必要的 Script 脚本信息，为快照的实现减小文本规模。

按照类似方法，能够获取 MSN，QQ，Sohu 等门户网站的体育新闻信息。

2. 索引构建

索引构建主要包括网页过滤抽取、相关数据库表建立等。具体数据结构和构建方法将分别在 2.1 和 2.2 中介绍，整个索引构建的流程如下：

(1) 从**原始网页库**读取网页；

(2) 抽取关键信息(title，keywords，description 等)生成**网页信息表**(PagesIndex)并写进数据库；

(3) 抽取正文进行分词，并根据分词结果生成文档索引（DocsIndex，即**正向索引**），同时生成**词项 ID-词项映射表**（TermsIndex），并写进数据库；

(4) 读取文档索引表，合并相同词项，统计 TF，DF，词项在文档中位置 POS，生成**倒排索引**（InvertedIndex），并写进数据库；

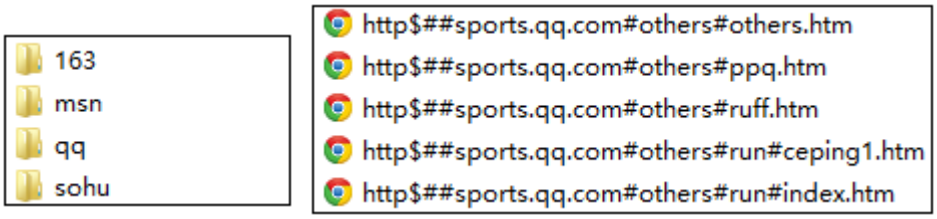
(5) 索引构建结束。

2.1 数据结构

存储在 Mysql 数据库中的表格包括：网页信息表、文档索引、词项 ID 映射表、倒排索引表，而原始网页库存储在文件系统。

原始网页库（RawPages）

原始网页库以 html 文件形式存储，以其 url 命名，其中，将 “/”替换成 “#”，将“：”替换成“$”，并以分类目录方式存储，如：sina、sohu。



网页信息表（PagesIndex）

属性名	文档ID	URL	标题	摘要	日期	类型	关键字
对应名称	DocID	Url	Title	Discription	Date	Type	Keywords
实例	0	http$##msn.sports.ynet.com#2.1.0#28778_3.html	精彩评论_体育_MSN中国	精彩的图集，深刻的解读，独特视角看奥运	null	msn	[中国, 军团, 北京, 奥运会, 姚明,]
类型	INT	varchar	varchar	text	varchar	varchar	varchar
范围		300B	300B	64KB	30B	30B	300B

文档 ID：主键

URL：候选键，与文档 ID 一一对应；

标题：用于结果排序、相似度计算和显示；

摘要：用于结果排序、相似度计算和显示；

日期：网页发布日期，用于结果排序、显示；

类型：门户网站类型，用于快照时根据类型进入原始网页库目录读取文件；

关键字：用于热词推荐、显示。

词项-词项 ID 映射表（TermsIndex）

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

将文档正文进行了分词，过滤停用词之后，将词项映射成 ID 。

文档索引表（DocsIndex）

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

TermIDs 属性的值是 TermIDs 的一个序列，用#分开。

倒排索引表（InvertedIndex）

属性名	词项 ID	文档 IDs
对应名称	TermID	DocumentIDs

实例	1	2|1:10:[1, 13]#2:10:[1, 13]#
类型	int	text
范围	0 - 4294967295	64KB

文档 IDs 格式：一个 termID 对应的一条 documentIDs 在数据库中的存储格式（其中位置信息取词项在文档索引中的 index 号）：

```
DF|docID1: tf-idf:[pos1,pos2...]#docID2: tf-idf:[pos1,pos2…]…
```

将每篇文档对每个词项的 tf-idf 权重预先计算并存储，可以在检索时加快计算文档得分速度。

2.2 构建方法

工具：Mysql， Navicat for MySQL

语言：SQL

生成表格：PagesIndex（网页信息表）、DocsIndex（文档索引表）、TermsInde（词项 ID-词项映射表）、InvertedIndex（倒排索引表）

创建 PagesIndex（网页信息表）

```
CREATE TABLE PagesIndex
(
        DocID INTEGER primary key,
        Url varchar(300),
        Title varchar(300),
        Description text,
        Date varchar(30),
        Type varchar(30),
        Keywords varchar(300)
)
```

创建 DocsIndex（文档索引表）

```
CREATE TABLE DocsIndex
(
        DocID   INTEGER primary key,
        TermIDs   text
```

```
)
```

创建 TermsIndex（词项 ID-词项映射表）

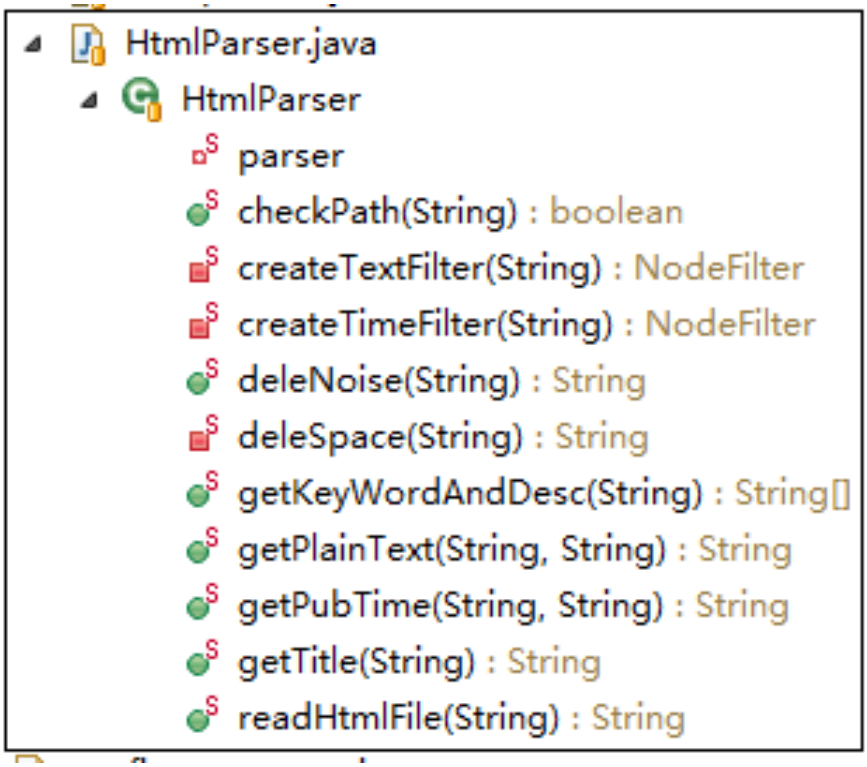
```
CREATE TABLE TermsIndex
(
        TermID    INTEGER primary key,
        Term    varchar(20)
)
```

创建 InvertedIndex（倒排索引表）

```
CREATE TABLE InvertedIndex
(
        TermID    INTEGER primary key,
        DocumentIDs    text
)
```

2.3 网页过滤

本项目的网页过滤部分采用开源第三方工具 htmlparser（http://htmlparser.sourceforge.net/），并结合了正则表达式过滤等方法，加以封装，以实现项目需求。封装后的代码结构如下：



该类主要实现了对网页信息的抽取，包括 title、description、keywords、public time 以及网页正文等。目前可实现对搜狐、网易、腾讯、MSN 四大门户网站的新闻网页过滤。其中四大门户网站网页中的 title、description 和 keywords 内容均可以通过相同的网页标签定位并获取。但是对于 public time，四大网站的标签均不同，并且对于同一个网站，其不同年份的 public time 标签也都不同，因此此处并未采用 htmlparser 来抽取，而是使用了正则表达式。

2.4 文本分词

项目中的分词模块使用了 Ansj（https://github.com/NLPchina/ansj_seg）第三方开源中文分词工具。Ansj 作者孙健，其实现是基于中科院的 ictclas 中文分词算法，比其他常用的开源分词工具（如 mmseg4j）的分词准确率更高。

分词方法

Ansj 主要分词方法有：基本分词（BaseAnalysis）、精准分词（ToAnalysis）、nlp 分词（NlpAnalysis）、面向索引的分词（IndexAnalysis）。四种方法的对比如下表：

	用户自定义词典	数字识别	人名识别	机构名识别	新词发现
基本分词	×	√	×	×	×
精准分词	√	√	√	×	×
NLP分词	√	√	√	√	√
面向索引分词	√	√	√	×	×

考虑到基本分词功能单一，因此不采用该方法。而对于面向索引分词，特点是充分考虑了歧义句的因素，比如对“主副食品”分词，能够产生[主副食品/n, 主副食, 副食, 副食品, 食品]五个分词结果，鉴于其分词结果会派生出很多新词，会极大增加倒排索引的规模，因此不在本项目总采用。

NLP 分词功能强大，能够支持新词发现功能。但是执行时间相对较长，并且非常消耗内存。而精准分词在易用性、稳定性、准确性、以及分词效率上都取得了一个不错的平衡，也是作者亲自推荐的方法（https://github.com/NLPchina/ansj_seg/issues/156）。

我们经过粗略测试，在同样的软硬件环境下，NLP 方法初始化时间约为 100 秒，而精准分词方法只需 10 秒左右来初始化。而在内存占用方面，NLP 方法相比于精准分词方法占用了极大部分内存，而且其分词时间也更久。

综合上述考虑，本项目中对于索引构建过程中对文本的分词采用精准分词方法，为了保持一致性，对于搜索语句的分词也采用该方法。

词典

Ansj 默认分词词典有 386211 个词项，共 6MB 左右。即便如此，该词典也只涵盖了首字母从 A-G 的常用词项，不过在实际体验中对分词结果的影响不大。

停用词词典需要自行添加，本项目添加的停用词有 1534 个，词典大小共 10.2KB。

3. 检索策略

Geeking 搜索引擎的检索过程从服务器端初始化开始，具体流程如下：

(1) 启动服务器，初始化搜索服务，包括初始化配置、数据库，加载词典、热词、和部分倒排索引表到内存；

(2) 用户输入查询词；

(3) 根据热词和历史搜索记录对查询词自动补全；

(4) 对查询词分词；

(5) 根据分词结果从词项映射表获取词项 ID；

(6) 根据词项 ID 从倒排索引表中获得各个词项 ID 对应的倒排文档集合；

(7) 如果查找不到相关文档，则从数据库中读取该词项 ID 对应的倒排文档集合；

(8) 若仍无相关文档，则执行相似单词查找，将推荐词返回给前端。

(9) 将文档集合按照其长度升序排列做合并操作；

(10) 根据 tf-idf 权重计算文档集合中每篇文档得分，并排序；

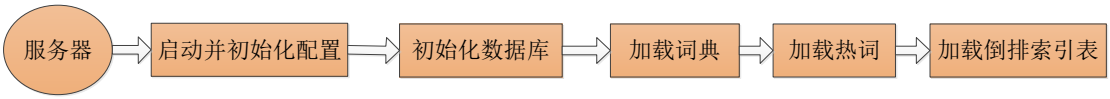
(11) 对排序后的相关文档集聚类，将聚类后的最终结果返回给前端；

(12) 前端显示搜索结果。

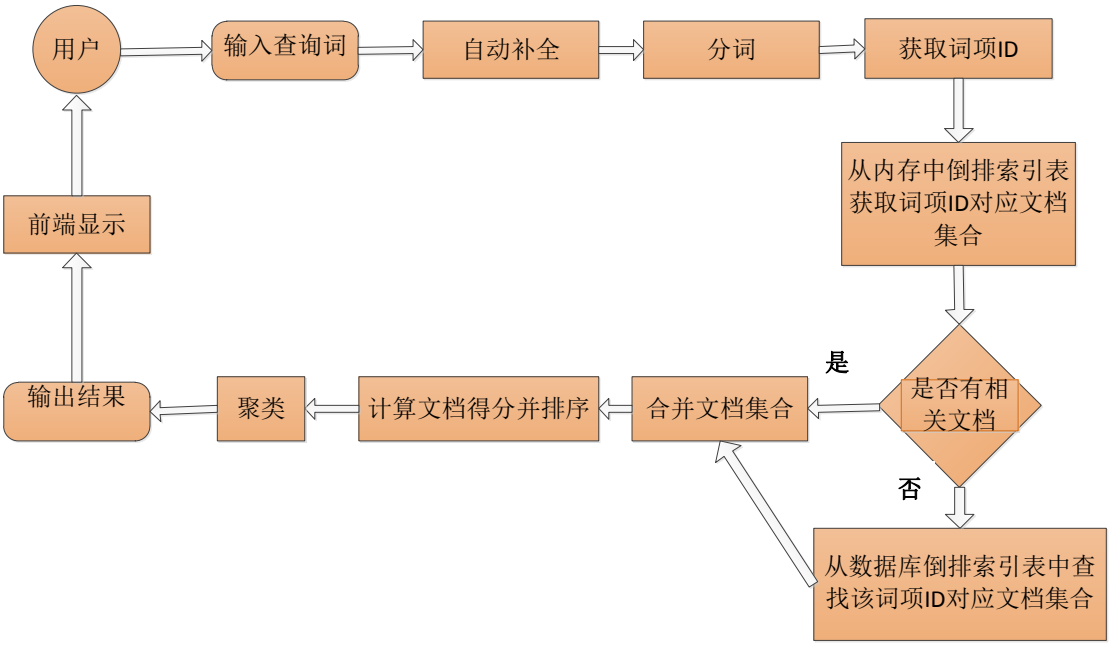
3.1 检索流程

用户检索的过程从服务器初始化之后开始，主要的步骤和流程如下所示：

服务器启动



检索流程



3.2 结果排序

胜者表

由于倒排索引中已经存储了每篇文档的 tf-idf 信息，因此，在搜索引起首次启动，加载倒排索引到内存的时候，程序会对每个词项的倒排索引按照和 tf-idf

值由高到底排序，并且截取出 TopK（在 configuration.properties 里可配置，默认为 50）篇文档。只有这 TopK 篇文档才可以参与最后的相似度运算。

```
public class InvertedIndex {
    private final long termID;
    private long docFreq = 0; //document frequency
    //该词项在各个文档中的统计信息
    private Map<Long, TermStat> statsMap = new TreeMap<Long, TermStat>();
    //降序排列的statsMap
    private Map<Long, TermStat> topKStatsMap = new TreeMap<>();
```

合并算法

在对每个搜索词项的相关文档集进行合并之前，程序会先按照相关文档的规模从小到大进行排序，然后再按照该顺序进行相关文档合并，从而减少不必要的合并计算。

```
/*按照检索词项的相关文档规模排序，便于merge*/
private List<Long> sortQueryIDs(List<Long> queryIDs)
```

相似度计算

本项目权重计算基于 **tf-idf**，并且充分考虑了体育新闻的**时效性**，计算了时间权重。此外，考虑到如果搜索词出现在某篇文档的**标题或者摘要**中，那么该文档的相关度应该更高。因此我们采取的相关度权重算法为：

$$score(q,d) = \sum_{t=1}^{q}\left(w_{t,q} * tf\_idf_{t,d} + 10 * count_{t,d}\right) + w(d, curMill)$$

其中，括号中前加号前半部分是 nnn.ntn 算法，而后半部分则是考虑到搜索词在“标题+摘要”中的权重。如果搜索词在标题和摘要中出现了$count_{t,d}$次，那么其权重将会增加$10 * count_{t,d}$。而具体是增加 10 分的权重或者是更多，有待日后不断调试，以获得最优方案。最后的求和的部分是时效性权重，计算公式为：

$$w(d, curMill) = (currMill \div (curMill - pubMill_d))^2$$

其中，$curMill$是指 epoch 时间，即是从 Epoch（1970 年 1 月 1 日 00:00:00 UTC）开始到现在所经过的秒数，而$pubMill_d$是指从 epoch 到网页发布那天所经历过的秒数。公式的意思是，如果网页发布时间距离用户搜索当天越近，那么权重分数将会更高，反之更低。

```
//计算权重
for (long term : queryIDs) {
    TermStat stat = invIdxMap.get(term).getStatsMap().get(doc.getKey());
    if (stat == null) {
        System.out.println("can not find doc stat in term: "+term);
    }
    //计算"标题+描述"中搜索词出现次数，1次weight+10,以及时间权重
    long titWeight = page.countInTitleDesc(queryTerms.get(term));

    //累计相似度结果：
    //weight = Σ{检索词项权重(1)*该文档权重(tf-idf)+标题中搜索词出现次数*10}
    doc.getValue().addWeight(stat.getTfIdf()+titWeight);
}
//计算并加入日期权重
doc.getValue().addWeight(page.countPubTimeWeight());
```

代码实现如上图，整个权重分数的算法充分考虑了 tf、idf、标题、发布时间这几个因素，在实际使用当中有很不错的效果。

3.3 结果聚类

3.3.1 对百度新闻搜索结果聚类的调研

调研过程

在百度上搜索“北京冬奥会”的新闻，显示前 3 条新闻结果如下图：

在每条新闻右下角，矩形红色框住的地方“*条相同新闻”就是百度搜索引擎对该新闻与之同类的其他新闻的聚类。点击第 1 条新闻的“35 条相同新闻”进去，

对这些同类新闻进行分析，查找其聚类依据，以及排序规律。将这些新闻的信息做成如下表格。序号 0 表示该显示新闻，序号 1-10 表示排序后隐藏的相同新闻。

表 1 百度新闻搜索结果相同新闻聚类分析

门户	title	date	keywords	description
网易	13亿中国人是北京申办冬奥坚强后盾(图)	2014-11-8 14:01	冬奥会	同标题
新浪	北京市长:13亿中国人是北京申办冬奥坚强后盾	2014-11-8 21:01	北京,冬奥会	同正文第1段
东方网	北京市市长:13亿中国人是北京申办冬奥坚强后盾	2014-11-8 17:25	冬奥会 北京市	同正文第1段
东方网	北京市市长:14亿中国人是北京申办冬奥坚强后盾	2014-11-8 16:38	冬奥会	同正文第1段
东方网	北京市市长:15亿中国人是北京申办冬奥坚强后盾	2014-11-8 16:25	冬奥会	同正文第1段
人民网	王安顺：13亿中国人是北京申办冬奥坚强后盾	2014-11-8 19:51	冬奥	同标题
中国经济网	王安顺：13亿中国人是北京申办冬奥坚强后盾	2014-11-8 19:51	冬奥会	同正文第1段
大众网	北京市市长:13亿中国人是北京申办冬奥坚强后盾	2014-11-8 16:21	冬奥会 北京市市长	同正文第1段
人民网体育	北京市市長:13億中國人是北京申辦冬奧堅強后盾	2014-11-8 16:16	冬奧會,北京市	同正文第1段
大河网	北京市市长:13亿中国人是北京申办冬奥坚强后盾	2014-11-8 16:23	冬奥会，北京市	同正文第1段
南报网	北京市市長:13億中國人是北京申辦冬奧堅強后盾	2014-11-8 20:33	北京	同标题
东方网	王安顺：13亿中国人是北京申办冬奥坚强后盾	2014-11-8 20:07	无	同正文第1段
东方网	北京市市长:13亿中国人是北京申办冬奥坚强后盾	2014-11-8 16:25	冬奥会	同正文第1段

调研结论

(1) 同类新闻之间标题很类似，不同类新闻间标题差异很大；

(2) 聚类依据是标题；

聚类过程与 keywords、description 无关，如上表序号 11 其 keywords 不包含与搜索词项相关的关键字，如表中 description 有的和标题相同，有的和正文第一段内容相同。只有标题是大同小异的，均包含和序号 0（即首条显示新闻）极其类似的内容。

(3) 同类新闻聚类后的相对顺序同其未聚类前的相对顺序。

同类新闻内的排序依据包括：门户、发布时间、关键字。如 10、11 的时间靠前，但是因为不含 idf 高的关键词“冬奥会”，故排在较后。这些排序依据和搜索引擎在首页显示不同栏目新闻的排序依据一致。

3.3.2 聚类算法

相似度定义及计算方法

根据调研结论(2)，相似度定义为标题的相似性。考虑到我们建立索引时候已经将标题分词，这里我们采用最长公共子序列衡量，子序列每个元素是一个词项，该计算方法与 k-gram 相比可取得同样的准确率却有更高的效率。

聚类过程

根据调研结论，由于同类新闻之间标题很类似，不同类新闻间标题差异很大，故可以通过调节合适的阈值，将一个类聚好后，再去聚下一个类。

输入： pageList，包含排好序的 n 篇文档；

输出： result，包含 m 个类 cluster；

(1) 按序遍历 pageList 取其第 k 篇网页 pageList[k]，构造空数组 claster，将 pageList[k]放入 claster，并从 pageList 中删除 pageList[k]；

(2) 按序遍历 pageList 取其第 i 篇网页 pageList[i]，计算 pageList[k]和 pageList[i]的标题相似度；

(3) 若相似度满足阈值要求，则将 pageList[i]从放到 claster 尾部，并从 pageList 删除 pageList[i]；

(4) 步(2)若未遍历完 pageList 则回到步(2)，若遍历完，则将 claster 作为一个元素放到 result 中；

(5) 步(1)若未遍历 pageList 完则回到步(1)，若遍历完则结束。

代码如下：

```
for (int k = 0; k < pageList.size(); k++) { //取第k篇文档
    cluster = new ArrayList<PageInfo>();        //第k篇文档放进新的数组
    page = (PageInfo)pageList.get(k).clone();
    cluster.add(page);
    for (int i = k+1; i < pageList.size(); i++) {       //遍历其他文档
        tmp = (PageInfo)pageList.get(i).clone();
        if (isSimilarPage(page, tmp)) {     //计算其他文档和该文档相似度
            cluster.add(tmp);                   //相似的文档放进新数组尾部
            pageList.remove(i--);           //并从原输入数组中删除
        }
    }
    result.add(cluster);                //新数组作为一个类添加到待输出的结果
}
```

计算相似度（即最长公共子序列）过程：

输入：title1，title2（2 篇文档标题）；

输出：matrix[m][n]（最长公共子序列长度）；

(1) 将标题 title1 分词，放进字符串数组 chars_title1，m 为 chars_title1 长度；

(2) 将标题 title1 分词，放进字符串数组 chars_title1，n 为 chars_title2 长度；

(3) 用动态规划方法求解矩阵矩阵 matrix[1...m][1...n]的值。

```
private static int longestCommonSubstring(String title1, String title2) {
    char[] chars_title1 = title1.toCharArray(); //标题转换为字符串数组
    char[] chars_title2 = title2.toCharArray();
    int m = chars_title1.length;                //标题1字符串数组长度
    int n = chars_title2.length;                //标题2字符串数组长度
    int[][] matrix = new int[m + 1][n + 1];
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (chars_title1[i - 1] == chars_title2[j - 1])
                matrix[i][j] = matrix[i - 1][j - 1] + 1;
            else
                matrix[i][j] = Math.max(matrix[i][j-1], matrix[i - 1][j]);
            }
        }
    return matrix[m][n];
}
```

时间复杂度

时间复杂度为 O(n^2)，n 为文档篇数。

由于每次计算文档相似性时采用标题的最长公共子序列衡量，时间复杂度的系数较小。假设标题的平均长度为 m，分词后的平均词项个数为 m/2，每次求最长公共子序列的时间为(m/2)*(m/2) = m2/4。又由于算法中每次聚类后的文档不参与其他新类的计算，实际时间复杂度没有 n2，时间复杂度和文档能聚成的类别数目有关，若结果聚成 c 个类时，时间复杂度为 O(c*n)。最好的情况是只聚成 1 个类时，时间复杂度仅为 n*m2/4。

4. 前端处理

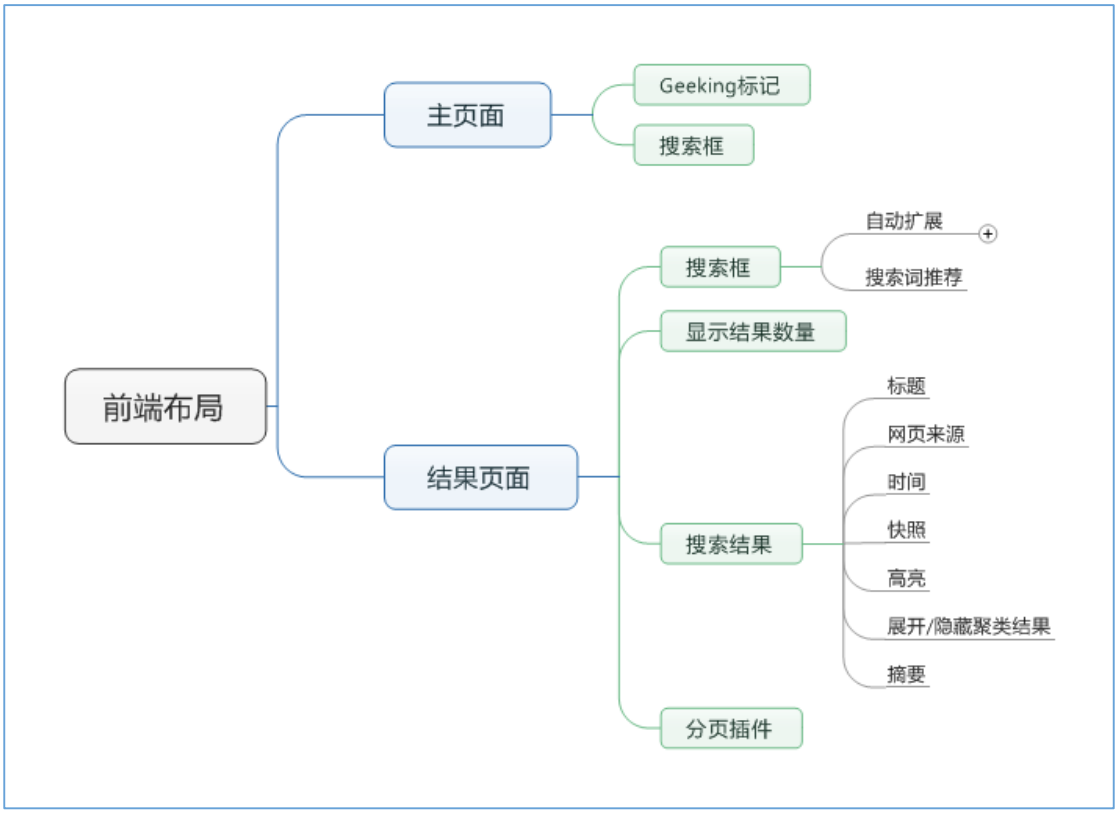
前端总体框架采用 Ajax+Jquery+JSP 的富客户端方式实现。使用 Javascript 绑定和处理所有数据，操作 Document Object Model 进行动态显示及交互。

AJAX 具有无刷新更新技术、异步与服务器通信、前端与后端负载平衡和界面与应用分离等优点，是一个强大灵活的 webservice 工具。

除了基本的显示搜索结果页面外，实现功能包括**无刷新分页、自动补齐、搜索词推荐、结果聚类、网页快照**等功能。功能完善，能带给用户良好的体验。

4.1 页面元素

前端布局：


前端布局
主页面
Geeking标记
搜索框
结果页面
搜索框
自动扩展
搜索词推荐
显示结果数量
搜索结果
标题
网页来源
时间
快照
高亮
展开/隐藏聚类结果
摘要
分页插件

主页面


logo
搜索一下
搜索框

搜索结果页面：

4.2 自动补齐

前端

当用户输入查询词的时候，会在输入框的下方面动态显示一个下拉框。 同时，Jquery 会监听用户在搜索框的每次输入，通过 Ajax 请求，从服务端获取查询结果，然后结果以 JSON 的数据类型返回，动态的显示在下拉框中。用户可以选择下拉框中的任意一个词，按回车键即可完成搜索。

后端

Server 端使用简单的 jsp，在初始化的时候使用关键词词库构建了一颗哈希树，接收到查询请求后，开始查询，将查询结果按照热度进行排序后，以 JSON 的数据格式返回。最多返回 5 个单词。

页面展示

4.3 搜索词推荐

前端

当发现某次搜索结果为空的时候，认为用户可能输入有误，会查找热词词典，返回最相似的最多三个单词，显示在搜索结果页面上，用户可以点击这些词进行搜索。分 3 情况显示：

(1) 有搜索结果时，不推荐热词；

(2) 无搜索结果时，并且能查找到相似热词，推荐热词；

(3) 无搜索结果时，并且不能查找到相似热词时，不推荐。

后台

为了将用户的输入和字典里的词相比较，找到相似度最高的单词，采用 **N-gram** 算法和 **LevensteinDistance 编辑距离**。

(1) 首先来说一个 N-Gram 的概念， N-Gram 是指将一个单词划分成若干等长的字串，每一个成为一个 gram，n 就是用来控制每个 gram 的长度的；

(2) 对于一个单词我们往往会给不同的 N 做多次切割，这样便于做搜索建议和拼写检查；

(3) 构建词典；

i. 对于每个词典中的单词，根据两个值 minN~maxN（minN, maxN 是根据单词长度由专门的函数产生的）进行分割，介于两个值之间的 N 都算作一种分割，伪码如下：

```
for ng = minN to maxN
        for i=0 to wordLen-ng +1
            gram = word.substr(i,i+ng);
             add this gram as a field to the current lucene document
        end for
end for
```

ii. 针对每个 N，开头结尾的 gram 也加入索引域。

举例来说：对于 three 这个词，minN = 2, maxN =3;那么针对 three 这个词

的对应 document 对象包含的 Field 有如下：

```
gram2:    th hr re ee
start2: th
end2: ee
gram3: thr hre ree
start3: thr
end3:ree
```

通过如上的方式将词典里的每个单词都进行了索引。

(4) 拼写检查；

i. 根据输入 word 构建查询条件；

也是通过 minN, maxN 来讲输入进行 gram 化，每个对应的分割都构成一个查询 Term，同时针对每种分割也有对应的起始和结束查询，过程和构建词典的过程一样。这些条件之间是或的关系。这个用 BooleanQuery 可以实现。

ii. 开始从词典构建的索引中使用上述条件进行查询。

a. 从查询结果中过滤掉自身（如果查询结果包含自身的话）；
b. 计算查询结果中的每个单词和输入 word 的编辑距离，过滤掉部分距离过小的单词；
c. 放入以编辑距离为衡量指标的优先队列中，队列长度人为确定，具体根据要查询的建议单词的多少确定；
d. 当队列长度满之后，返回结果，注意此处的结果要是以编辑距离增序的结果（使用优先队列的函数可以达到此要求）。

(5) 编辑距离解释。

编辑距离是衡量两个字符串相似度的值，介于 0 和 1 之间，0 表示两个字符串完全相同，1 表示两个字符串具有最大的不相似程度。常用的编辑距离有：LevensteinDistance 和 JaroWinklerDistance. 这里使用 Levenshtein 算法中，两个字符串之间的距离定 义为将一个字符串转换为另一字符串所需的最少编辑次数，允许的编辑操作有插入、删除、单个字符的替换。该算法由 Vladimir Levenshtein 在 1965 年提出，并以作者名来命名。

效果展示：

4.4 摘要快照及高亮

前端

摘要目前是静态摘要，将新闻的导语作为摘要显示出来。并且查找到关键词，进行高亮显示，主要使用 jquery 动态添加标签，实现了高亮效果。

为了防止摘要过长，自动截取一部分显示。向后台发起 Ajax 请求，返回分词时候的 query 列表，使用 jquery 插件，将 title 和正文中的 query 都高亮显示。

快照则是在“Geeking 快照”中加入一个超链接，指向缓存的网页所在的地址。

后端

将 query 的结果分词后，接受客户端的 Ajax 请求，将分词后的列表返回给前端。

效果展示

三．测试与评估

本项目测试主要分为两个部分，功能测试和性能测试。其中功能测试包括了黑盒、白盒及回归测试，涵盖了项目开发过程中主要模块的功能测试以及 bug 修复统计。性能测试包括了索引构建、服务启动和检索过程中分词、排序、聚类等每一步的时间开销。

1. 测试环境

硬件环境：

CPU：Intel Core2 Duo 2.2GHz

Memory：3.0GB DDR2

硬盘：机械硬盘

软件环境：

操作系统：Windows 8.1 Enterprise

服务器：Apache Tomcat 7.0.55

编译器：Eclipse 4.3 Kepller

数据库：Mysql 5.1

JDK：1.8.0

语料集：

来源：网易体育，腾讯体育，搜狐体育，MSN Sports

网页数目：

搜狐	网易	腾讯	MSN	合计
45831	15669	37622	5791	104913

2. 测试结果

2.1 功能测试

此处功能测试并不基于上述测试环境，而是整个系统开发过程中主要功能基本实现之后，进行功能测试时所遇到的 bug 和修复统计，具体如下表：

项目名称		Geeking 搜索引擎	**测试方法**	黑盒,白盒,回归	
模块开发者		高妍、林裕杰	**程序版本号**	0.2	
功能/模块	**序号**	**功能点**	**测试用例数**	**发现 bug 数量**	**修复 Bug 数量**
Web UI 与交互模块	1	自动补齐功能	20	10	10
	2	检索词推荐功能	20	15	15
	3	检索结果聚类	20	12	12
	4	整体布局调整	20	10	9
	5	关键词高亮	25	10	10
	6	无刷新分页	25	5	5
	7	快照功能	25	2	2
	8	聚类结果展开/隐藏	25	12	12
SpellCheck 模块	1	N-gram 词典构建	25	12	12
	2	LevensteinDistance 编辑距离。	25	2	2
	3	关键词推荐处理逻辑	25	12	12
	高妍	**日期**	2014/12/9		
自测执行者		陈新荃	**日期**	2014/12/9	

2.2 性能测试

基于上述测试环境，我们利用大型语料集，主要进行了索引构建、用户检索等性能测试。

索引构建：

索引构建所用时间：

构建内容	花费时间(分钟)
网页预处理（网页抽取过滤, 网页信息表, 正向索引表）	130.68
词项ID映射表构建	109.83
倒排索引构建	176.35
总时间	416.86

构建完成后的数据库信息：

数据库表	记录条数
网页信息表	104828
正向索引表	61725
词项ID映射表	291402
倒排索引表	283587

分析：

索引构建采用单机、单线程构建，在大型语料集下，性能较低。除此之外，由于构建索引所用计算机是五年前出厂的笔记本电脑，配置较低，也是影响索引构建速度的因素。

检索服务：

在小规模的语料集下，词项 ID 映射表和倒排索引均全部载入内存，因此检索速度基本在 1 秒之内，对此我们不再进行测试分析。以下测试基于上述大型语料集。

以用户输入查询词“曲棍球”为例，返回 50 篇相关文档。整个检索过程所耗费的时间如下：

检索流程	所用时间(ms)	百分比
1. 分词并映射到词项ID	373	0.50%
2. 查找相关文档	5143	7.70%
3. 获取网页信息	61459	91.71%
4. 计算相关度	10	0.02%
5. 排序	9	0.02%
6. 聚类	22	0.05%
合计	67016	100%

分析：

可以看到，执行一次检索总共需要花费 67 秒的时间，这是令用户无法接受的。

性能低的原因在于词项 ID 映射表和倒排索引表规模较大，无法一次性载入内存，而程序的检索策略是如果在内存中找不到对应的词项 ID 或者倒排索引，就必须从数据库中查找，这涉及到了数据库的查找操作，同时伴随着磁盘(机械盘)的读写，而**数据库查询操作是影响性能的最主要因素**。

而整个检索过程花费时间最多的步骤在于前面三步。其中，第 1 步主要涉及到词项 ID 映射表的查询和读取；第 2 步主要涉及到倒排索引表的查询和读取；第 3 步主要涉及到网页信息表的查询和读取。值得一提的是，在查询词只有一个的条件下，第 1 步和第 2 步只需要对数据库执行 1 次查询读取操作。然而对于第 3 步，因为相关文档有 50 篇，因此需要对数据库的网页信息表执行 50 次的查询读取操作。

又因为本项目采用的是 mysql 数据库，在较低的硬件环境下其查询性能非常耗费时间。

因此综合上述原因，在大规模语料集下，目前本项目的检索性能并不理想。

四．创新点

1. 相似度计算

除了在结果排序过程中使用到了胜者表以及合并算法进行优化之外，本项目还对相似度分数计算进行了创新，充分考虑了四种因素：tf、idf、标题、发布时间。

$$score(q,d) = \sum_{t=1}^{q} \left(w_{t,q} * tf\_idf_{t,d} + 10 * count_{t,d} \right) + w(d, curMill)$$

如果查询词在标题中出现的次数越多，那么权重分数将会更高；如果网页发布时间距离用户搜索当天越近，那么权重分数将会更高。

目前来看，基于本项目语料集，该相关度评分算法效果非常不错。

2. 中间字自动补全

传统的搜索引擎的自动补全功能是通过字典树实现的，只能按照前缀匹配，我们的自动补全功能实现了查询词的任意段的匹配，具体原理就是将搜索词按照热度排序，维护一个 hashmap，监听搜索框的输入，对于用户实时输入的 query 遍历 hashmap 词典，查找是否包含这个词，选择包含用户的 query，并且热度最高的几个词，返回给前端，显示出来。这样，可以减少用户输入查询词的时间，提高用户体验。

但是，这种方式只适合小规模的 hashmap 词典，一旦词典变大，查询效率很低，不能做到实时响应，会影响到用户体验，后续还要继续改进。

3. 网页聚类算法

新闻聚类中，根据对百度搜索新闻聚类调研得到的结论，我们采用 2 个标题字符串的最长公共子序列的值衡量 2 篇网页的相似度，该方法能使聚类时间复杂度 $O(n^2)$的常数因子很小，为 $m^2/4$（m 为单个标题分词后的词项个数）；并且，根据同类新闻相似度大而不同类新闻间相似度很小的实际情况，我们通过预设阈值，先将一个类聚好后，再去聚下一个类。与普通的单连接算法相比，该算法在满足同样聚类准确率的前提下，能显著提高效率，若结果聚成 c 个类时，时间复杂度为 O(c*n)。

五．总结

1. 项目总结

Geeking 项目开始于 2014 年 9 月 23 日(第一次会议)，并于 12 月 14 日如期完成。

本项目并未使用过多第三方工具，核心功能均由项目组成员自主实现。并且

在项目执行过程中，成员之间有着良好的沟通与思想碰撞，产生了许多创新的思想，并将其中的一部分在项目中实现出来，比如文档和检索词计算的公式、中间字的自动补全、根据标题聚类的算法等等。

不过，在进行大型语料集的索引构建和检索测试时，本项目表现出的性能并不理想。这主要是由于大型的倒排索引无法载入内存，以及 mysql 数据库系统的查询性能低导致的。

因此，本项目后续还可以进行如下方面的改进：

(1) 考虑利用 Hadoop 的 MapReduce 编程思想进行倒排索引的构建；

(2) 考虑对倒排索引进行压缩，使其可一次性载入内存；

(3) 考虑使用 keyValue 数据库（比如 Redis），提高数据库查询性能；

(4) 考虑引入 MVC 开发框架等；

GeeK 项目组团队成员来自两个不同学院，在这几个月的期间里，虽然大家都学业繁忙，但是所有人都积极参与，发挥各自所长，使得项目进展有条不紊，最后如期完成了任务。感谢每个人的努力与付出！

2. 个人总结

陈新荃：

第一次像模像样的带领一个团队完成一个项目。花了很多时间和精力在这个项目上，但是获益良多。经历过这样一个项目，除了信息检索相关的知识得到巩固和加强之外，在编程能力上也有了提高，代码的健壮性和可读性方面得到了加强。

深深感受到了项目开发过程的复杂，除了要理清思路把握项目整体的方向和进展，还需要考虑成员之间的分工合作。大到制定开发计划，小到考虑某段代码的开销等等，都需要清晰地思路和周全的考虑。这些对我来说都是历练。在此非常感谢团队成员的积极配合，项目能够圆满完成是大家齐心协力的结果，能与大家一起共事我感到很幸运。

高妍：

通过这次信息检索大作业，我对搜索引擎有了更深一步的理解，增强了编程能力，对于算法效率、可用性、可扩展性有了进一步的理解。掌握了 JSP、Jquery、Ajax、Java 等多项技术，深入理解了 N-gram 算法和 LevensteinDistance 编辑距离。其中 spellcheck 模块是通过阅读 Lucene 源码给了我跟多启发，在此基础上完成

的。更重要的是，在团队合作开发的过程中，积累了团队合作经验，学会使用github，特别感谢组长陈新荃和林裕杰、肖卡飞两位队友，我们的队伍里没有大神，大家都是一起努力的菜鸟，在整个开发过程中，面对开发量大，难度高，缺乏经验这些困难，我们相互帮助，相互支持，相互鼓励，都一一克服了。在大家的共同努力下，我们很好的完成了任务。

林裕杰：

首先我要感谢老师给予了我们做这个项目的机会，不仅让我对信息检索过程有了更清晰的认识，还让我学习到了调研分析问题的方法。比如网页聚类的问题，根据对百度搜索新闻聚类调研的实际情况分析，得到有助于简化问题的结论，针对具体聚类对象，得到有效的能够提升聚类效率的算法。

其次我要感谢为我们项目作出许多贡献的组长陈新荃，是组长的聪明和才干让我们的团队能够做好这个项目，以及高妍、肖卡飞两位队友，是他们让这个项目变得富有意义，让我在这个项目过程中受益良多，他们都是我的学习榜样。

肖卡飞：

这是我第一份比较正式地和团队完成的计算机 project。通过一学期的有条不紊地计划开展，分工协作，具实践，使我加深了对本课程理论的理解，对开发搜索引擎整套流程的掌握和细节的了解。我编程能力不强从前对 java 更是一无所知，在本项目中，我主要负责爬虫等模块。学会了 java 的基本知识，提高了编程能力，和爬虫的工作原理和对开源代码的使用和借鉴。

还有就是很高兴更认识我的队友们，在我求组无望的情况下，如今的 Geek 团队收留了我这只菜鸟。组长的专业、认真、负责让我敬佩，高妍的能力、水平，裕杰的态度、努力都让我深深感动，都值得我好好学习。他们是我的榜样，很感谢这次项目让我认识我的朋友们。

—— GeeK 团队

2014.12.15