
自己动手写搜索引擎

——建一个自己的 Google

罗刚

2009

自己动手写搜索引擎.....	1
第 1 章 了解搜索引擎.....	1
1.1 Google 神话.....	1
1.2 体验搜索引擎.....	1
1.3 你也可以做搜索引擎.....	4
1.4 本章小结.....	4
第 2 章 遍历搜索引擎技术.....	5
2.1 30 分钟实现的搜索引擎.....	5
2.1.1 准备工作环境（10 分钟）.....	5
2.1.2 编写代码（15 分钟）.....	6
2.1.3 发布运行（5 分钟）.....	9
2.2 搜索引擎基本技术.....	14
2.2.1 网络蜘蛛.....	14
2.2.2 全文索引结构.....	14
2.2.3 Lucene 全文检索引擎.....	15
2.2.4 Nutch 网络搜索软件.....	15
2.2.5 用户界面.....	17
2.3 商业搜索引擎技术介绍.....	17
2.3.1 通用搜索.....	17
2.3.2 垂直搜索.....	18
2.3.3 站内搜索.....	19
2.3.4 桌面搜索.....	21
2.4 本章小结.....	21
第 3 章 获得海量数据.....	22
3.1 自己的网络蜘蛛.....	22
3.1.1 BerkeleyDB 介绍.....	27
3.1.2 抓取网页.....	28
3.1.3 MP3 抓取.....	29
3.1.4 RSS 抓取.....	30
3.1.5 图片抓取.....	33
3.1.6 垂直行业抓取.....	34
3.2 抓取数据库中的内容.....	36

3.2.1 建立数据视图.....	36
3.2.2 JDBC 数据库连接.....	36
3.2.3 增量抓取.....	38
3.3 抓取本地硬盘上的文件.....	38
3.3.1 目录遍历.....	38
3.4 本章小结.....	40
第 4 章 提取文档中的文本内容.....	41
4.1 从 HTML 文件中提取文本	41
4.1.1 HtmlParser 介绍.....	49
4.1.2 结构化信息提取.....	52
4.1.3 网页去噪.....	58
4.1.4 网页结构相似度计算.....	61
4.1.5 正文提取的工具 FireBug.....	62
4.1.6 正文提取的工具 NekoHTML.....	64
4.1.7 正文提取.....	66
4.2 从非 HTML 文件中提取文本	77
4.2.1 TEXT 文件.....	77
4.2.2 PDF 文件	77
4.2.3 Word 文件	85
4.2.4 Rtf 文件.....	86
4.2.5 Excel 文件.....	87
4.2.6 PowerPoint 文件	88
4.3 流媒体内容提取.....	89
4.3.1 音频流内容提取.....	89
4.3.2 视频流内容提取.....	91
4.4 抓取限制应对方法.....	93
4.5 本章小结.....	94
第 5 章 自然语言处理.....	95
5.1 中文分词处理.....	95
5.1.1 Lucene 中的中文分词	95
5.1.2 Lietu 中文分词的使用	96
5.1.3 中文分词的原理.....	96
5.1.4 查找词典算法.....	99
5.1.5 最大概率分词方法.....	102
5.1.6 新词发现.....	105

5.1.7 隐马尔可夫模型.....	106
5.2 语法解析树.....	108
5.3 文档排重.....	108
5.4 中文关键词提取.....	110
5.4.1 关键词提取的基本方法.....	110
5.4.2 关键词提取的设计.....	111
5.4.3 从网页提取关键词.....	111
5.5 相关搜索.....	111
5.6 拼写检查.....	114
5.6.1 英文拼写检查.....	114
5.6.2 中文拼写检查.....	115
5.7 自动摘要.....	120
5.7.1 自动摘要技术.....	120
5.7.2 自动摘要的设计.....	120
5.7.3 Lucene 中的动态摘要.....	128
5.8 自动分类.....	128
5.8.1 Classifier4J.....	129
5.8.2 自动分类的接口定义.....	131
5.8.3 自动分类的 SVM 方法实现.....	132
5.8.4 多级分类.....	132
5.9 自动聚类.....	134
5.9.1 聚类的定义.....	134
5.9.2 K 均值聚类方法.....	135
5.9.3 K 均值实现.....	137
5.10 拼音转换.....	142
5.11 语义搜索.....	143
5.12 跨语言搜索.....	147
5.13 本章小结.....	148
第 6 章 创建索引库.....	149
6.1 设计索引库结构.....	150
6.1.1 理解 Lucene 的索引库结构.....	150
6.1.2 设计一个简单的索引库.....	152
6.2 创建和维护索引库.....	153

6.2.1 创建索引库.....	153
6.2.2 向索引库中添加索引文档.....	153
6.2.3 删除索引库中的索引文档.....	155
6.2.4 更新索引库中的索引文档.....	155
6.2.5 索引的合并.....	155
6.2.6 索引的定时更新.....	156
6.2.7 索引的备份和恢复.....	157
6.2.8 修复索引.....	158
6.3 读写并发控制.....	158
6.4 优化使用 Lucene.....	159
6.4.1 索引优化.....	159
6.4.2 查询优化.....	161
6.4.3 实现时间加权排序.....	166
6.4.4 实现字词混合索引.....	167
6.4.5 定制 Similarity.....	173
6.4.6 定制 Tokenizer.....	174
6.5 查询大容量索引.....	176
6.6 本章小结.....	177
第 7 章 用户界面设计与实现.....	178
7.1 Lucene 搜索接口(search 代码).....	178
7.2 搜索页面设计.....	179
7.2.1 用于显示搜索结果的 taglib.....	179
7.2.2 用于搜索结果分页的 taglib.....	181
7.2.3 设计一个简单的搜索页面.....	183
7.3 实现搜索接口.....	186
7.3.1 布尔搜索.....	186
7.3.2 指定范围搜索.....	186
7.3.3 设置过滤条件.....	错误!未定义书签。
7.3.4 搜索结果排序.....	191
7.3.5 搜索页面的索引缓存与更新.....	192
7.4 实现关键词高亮显示.....	194
7.5 实现多维视图.....	196
7.6 实现相似文档搜索.....	202
7.7 实现 AJAX 自动完成.....	205
7.7.1 总体结构.....	206
7.7.2 服务器端处理.....	206
7.7.3 浏览器端处理.....	208
7.7.4 服务器端改进.....	209
7.7.5 部署总结.....	219

7.8 jQuery 实现的自动完成.....	219
7.9 集成其他功能.....	225
7.9.1 拼写检查.....	225
7.9.2 分类统计.....	226
7.9.3 相关搜索.....	226
7.9.4 再次查找.....	229
7.9.5 搜索日志.....	229
7.10 搜索日志分析.....	231
7.11 本章小结.....	234
第 8 章 其他高级主题.....	235
8.1 使用 Solr 实现分布式搜索.....	235
8.1.1 Solr 服务器端的配置与中文支持.....	235
8.1.2 把数据放进 Solr.....	240
8.1.3 删除数据.....	243
8.1.4 客户端搜索界面.....	244
8.1.5 Solr 索引库的查找.....	245
8.1.6 索引分发.....	249
8.1.7 Solr 搜索优化.....	249
8.1.8 Solr 中字词混合索引.....	252
8.1.9 相关检索.....	256
8.1.10 搜索结果去重.....	258
8.1.11 分布式搜索.....	263
8.1.12 SolrJ 查询分析器.....	267
8.1.13 扩展 SolrJ.....	279
8.1.14 扩展 Solr.....	280
8.1.15 Solr 的 .net 客户端.....	288
8.1.16 Solr 的 php 客户端.....	289
8.2 图片搜索.....	294
8.2.1 图像的 OCR 识别.....	295
8.3 竞价排名.....	299
8.4 Web 图分析.....	300
8.5 使用并行程序分析数据.....	305
8.6 RSS 搜索.....	306
8.7 本章小结.....	307
参考资源.....	308
书籍.....	308
网址.....	308
本书中的章节和代码对照表.....	309

第1章 了解搜索引擎

1.1 Google 神话

1995 年，两个年轻的学生 Larry Page 和 Sergey Brin 在一点上达成了共识——从大量数据检索信息是计算系统面临的最大的挑战之一。1996 年，他们创建了一个叫做 BackRub 的搜索引擎。这个搜索引擎后来叫做 Google。1998 年，Page 和 Brin 在 Larry 的大学宿舍创立 Google 公司的第一个数据中心。2000 年，Google 开始成为全球最大的搜索引擎一直到现在。2005 年，Google 股票市值超过 1000 亿美元。公司首次登陆华尔街时，“让世界更美好”便是他们阐明的目标之一。

1.2 体验搜索引擎

你想起什么大脑没有记住的知识了吗？用搜索引擎吧。它往往不会让你失望。

事实上这正是 Google 的创始人设想的。“如果你想搜就能搜，几乎像拥有第二个大脑，那就妙极了，”他说。就在 Brin（一直是两人中站在前台的那个）说在兴头上的时候，Page 在不声不响地稍稍露面之后，便带着“共谋者般的微笑”溜出了房间。

但是不要感到 Google 已经完成所有的事情，没有什么事情留下来给我们做了。下面是搜索“建国门饭店 电话”的结果。显然，你很清楚我要寻找的信息可能是建国门饭店的电话。下面是 Google 返回的前两条结果：



网页 图片 资讯 论坛 网页目录 更多»

建国门饭店 电话

搜索 高级搜索 使用偏好

☒ 搜索所有网页 ☐ 搜索所有中文网页 ☐ 搜索简体中文网页

网页 约有 342,000 项符合

建议： 无需单击“搜索”，按回车键可节省时间。

[北京站建国门崇文门同仁医院地区附近酒店/北京超低价经济型酒店预订 ...](#)
市内均装配中央空调、程控电话、彩色电视、卫生间等设备；饭店内还设有商品部、酒吧 间、美容美发、商务中心，可随时向客人提供电传、传真、打字、复印、订票等服务项目。 建国门饭店拓展的《高尔乔》巴西烤肉，让客人足不出店就可以品尝到鲜嫩可口的异国 ...
www.cbts.cn/100hotel/100bjhotel_beijingzhan.asp - 44k - 网页快照 - 类似网页

[大陆世纪卡成员宾馆-北京市](#)
门市价：标房1050元宝辰饭店★★★★地址：建国门内大街甲18号电话：010-65266688 优惠价：标房494元（参考价） 门市价：标房1012元西苑 ... 门市价：标间764元、豪华 间1062元国贸饭店★★★★地址：大北窑建国门外大街1号电话：010-65052277 ...
www.dalu.com/ad/vip/beij.html - 46k - 网页快照 - 类似网页

即使把"建国门饭店"打上引号，用短语的方式搜索，效果也是差强人意。

那让我们尝试做一个自己的搜索引擎吧。是不是更好用，看结果才知道。

专业的搜索引擎一般都会实现一个搜索语法，这个百度叫做高级搜索：

- 把搜索范围限定在网页标题中——intitle

网页标题通常是对网页内容提纲挈领式的归纳。把查询内容范围限定在网页标题中，有时能获得良好的效果。使用的方式，是把查询内容中，特别关键的部分，用“intitle:”领起来。

例如，找林青霞的写真，就可以这样查询：写真 intitle:林青霞

注意，intitle:和后面的关键词之间，不要有空格。

- 把搜索范围限定在特定站点中——site

有时候，您如果知道某个站点中有自己需要找的东西，就可以把搜索范围限定在这个站点中，提高查询效率。使用的方式，是在查询内容的后面，加上“site:站点域名”。

例如，要从天空网下载软件查找 msn 聊天工具，就可以这样查询：msn site:skycn.com

注意，“site:”后面跟的站点域名，不要带“http://”；另外，site:和站点名之间，不要带空格。

Site 语法的另外一个用处是查看一个网站被搜索引擎收录的情况，例如通过 psite:search.rayli.com.cn 可以看出 Google 中收录了 26,800 条瑞丽搜索的信息：



这些信息对于搜索引擎优化 SEO 是有参考价值的。

- 把搜索范围限定在 url 链接中——inurl

网页 url 中的某些信息，常常有某种有价值的含义。于是，您如果对搜索结果的 url 做某种限定，就可以获得良好的效果。实现的方式，是用“inurl:”，后跟需要在 url 中出现的关键词。

例如，找关于 photoshop 的使用技巧，可以这样查询：photoshop inurl:jqiao

上面这个查询串中的“photoshop”，是可以出现在网页的任何位置，而“jqiao”则必须出现在网页 url 中。

注意，inurl:语法和后面所跟的关键词，不要有空格。

精确匹配——双引号和书名号

如果输入的查询词很长，百度在经过分析后，给出的搜索结果中的查询词，可能是拆分的。如果您对这种情况不满意，可以尝试让百度不拆分查询词。给查询词加上双引号，就可以达到这种效果。

例如，搜索 上海科技大学，如果不加双引号，搜索结果被拆分，效果不是很好，但加上双引号后，“上海科技大学”，获得的结果就全是符合要求的了。

书名号是百度独有的一个特殊查询语法。在其他搜索引擎中，书名号会被忽略，而在百度，中文书名号是可被查询的。加上书名号的查询词，有两层特殊功能，一是书名号会出现在搜索结果中；二是被书名号扩起来的内容，不会被拆分。书名号在某些情况下特别有效果，例如，查名字很通俗和常用的那些电影或者小说。比如，查电影“手机”，如果不加书名号，很多情况下出来的是通讯工具——手机，而加上书名号后，《手机》结果就都是关于电影方面的了。

要求搜索结果中不含特定查询词

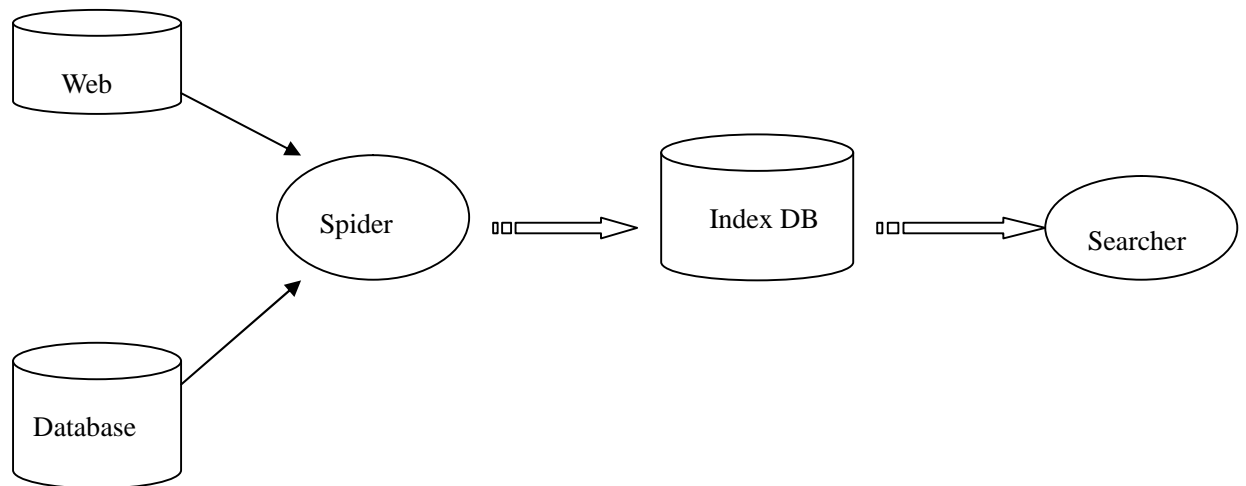
如果您发现搜索结果中，有某一类网页是您不希望看见的，而且，这些网页都包含特定的关键词，那么用减号语法，就可以去除所有这些含有特定关键词的网页。

例如，搜 神雕侠侣，希望是关于武侠小说方面的内容，却发现很多关于电视剧方面的网页。那么就可以这样查询：神雕侠侣 -电视剧

注意，前一个关键词，和减号之间必须有空格，否则，减号会被当成连字符处理，而失去减号语法功能。减号和后一个关键词之间，有无空格均可。

1.3 你也可以做搜索引擎

一个最简单的搜索引擎由搜索和抓取两部分组成：



数据来源可以是 Web 或者数据库等，也可以是本地路径等。

1.4 本章小结

第2章 遍历搜索引擎技术

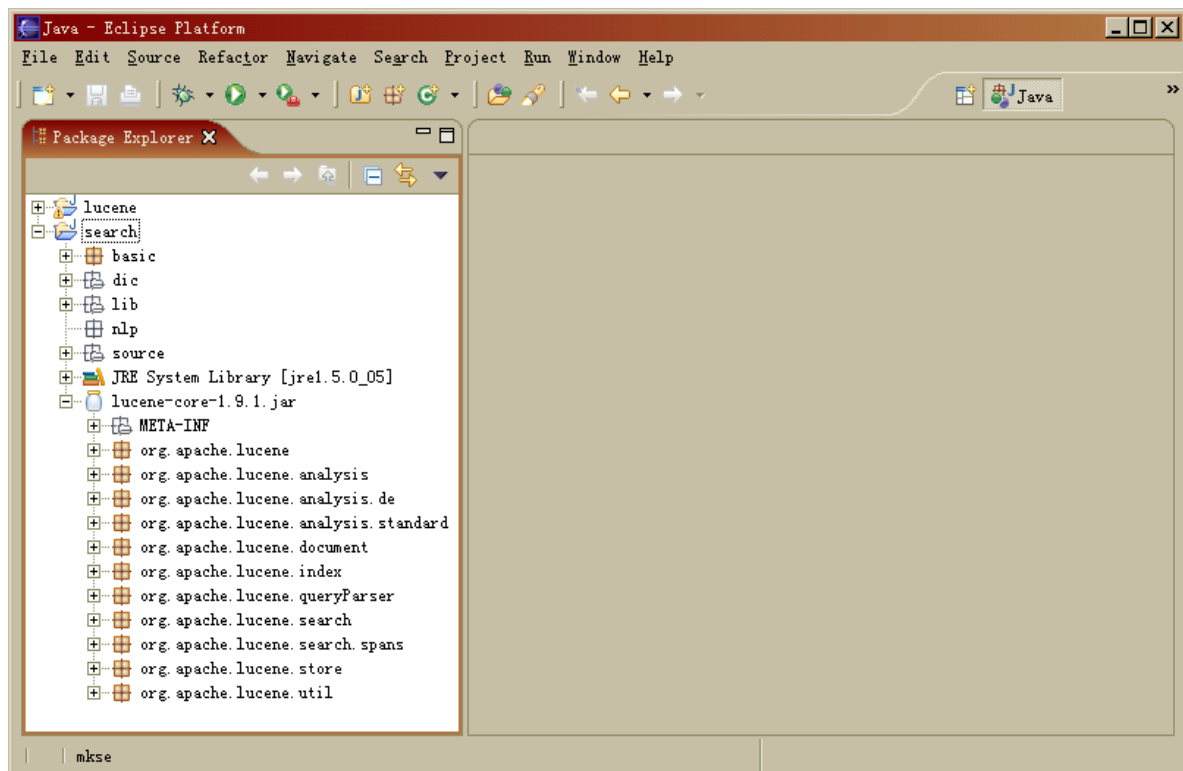
2.1 30 分钟实现的搜索引擎

我们从一个简单的搜索引擎入手，实现一个简单的指定目录文件的搜索引擎。实现之前需要有 java 开发方面的基础知识。

2.1.1 准备工作环境（10 分钟）

首先要准备一个 Java 的开发环境。当前可以使用 JDK1.6。JDK1.6 可以从 Sun 的官方网站 java.sun.com 下载得到。使用缺省方式安装即可。

然后要使用的是 Lucene 全文检索包。当前可以从 <http://lucene.apache.org/java/docs/index.html> 下载到最新的 Lucene，当前的版本是 2.3。另外，我们使用的集成开发环境是 Eclipse，下面是开发界面。



（注：制作过程中所用的程序在所赠光盘中都能找到）

如果需要用 Web 界面搜索，还要下载 Tomcat，当前可以从 <http://tomcat.apache.org/> 下载到，推荐使用 Tomcat5.5 以上的版本。

如果是在 linux 下，修改 ./catalina.sh 增加行 `JAVA_OPTS=-Xmx600m`

如果是在 windows 下，修改 ./catalina.bat 增加行 `set JAVA_OPTS=-Xmx600m`

对 catalina.bat 进行的修改，用 startup.bat 时可以生效起作用，但用 service.bat 时不起作用。

2.1.2 编写代码（15 分钟）

搜索引擎的基础在于对全文索引库的管理，在 Lucene 中，通过 IndexWriter 来写入索引库。下面是简单的代码：

```
index = new IndexWriter(new File(indexDir), new CnAnalyzer(),!incremental);
```

//说明 IndexWriter 是将要搜索的内容写入索引库的类。

```
public void go() throws Exception {

    long start = System.currentTimeMillis();

    // 创建索引目录或者建立增量索引

    if (verbose) {

        System.out.println("Creating index in: " + indexDir);

        if (incremental) System.out.println("    - using incremental mode");

    }

    index = new IndexWriter(new File(indexDir), new StandardAnalyzer(),

                            !incremental);//打开或创建索引库

    File dir = new File(sSourceDir);

    indexDir(dir);//索引路径
```

```
index.optimize();//索引优化

index.close();//关闭索引库

if(verbose)

    System.out.println("index complete in :"+(System.currentTimeMillis() - start)/1000);

}
```

实际把文件内容加到索引库是靠下面这段代码:

```
private void indexFile(File item) {

    if (verbose) System.out.println("Adding FILE: " + item);

    News news = loadFile(item);//把文件中的内容加载到 news 对象

    if (news!= null && news.body != null) {

        Document doc = new Document();

        //创建网址列

        Field f = new Field("url", news.URL ,

            Field.Store.YES, Field.Index.UN_TOKENIZED,

            Field.TermVector.NO);

        doc.add(f);

        //创建标题列

        f = new Field("title", news.title ,

            Field.Store.YES, Field.Index.TOKENIZED,

            Field.TermVector.WITH_POSITIONS_OFFSETS);
```

```
doc.add(f);

//创建内容列

f = new Field("body", news.body.toString() ,

            Field.Store.YES, Field.Index.TOKENIZED,

            Field.TermVector.WITH_POSITIONS_OFFSETS);

doc.add(f);


try{

    //文档增加到索引库

    index.addDocument(doc);

}

catch(Exception e)

{

    e.printStackTrace();

    System.exit(-1);

}

}
```

完整的代码可以在本书附带的光盘找到。

运行以后，一般会生成三个索引文件，例如：

_0.cfs

segments.gen

segments_2

其中任何索引库都会包括的一个文件是：segments.gen。我们可以通过判断一个路径下是否包括这个文件来判断一个路径下是否已经存在 Lucene 索引。

2.1.3 发布运行（5 分钟）

把搜索页面 index.jsp 放到 Tomcat。

```
<% if (!query.equals("")) { %>

<jsp:useBean id="search" class="com.bitmechanic.spindle.Search" scope="application">

<!-- Specify the directory that stores our Lucene index created by the spindle spider -->

<% search.init("d:/index"); %>

</jsp:useBean>

<jsp:setProperty name="search" property="query"/>

<list:init name="customers" listCreator="search" max="10">

<hr>

<list:hasResults>

    <list:iterate>

        <TABLE width="65%">

            <TR>

                <TD>

                    <a href="<list:iterateProp property="url"/>">

                        <B><FONT style="FONT-SIZE:
14px"><list:iterateProp property="title"/></FONT></B></a>&nbsp;&nbsp;&nbsp;<FONT size=-1
color=#6f6f6f>(<list:iterateProp property="source"/>)&nbsp;&nbsp;&nbsp;<list:iterateProp
property="accessDate"/></FONT>
```

```

        </TD>

    </TR>

    <TR>

        <TD>

            <FONT
                size=-1><list:iterateProp
property="desc"/></FONT>

        </TD>

    </TR>

</TABLE>

</list:iterate>

<pg:pager

    url="/index.jsp"

    items="<%=Integer.parseInt(listSize)%>"

    maxPageItems="10"

    maxIndexPages="10"

    export="currentPageNumber=pageNumber"

    scope="request">

    <pg:param name="query" value="<%=query%>"/>

    <pg:index export="totalItems=itemCount">

```



```
<pg:page export="firstItem, lastItem">
```

```
<div class="resultInfo">
```

```
    当前显示结果 <strong><%= firstItem %>-<%= lastItem %></strong> 找到相关文档
    <strong><%=listSize%></strong> 篇
```

```
</div>
```

```
</pg:page>
```

```
<div class="rnav">
```

```
    <table border=0 cellpadding=0 width=1% cellspacing=0 align=center><tr align=center
    valign=top>
```

```
        <td valign=bottom nowrap><font size=-1>结果页码: &nbsp; &nbsp;
```

```
        <pg:first unless="current">
```

```
            <td nowrap><a href="<%= pageUrl %>" class="rnavLink"><nobr> 首页
        </nobr></a>
```

```
        </pg:first>
```

```
        <pg:prev export="pageUrl">
```

```
            <td nowrap><a href="<%= pageUrl %>" class="rnavLink">&#171;&nbsp;&nbsp;上
        一页</a>&nbsp;&nbsp;
```

```
        </pg:prev>
```

```
        <pg:pages><%=
```

```

        if (pageNumber.intValue() < 10) {

            %>&nbsp;<%

        }

        if (pageNumber == currentPageNumber) {

            %><b><%= pageNumber %></b><%

        } else {

            %><td nowrap>&nbsp;<a href="<%= pageUrl %>"><%=
pageNumber %></a>&nbsp;<%

        }

        %>

</pg:pages>

<pg:next export="pageUrl">

&nbsp;&nbsp;<td nowrap><a href="<%= pageUrl %>" class="rnavLink">下
一页&nbsp;&#187;</a>

</pg:next>

<pg:last unless="current">

&nbsp;<td nowrap><a href="<%= pageUrl %>" class="rnavLink"><nobr>末
页</nobr></a>

</pg:last>

</table>

</div>

```

</pg:index>

</pg:pager>

</list:hasResults>

<list:hasNoResults>

对不起，没有结果返回。<p>建议您换用更少的文字查询以扩大搜索范围。

</list:hasNoResults>

</list:init>

<% } %>

Tomcat 版本 5.5.X 或 6.0。为了让 Tomcat 支持中文，要修改 server.xml。设置值 URIEncoding:

<Connector

port="8080" maxThreads="150" minSpareThreads="25"
maxSpareThreads="75"

enableLookups="false" redirectPort="8443" acceptCount="100"

debug="0" connectionTimeout="20000"

disableUploadTimeout="true" URIEncoding="UTF-8" useBodyEncodingForURI="true" />

同时修改 WEB-INF 目录下的 web.xml，设置 Filter

```
<filter>
  <filter-name>Set Character Encoding</filter-name>
  <filter-class>filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>Set Character Encoding</filter-name>
  <url-pattern>/*</url-pattern>

  </filter-mapping>
```

2.2 搜索引擎基本技术

2.2.1 网络蜘蛛

网络蜘蛛(Spider)又被称作网络机器人(Robot)., Worm 或着 Random, 它的主要目的是为获取在 Internet 上的信息。网络蜘蛛利用主页中的超文本链接遍历 Web, 通过 URL 引用从一个 HTML 文档爬行到另一个 HTML 文档。网络蜘蛛收集到的信息可有多种用途, 如建立索引、HTML 文件的验证、URL 链接验证、获取更新信息、站点镜像等。网络蜘蛛建立的页面数据库, 包含有根据页面内容生成的文摘, 这是一个重要特色。

在抓取网页时大部分网络机器人会遵循 Robot.txt 协议。

网站本身可以声明不想被搜索引擎收入的内容。可以有两种方式实现: 第一种方式是在你的站点增加一个纯文本文件<http://www.yourdomain.com/robots.txt>; 另外一种方式是直接在 HTML 页面中使用 robots 的 meta 标签。

2.2.2 全文索引结构

有如下两个文档:

Doc1: When in Rome, do as the Romans do.


Doc2: When do you come back from Rome?


经过如下停用词表的过滤:


in, as, the, from

形成的索引结构如下：

term	(doc, freq)	pos
back	(2, 1)	(5)
come	(2, 1)	(4)
do	(1, 2), (2, 1)	(3, 5), (2)
romans	(1, 1)	(4)
rome	(1, 1), (2, 1)	(2), (6)
when	(1, 1), (2, 1)	(1), (1)
you	(2, 1)	(3)


 .tis


 .frq


 .prx

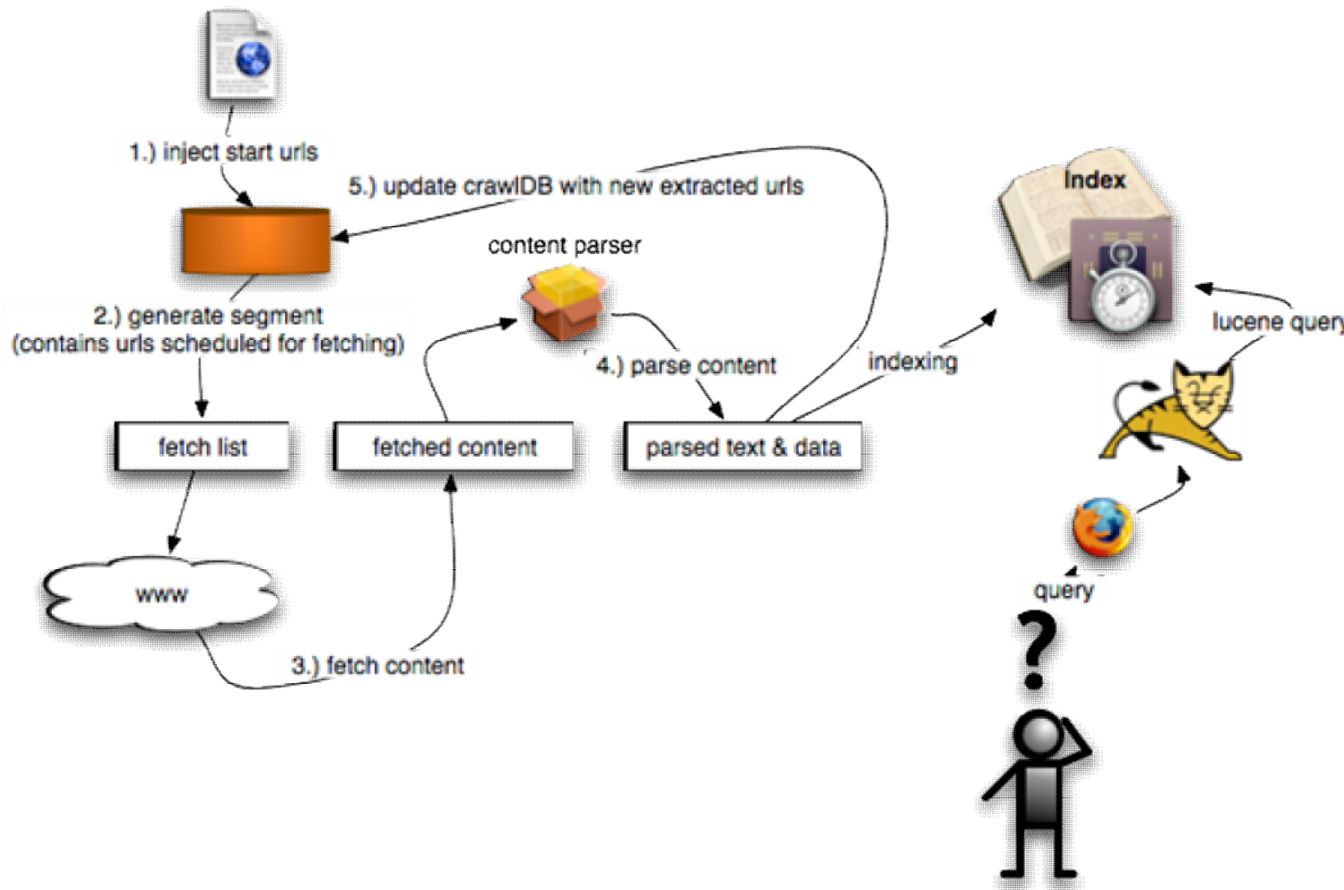
2.2.3 Lucene 全文检索引擎

Lucene 是一个开放源代码的全文索引库。经过 10 多年的发展，Lucene 拥有了大量的用户和活跃的开发团队。如果说 Google 是拥有最多用户访问的搜索引擎网站，那么拥有最多开发人员支持的搜索包也许是 Lucene。它最初由 Java 开发而成，现在有了 C#和 c++等移植版本。

它的官方网站地址是<http://lucene.apache.org/>。

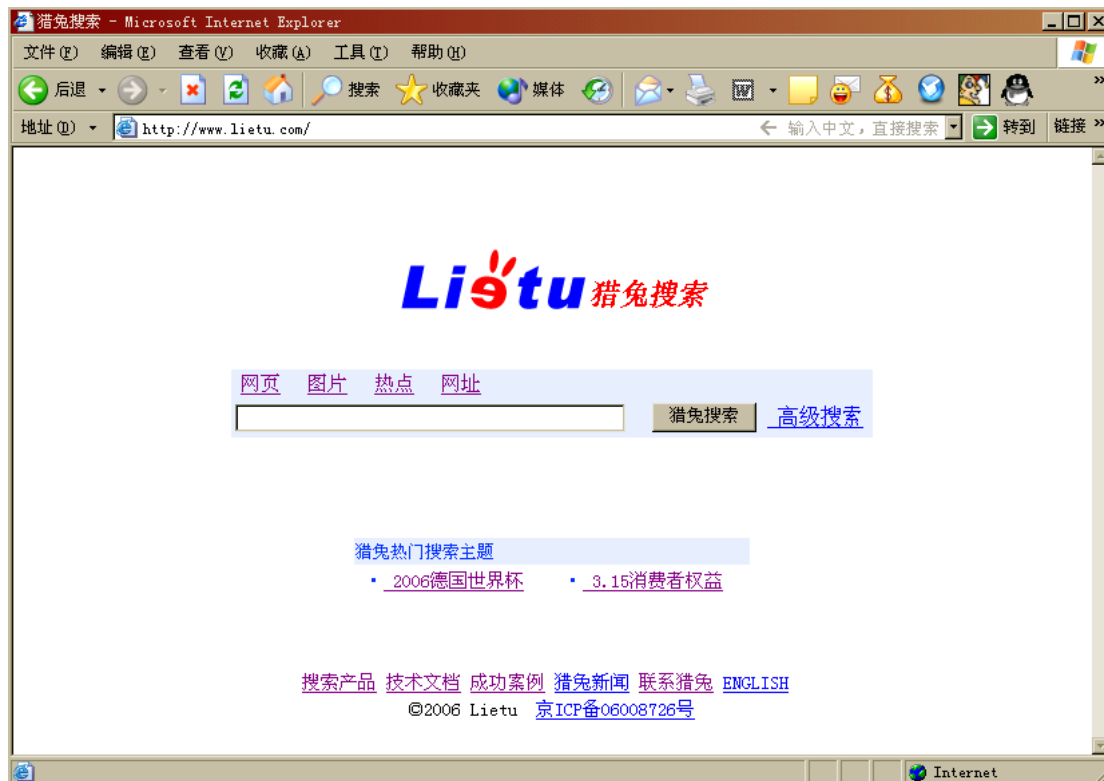
2.2.4 Nutch 网络搜索软件

Nutch 是构架于 Lucene 索引库基础上的网络搜索引擎。按照 Nutch 的设计，它可以搜索的范围可以是本地的局域网、Intranet、垂直的行业信息或者是整个互联网。抓取部分和搜索部分是它的两个基本组成部分。基本结构如下：



Nutch 从 0.8 版本开始，把分布式计算处理部分独立出来，命名为 Hadoop。Hadoop 主要包括分布式的文件系统(DFS 的实现在 `org.apache.hadoop.dfs` 中)和一个 MapReduce 分布式数据处理器(MapReduce 的实现在 `org.apache.hadoop.mapred` 中)。

2.2.5 用户界面



2.3 商业搜索引擎技术介绍

2.3.1 通用搜索

目前通用搜索引擎的组织方式主要有网络综合搜索引擎和网络主题资源搜索引擎两种。其中网络综合搜索引擎能够广泛地采集各 Internet 站点资源，并对其进行页面搜索，将索引结果存入索引数据库，供网络用户检索，提供 Internet 网络资源地导航功能的工具，如 google、baidu 等。

图 2.1 列出了国内外比较出名的几种通用搜索的排名，从图中我们可以看出在国内有影响力 Google 位列第一，当然我们也看到在通用搜索这个领域内，还有几家也是仅次于 google，通用搜索这一领域竞争也非常激烈。

名 次	站 名	类 别	全球排序
1.	Google	通用搜索	Click
2.	Ask Jeeves, Inc.	通用搜索	Click
3.	Google Groups	通用搜索	Click
4.	AllTheWeb	通用搜索	Click
5.	Google Image Search	通用搜索	Click

图 2.1 通用搜索引擎排名

2.3.2 垂直搜索

垂直搜索是针对某一个行业的专业搜索引擎，例如生活信息搜索 <http://www.kooxoo.com>，职位搜索 <http://www.jobui.com>。垂直搜索是搜索引擎的细分和延伸，是对网页库中的某类专门的要的数据进行处理后再以某信息进行一次整合，定向分子段抽取所需形式返回给用户。

垂直搜索需要从茫茫的互联网中获取行业信息，信息按行业过滤和分类是必不可少的。垂直搜索引擎和普通的网页搜索引擎的另一个最大区别是对网页信息进行了结构化信息抽取，也就是将网页的非结构化数据抽取成特定的结构化信息数据，好比网页搜索是以网页为最小单位，基于视觉的网页块分析是以网页块为最小单位，而垂直搜索是以结构化数据为最小单位。然后将这些数据存储到数据库，进行进一步的加工处理，如：去重、分类等，最后分词、索引再以搜索的方式满足用户的需求。

整个过程中，数据由非结构化数据抽取成结构化数据，经过深度加工处理后以非结构化的方式和结构化的方式返回给用户。

垂直搜索引擎的应用方向很多，比如企业库搜索、供求信息搜索引擎、购物搜索、房产搜索、人才搜索、地图搜索、mp3 搜索、图片搜索……几乎各行各业各类信息都可以进一步细化成各类的垂直搜索引擎。

举个例子来说明会更容易理解，比如购物搜索引擎，整体流程大致如下：抓取网页后，对网页商品信息进行抽取，抽取出商品名称、价格、简介然后对信息进行清洗、去重、分类、分析比较、数据挖掘，最后通过分词索引提供用户搜索、通过分析挖掘提供市场行情报告。

垂直搜索引擎大体上需要以下技术：

- 1 . Spider;
- 2 . 网页结构化信息抽取技术或元数据采集技术;
- 3 . 分词、索引;

4. 其他信息处理技术。

垂直搜索引擎的技术评估应从以下几点来判断：

1. 全面性；
2. 更新性；
3. 准确性；
4. 功能性。

垂直搜索的进入门槛很低，但是竞争的门槛很高。没有专注的精神和精湛的技术是不行的。行业门户网站具备行业优势但他们又是没有技术优势的，绝对不要想像着招几个人就可以搞定垂直搜索的全部技术，作为一个需要持续改进可运营的产品而不是一个项目来说对技术的把握控制程度又是垂直搜索成功的重要因素之一。与专业的搜索技术提供商合作共赢是一种现实的解决方法。当前国内搜索技术提供商有：

厂商	特点
Atonomy	国际最大的企业搜索提供者，价格相对较贵。
TRS	国内最大的企业搜索提供者，政府型项目居多。
海量	提供中文分词和搜索方案的厂家。
猎兔	提供基于 Lucene 和自然语言处理的企业搜索的厂家。

2.3.3 站内搜索

首先我们先从下图的比较中认识什么是真正的站内搜索。

真正的全文检索应具备相关性排序技术和分词索引功能。如果需要进行互联网的信息抓取和采集那么还需要网络蜘蛛模块。分词、索引、排序这是全文检索的基本和核心，缺一不可。全文检索至少需要具备中文分词、索引、相关性排序功能。

所以简单考查一个站内搜索引擎的真伪只需要知道：能否实现相关性排序、国际标准的搜索语法、动态摘要、飘红、支持海量数据多并快速发查询、搜索耗时极短。

常用的站内搜索技术比较：

	基于数据库的搜索	基于 spider 抓取的站内搜索	站内搜索软件系统
原理	数据库搜索	通过 Spider 抓取网页，经 html 解析，分词，索引实现网页式站内搜索。	对数据库数据进行 html 解析、图片缩略，分词，索引，实现站内站内搜索。
检索效率	非常低下 消耗大量硬件资源	高效	高效
检索范围	无法完成全文检索（可以用 sql 的单字索引功能最简单的完成索引功能实现最低级的全文检索），只能进行标题检索。数据库效率太低，无法开展各种附加功能。	网页检索 优点：不需要做各种工作，直接即可使用 缺点：1.有大量的不必要的信息影响搜索结果的排序和显示的效果。严重影响精确度。2.部分页面无法抓取到。3.用户对搜索范围和内容以及体现的结果无法精确控制	标题+内容 基于内容分析的排序方法。基于内容分析排序是最佳的排序方法。标题和内容可控制，搜索结果准确到位。内容内可控，用户可对搜索的内容范围和体现的结果进行精确的控制
检索语法	无	支持标准的国际搜索语法。	支持标准的国际搜索语法。
动态摘要	无	摘要内容不清晰各种垃圾信息过多	提供动态摘要，摘要清晰精确，便于用户快速寻找到需要的信息。
关键词飘红	无	有	有
内容的范围	可控制	不可有效控制，动态网页抓取效果不佳，没有链接的网页无法抓取，页面出现杂乱信息影响搜索结果。对于时间控制也无法做到精准，对于栏目的归属无法做到准确。	可有效控制，您可以把多个字段拆分合并，可以确定那些需要，哪些不需要。所有动态网页和没有链接的网页均可有效收入。栏目控制精准。
图片缩略	无	无	有
同义词	无	无	有
相关性排序	无	有	有
其他	低效率低质量的平台无法开展增值服务	二次研发成本高	具有持续不断的升级能力和良好的售后服务。
成本	低	价格高	以产品形式运作，多家客户分摊成本，成本相对低廉。运维成本低。
维护成本	不大	维护量不大。	程序维护无须投入，但是需要进行一定量的内容维护。经过简单培训即可胜任。

猎兔（<http://www.lietu.com/>）企业搜索的实现正是这样一种站内全文搜索的实现。

2.3.4 桌面搜索

桌面搜索引擎允许用户可以很方便的快速查找到存在于计算机本地磁盘上的相关文档。这种搜索方式就像读者可以通过搜索引擎快速的在因特网上查到自己需要的资料一样。

桌面搜索技术本身不是一个新的概念，但是最近有些知名的搜索引擎（如 Google，Yahoo）已经开始涉足这个领域使桌面搜索技术在应用方面变得可视化。（读者可以去 <http://desktop.google.com/zh/> 下载 Google 的 DeskTop 体验桌面搜索）。

为了了解目前市场上的桌面搜索工具的威斯康星-麦迪逊大学的2位学者对12个桌面搜索工具在可用性(usability)，多用性(versatility)，效率(efficiency)，准确度(accuracy)，安全性(security)，企业适用性(enterprise readiness)6个方面做了比较，前5名为

copernic Desktop Search
yahoo! Desktop Search
likasoft archivarius 3000
mSn Toolbar Suite
google Desktop

新一代的桌面搜索能同时帮助用户发现企业和互联网中的文档。在知识经济的时代，公司期望通过它提高员工的生产效率和创新性。

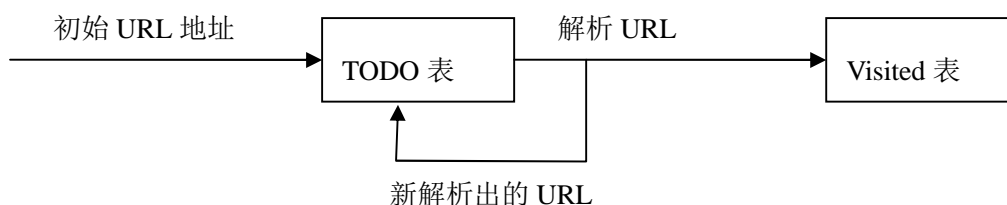
2.4 本章小结

至此我们对搜索引擎的分类可以告一段落。回想一下，我们在这章详细介绍了商业搜索引擎的技术。其实大多数搜索引擎之间没有绝对的界限，往往是交叉运用的。

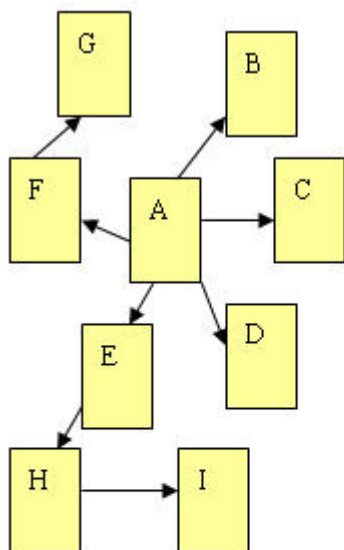
第3章 获得海量数据

3.1 自己的网络蜘蛛

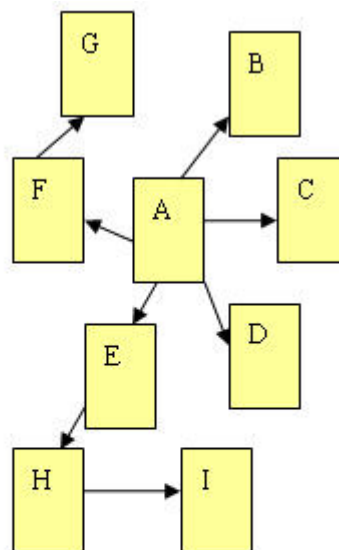
网络蜘蛛是对URL链接的遍历。下面分析一个支持多线程的网络蜘蛛实现。最基本的数据结构包括一个待扩展的URL表和一个已经访问过的URL地址表。



在抓取网页的时候，网络蜘蛛一般有两种策略：广度优先和深度优先（如下图所示）。广度优先是指网络蜘蛛会先抓取起始网页中链接的所有网页，然后再选择其中的一个链接网页，继续抓取在此网页中链接的所有网页。这是最常用的方式，因为这个方法可以让网络蜘蛛并行处理，提高其抓取速度。深度优先是指网络蜘蛛会从起始页开始，一个链接一个链接跟踪下去，处理完这条线路之后再转入下一个起始页，继续跟踪链接。这个方法有个优点是网络蜘蛛在设计的时候比较容易。两种策略的区别，下图的说明会更加明确。



广度优先的抓取顺序：
A—B.C.D.E.F—H.G—I



深度优先的抓取顺序：
A—F—G
E—H—I
.....

广度遍历，采用队列的方式实现 `todo` 表的扩展，先访问的网页先扩展，对于上图中的 `todo` 表和 `visited` 表的执行状态如下：

`todo: a`

`visited: null`

`todo: b c d e f`

`visited: a`

`todo: c d e f`

`visited: a b`

`todo: c d e f`

`visited: a b`

`todo: d e f`

`visited: a b c`

`todo: e f`

`visited: a b c d`

`todo: f h`

`visited: a b c d e`

`todo: h g`

`visited: a b c d e f`

`todo: g i`

`visited: a b c d e f h`

todo: i

visited: a b c d e f h g

todo: null

visited: a b c d e f h g i

深度遍历，采用堆栈的方式实现 todo 表的扩展，先访问的网页先扩展，对于上图中的 todo 表和 visited 表的执行状态如下：

todo: a

visited: null

todo: b c d e f

visited: a

todo: b c d e g

visited: a f

todo: b c d e

visited: a f g

todo: b c d h

visited: a f g e

实现的部分代码如下：

```
/** current tasks */
```

```
protected ToDoTaskList todo = null;
```

```
/** a list of all URLs we got already */
```

```

protected VisitedTaskList visited = null;

//然后通过enqueueURL放入todo表。
public synchronized void enqueueURL(NewsSource newitem) {
    if (!visited.contains(newitem.URL)) {
        todo.add(newitem);
        //唤醒阻塞的线程
        notifyAll();
    }
}

//下一步是从todo取出要分析的URL地址，同时把它放入visited表。
public synchronized NewsSource dequeueURL() throws Exception {
    while (true) {
        if (!todo.isEmpty()) {
            NewsSource newitem = (NewsSource)todo.removeFirst();
            visited.add(newitem.URL);
            return newitem;
        }
        else {
            threads--;
            if (threads > 0) {
                wait();
                threads++;
            }
            else {
                notifyAll();
                return null;
            }
        }
    }
}

```

多线程处理:

主线程部分:

```

threadList = new ArrayList<Thread>(THREAD_NUM);

for (int i = 0; i < THREAD_NUM; i++) {
    Thread t = new Thread(this, "Spider Thread #" + (i+1));
    t.start();
    threadList.add(t);
}

```

```
//current thread wait until child thread exit.
while (threadList.size() > 0) {
    Thread child = (Thread)threadList.remove(0);
    child.join();
}
```

子线程:

```
public synchronized NewsSource dequeueURL() throws Exception {
    while (true) {
        if (!todo.isEmpty()) {
            NewsSource newitem = (NewsSource)todo.removeFirst();
            visited.add(newitem.URL,newitem.source);
            return newitem;
        }
        else {
            threads--;
            if (threads > 0) {
                wait();
                threads++;
            }
            else {
                notifyAll();
                return null;
            }
        }
    }
}
```

```
public synchronized void enqueueURL(NewsSource newitem) {
    if (!visited.contains(newitem.URL)) {
        todo.add(newitem);
        //visited.add(newitem.URL,newitem.source);
        notifyAll();
    }
}
```

```
public void run() {
    NewsSource item;
```



```

try {
    while ((item = dequeueURL()) != null) {
        indexURL(item);
    }
}
catch(Exception e) {
    e.printStackTrace();
}

threads--;
}

```

3.1.1 BerkeleyDB 介绍

todo 表或者 visited 表一般用 ArrayList 或者 HashMap 实现，它们只能在内存中，但内存是有限的。开始的时候，有人把 todo 表或者 visited 表放在数据库中。但数据库对于这种简单的结构化存储来说，不够轻量级。

BerkeleyDB 是一个嵌入式数据库。底层实现采用 B 树。可以看成可以存储大量数据的 HashMap。它简称 BDB，官方网址是：<http://www.oracle.com/database/berkeley-db/index.html>。c++版本，然后实现了 Java 本地版本。

BerkeleyDB 用到的主要对象有：

新建环境：

```

EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setTransactional(false);
envConfig.setAllowCreate(true);

exampleEnv = new Environment(envDir, envConfig);

```

释放环境变量：

```

exampleEnv.sync();
exampleEnv.close();

exampleEnv = null;

```

创建数据库：

```

String databaseName= "ToDoTaskList.db";
DatabaseConfig dbConfig = new DatabaseConfig();
dbConfig.setAllowCreate(true);
dbConfig.setTransactional(false);

```

```

// Open the database that you use to store your class information.
// The db used to store class information does not require duplicates
// support.
dbConfig.setSortedDuplicates(false);
Database myClassDb = exampleEnv.openDatabase(null, "classDb",
dbConfig);

// Instantiate the class catalog
catalog = new StoredClassCatalog(myClassDb);

TupleBinding keyBinding =
    TupleBinding.getPrimitiveBinding(String.class);

// use String serial format and binding for values
SerialBinding valueBinding = new SerialBinding(catalog,
NewsSource.class);

store = exampleEnv.openDatabase(null, databaseName, dbConfig);

```

建立映射:

```

// create a map view of the data store

this.map = new StoredSortedMap(store, keyBinding, valueBinding,
true);

```

3.1.2 抓取网页

网页抓取首先从一个 URL 地址列表开始遍历,要通过 DNS 取得该 URL 域名的 IP 地址。在 Linux 下 DNS 解析的问题可以用 dig 或 nslookup 命令来分析,例如:

```
#dig www.sanyuan-com.com
```

```
#nslookup www.lietu.com
```

如果需要可以编辑 DNS 配置文件更换更好的 DNS 域名解析服务器。

```
#vi /etc/resolv.conf
```

不同的网站间转载内容的情况很常见。即使在同一个网站,有时候不同的 URL 地址可能对应同一个页面,或者存在同样的内容以多种方式显示出来。网页的文档排重是必要的。

判断文档的内容重复有很多种方法，语义指纹的方法比较高效。语义指纹直接提取一个文档的二进制数组表示的语义，通过比较相等来判断网页是否重复。语义指纹是一个很大的数组，全部存放在内存会导致内存溢出，普通的数据库效率太低，所以这里采用内存数据库 BerkeleyDB。

3.1.3 MP3 抓取

百度公司内部数据表明，MP3 搜索占其全部搜索流量的 4 成左右，多数年轻网民使用百度的主要功能就是 MP3 搜索。

Mp3 索引应该需要如下的数据：

歌手名（非必需）

歌曲名（必需）

url

歌词（非必需）

LRC 是一种歌词档案格式，含有用作将歌词与声音/影像档案作同步处理的时间标签 time tag。在互联网上，被广泛使用的 LRC 格式有两种：

简易 LRC 格式的歌词显示器（相信是第一个模拟卡拉 OK 的软件）所采用的歌词格式。“简易 LRC 格式”只含有行时间标签 Line Time Tags [mm:ss.xx]：

```
[00:23.93](男)半夜睡不著觉 把心情哼成歌
[00:29.14]只好到屋顶找另一个梦境
[00:40.06](女)睡梦中被敲醒 我还是不確定
[00:45.62]怎曾有动人 弦律在对面的屋顶
[00:51.08]我悄悄关上门 带著希望上去
[00:56.32]原来是我梦里常出现的那个人
[01:00.95](男)那个人不就是我梦里
[01:04.00]那模糊的人 我们有同样的默契
```

因为这种格式没有“时间签标”标示一个字（或音节）的开始及结束时间，所以不能支援进行即时歌词逐字填色。加强 LRC 格式，为原来的格式加多一个“时间签标” - 字时间签标 Word Time tag <mm:ss.xx>：

```
[mm:ss.xx] <mm:ss.xx> 第一行第一个字 <mm:ss.xx> 第一行第二个字 <mm:ss.xx> ... 第一
行最后一个字 <mm:ss.xx>
```

[mm:ss.xx] <mm:ss.xx> 第二行第一个字 <mm:ss.xx> 第二行第二个字 <mm:ss.xx> ... 第二行最后一个字 <mm:ss.xx>

...

3.1.4 RSS 抓取

RSS 抓取的第一步是解析 RSS 数据源<http://informa.sourceforge.net>提供了一个解析包。ROME 是另外一个常用的解析包。

为了读取一个 RSS 种子，首先定义读取种子的源，然后定义构建新闻频道对象模型的 ChannelBuilder。

```
ChannelBuilder builder = new ChannelBuilder();
String url = "http://rss.news.yahoo.com/rss/topstories";
Channel channel = (Channel) FeedParser.parse(builder, url);
System.out.println("标题: " + channel.getTitle());
System.out.println("描述: " + channel.getDescription());
System.out.println("内容:");
for (Object x : channel.getItems())
{
    Item anItem = (Item) x;
    System.out.print("标题:" + anItem.getTitle() + " 描述: ");
    System.out.println(anItem.getDescription());
}
```

<link href="http://blog.csdn.net/myth1979/rss.aspx" title="RSS" type="application/rss+xml" rel="alternate" />

type="application/rss+xml"

forumrss - rss 地址

VisitTime - rss 种子访问时间

VisitFreq - 访问频率

采用不同的频率抓取不同更新频率的 RSS。

提取 RSS 描述里面的字符串：

```
public static String removeHTML(String s) {

    Lexer lexer = new Lexer(s);

    StringBuilder body = new StringBuilder();

    try {

        Node n = lexer.nextNode();

        while (n!=null)

        {

            if(n instanceof org.htmlparser.lexer.nodes.StringNode)

            {

                //System.out.println("node:"+n);

                //System.out.println("node:"+n.getClass());

                body.append(((org.htmlparser.lexer.nodes.StringNode)n).getText());

            }

            n = lexer.nextNode();

        }

    } catch (ParserException e) {

        // TODO Auto-generated catch block

        e.printStackTrace();

    }

    //System.out.println("body:"+body.toString());
```

```
        return body.toString();  
  
    }  
}
```

字符串最后的处理:

```
TextHtml.html2text(removeHTML(desc)).replaceAll("[\t\n\f\r ]+","")
```

sf.net 上面还有个在 informa 基础上构架的新闻爬虫, WebNews Crawler, <http://senews.sourceforge.net/>。

WebNews Crawler 是一个通过 HTTP 下载资源的 java 爬虫。这个爬虫可以解析 RSS 种子, 也可以通过 HTML2XML 库把 HTML 数据表示成 XML 格式。

首先准备一个要抓取的 URL 列表, 把它放到'bin/news-rss.crl' 文件。

根据需要修改配置文件。

打开和编辑'bin/conf/crawler.properties'文件, 它包括了和抓取进程相关的的所有的重要参数。

在控制台启动爬虫

```
shell> cd unpacked_package_dir/bin
```

开始爬虫进程直到它结束:

```
shell> java -jar webnews-crawler.jar -cmd start
```

如果你计划抓取许多 URL 可以考虑使用服务器模式而不是上面的方式。下面在 12345 端口启动爬虫服务器:

```
shell> java -jar webnews-crawler.jar -cmd server -p 12345
```

使用 TanaSend.jar 发送一个启动命令

```
shell> java -jar TanaSend.jar localhost:12345 cmd:start
```

类似的命令有 stop、resume、shutdown、exit 等。

在上面的步骤中，爬虫已经下载了一些内容并且把它保存在内部数据库。下面是从内部数据导出数据的方法：

```
shell> java -jar webnews-crawler.jar -cmd export
```

导出过程结束后，可以看到一个路径'export-<timestamp>' 这里 <timestamp>是一个 Unix 时间。在这个路径里面，每个导出的资源有一个'meta' 和一个'original' 文件。而且，如果资源文件的 HTML2XML 过程执行成功的话， 会有个附加的'xml'文件。

3.1.5 图片抓取

为了能够节约网络流量。抓取过来的图片经常需要缩小到一定的尺寸。比如 100*100 或 80*80。

```
// load image from INFILE
Image image = Toolkit.getDefaultToolkit().getImage(inFile);
MediaTracker mediaTracker = new MediaTracker(new Container());
mediaTracker.addImage(image, 0);
mediaTracker.waitForID(0);
// determine thumbnail size from WIDTH and HEIGHT
int thumbWidth = Integer.parseInt(args[2]);
int thumbHeight = Integer.parseInt(args[3]);
double thumbRatio = (double)thumbWidth / (double)thumbHeight;
int imageWidth = image.getWidth(null);
int imageHeight = image.getHeight(null);
double imageRatio = (double)imageWidth / (double)imageHeight;
if (thumbRatio < imageRatio) {
    thumbHeight = (int)(thumbWidth / imageRatio);
} else {
    thumbWidth = (int)(thumbHeight * imageRatio);
}
// draw original image to thumbnail image object and
// scale it to the new size on-the-fly
BufferedImage thumbImage = new BufferedImage(thumbWidth,
    thumbHeight, BufferedImage.TYPE_INT_RGB);
Graphics2D graphics2D = thumbImage.createGraphics();
graphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
graphics2D.drawImage(image, 0, 0, thumbWidth, thumbHeight, null);
// save thumbnail image to OUTFILE
BufferedOutputStream out = new BufferedOutputStream(new
    FileOutputStream(outFile));
```

```

JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
JPEGEncodeParam param = encoder.
    getDefaultJPEGEncodeParam(thumbImage);
int quality = Integer.parseInt(args[4]);
quality = Math.max(0, Math.min(quality, 100));
param.setQuality((float)quality / 100.0f, false);
encoder.setJPEGEncodeParam(param);
encoder.encode(thumbImage);

out.close();

```

3.1.6 垂直行业抓取

垂直搜索的抓取用来抓取行业相关信息。要解决的一个问题是从大量的网页中有效的取得行业相关信息。有两种流行的方法控制抓取相关性和质量：

- 限制 spiders 只抓取特定网站域名的信息。例如，www.ctrip.com中的大多数网页都是和旅游相关的。
- 基于网页内容过滤收集到的页面。例如，可以删除一个页面，如果它的信息中包括的相关关键词个数少于一个阈值。

有人比较过 BFS (breadth-first search) spider、PageRank spider 和 Hopfield Net spider。BFS spider 基于这样一个假设：如果一个网页的内容与搜索领域相关的，那么由这个网页出去的链接也很有可能与这个领域相关。由于领域内最重要的那些网页通常都包含大量的 in-link，BFS 能够在搜索的早期阶段发现大量高质量的网页。PageRank spider 基于 BFS spider，但是使用 PageRank 算法作为启发，具有更多 in-link 的网页会获得更高的得分。每一轮，PageRank spider 会抓取最高得分的 URL，对内容进行解析后将链接放入队列中并重新计算得分。Hopfield Net spider 将整个 Web 看作一个巨大的 Hopfield 神经网络，其中每一个网页就是神经元，网页之间的超链就是神经元之间连接。Hopfield 网用并行的方式工作，不同的神经元同时受到激发，并且来自不同神经元的激发会通过一定方式合并，继续影响其它神经元，直到整个网络收敛到一个稳定值。试验表明，Hopfield Net spider 的评价最好，BFS spider 的评价次之，PageRank spider 的评价最差。尽管 PageRank 算法对于大量的页面是稳定的，但是当处理小数据量的时候，它的分值可能会误导 Spider。

下面我们实现一个 Best-First 的爬虫，也就是说每次取的 URL 地址都是 Todo 列表中的最好的 URL 地址。为了实现快速的查找和操作，一般使用 Berkeley DB 来实现 TodoList。

Berkeley DB 中存储的一般是一个关键字和值的 Map。为了同时实现按照 URL 地址查找和按照 url 地址分值排序，简单的 map 不能满足要求，这时候可以使用 SecondaryIndex 和 PrimaryIndex 来达到多列索引的目的。实现持久化的基本类如下：

```

@Entity
public class NewsSource {

```



```

@PrimaryKey public String URL;
public String source;
public int level;
public int rank;

public String urlDesc = null;
@SecondaryKey(related=Relationship.MANY_TO_ONE) public int score;
}

```

Todo 列表的构造方法如下:

```

public TodoTaskList(Environment env) throws Exception {
    StoreConfig storeConfig = new StoreConfig();
    storeConfig.setAllowCreate(true);
    storeConfig.setTransactional(false);

    store = new EntityStore(env, "classDb", storeConfig);
    newsByURL = store.getPrimaryIndex(String.class,
NewsSource.class);
    secondaryIndex = store.getSecondaryIndex(this.newsByURL,
Integer.class, "score");
}

```

从 Todo 列表取得最大分值的 URL 地址的方法如下:

```

public NewsSource removeBest() throws DatabaseException
{
    Integer score = secondaryIndex.sortedMap().lastKey();
    if (score != null){
        EntityIndex<String,NewsSource> urlLists =
secondaryIndex.subIndex(score);
        EntityCursor<String> ec = urlLists.keys();
        String url = ec.first();
        ec.close();
        NewsSource source = urlLists.get(url);
        urlLists.delete(url);

        return source;
    }

    return null;
}

```

3.2 抓取数据库中的内容

3.2.1 建立数据视图

传统的方式下，我们建立起一个统一的数据视图，比如标题，内容，日期等放在一个视图中。在更新的时候，需要从索引库中删除整条记录，然后增加整条记录。

当创建购物搜索的时候，当价格等打折信息需要放入索引库，而像商品描述等信息一般不会变化。又比如像视频搜索中的视频点击次数也会频繁的更新。这时候使用统一的视图会影响索引库的及时更新性能。这时候可以把频繁更新的索引和不频繁更新的索引列分开，然后通过 `ParallelReader` 来合并结果。

```
String _dir1 = "c:/index1";
IndexReader reader1 = IndexReader.open(_dir1 );
String _dir2 = "c:/index2";
IndexReader reader2 = IndexReader.open(_dir2 );
ParallelReader preader=new ParallelReader();
preader.add(reader1);
preader.add(reader2);

IndexSearcher searcher=new IndexSearcher(preader);
```

之后的操作和一般的搜索相同。

`ParallelReader` 用到的多个索引目录中的 `Doc_id` 是同步的。是为了同步多个索引目录，需要使用 `ParallelWriter` 来做索引。

```
Directory[] directories;
```

```
ParallelWriter writer = new ParallelWriter(fieldDirectories, new StandardAnalyzer(), create);
```

3.2.2 JDBC 数据库连接

一个简单的对整个表中的记录建立索引的例子：

```
String sql = "select " +
    "id," +
    "title," +
    "content," +
    "type," +
```

```

        "Country_id," +
        "input_time,"+
        "click_count"+
        " from commerce_info";

PreparedStatement stmt = con.prepareStatement(sql);
ResultSet rs = stmt.executeQuery();
int i = 1 ;
while(rs.next())
{
    Document doc = new Document();
    Field f;

    System.out.println(String.valueOf(i));
    f = new Field("id", String.valueOf(rs.getInt("id")) ,
        Field.Store.YES, Field.Index.UN_TOKENIZED,
        Field.TermVector.NO);
    doc.add(f);

    f = new Field("title", rs.getString("title") ,
        Field.Store.YES, Field.Index.TOKENIZED,
        Field.TermVector.WITH_POSITIONS_OFFSETS);
    doc.add(f);

    f = new Field("content", rs.getString("content") ,
        Field.Store.YES, Field.Index.TOKENIZED,
        Field.TermVector.WITH_POSITIONS_OFFSETS);
    doc.add(f);

    f = new Field("type", String.valueOf(rs.getInt("type")) ,
        Field.Store.YES, Field.Index.UN_TOKENIZED,
        Field.TermVector.NO);
    doc.add(f);

    f = new Field("country",
String.valueOf(rs.getInt("Country_id")) ,
        Field.Store.YES, Field.Index.UN_TOKENIZED,
        Field.TermVector.NO);
    doc.add(f);

    f = new Field("time",

DateTools.dateToString(rs.getDate("input_time"),Resolution.DAY) ,
        Field.Store.YES, Field.Index.UN_TOKENIZED,

```

```

        Field.TermVector.NO);
    doc.add(f);

    f = new Field("pop",
        String.valueOf(rs.getInt("click_count")),
        Field.Store.YES, Field.Index.UN_TOKENIZED,
        Field.TermVector.NO);
    doc.add(f);

    i++;
    try{
        index.addDocument(doc);
    }
    catch(Exception e)
    {
        e.printStackTrace(System.out);
    }
} //while2 end !!!
rs.close();

stmt.close();

```

如果要对很大的表建立索引，使用微软 SQL Server 的 JDBC 驱动一次读取全部的数据到内存中，会导致内存溢出，如果内存足够大，增加 JVM 的使用内存就可以了。当碰到更大的表需要索引的时候，需要使用比较新的 Sql Server JDBC 驱动，设置 connection 连接参数 responseBuffering=adaptive。

3.2.3 增量抓取

如果只是增加数据而不考虑更新或删除，可以用一个表记录当前已经索引的最大 id 号码。

```
create table indexLog(itemName varchar(50),maxID int)
```

3.3 抓取本地硬盘上的文件

3.3.1 目录遍历

用递归的方法遍历路径。

```

public void go() throws Exception {
    long start = System.currentTimeMillis();

    // create the index directory -- or append to existing
    if (verbose) {
        System.out.println("Creating index in: " + indexDir);
        if (incremental) System.out.println("    - using incremental
mode");
    }
    index = new IndexWriter(new File(indexDir), new StandardAnalyzer(),
        !incremental);

    File dir = new File(sSourceDir);

    indexDir(dir);

    index.optimize();
    index.close();
    if(verbose)
        System.out.println("index complete
in :"+(System.currentTimeMillis() - start)/1000);
}

private void indexDir(File dir)
{
    File[] files = dir.listFiles();

    for (int i = 0; i < files.length; i++) {
        File f = files[i];
        if (f.isDirectory()) {
            indexDir(f); // recurse
        } else if (f.getName().endsWith(".txt")) {
            indexFile(f);
        }
    }
}

private void indexFile(File item) {
    if (verbose) System.out.println("Adding FILE: " + item);

    News news = loadFile(item);

    if (news != null && news.body != null) {
        Document doc = new Document();

```

```
Field f = new Field("url", news.URL ,
    Field.Store.YES, Field.Index.UN_TOKENIZED,
    Field.TermVector.NO);
doc.add(f);

f = new Field("title", news.title ,
    Field.Store.YES, Field.Index.TOKENIZED,
    Field.TermVector.WITH_POSITIONS_OFFSETS);
doc.add(f);

f = new Field("body", news.body.toString() ,
    Field.Store.YES, Field.Index.TOKENIZED,
    Field.TermVector.WITH_POSITIONS_OFFSETS);
doc.add(f);

try{
    index.addDocument(doc);
}
catch(Exception e)
{
    e.printStackTrace();
    System.exit(-1);
}
}
```

3.4 本章小结

第4章 提取文档中的文本内容

4.1 从 HTML 文件中提取文本

在实现从 Web 网页提取文本之前，首先要识别网页的编码，如果有必要，也要识别网页所使用的语言。整体流程如下：

1. 从 Web 服务器返回的 `content type` 中提取编码，如果是 `gb2312` 类型的编码要当成 `GBK` 处理。
2. 从网页的 `Meta` 信息中识别字符编码，如果和 `content type` 中的编码不一致，以 `Meta` 中声明的编码为准。
3. 如果仍然无法确定网页所使用的字符集，需要从返回流的二进制格式判断。同时要确定网页所使用的语言，例如 `UTF-8` 编码的语言可以是中文，英文，日文或韩文等任何语言。

下面的代码从头信息提取网页编码。

//输入头信息，返回网页编码

```
public static String getCharset (String content)
{
    final String CHARSET_STRING = "charset";
    int index;
    String ret;

    ret = null;
    if (null != content)
    {
        index = content.indexOf (CHARSET_STRING);

        if (index != -1)
        {
            content = content.substring (index + CHARSET_STRING.length
            ()).trim ();
            if (content.startsWith ("="))
            {
                content = content.substring (1).trim ();
                index = content.indexOf (";");
                if (index != -1)
                    content = content.substring (0, index);
            }
        }
    }
}
```

```

        //remove any double quotes from around charset string
        if (content.startsWith("\"") && content.endsWith("\"")
&& (1 < content.length ()))
            content = content.substring (1, content.length () -
1);

        //remove any single quote from around charset string
        if (content.startsWith("'") && content.endsWith("'")
&& (1 < content.length ()))
            content = content.substring (1, content.length () -
1);

        ret = findCharset (content, ret);
    }
}

return (ret);
}

```

有的页面在 head 信息中不包括编码格式内容，需要从 meta 信息中提取。例如下面这个

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

下面利用 Htmlparser 提取 meta 信息。

```
String contentCharSet = tagNode.getAttribute("CONTENT");
```

另外一个和字符编码相关的问题是，有时候碰到 gb2312 编码的网页会有乱码问题，因为浏览器能正常显示包含 GBK 字符的 gb2312 编码网页。我们把 org.htmlparser.lexer.InputStreamSource 中的设置编码方法修改一下，把设置字符集为 GB2312 改成 GBK：

```

public void setEncoding (String character_set)
    throws
        ParseException
{
    if(character_set!= null &&
character_set.toLowerCase().equals("gb2312"))
    {
        character_set = "GBK";
    }
}

```


...

提取正文的完整过程如下：

```
public void parseHTML(HttpURLConnection uc) throws ParseException
{
    Node node;
    String stringText;

    String contentType = uc.getContentType();
    String charSet = getCharset (contentType);
    Lexer lexer = null;
    if(charSet!=null)
    {
        try {
            lexer = new Lexer (new Page(uc.getInputStream(), charSet ));
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return ;
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace(System.out);
            return ;
        }
    }
    else
    {
        try {
            lexer = new Lexer (new Page(uc.getInputStream(),
"GB2312" ));
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return ;
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return ;
        }
    }
    lexer.setNodeFactory(typicalFactory);
    boolean tryAgain = false;
```

```
while (null != (node = lexer.nextNode ()))
{
    //omit script tag
    if (node instanceof org.htmlparser.tags.ScriptTag)
    {
        while (null != (node = lexer.nextNode ()))
        {
            if (node instanceof org.htmlparser.tags.Tag)
            {
                org.htmlparser.tags.Tag tag =
(org.htmlparser.tags.Tag)node;
                if( tag.isEndTag() &&
"SCRIPT".equals(tag.getTagName()) )
                {

                    //System.out.println("tagname:"+tag.getTagName());
                    break;
                }
            }
        }
        if(null == node)
            break;
    }
    //omit script tag
    else if (node instanceof org.htmlparser.tags.StyleTag)
    {
        while (null != (node = lexer.nextNode ()))
        {
            if (node instanceof org.htmlparser.tags.Tag)
            {
                org.htmlparser.tags.Tag tag =
(org.htmlparser.tags.Tag)node;
                if( tag.isEndTag())
                    break;
            }
        }
        if(null == node)
            break;
    }
    else if (node instanceof StringNode)
    {
        stringText = node.toPlainTextString();
        if("".equals(title) )
            continue;
    }
}
```

```

        stringText = stringText.replaceAll("[ \\t\\n\\f\\r ]+", " ");
        stringText = TextHtml.html2text(stringText.trim());
        if (!"".equals(stringText))
        {

            //System.out.println("stringText.len:"+stringText.length());
            this.body.append(stringText);
            this.body.append(" ");
            //                System.out.println(this.body);
        }
    }
    else if (node instanceof TagNode)
    {
        TagNode tagNode = (TagNode)node;
        String name = ((TagNode)node).getTagName();
        if(name.equals("OPTION"))
        {
            //omit option
            lexer.nextNode ();
            lexer.nextNode ();
            //System.out.println("tag name:"+name);
        }
        else if (name.equals("A") && !tagNode.isEndTag() && expand)
        {
            String href = tagNode.getAttribute("HREF");

            String urlDesc = null;
            node = lexer.nextNode ();
            if(node instanceof StringNode)
            {
                StringNode sNode = (StringNode)node;
                String title = sNode.getText().trim();

                if(title.length()>=4)
                {
                    urlDesc= title;
                    //System.out.println("next node:"+title);
                }
            }

            addLink(this.url, href,urlDesc);
        }
        else if (name.equals("FRAME") &&
            !tagNode.isEndTag() && expand)
    }

```

```

        {
            // FRAME SRC=

addLink(url,include,tagNode.getAttribute("SRC"),null);
            // handle internal frame (iframes) as well
        }
        else if (name.equals("TITLE") &&
            !tagNode.isEndTag() )
        {
            node = lexer.nextNode ();
            stringText = node.toPlainTextString().trim();
            if(!"".equals(stringText))
            {
                this.title=stringText;
            }
        }
        else if(name.equals("META") && charSet == null)
        {
            String contentCharSet =
tagNode.getAttribute("CONTENT");
            charSet = URLSummary.getCharset (contentCharSet);
            tryAgain = true;
            break;
        }
    }
}

//如果在Meta信息中检测到新的字符编码，则需要按照meta信息中的编码再次解析网页。
if (tryAgain)
{
    this.body = new StringBuffer();

    try {
        uc = (HttpURLConnection) uc.getURL().openConnection();
        lexer=new Lexer (new Page(uc.getInputStream(), charSet));
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    lexer.setNodeFactory(typicalFactory);
}

```

```
while (null != (node = lexer.nextNode ()))
{
    //System.out.println("node:"+node);
    //omit script tag
    if (node instanceof org.htmlparser.tags.ScriptTag)
    {
        while (null != (node = lexer.nextNode ()))
        {
            if (node instanceof org.htmlparser.tags.Tag)
            {
                org.htmlparser.tags.Tag tag =
(org.htmlparser.tags.Tag)node;
                if( tag.isEndTag() &&
"SCRIPT".equals(tag.getTagName()) )
                {
                    break;
                }
            }
        }
        if(null == node)
            break;
    }
    //omit script tag
    else if (node instanceof org.htmlparser.tags.StyleTag)
    {
        while (null != (node = lexer.nextNode ()))
        {
            if (node instanceof org.htmlparser.tags.Tag)
            {
                org.htmlparser.tags.Tag tag =
(org.htmlparser.tags.Tag)node;
                if( tag.isEndTag())
                    break;
            }
        }
        if(null == node)
            break;
    }
    else if (node instanceof StringNode)
    {
        stringText = node.toPlainTextString();
        if("".equals(title) )
            continue;
    }
}
```

```

        stringText = stringText.replaceAll("[ \\t\\n\\f\\r ]+", "
");

        stringText = TextHtml.html2text(stringText.trim());
        if (! "".equals(stringText))
        {
            this.body.append(stringText);
            this.body.append(" ");
        }
    }
    else if (node instanceof TagNode)
    {
        TagNode tagNode = (TagNode)node;
        String name = ((TagNode)node).getTagName();
        if(name.equals("OPTION"))
        {
            //omit option
            lexer.nextNode ();
            lexer.nextNode ();
        }
        else if (name.equals("A") && !tagNode.isEndTag() &&
expand)
        {
            String href = tagNode.getAttribute("HREF");

            String urlDesc = null;
            node = lexer.nextNode ();
            if(node instanceof StringNode)
            {
                StringNode sNode = (StringNode)node;
                String title = sNode.getText().trim();

                if(title.length()>=4)
                {
                    urlDesc= title;
                }
            }

            addLink(this.url,include,href,urlDesc);
        }
        else if (name.equals("FRAME") &&
!tagNode.isEndTag() && expand)
        {
            // FRAME SRC=

```

```

addLink(url,include,tagNode.getAttribute("SRC"),null);
        // handle internal frame (iframes) as well
    }
    else if (name.equals("TITLE") &&
        !tagNode.isEndTag() )
    {
        node = lexer.nextNode ();
        stringText = node.toPlainTextString().trim();
        if(!"".equals(stringText))
        {
            this.title=stringText;
        }
    }
}
}
}
}
}
}
}
}
}
}

```

4.1.1 HtmlParser 介绍

Htmlparser 是一个良好实现的提取 Html 文件解析程序库。可以使用它完成对非规范的 HTML 文件解析。HtmlParser 将网页转换成一个个串联的 Node。例如下面把一个 url 中的标签都打印出来。

```

lexer = new Lexer (url.openConnection ());
while (null != (node = lexer.nextNode ()))

    System.out.println (node.toString ());

```

Node分成三类:

- RemarkNode:代表Html中的注释;
- TagNode: 标签节点, 是种类最多的节点类型, 上述Tag的具体节点类都是TagNode的实现;
- TextNode: 文本节点。

例如:

```

<!--enpproperty <date>2006-01-06</date><author></author><title>广告业务
</title><nodename></nodename><cnodename>联系我们
</cnodename><parentid>2788</parentid><parentname>BTV广告

```

```
</parentname><Pparentid>2781</Pparentid><Pparentname>网站栏目
</Pparentname><source></source><keyword></keyword><SiteID>1</SiteID><ip></ip><ChannelID>2781
</ChannelID><imp>0</imp><subtitle></subtitle><introtitle></introtitle> /enpproperty-->
```

这整段就是Remark Node。

例如：

```
<font class="nrbt">广告业务</font>
```

由下面三个Node组成：

```
<font class="nrbt">是TagNode，
```

```
广告业务 是Text Node，
```

```
</font>也是 TagNode，不过它的 isEndTag()是真。
```

各种不同的 HTML 标签则用 Node 的子类 Tag 表示，Html 标签是 Tag，而像字符正文则是 StringNode。

在 Lexer 类中实现了对 HTML 语法的基本解析，也就是把页面中的字符组装成一个一个的 Node。其中用到 Cursor 类来标定标记的位置，以 Lexer 中的解析字符串为例：

```
protected Node parseString (Cursor cursor, boolean quotesmart)
    throws
        ParseException
{
    boolean done;
    char ch;
    char quote;
    Node ret;

    done = false;
    quote = 0;
    while (!done)
    {
        ch = mPage.getCharacter (cursor);
        if (0 == ch)
            done = true;
        else if (0x1b == ch) // escape
        {
            ch = mPage.getCharacter (cursor);
            if (0 == ch)
                done = true;
        }
    }
}
```



```

else if ('$' == ch)
{
    ch = mPage.getCharacter (cursor);
    if (0 == ch)
        done = true;
    else if ('B' == ch)
        scanJIS (cursor);
    else
    {
        cursor.retreat ();
        cursor.retreat ();
    }
}
else
    cursor.retreat ();
}
else if (quotesmart && (0 == quote) && (('\' == ch) || ('"
== ch)))
    quote = ch; // enter quoted state
// patch contributed by Gernot Fricke to handle escaped closing
quote
else if (quotesmart && (0 != quote) && ('\\' == ch))
{
    ch = mPage.getCharacter (cursor); //try to consume escaped
character
    if ( (ch != '\\') // escaped backslash
        && (ch != quote)) // escaped quote character
        // ( reflects [" or [' whichever opened the
quotation)
        cursor.retreat(); // unconsume char if character was not
an escapable char.
    }
else if (quotesmart && (ch == quote))
    quote = 0; // exit quoted state
else if ((0 == quote) && ('<' == ch))
{
    ch = mPage.getCharacter (cursor);
    if (0 == ch)
        done = true;
    // the order of these tests might be optimized for speed:
    //change by luogang //add '?' == ch
    else if ('/' == ch || Character.isLetter (ch) || '!' == ch
|| '%' == ch || '?' == ch )
    {

```

```

        done = true;
        cursor.retreat ();
        cursor.retreat ();
    }
    else
    {
        // it's not a tag, so keep going, but check for quotes
        cursor.retreat ();
    }
}

return (makeString (cursor));
}

```

4.1.2 结构化信息提取

从 HTML 提取有效的文本，经常碰到的有两种类型。一种是针对特定的网页特征提取结构化信息，还有一种就是通用的网页去噪。下面首先介绍结构化信息的提取方法。

例如我们需要从下面这个职位发布的网页提取公司名称：

北京亿达网通科技发展有限公司

[查看公司简介](#)

公司行业：计算机服务（系统、数据服务） 公司性质：民营/私营公司 公司规模：少于50人

项目经理					
电子邮箱：	yidawangtong@163.com				
发布日期：	2007-11-15	工作地点：	北京市	招聘人数：	1
外语要求：	英语 良好	薪水范围：	面议	学 历：	本科
工作年限：	五年以上				

职位描述：

- 1、根据客户的需求撰写项目建议书与方案、制作标书；
- 2、制定项目实施计划、组织成员完成各项具体工作、控制项目的进度、最终成功的完成预期的项目；
- 3、组织对客户进行产品知识和技术应用方面的指导和培训；

基本要求：

- 1、大学本科以上学历，理工类计算机、信息工程、自动控制等相关专业，5年以上工作经验；
- 2、有丰富的系统集成专业知识和成功的项目管理经验；
- 3、具备较强的团队领导能力、沟通能力、项目进度掌控能力和突发事件的处理能力；
- 4、有强烈的责任心、工作热情和积极主动性。

[立即申请职位](#)

[马上填写简历](#)

[放入职位收藏夹](#) [介绍给朋友](#) [该公司其他职位](#)

查看公司名称周围的特征：

<td align="left" class="title02">北京亿达网通科技发展有限公司</TD>

可以利用下面这个 Filter 来处理：

```
//提取公司名字的Filter
NodeFilter filter_title = new AndFilter(new
TagNameFilter("TD"),

    new HasAttributeFilter("class","title02"));
```

然后就可以提取公司名称:

```
NodeList nodelist =
parser.extractAllNodesThatMatch(filter_title);
Node node_title = nodelist.elementAt(0);
String comName = extractText(node_title.toHtml());

comName = comName.replaceAll("[\t\n\f\r ]+","");
```

定义好一个职位类:

```
public class Job {
    public String comName = null;
    public String positionName = null;
    public String email = null;
    public String releaseData = null;
    public String city = null;
    public String number = null;
    public String experience = null;
    public String salary = "面议";
    public String knowledge = null;
    public String acceptResumeLanguage = null;
    public String positionDescribe = null;
    public String comintro = null;
    public String comHomePage = null;
    public String comAddress = null;
    public String postCode = null;
    public String fax = null;
    public String connectPerson = null;
    public String telephone = null;
    public String languageAbility = null;
    public String funtype_big = null;
    public String funtype = null;
    public String province = null;

    public String toString(){
        return "公司名称: "+comName+"\n职位名称: "+positionName+"\n电子邮箱:
"
        +email+"\n发布日期: "+releaseData+"\n工作地点: "+city+"\n招聘人数:
"
```

```

        +number+"\n工作年限: "+experience+"\n薪水范围: "+salary+"\n学历:
"+knowledge
        +"\n接受简历语言: "+acceptResumeLanguage+"\n职位描述:
"+positionDescribe+"\n公司简介: "
        +comintro+"\n公司网站: "+comHomePage+"\n地址: "+comAddress
        +"\n邮政编码: "+postCode+"\n传真: "+fax+"\n联系人:
"+connectPerson+"\n电话: "+telephone+"\n外语要求: "+languageAbility;
    }

```

完整的提取方法如下:

```

public Job extractContent(String url){

    Job position = new Job();
    try{
        Parser parser = new Parser(url);
        //设置编码方式
        parser.setEncoding("GB2312");
        //提取公司名字的Filter
        NodeFilter filter_title = new AndFilter(new
TagNameFilter("TD"),
            new HasAttributeFilter("class","title02"));

        //提取公司名称
        NodeList nodelist =
parser.extractAllNodesThatMatch(filter_title);
        Node node_title = nodelist.elementAt(0);
        position.comName = extractText(node_title.toHtml());
        position.comName =
position.comName.replaceAll("[\t\n\f\r ]+","");

        //提取职位名称的Filter
        NodeFilter filter_job_name = new AndFilter(new
TagNameFilter("td"),
            new HasAttributeFilter("bgcolor","#FFEEEE"));

        NodeFilter job_description_end1 = new AndFilter(new
TagNameFilter("table"),
            new HasAttributeFilter("width","100%"));
        NodeFilter job_description_end2 = new
HasAttributeFilter("cellpadding","5");
        NodeFilter company_description = new AndFilter(new
TagNameFilter("table"),
            new HasAttributeFilter("width","98%"));
    }
}

```

```

        //提取职位的名称
        parser.reset();
        nodelist.removeAll();
        nodelist=parser.extractAllNodesThatMatch(filter_job_name);
        Node node_job_name = nodelist.elementAt(0);
        position.positionName = extractText(node_job_name.toHtml());
        position.positionName =
position.positionName.toString().replaceAll("[\t\n\f\r ]+", "");

        //提取职位描述
        NodeFilter job_description_end = new
AndFilter(job_description_end1, job_description_end2);
        parser.reset();
        NodeList nodelist_description =
parser.extractAllNodesThatMatch(job_description_end);
        Node node_job_description =
nodelist_description.elementAt(0);
        position.positionDescribe =
extractText(node_job_description.toHtml());
        position.positionDescribe =
position.positionDescribe.replaceAll("[\t\n\f\r ]+", " ");

        //提取公司简介
        parser.reset();
        nodelist_description.removeAll();
        nodelist_description =
parser.extractAllNodesThatMatch(company_description);

        Node node_company_description =
nodelist_description.elementAt(0);

        //处理公司简介
        position.comintro =
doCompanyDescription(node_company_description);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return position;
    }
}

```

上面这个方法，输入一个 url 地址，返回提取好正文的职位类。

自动提取结构化信息的关键是从同样类型的实例中发现编码模版。一个自然的方法是从HTML 编码字符串检测重复的模式。检测的方法有字符串编辑距离和树编辑距离。

计算两个字符串的编辑距离的实现如下：

```
public class Distance {
    // *****
    // Get minimum of three values
    // *****
    private static int Minimum(int a, int b, int c) {
        int mi;
        mi = a;
        if (b < mi) {
            mi = b;
        }
        if (c < mi) {
            mi = c;
        }
        return mi;
    }

    /**
     *
     * @param s 输入源串
     * @param t 输入目标串
     * @return 源串和目标串之间的编辑距离
     */
    public static int LD(String s, String t) {
        int d[][]; // matrix
        int n; // length of s
        int m; // length of t
        int i; // iterates through s
        int j; // iterates through t
        char s_i; // ith character of s
        char t_j; // jth character of t
        int cost; // cost

        // Step 1 初始化
        n = s.length();
        m = t.length();
        if (n == 0) {
            return m;
        }
        if (m == 0) {
            return n;
        }
    }
}
```

```

    }
    d = new int[n + 1][m + 1];

    // Step 2 Initialize the first row to 0..n.
    for (i = 0; i <= n; i++) {
        d[i][0] = i;
    }

    //Initialize the first column to 0..m.
    for (j = 0; j <= m; j++) {
        d[0][j] = j;
    }

    // Step 3 Examine each character of s (i from 1 to n).
    for (i = 1; i <= n; i++) {
        s_i = s.charAt(i - 1);

        // Step 4 Examine each character of t (j from 1 to m).
        for (j = 1; j <= m; j++) {
            t_j = t.charAt(j - 1);

            // Step 5
            // If s[i] equals t[j], the cost is 0.
            // If s[i] doesn't equal t[j], the cost is 1.
            if (s_i == t_j) {
                cost = 0;
            } else {
                cost = 1;
            }

            // Step 6
            //Set cell d[i,j] of the matrix equal to the minimum of:
            //a. The cell immediately above plus 1: d[i-1,j] + 1.
            //b. The cell immediately to the left plus 1: d[i,j-1] +
1.
            //c. The cell diagonally above and to the left plus the cost:
d[i-1,j-1] + cost.
            d[i][j] = Minimum(d[i - 1][j] + 1, d[i][j - 1] + 1,
                d[i - 1][j - 1] + cost);
        }
    }

    // Step 7
    // After the iteration steps (3, 4, 5, 6) are complete, the distance

```

```

is found in cell d[n,m].
    return d[n][m];
}
}

```

两个树 A 和 B 之间的树编辑距离是从 A 转换成 B 所需要的最小操作次数。可以包括如下的操作：

- 删除节点；
- 插入节点；
- 替换节点。

每个操作赋值一个代价权重。

“Zhang and Shasha algorithm”采用动态规划的方法实现了树编辑距离(Tree Distance)的计算。headliner.treedistance 中实现了“Zhang and Shasha algorithm”。

4.1.3 网页去噪

一个页面中的经常包括导航栏或底部的公司介绍等信息，这样的信息在很多页面都会出现，可以看作噪音信息。例如下面两段文字：

“今天是： 当前位置:新闻中心-风光快讯 我公司新产品 矿用提升机高压变频器、油田钻机专用变频器 成功通过科技成果暨新产品鉴定会 会议在山东济南市玉泉森信大厦泉荷厅举行 鉴定委员会主任委员陈伯时教授在认真审查资料 公司技术人员做技术研究报告 2005 年 12 月 18 日，在济南由山东省科技厅、山东省经济贸易委员会联合组织并主持召开了山东新风光电子科技发展有限公司研制的 JD-BP 系列矿井提升机专用高压变频调速器和 JD-BP 系列石油钻机专用变频调速器的技术鉴定会。鉴定委员会听取了新风光电子公司的研制工作报告、技术总结报告、经济效益分析报告、国家电控配电设备质量监督检验中心检验报告及河北省峰峰集团的用户运行报告，观看了生产试验现场和样机运行现场的影像资料，并审查了会议的全套资料。 鉴定委员会由被业内尊为变频器之父的上海大学教授、博士生导师陈伯时担任鉴定委员会主任委员。 鉴定委员会一致认为，风光石油钻机变频器符合石油钻机工况要求，技术指标先进、可靠性高，具有较高的性价比，率先实现了石油钻机专用变频器的国产化，可替代国外同类产品，达到国内领先水平。 鉴定委员会一致认为，JD-BP 系列矿井提升机高压变频调速器具有技术指标先进、可靠性高等优点，在级联高压逆变器能量回馈控制方面填补了国内空白，达到国内领先水平，具有较大的经济、社会效益和推广价值，可以批量生产。 ||| 山东新风光电子科技发展有限公司©2004-2005 版权所有”

和

“今天是： 当前位置:新闻中心-风光快讯 公司当选为中国电器工业协会 变频器分会副理事长单位 07 年 3 月 30 日，在天津召开的 中国电器工业协会变频器分会 成立大会上，公司当选为中国电器工业协会 变频器分会副理事长单位。 其他相关信息：（按原文顺序） 理事长：天津电气传动设计研究所 副理事长：冶金自动化研究设计院 西门子（中国）有限公司 成都希望森兰变频器制造有限公司 上海电器科学研究所（集团）有限公司 山东新风光电子科技发展有限公司 深圳市普传科技有限公司 上海发电设备成套设计研究院 北京利德华福电气技术有限公司 会员单位：100 家，（名单略）。 撰稿：李明伦 ||| 山东新风光电子科技发展有限公司©2004-2005 版权所有”

有多种方法解决这个问题，一个思路是，利用多个网页的正文信息比对的方法，把公共子串检测出来，较长的公共子串可以看成是噪音信息。从两个字符串中提取公共子串的方法如下：

```
public static String getLCString(String str1, String str2) {
    StringBuffer str = new StringBuffer();
    int i, j;
    int len1, len2;
    len1 = str1.length();
    len2 = str2.length();
    int maxLen = len1 > len2 ? len1 : len2;
    int[] max = new int[maxLen];
    int[] maxIndex = new int[maxLen];
    int[] c = new int[maxLen];

    for (i = 0; i < len2; i++) {
        for (j = len1 - 1; j >= 0; j--) {
            if (str2.charAt(i) == str1.charAt(j)) {
                if ((i == 0) || (j == 0))
                    c[j] = 1;
                else
                    c[j] = c[j - 1] + 1;
            } else {
                c[j] = 0;
            }

            if (c[j] > max[0]) {
                max[0] = c[j];
                maxIndex[0] = j;

                for (int k = 1; k < maxLen; k++) {
                    max[k] = 0;
                    maxIndex[k] = 0;
                }
            } else if (c[j] == max[0]) {
                for (int k = 1; k < maxLen; k++) {
```

```

        if (max[k] == 0) {
            max[k] = c[j];
            maxIndex[k] = j;
            break;
        }
    }
}

for (j = 0; j < maxLen; j++) {
    if (max[j] > 0) {
        str.append(str1.substring(maxIndex[j] - max[j] +
1,maxIndex[j]+1));
    }
}
return str.toString().trim();
}

```

提取长度大于 20的最多五个公共子串。

```

public static ArrayList<String> removeContent(String str1,String
str2){
    ArrayList<String> com = new ArrayList<String>(5);

    //最多提取5段公共子串
    for(int i = 0;i<5;i++){
        String commonStr = LCString.getLCString(str1, str2);
        if(commonStr.length()<20){
            break;
        }
        com.add(commonStr);
        str2 = str2.replace(commonStr, "");
    }
    return com;
}

```

提取出这样的公共子串后，把它从正文删除即可。

```

ArrayList<String> com = LCString.removeContent(tempbody, body);
if(com!=null){
    for(int i = 0;i<com.size();i++){
        if(body.contains(com.get(i))){

```

```

        body = body.replace(com.get(i), "");
    }
}
}

```

利用网页的 DOM 树还可以去掉 FORM ,Select, IFRAME ,INPUT, STYLE 中的节点。

4.1.4 网页结构相似度计算

为了确定两个网页是否由同一个网页模板生成，需要计算两个网页的结构相似度。一个简单的实现方法：

1. 把网页抽象成一个 Node 数组；
2. 比较 Node 数组的编辑距离。

第一步，由网页生成一个Node数组的过程：

```

public static int getPageDistance(String urlStr1,String urlStr2)
throws ParseException, IOException
{
    ArrayList<Node> pageNodes1 = new ArrayList<Node>();

    URL url = new URL(urlStr1);
    Node node;
    Lexer lexer = new Lexer (url.openConnection ());
    lexer.setNodeFactory(new PrototypicalNodeFactory ());
    while (null != (node = lexer.nextNode ()))
    {
        pageNodes1.add(node);
    }

    ArrayList<Node> pageNodes2 = new ArrayList<Node>();
    URL url2 = new URL(urlStr2);

    lexer = new Lexer (url2.openConnection ());
    lexer.setNodeFactory(new PrototypicalNodeFactory ());
    while (null != (node = lexer.nextNode ()))
    {
        pageNodes2.add(node);
    }
}

```

```
return PageDistance.LD(pageNodes1, pageNodes2);  
}
```

第二步，计算编辑距离的过程：

```
if (s_i.getClass().equals(t_j.getClass())) {  
    cost = 0;  
} else {  
    cost = 1;  
}
```

4.1.5 正文提取的工具 FireBug

用 DOM 查看器查看树型结构组织的网页。

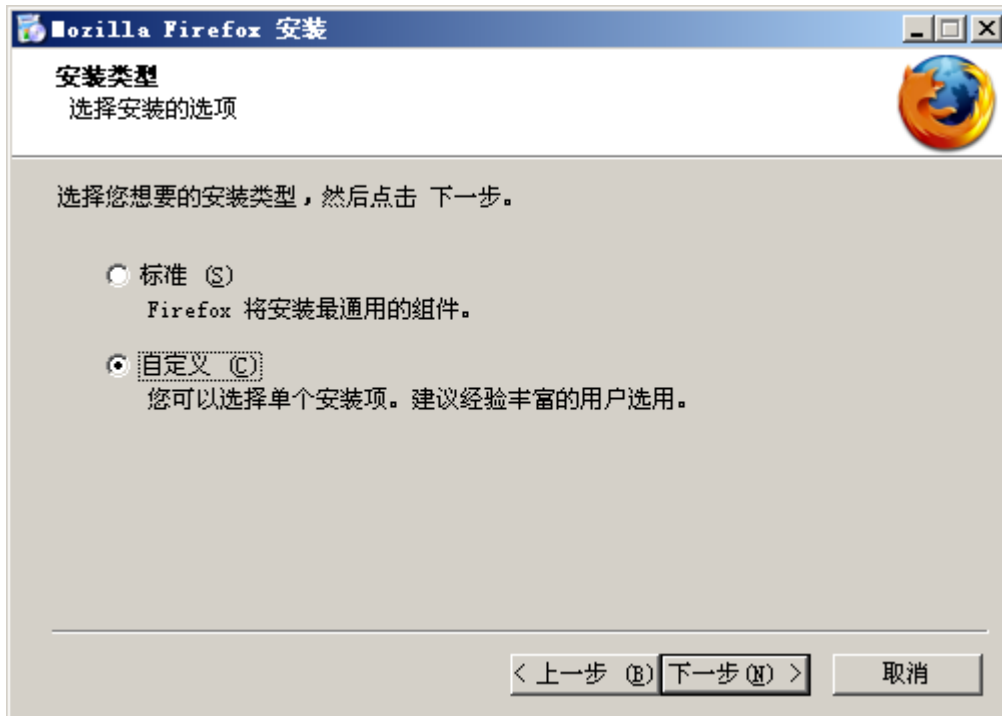
DOM 查看器能够查看任何页面的已解析的文档对象模型(DOM)。可以获得每个 HTML 元素、属性和文本节点的详细信息；也可以看到每个页面样式表中的所有 CSS 规则；也可以看到每个对象的所有脚本属性。它的功能非常强大。

DOM 查看器包含在 Firefox 的安装程序中，但是由您的平台而定，它可能默认没装。如果您在“工具”菜单中看不到“DOM 查看器”，那么您需要重新安装 Firefox。重新安装 Firefox 不会影响您现有的书签，设置，扩展以及用户脚本。

安装 DOM 查看器过程如下：

运行 Firefox 的安装程序。

接受用户协议后，选择自定义安装。

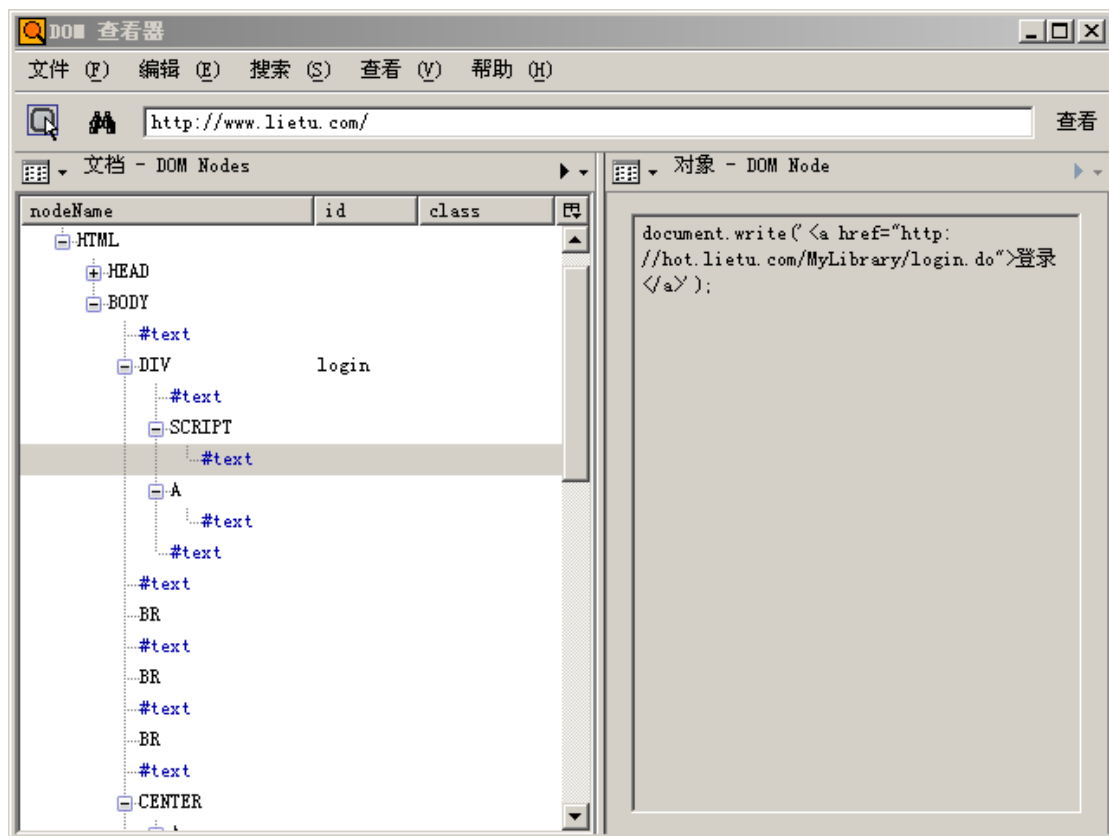


选择安装路径后，安装向导会询问安装的组件。选择 DOM 查看器。



安装结束后，运行 Firefox。您会看到“工具”→“DOM 查看器”。

访问 <http://www.lietu.com/>。在菜单中，选择“工具”→“DOM 查看器”，打开 DOM 查看器。在 DOM 查看器的左侧视图中，会看到 DOM 节点的树状图。



如果觉得依次展开 DOM 节点树中每层很不方便,可以使用 Inspect Element 扩展,它能迅速找到 DOM 查看器中的指定元素。

4.1.6 正文提取的工具 NekoHTML

NekoHTML 是一个简单的 HTML 扫描器和标签补偿器(tag balancer),使得程序能解析 HTML 文档并形成标准的 DOM 树,下面的程序自顶向下打印输出 DOM 树:

```
public static void main(String[] argv) throws Exception {
    DOMParser parser = new DOMParser();
    parser.parse("http://www.lietu.com");
    print(parser.getDocument(), "");
}

public static void print(Node node, String indent)
{
    //System.out.println("打印该节点");
    if (node.getNodeValue() != null)
    {
        if(!"".equals(node.getNodeValue().trim()))
        {
            System.out.print(indent);
            System.out.println(node.getNodeValue());
        }
    }
}
```

```

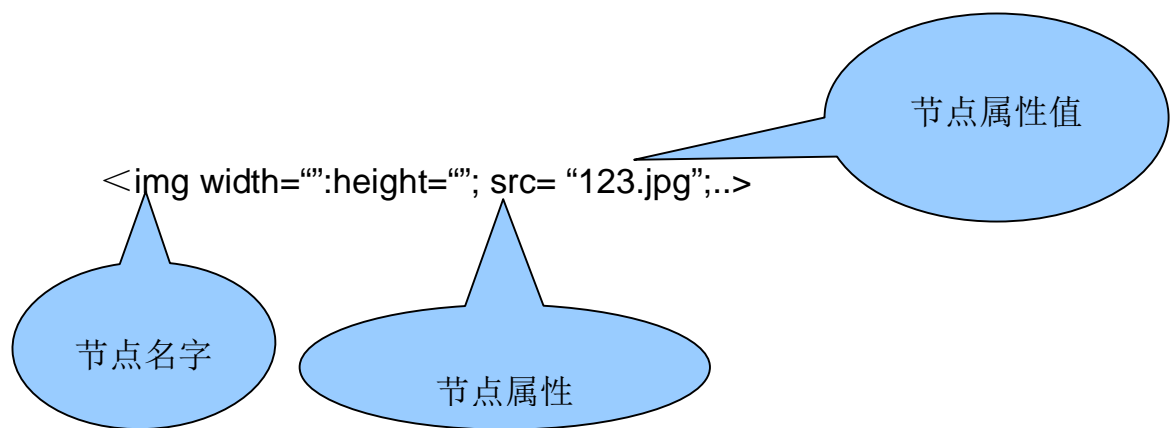
    }
}

//System.out.println("取得第一个孩子节点");
Node child = node.getFirstChild();

while (child != null)
{
    print(child, indent+" ");
    child = child.getNextSibling();
    //System.out.println("取得该节点的兄弟节点");
}
//System.out.println("打印该节点结束");
}

```

基 本 概 念 :



xerces-2_9_1 包括了一个 `LSSerializer` 对象可以保存 DOM 解析后的文档或节点，将包含内容部分的主节点输出到控制台：

```

registry = DOMImplementationRegistry.newInstance();

DOMImplementationLS impl =

(DOMImplementationLS)registry.getDOMImplementation("LS");
LSSerializer writer = impl.createLSSerializer();
LSOutput output = impl.createLSOutput();
output.setByteStream(System.out);
output.setEncoding(System.getProperty("file.encoding"));

writer.write(iNode, output);

```

还可以使用 `writeToString` 方法将文档或文档中的节点写入字符串：

```
String nodeString=domWriter.writeToString(iNode);
```

常用方法介绍：

`removeAttribute` 一移除节点属性

```
private void removeAttribute(final Node iNode, final String iAttr)
{
    iNode.getAttributes().removeNamedItem(iAttr);
}
```

4.1.7 网站风格树去除文档噪音

每个超文本标记语言页对应于一个 DOM 树，其中标签是内节点并且详细的文本，图象或者超链接是叶节点。图 2 示出了超文本标记语言代码的一部分以及它相应的 DOM 树。在 DOM 树中，每个长方形是标签节点。阴影方块是结点的实际内容，例如：标签 `IMG`，它的实际内容是“`src=image.gif`”。注意这一点，我们从 `BODY` 标签开始研究 HTML 网页，因为所有的可视部分都在 `BODY` 范围内。每个节点与它的显示属性相连。为便于分析，在该 DOM 树内，我们增加一个虚拟根目录结，`root` 节点没有任何属性，作为 `BODY` 的父亲结点。

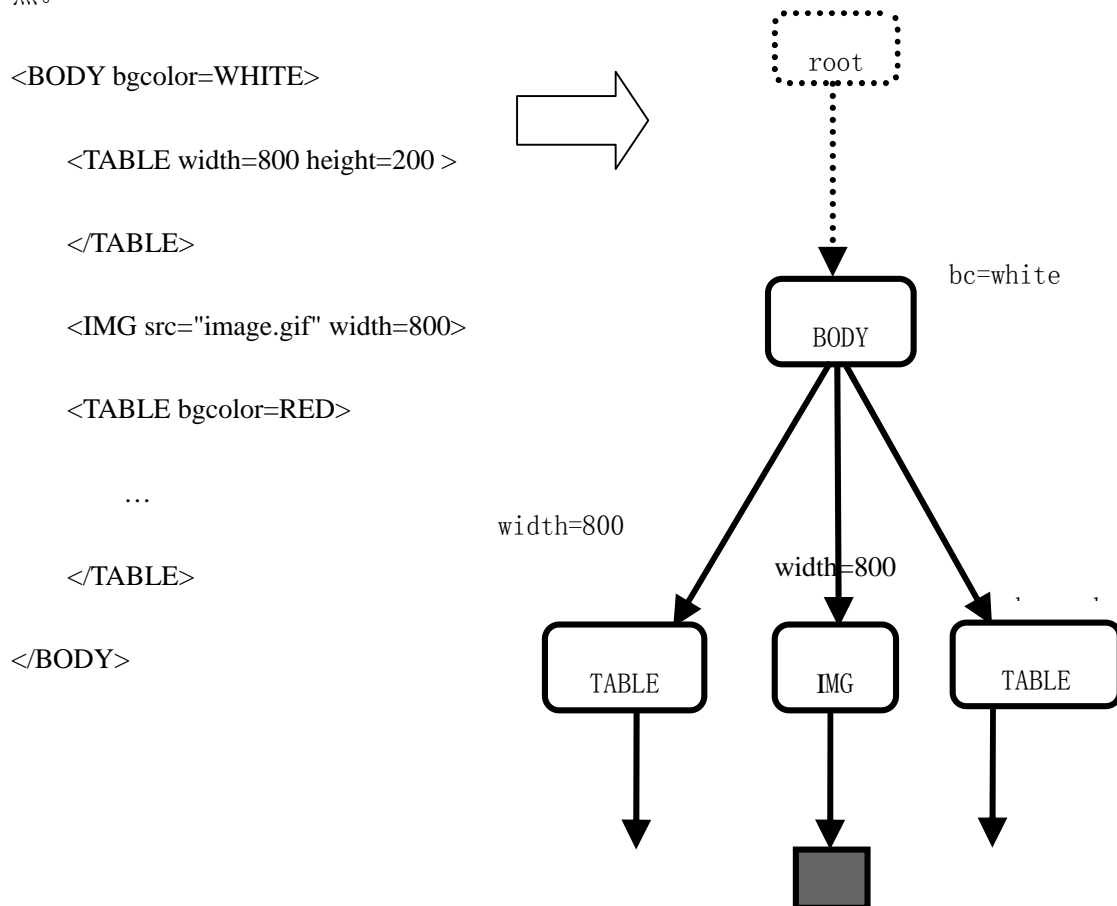


图 2：一个 DOM 树的例子（较低水平的标签被删除）

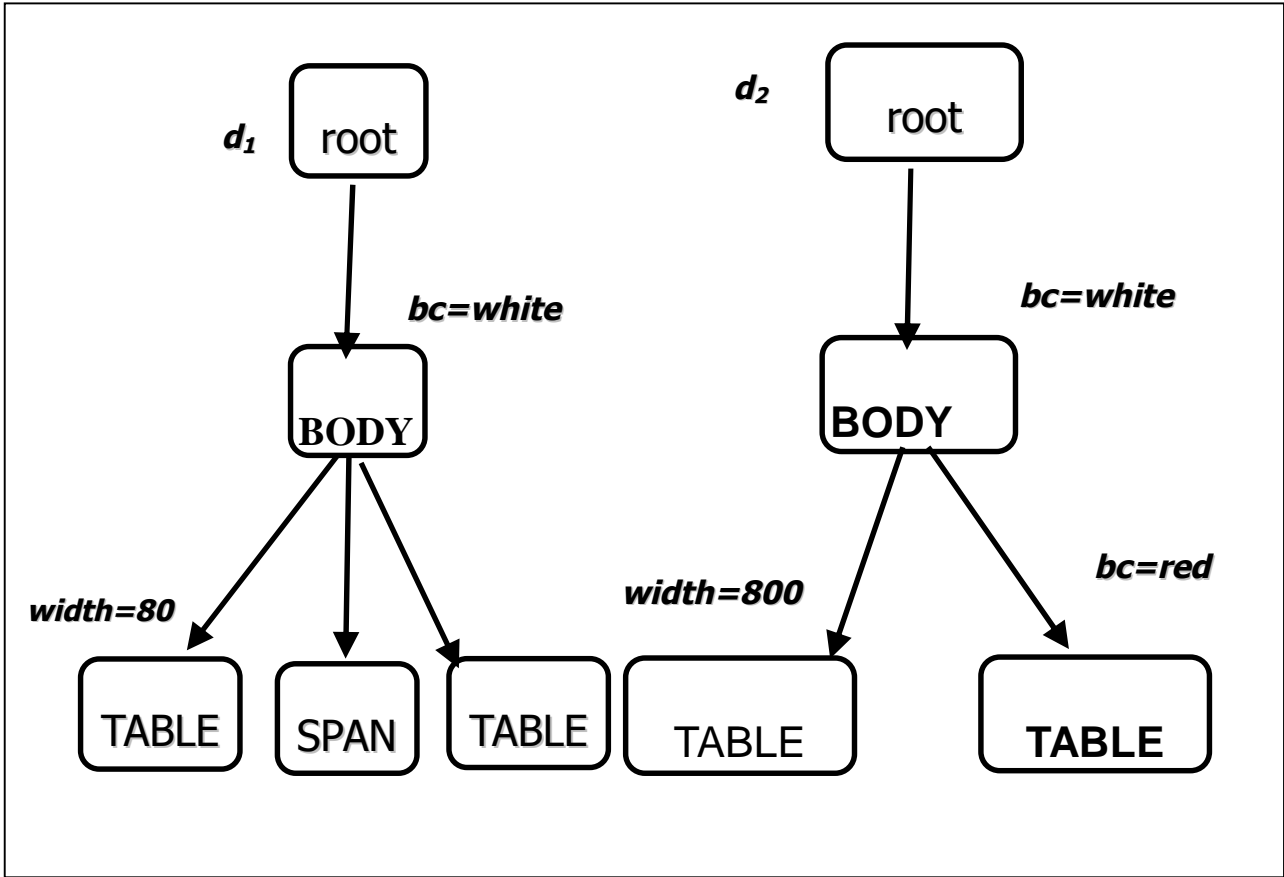
虽然 DOM 树已经足以描述此布局，或者描述一个单一超文本标记语言页的呈现风格，但是我们很难学习所有的呈现风格以及一组超文本标记语言页全部内容并且很难清除基于单个单一超文本标记语言页的内容。因此，在同时考虑呈现风格和网页的实际内容时，DOM 树在我们做清除工作时已不再适用。为此，我们需要一个更强有力的结构。这个结构至关重要，因为我们的算法需要它从一个站点发现网页的共同的风格，以便消灭噪声。我们引入一种新的树结构，把它称为风格树(ST)，该树结构能压缩一组相关网页的共同呈现风格。

图3给出了一风格树的例子，作为DOM树d1和d2的联合体。我们观察到，除了四个标（P,IMG,P 和 A）位于底层，d1中的所有标签在标签d2中都有其相应的标签。因此，d1和d2是可以被压缩的。

我们使用计数器来指示在风格树的一个特定层上具有一种特定风格的网页有多少。在图3中，我们能看到两个网页都以BODY开始，并且因此BODY具有一计数2。

Below BODY, 两页也有同样的TABLE-IMG-TABLE风格。我们把这整个TABLE-IMG-TABLE 标签序列叫做风格节点，他们包围在图3中虚线长方形框内。此时，它代表一种特定的呈现风格。因此，在DOM树中，一个风格节点是一个标签节点序列。我们把这些标签节点称为元素节点，以便在DOM树内把他们从标签节点中鉴别出来。例如，TABLE-IMG-TABLE 风格节点具有三个元素节点，TABLE, IMG 和 TABLE。

在一个 DOM 树中，来自一个标记结的元素结也包含一些不同的信息，以后将被定义。



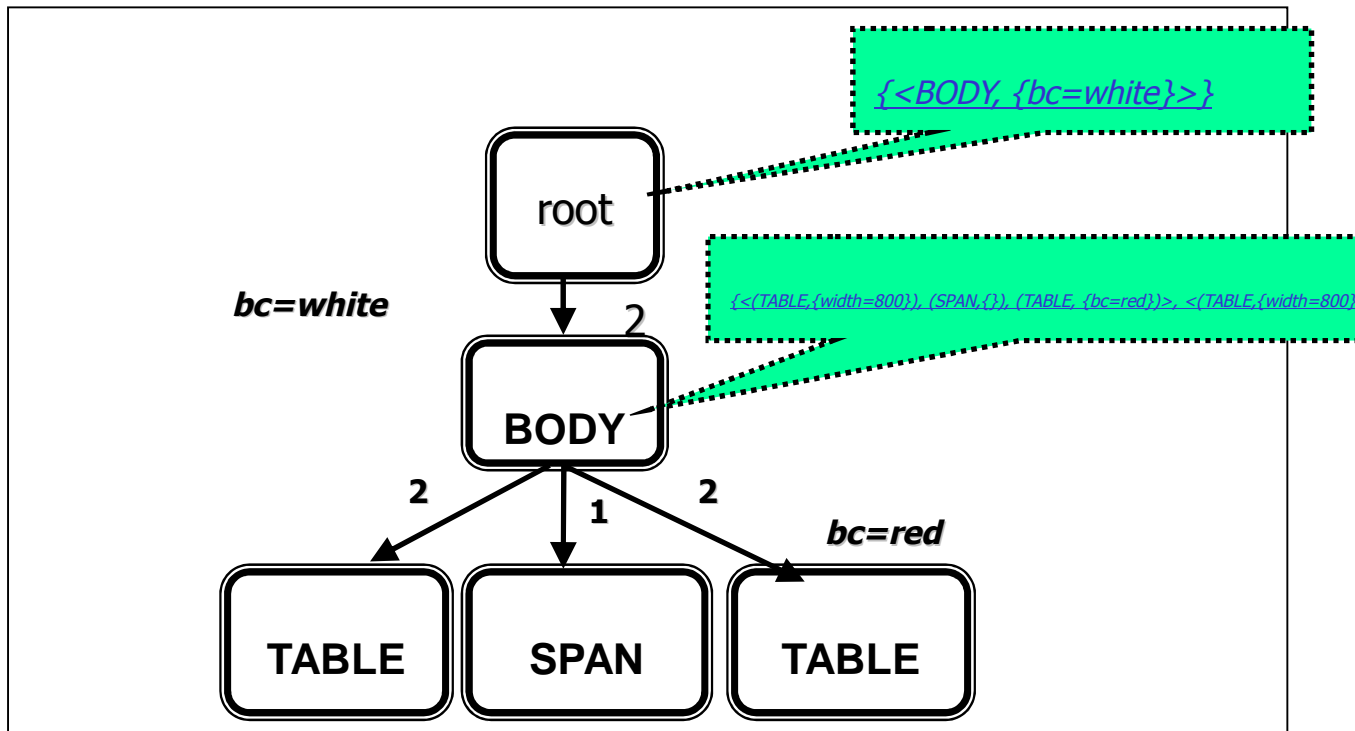


图3: DOM树及风格树

在图3, 我们可以看到右侧大多数TABLE标签, d1 and d2分岔, 在风格树中由二个不同风格的节点所反映。二个风格节点分别是P-IMG-P-A和P-BR-P。这意味着在右侧TABLE节点之下, 我们有两种不同的呈现风格。这两个风格节点的网页计数器都是1。很清楚, 风格树是两个DOM树的压缩表现形式。它使我们看哪DOM树的那些部分是共同的, 那些部分是不同的。

1. 风格树 (ST)

我们现在定义风格树, 其中包括两种类型的节点, 即风格节点和元素节点。

定义: 一种风格节点代表了一种布局或呈现风格, 它有两个组成部分, 记作 (ES, n), 其中 ES 是元素节点的一个序列 (见下文), 和 N 是网页的页码, 该页码具有这种在次节点层上的特定风格。

在图3中, 风格节点 (在一虚线方框内) 的 P - IMG-P-有4个元素节点, P, IM, P 和 A, 并且 n=1。

定义: 一个元素节点 E 有三个组成部分, 记作 (TAG, Attr, Ss), 其中

- TAG 是标签的名称, 例如, "TABLE" 和 "IMG";

- Attr 是 TAG 的一套显示属性, 例如, bgcolor = RED, weidth=100, 等等。

•Ss 是低于 E 的一套风格节点。

请注意，在 DOM 树内，一个元素节点对应一个标签节点，但是指向一组孩子风格节点 Ss（见图3）。为了方便起见，我们通常通过标签的名称标记元素节点，通过与风格的节点序列标记名称相应的元素节点序列标记一风格节点。

为网站的网页创建一个风格树（称为网站风格树或 SST）非常简单。首先我们给每一个网页创建一个 DOM 树，然后合并成由上而下样式的风格树。在风格树的一个特定元素节点 E 处（在风格树内），我们检查一下 DOM 树中 T 的子标签节点序列是否与位于风格节点 S 下方元素节点的序列相同。如果相同，我们只是增加了风格节点 S 的网页计数器，然后合并风格树和和 DOM 树中剩下的节点。如果不相同，风格树中的元素节点 E 的下方就可以创建一个新的风格节点。在转换为节点和元素节点的风格树后，DOM 树中标签节点 T 的子树复制到风格树内。

2. 在风格树内确定噪声元素

在我们的工作中，噪声的定义是基于以下假设：（1）一个要素节点具有的呈现风格越多，它越重要，反之亦然。（2）某个元素节点的实际内容具有的分岔越多，该元素节点越重要，反之亦然。这两个重要性值用于评估的元素节点的重要性。表现其重要性的目的是用常规的呈现风格检测噪声，与此同时，介绍的内容重要性的目的是鉴别这些主要内容的网页能够以类似的呈现风格呈现。因此，以这种提出的设计方法，通过结合呈现重要性和内容重要性，给出元素节点的重要性。一个元素节点的结合重要性越大，就越有可能是主要的网页内容。

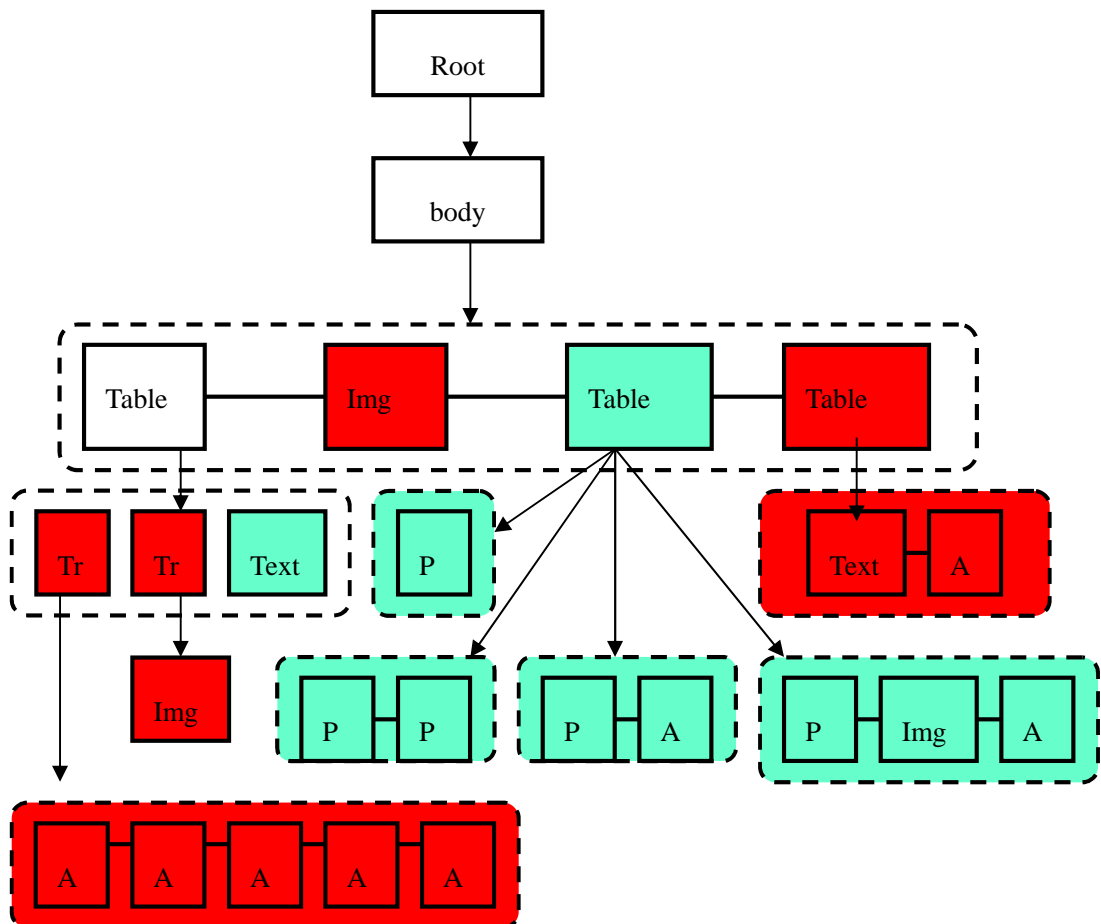


图4：网站风格树的一个例子（SST）

在图4的这个例子中，SST的阴影部分很可能是噪声，因为他们的呈现风格是很规则和重要的（连同其实际内容，在图中未示出），因此不那么重要。该双线TABLE元素节点有许多子风格节点，也就是说，该元素节点很可能是重要的。也就是说，双排队表更可能包含的主要内容的网页。特别是双线TEXT也有意义，因为它的内容是不同但其呈现风格是固定的。让使得创建风格树时需要采用一个网站的所有网页。

我们需要一个公式来衡量呈现风格的重要性。信息理论（或熵）是一种自然的选择。

定义（节点重要性）：在SST中，对于某个元素节点E，设M是包含E的网页数，L是E子风格节点的数（即 $l = |E.Ss|$ ），E的节点重要性，由 $NodeImp(E)$ 表示，定义为：

$$NodeImp(E) = \begin{cases} -\sum_{i=1}^l p_i \log_m p_i & \text{if } m > 1 \\ 1 & \text{if } m = 1 \end{cases} \quad (1)$$

其中， p_i 是在E.Ss中，网页使用ith风格节点的可能性。直观上，如果l小，E在不同的风格中被呈现出来的可能性就小。因此， $NodeImp(E)$ 的值小。如果E包含多种呈现风格，那么 $NodeImp(E)$ 的值就大。例如，在图4的SST，元素节点BODY的重要性是0（ $\log_{100} 1 = 0$ ），因为l=1。即，在BODY下，仅有一种呈现风格Table-Img-Table-Table。双线Table的重要性是：

$$-0.35 \log_{100} 0.35 - 2 * 0.25 \log_{100} 0.25 - 0.15 \log_{100} 0.15 = 0.292 > 0$$

然而，我们不能仅仅考虑节点重要性，说BODY是一个噪声，因为它并没有考虑其后代的重要性。我们使用合成重要性去测量一个元素节点及其后代的重要性。

定义（综合重要性）：在SST内，对于一个内部元素节点E，设 $l = |E.Ss|$ 。E的综合重要性，由 $CompImp(E)$ 表示，由 $CompImp(E)$ 表示。定义为，

$$CompImp(E) = (1 - \gamma') NodeImp(E) + \gamma' \sum_{i=1}^l (p_i compImp(s_i)) \quad (2)$$

，其中 p_i 是在E.Ss中E具有ith子风格节点的可能性。在上述等式中， $CompImp(S_i)$ 是一个风格节点 $S_i (\in E.Ss)$ 的综合重要性，定义为，

$$CompImpP(S_i) = \frac{\sum_{j=1}^k CompImpP(E_j)}{k} \quad (3)$$

其中 E_j 是在 $S_i.E$ 的一个元素节点，并且 $k = |S_i.Es|$ ，是在 S_i 中元素节点的数目。

在 (2)， γ 是衰减因字，设置为 0.9。当 L 大时，它增加了 $NodeImp(E)$ 的权重。当 L 小时，它减少了 $NodeImp(E)$ 的权重。这意味着，一个风格节点具有的子风格节点越多，其本身的综合重要性越大，所具有的子风格节点数越少，其子代的综合重要性越大。

叶节点与内部节点不同，因为它们只有实际内容没有标签。我们基于在其实际内容（即，文本，图片，连接，等等）中的信息定义一个叶元素节点的综合重要性。

定义：在 SST 中的一个叶元素节点，设 l 是出现在 E 中的特征的数目（即，文字，图片文件，连接参考，等等），并且设 m 是包含 E 的网页数目， E 的综合重要性定义为：

$$CompImpP(E) = \begin{cases} 1 & \text{if } m=1 \\ \frac{\sum_{i=1}^l H(a_i)}{l} & \text{if } m>1 \end{cases} \quad (4)$$

其中， a_i 是 E 中内容的实际 t 特征。 $H(a_i)$ 是在 E 中的 a_i 熵，

$$H_E(a_i) = \begin{cases} 0 & \text{if } m=1 \\ -\sum_{j=1}^m p_{ij} \log_m p_{ij} & \text{if } m>1 \end{cases} \quad (5)$$

其中 p_{ij} 是网页 j 的 E 中 a_i appears 出现的可能性

如果标记 $m=1$ ，这意味着只有一个网页含有 E ，那么 E 是一个非常重要的节点，并且它的

$CompImp$ 是 1（ $CompImp$ 的全部值在归一化成 0 与 1 之间的数值）。

所有的元素节点与风格检点的综合重要性计算（应用 $CalcCompImp(E)$ 程序）可以通过转换

SST来实现。

4.1.8 正文提取

网页有目录导航式页面和主体型页面（详细页面）。主体型页面需要抽取的正文信息包括标题和内容等。

详细页面的特征有：

- 文字较多（非锚文本）
- 一般都有明显的文本段落，文字较多，相应的标点符号也较多。
- URL 较长。在一般的 Web 网站链接导航树上，主题型网页主要分布于底层，多为叶节点。对于同一网站而言，主题型网页的 URL 相对较长。URL 体现了网站内容管理的层次，对于大型网站而言，URL 往往非常有规律。
- 链接较少。主题型网页的主体在于“文字”，相对于导航型网页，其链接数较少。

详细页面中网页噪音特征：

- 多以链接的形式出现
- 有很多锚文本，但标点符号较少
- 有许多常见的噪音文本，如版权声明等
- 在视觉上，多出现于网页的边缘

提取网页正文首先进行网页去噪。网页去噪的方法基本方法是利用各种通用的特征来区分有效的正文和页眉，页脚，广告等其他信息。其中一个常用的特征是链接文字比率。可以把 Html 转换成 DOM 树，对每个节点计算链接文字比率。如果该节点的链接文字比率高于 1/4 左右，就把这个节点去掉。

下面首先用递归调用的方法计算一个节点下的链接数。

```
/**
 * Counts the number of links from one node downward
 *
 * @param iNode
 *         the node to start counting from
 * @return the number of links
 */
public static int getNumLinks(final Node iNode) {
```

```

    int links = 0;

    if (iNode.hasChildNodes()) {
        Node next = iNode.getFirstChild();

        while (next != null) {
            Node current = next;
            next = current.getNextSibling();
            links += getNumLinks(current);
        }
    }

    if (isLink(iNode))
        links++;

    return links;
}

```

计算一个节点下的有效正文长度，忽略锚点上的字。

```

/**
 * Counts the number of words from one node downward
 *
 * @param iNode
 *         the node to start counting from
 * @return the number of words
 */
private int getNumWords(final Node iNode) {
    int words = 0;

    if (iNode.hasChildNodes()) {
        Node next = iNode.getFirstChild();

        while (next != null) {
            Node current = next;
            next = current.getNextSibling();

            //If it is a link, don't go any deeper into it
            if (!isLink(current))
                words += getNumWords(current);
        }
    }

    //Check to see if the node is a Text node or an element node
    int type = iNode.getNodeType();

```

```
//Text node
if (type == Node.TEXT_NODE) {
    String content = iNode.getNodeValue();
    words += getHTMLLen(content);
    //((double) content.length()) / LETTERS_PER_WORD;
} //if

return words;
}
```

下面是计算一个文本中大概包括的正文长度方法：

```
private int getHTMLLen(String text)
{
    int len = 0;
    for(int i=0;i<text.length();++i)
    {
        if(text.charAt(i)==' ')
        {

        }

        else if(text.charAt(i)==' ')
        {

        }

        else if(text.charAt(i)==' ')
        {

        }

        else if(text.charAt(i)=='&')
        {
            i+=5;
        }
        else
        {
            ++len;
        }
    }
    if( len<10)
        len = 0;
    return len;
}
```

根据链接文字比删除无效节点。


```

/**
 * Removes a table cell if the link ratio is appropriate
 *
 * @param iNode
 *         the table cell node
 */
public void testRemoveCell(final Node iNode) {
    //Ignore if the cell has no children
    if (!iNode.hasChildNodes())
        return;

    double links;
    double words;

    //Count up links and words
    links = getNumLinks(iNode);
    words = getNumWords(iNode);

    //Compute the ratio and check for divide by 0
    double ratio = 0;
    if (words == 0)
        ratio = settings.linkTextRatio + 1;
    else
        ratio = links / words;

    if (ratio > settings.linkTextRatio) {
        Node next = iNode.getFirstChild();
        while (next != null) {
            Node current = next;
            next = current.getNextSibling();

            removeAll(current);
        }
    }
} //testRemoveCell

```

找到覆盖主要正文的内容节点。找到这样一个节点后，可以把这个节点用 Xpath 表示出来，例如：

```

/HTML[1]/BODY[1]/DIV[1]/TABLE[1]/TBODY[1]/TR[1]/TD[1]/TABLE[1]/TBODY[1]/TR[1]/TD[1]

```

但是这种 Xpath 的绝对路径表示方式当 DOM 树的节点有删除或修改后就失效了。

网页链接中的锚点文本是对目标网页主题内容的概括,所以往往用它作为一个网页的标题描述。

- [发展中国家强调以技术合作应对气候变化](#) 20:51
- [经济适用房新政解读 重新回归“自身居住”功能](#) 17:13
- [“红岛”渔排上的“红色”演出](#) 19:51
- [\[“伊朗威胁论”的前世今生\]美国的前后矛盾](#) 18:25
- [中国愿支援台湾发展航太技术](#) 18:15
- [陕西咸阳市大力推进农村节能减排](#) 21:13
- [财经观察：纳斯达克为何而来](#) 17:26
- [坡头区开展“12·4”法制宣传日活动](#) 20:46
- [秦刚 战略经济对话是中美合作的重要机制和平台](#) 17:52
- [“土地储备”首度立规 “土地融资”将受约束](#) 21:15

发展中国家强调以技术合作应对气候变化

2007年12月04日 20:51:05 来源：新华网

【字号 大 中 小】 【留言】 【打印】 【关闭】 【Email推荐: 提交】

新华网印度尼西亚巴厘岛12月4日电(记者齐紫剑 林小春)《联合国气候变化框架公约》秘书处执行秘书德博埃尔4日在巴厘岛说,在[联合国](#)气候变化大会上,很多发展中国家强调应对气候变化的谈判应解决相关的技术合作与技术转让问题,要关注它们目前的“关切”。

取得了网页标题以后,还可以利用标题信息计算网页的内容和标题之间的距离。

```
public static double getSimilarity(String title,String body)
{
    int matchNum = 0;

    for(int i=0;i<title.length();i++)
    {
        if(body.indexOf(title.charAt(i))>=0)
        {
            ++matchNum;
        }
    }
    double score = (double)matchNum/( (double)title.length() );
    return score;
}
```

```
}
```

4.2 从非 HTML 文件中提取文本

4.2.1 TEXT 文件

```
File textFile;  
Scanner scanner = new Scanner(textFile, "GBK");  
scanner.useDelimiter("\\z");  
  
String buffer = "";  
while (scanner.hasNext())  
{  
    buffer += scanner.next();  
}
```

4.2.2 PDF 文件

PDF 是 Adobe 公司开发的电子文件格式。这种文件格式与操作系统的平台无关，可以在多数操作系统上通用。我们经常会遇到这种情况，就是想把 PDF 文件中的文字复制下来，可是会发现经常不能复制。那么怎么把 PDF 文件中的内容提取出来？这就是我们这一节要解决的问题。现在已经有很多工具可以让我们利用来完成这项任务了，PDFBox 就是专门用来解析 PDF 文件的。

PDFBox 的下载

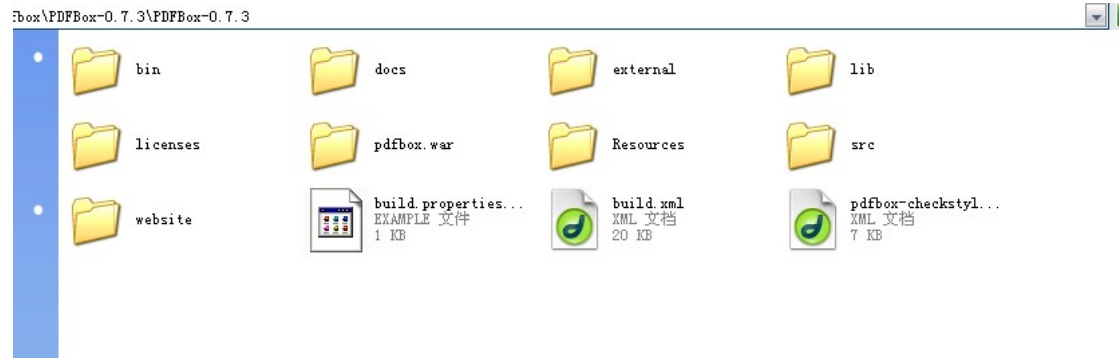
PDFBox 它是一个开源软件，可以到<http://sourceforge.net/projects/pdfbox/>下载。本书用的版本是 PDFBox-0.7.3 版。接下来将对它如何配置以及怎么提取文件内容做详细的说明。

PDFBox 在 Eclipse 中的配置

配置步骤如下：

1>>在 Eclipse 的 workspace 创建一工程：PDFParser，并在工程下建立 lib 包。

2>>把 PDFBox-0.7.3.zip 解压，解压后的目录结构如下图所示：

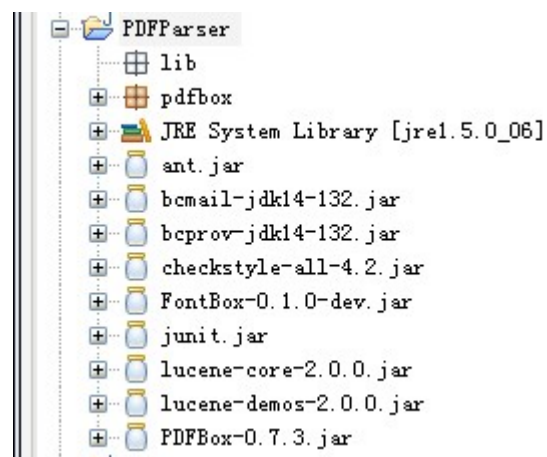


3>>然后进入 external 目录下可以看到它包含：

名称	大小	类型	修改日期	属性
ant.jar	977 KB	Executable Jar ...	2004-8-28 18:43	A
bcmail-jdk14-132.jar	160 KB	Executable Jar ...	2006-4-13 21:08	A
bcprov-jdk14-132.jar	1,033 KB	Executable Jar ...	2006-4-13 21:08	A
checkstyle-all-4.2.jar	1,331 KB	Executable Jar ...	2006-9-26 21:03	A
FontBox-0.1.0-dev.jar	62 KB	Executable Jar ...	2006-3-4 15:16	A
junit.jar	119 KB	Executable Jar ...	2004-8-28 18:43	A
lucene-core-2.0.0.jar	395 KB	Executable Jar ...	2006-6-29 21:15	A
lucene-demos-2.0.0.jar	47 KB	Executable Jar ...	2006-6-29 21:15	A

这里包含了 PDFBox 所有用到的外部包。复制所有 jar 包的工程的 lib 目录下，在从 PDFBox 的 lib 目录下复制 PDFBox-0.7.3.jar 到工程的 lib 目录下。

4>>这一步要做的是把上面所有的 jar 加入工程的 Build Path (具体步骤：在工程上点击右键，在弹出的菜单中选择“Build Path-->Configure Build Path→Add Jars”把所有 jar 包加入)。效果图如下：



用 PDFBox 解析 PDF 文档

在工程下创建一个 PDF 包，在此包下建立一个 TestPDF 类，具体代码如下：

```
package pdfbox;

import java.io.*;

import java.net.*;

import org.pdfbox.pdmodel.PDDocument;

import org.pdfbox.util.PDFTextStripper;

public class TestPDF {

    public void getText(String file)throws Exception{

        //是否排序

        boolean sort=false;

        //pdf 文件名

        String pdfFile=file;

        //输入文本文件名称

        String textFile=null;

        //编码方式

        String encoding="UTF-8";

        //开始提取页数

        int startPage=1;

        //结束提取页数
```

```
int endPage=Integer.MAX_VALUE;

//文件输出流，生成文本文件

Writer output=null;

//内存中存储的 PDF Document

PDDocument document=null;

try{

    try{

        //首先当作一个 URL 来装载文件，如果得到异常再从本地文件系统去装
载文件

        URL url=new URL(pdfFile);

        document=PDDocument.load(url);

        //获取 PDF 文件名

        String fileName=url.getFile();

        // System.out.println(fileName);

        //以原来 PDF 的名称来命名新产生的 txt 文件

        if(fileName.length()>4){

            File          outputFile=new          File(fileName.substring(0,
fileName.length()-4)+".txt");

            textFile=outputFile.getName();

        }

    }catch(MalformedURLException e){
```

```
//如果作为 URL 装载得到异常则从文件系统装载

document=PDDocument.load(pdfFile);

if(pdfFile.length()>4){

    textFile=pdfFile.substring(0,pdfFile.length()-4)+".txt";

    System.out.println(textFile);

}

}

//文件输入流，写入文件到 textFile

output=new OutputStreamWriter(new FileOutputStream(textFile),encoding);

//用 PDFTextStripper 来提取文本

PDFTextStripper stripper =new PDFTextStripper();

//设置是否排序

stripper.setSortByPosition(sort);

//设置起始页

stripper.setStartPage(startPage);

//设置结束页

stripper.setEndPage(endPage);

//调用 PDFTextStripper 的 WriterText 提取并输出文本

stripper.writeText(document, output);

}finally{

    if(output!=null){

        output.close();
```

```
        }

        if(document!=null){

            document.close();

        }

    }

}

public static void main(String[] args){

    TestPDF pdfTest=new TestPDF();

    try{

        pdfTest.getText("C:\\Documents and Settings\\Administrator\\桌面\\ch10.pdf");

    }catch(Exception e){

        e.printStackTrace();

    }

}

}
```

在上面的代码中我是以桌面的 ch10.pdf 为例，getText 接受了一个 String 类型的参数，指定要提取的 PDF 文件的路径。这个路径可以是 URL 也可以是本地文件。首先它从 URL 装载文件给 document，若装载失败在从本地文本装载给 document。然后利用 org.pdfbox.util.PDFTextStripper 类定义 stripper 类，设置一些属性。最后用类 stripper 的 writeText 方法把 document 写入指定的文件。需要说明的是它的这个版本是支持中文的。对于 PDF 文件中的图片它是无能为力的。还需要说明的是，一般把提取出来的文档保存到“.txt”的文件中格式会比较乱，可以把“.txt”用“.doc”替换即把它存入 Word 文件中，看着会好一点。

提取 Pdf 文件的标题，可以用 Pdf 元数据中存储的文档标题：

```
// collect title
```



```

        PDDocumentInformation info =
document.getDocumentInformation();

        this.title = info.getTitle();

```

但是很多 Pdf 文件中存储的标题不准，这里取前三个非空行，取最短的行作为标题。

```

public String getTitle(String str){
    StringTokenizer st=new StringTokenizer(str,"\n");
    String title="
";

    int count =0;
    while (count<3)
    {
        if(!st.hasMoreTokens())
        {
            break;
        }
        String temp=st.nextToken().trim();

        if(!"".equals(temp))
        {
            if(temp.length()<title.length())
            {
                title = temp;
            }
            ++count;
        }
    }

    return title.trim();
}

```

我们也可以利用文本的位置信息帮助选取标题。

```

/**

 * This method will attempt to guess the title of the document.

 *

 * @param textIter The characters on the first page.

 * @return The text position that is guessed to be the title.

```

*/

```
protected TextPosition guessTitle(Iterator textIter) {

    float lastFontSize = -1.0f;

    int stringsInFont = 0;

    StringBuffer titleText = new StringBuffer();

    while (textIter.hasNext()) {

        Iterator textByArticle = ((List) textIter.next())

            .iterator();

        while (textByArticle.hasNext()) {

            TextPosition position = (TextPosition) textByArticle

                .next();

            float currentFontSize = position.getFontSize();

            if (currentFontSize != lastFontSize) {

                if (beginTitle != null) { // font change in candidate title.

                    if (stringsInFont == 0) {

                        beginTitle = null; // false alarm

                        titleText.setLength(0);

                    } else {

                        // had a significant font with some words: call it a title

                        titleGuess = titleText.toString();

                        afterEndTitle = position;

                        return beginTitle;
                    }
                }
            }
        }
    }
}
```

```
        }

        } else { // font change and begin == null

            if (currentFontSize > 13.0f) { // most body text is 12pt max I guess

                beginTitle = position;

            }

        }

        lastFontSize = currentFontSize;

        stringsInFont = 0;

    }

    stringsInFont++;

    if (beginTitle != null) {

        titleText.append(position.getCharacter() + " ");

    }

}

return beginTitle; // null

}
```

4.2.3 Word 文件

Apache 的 POI 可以用来在 windows 或 Linux 平台下提取 word 文档。

```
public static String readDoc(InputStream is) throws IOException{
    //创建WordExtractor
    WordExtractor extractor=new WordExtractor(is);
    // 对DOC文件进行提取
```

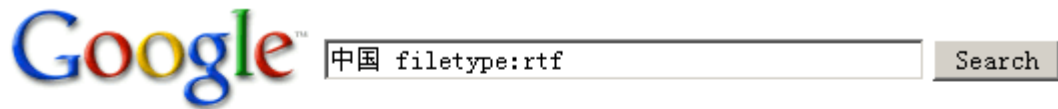
```
String text = extractor.getText();

    return text;
}

public static String getTitle(String str){
    StringTokenizer st=new StringTokenizer(str ,"\n");
    String title=st.nextToken();
    while("".equals(title.trim())){
        if(st.hasMoreTokens()){
            title=st.nextToken();
        }
        else
        {
            break;
        }
    }
    if(title.length()>50){
        if(title.indexOf('_')>0)
            title=title.substring(0, title.indexOf('_'));
        else if(title.indexOf('。')>0)
            title=title.substring(0, title.indexOf('。'));
        else if(title.indexOf('.')>0)
            title=title.substring(0, title.indexOf('.'));
    }
    return title;
}
```

4.2.4 Rtf 文件

许多开源的 rtf 文件解析器不能正确处理多字节编码内容，包括 Google 搜索中的 rtf 文件也有乱码：



[RTF] [??????勿则???](#)

File Format: Rich Text Format

[This site may harm your computer.](#)

中国生物材料委员会是由与生物材料有关的中国全国学会组成的一个联络性组织。... 常务委! 会由中国委员会全体会议选举产生, 常务委员会由主席领导, 并按本章程管理 ...

[www.biomater.com/ccbm/data/CCBM章程\(中文\).rtf](#) - [Similar pages](#) - [Note this](#)

[RTF] [坻漫?*彬??*???](#)

File Format: Rich Text Format - [View as HTML](#)

今天的中国正在将宏观的城市规划和城市理想通过大规模的城市建设予以实现。随着中国与国的接轨, 全球化已经对中国的城市规划和建筑界产生了重大的影响。 ...

[ghjz.tjuci.edu.cn/jingpin/waijianshi_class/kejian/zuoye/郝.rtf](#) - [Similar pages](#) - [Note this](#)

[DOC] [???谳喳??津谳????](#)

File Format: Microsoft Word - [View as HTML](#)

中国科学院院长奖申报表. 申报类别 院长特别奖/院长优秀奖. 姓名. 单位名称. 学科专业名称. 学科专业代码. 攻读学位. 导师姓名及职称. 中国科学院人事教育局 ...

[www.gibh.ac.cn/document/file/中国科学院院长奖申报表.rtf](#) - [Similar pages](#) - [Note this](#)

调用我们自己实现的 Rtf 文件格式解析器:

```
URL u = new
URL("http://www.silo.net/LaReja-2005-05-07/texto_chino_7M.rtf");

// Create a URLConnection object
URLConnection uc = u.openConnection();

// now lets get the response and print it out
InputStream is = uc.getInputStream();

RtfExtractor extractor=new RtfExtractor(is);
String text = extractor.getText();

is.close();

System.out.println("text:"+text);
```

4.2.5 Excel 文件

调用 Apache 的 POI 例子:

```
public static String readDoc(InputStream is) throws IOException{
    HSSFWorkbook wb = new HSSFWorkbook(new POIFSFileSystem(is));
```

```

ExcelExtractor extractor = new ExcelExtractor(wb);

extractor.setFormulasNotResults(true);
extractor.setIncludeSheetNames(false);
String text = extractor.getText();
return text;
}

public static String getTitle(String str){
    StringTokenizer st=new StringTokenizer(str, "\\n");
    if(!st.hasMoreTokens())
        return "";
    String title=st.nextToken();
    while(" ".equals(title.trim())){
        if(st.hasMoreTokens()){
            title=st.nextToken();
        }
        else
        {
            break;
        }
    }
    if(title.length()>50){
        if(title.indexOf('_')>0)
            title=title.substring(0, title.indexOf('_'));
        else if(title.indexOf('。')>0)
            title=title.substring(0, title.indexOf('。'));
        else if(title.indexOf('.')>0)
            title=title.substring(0, title.indexOf('.'));
    }
    return title;
}
}

```

4.2.6 PowerPoint 文件

调用 Apache 的 POI 例子:

```

public static String readDoc(InputStream is) throws IOException{
    //创建PowerPointExtractor
    PowerPointExtractor extractor=new PowerPointExtractor(is);
    // 对PPT文件进行提取
    String text = extractor.getText();
}

```

```

    return text;
}

public static String getTitle(String str){
    StringTokenizer st=new StringTokenizer(str ,"\n");
    String title=st.nextToken();
    while("".equals(title.trim())){
        if(st.hasMoreTokens()){
            title=st.nextToken();
        }
        else
        {
            break;
        }
    }
    if(title.length()>50){
        if(title.indexOf('_')>0)
            title=title.substring(0, title.indexOf('_'));
        else if(title.indexOf('.')>0)
            title=title.substring(0, title.indexOf('.'));
        else if(title.indexOf('.')>0)
            title=title.substring(0, title.indexOf('.'));
    }
    return title;
}

```

4.3 流媒体内容提取

相对于文本检索，音频和视频检索技术研究的比较少。常用的方法是提取出相关的文字描述来索引音频和视频。

4.3.1 音频流内容提取

音频是多媒体中的一种重要媒体。我们能够听见的音频频率范围是 60Hz~20kHz，其中语音大约分布在 300Hz~4kHz 之内，而音乐和其他自然声响是全范围分布的。声音经过模拟设备记录或再生，成为模拟音频，再经数字化成为数字音频。数字化时的采样率必须高于信号带宽的 2 倍，才能正确恢复信号。样本可用 8 位或 16 位比特表示。

以前的许多研究工作涉及到语音信号的处理，如语音识别。机器容易自动识别孤立的字词，如用在专用的听写和电话应用方面，而对连续的语音识别则较困难，错误较多，但目前在这方面已经取得了突破性的进展，同时还研究了辨别说话人的技术。这些研究成果将为音

频信息的检索提供很大帮助。

语音检索是以语音为中心的检索，采用语音识别等处理技术。如电台节目、电话交谈、会议录音等。

基于语音技术的检索是利用语音处理技术检索音频信息。过去人们对语音信号处理开展了大量的研究，许多成果可以用于语音检索。

(1)利用大词汇语音识别技术进行检索

这种方法是利用自动语音识别(ASR)技术把语音转换为文本，从而可以采用文本检索方法进行检索。虽然好的连续语音识别系统在小心地操作下可以达到 90% 以上的词语正确度，但在实际应用中，如电话和新闻广播等，识别率并不高。即使这样，ASR 识别出来的脚本仍然对信息检索有用，这是因为检索任务只是匹配包含在音频数据中的查询词句，而不是要求一篇可读性好的文章。例如，采用这种方法把视频的语音对话轨迹转换为文本脚本，然后组织成适合全文检索的形式支持检索。

(2)基于识别关键词进行检索

在无约束的语音中自动检测词或短语通常称为关键词的发现(Spotting)。利用该技术，识别或标记出长段录音或音轨中反映用户感兴趣的事件，这些标记就可以用于检索。如通过捕捉体育比赛解说词中“进球”的词语可以标记进球的内容。

(3)基于说话人的辨认进行分割

这种技术是简单地辨别出说话人语音的差别，而不是识别出说的是什么。它在合适的环境中可以做到非常准确。利用这种技术，可以根据说话人的变化分割录音，并建立录音索引。如用这种技术检测视频或多媒体资源的声音轨迹中的说话人的变化，建立索引和确定某种类型的结构(如对话)。例如，分割和分析会议录音，分割的区段对应于不同的说话人，可以方便地直接浏览长篇的会议资料。

MPEG-7 标准被称为“多媒体内容描述接口”，为各类多媒体信息提供一种标准化的描述，这种描述将与内容本身有关，允许快速和有效的查询用户感兴趣的资料。它将扩展现有内容识别专用解决方案的有限的能力，特别是它还包括了更多的数据类型。换言之，MPEG-7 规定一个用于描述各种不同类型多媒体信息的描述符的标准集合。

MPEG-7 的目标是支持多种音频和视觉的描述，包括自由文本、N 维时空结构、统计信息、客观属性、主观属性、生产属性和组合信息。对于视觉信息，描述将包括颜色、视觉对象、纹理、草图、形状、体积、空间关系、运动及变形等。

MPEG-7 的目标是根据信息的抽象层次，提供一种描述多媒体材料的方法以便表示不同层次上的用户对信息的需求。以视觉内容为例，较低抽象层将包括形状、尺寸、纹理、颜色、运动（轨道）和位置的描述。对于音频的较低抽象层包括音调、调式、音速、音速变化、音响空间位置。最高层将给出语义信息：如“这是一个场景：一个鸭子正躲藏在树后并有一个

汽车正在幕后通过。”抽象层与提取特征的方式有关：许多低层特征能以全自动的方式提取，而高层特征需要更多人的交互作用。MPEG-7 还允许依据视觉描述的查询去检索声音数据，反之也一样。

MPEG-7 的目标是支持数据管理的灵活性、数据资源的全球化和互操作性。

MPEG-7 标准化的范围包括：一系列的描述子（描述子是特征的表示法，一个描述子就是定义特征的语法和语义学）；一系列的描述结构（详细说明成员之间的结构和语义）；一种详细说明描述结构的语言、描述定义语言（DDL）；一种或多种编码描述方法。

在我们的日常生活中，日益庞大的可利用音视频数据需要有效的多媒体系统来存取、交互。这类需求与一些重要的社会和经济问题相关，并且在许多专业和消费应用方面都是急需的，尤其是在网络高度发展的今天，而 MPEG-7 的最终目的是把网上的多媒体内容变成象现在的文本内容一样，具有可搜索性。

4.3.2 视频流内容提取

视频信息一般由四部分组成：帧、镜头、情节、节目。视频流的内容可以从多个层次分析：

- 底层内容建模，包括颜色、纹理、形状、空间关系、运动信息等。
- 中层内容建模，指视频对象(Video Object)，在 MPEG-4 中包括了视频对象。对象的划分可根据其独特的纹理、运动、形状、模型和高层语义为依据。
- 高层内容建模（语义概念等），例如一段体育视频，可以提取出扣篮或射门的片断。

关键帧提取策略：

第一， 设置一个最大关键帧数 M；

第二， 每个镜头的非边界过渡区的第一帧确定为关键帧；

第三， 使用非极大值抑制法确定镜头边界系数极大值，并排序，以实现基于镜头边界系数的关键帧提取。

镜头边界检测方法：

第一，只使用镜头边界系数的镜头边界检测——固定阈值法

第二，结合相邻帧差的镜头边界检测——自适应阈值法

可以使用 Java 媒体框架 API (JMF)在 Java 语言中处理声音和视频等时序性的媒体。下

面我们首先通过 vid2jpg.java 类来提取视频中所有的帧。

```
/**
 * This will get called when there's data pushed from the PushBufferDataSource.
 */

public void transferData(PushBufferStream stream)
{
    try
    {
        stream.read(readBuffer);
    }
    catch(Exception e)
    {
        System.out.println(e);
        return;
    }

    // Just in case contents of data object changed by some other thread

    Buffer inBuffer = (Buffer)(readBuffer.clone());

    // Check for end of stream

    if(readBuffer.isEOM())
    {
```

```
        System.out.println("End of stream");

        return;

    }

    // Do useful stuff or wait

    useFrameData(inBuffer);

}
```

4.4 抓取限制应对方法

有些网站对于同一个 IP 在一段时间内的访问次数有限制。这时可以通过大量的 Socket 代理循环访问网站。

首先建立有效代理列表，proxyIP.txt：

```
219.93.178.162:3128

222.135.79.253:8080

203.160.1.38:554

132.239.17.225:3124

169.229.50.5:3124

203.160.1.146:554

203.160.1.49:554
```

有些可以自动发现有效代理的软件，例如“花刺代理”等。

然后建立 ProxyDB 类，循环使用这些 Socket 代理：

```
int pos = proxyIpList.get(count).toString().indexOf(":");
if (pos > 0) {
    String port =
```

```
proxyIpList.get(count).toString().substring(pos+1);
    proxyAddr =
proxyIpList.get(count).toString().substring(0,pos);

    proxyPort = Integer.parseInt(port);

    SocketAddress socketaddress = new
InetSocketAddress(proxyAddr,proxyPort);

    proxy = new Proxy(Proxy.Type.HTTP,socketaddress);
```

最后通过 URL 的 openConnection 方法使用它:

```
url.openConnection(ProxyDB.getProxy());
```

简单的模仿浏览器访问的方法:

```
URLConnection con = (URLConnection) url.openConnection();

con.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows; U;
Windows NT 5.2; zh-CN; rv:1.8.1.10) Gecko/20071115 Firefox/2.0.0.10");
con.setRequestProperty("Host", url.getHost());
con.setRequestProperty("Accept-Language", "zh-cn");

con.setRequestProperty("Accept-Charset",
"gb2312,utf-8;q=0.7,*;q=0.7n");
```

4.5 本章小结

第5章 自然语言处理

当前文本信息是搜索引擎主要处理的内容。自然语言处理，英文叫做 Natural Language Process，严格来说自然语言处理包括自然语言理解和自然语言生成两部分。但这里我们只涉及自然语言理解部分。

5.1 中文分词处理

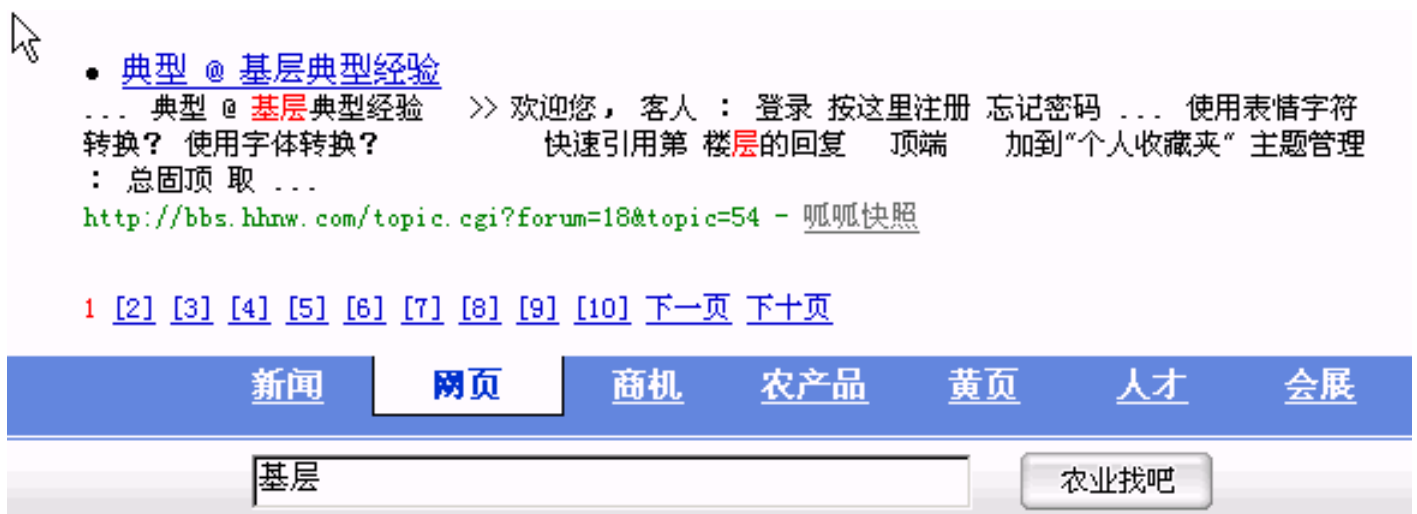
因为中文文本中词和词之间不像英文一样存在边界，所以中文分词是一个专业处理中文信息的搜索引擎首先面对的问题。在当前常用的语言中，简体中文，繁体中文和日文存在分词问题。

5.1.1 Lucene 中的中文分词

Lucene 中处理中文分词的常用方法有三种。以“咬死猎人的狗”这句话的输出结果为例：

1. 单字方式。[咬] [死] [猎] [人] [的] [狗]；
2. 二元覆盖的方式。[咬死] [死猎] [猎人] [人的] [的狗]；
3. 分词的方式。[咬] [死] [猎人] [的] [狗]。

Lucene 中 StandardAnalyzer 采用了单字分词的方式。我们可以通过下面这个搜索页面看到单字分词的不足。



CJKAnalyzer 采用了二元覆盖的方式实现。小规模搜索网站宜采用二元连接的方法，这样可以解决单字搜索“上海”和“海上”混淆的问题。采用中文分词的方法适用于中大规模的搜索引擎。当然，天津市海量科技发展有限公司(<http://www.hylanda.com/>)的中文分词系统是首先在商业领域得到广泛应用的系统。北京盈智星科技发展有限公司提供了一个基于 Lucene 接口的 java 版中文分词系统。

5.1.2 Lietu 中文分词的使用

Lucene 中负责语言处理的包主要是 `org.apache.lucene.analysis.Analyzer` 和 `org.apache.lucene.analysis.TokenStream` 的子类。`org.apache.lucene.analysis.TokenStream` 用来做基本的分词工作。`org.apache.lucene.analysis.Analyzer` 是 `TokenStream` 的外围包装类，负责整个解析工作。

Lietu 中文分词程序由 `seg.jar` 的程序包和一系列词典文件组成。通过系统参数 `dic.dir` 指定词典数据文件路径。我们可以写一个简单的分词测试代码：

```
String sentence = "有关刘晓庆偷税案";
String[] result = com.lietu.seg.result.Tagger.split(sentence);

for (int i=0; i<result.length;i++)
{
    System.out.println(result[i]);
}
```

使用分词的时候为了高亮显示关键字，必需保留位置信息。如果分词处理的 web 文档有很多无意义的乱码，这些乱码导致的无意义的位置信息存储甚至可能会导致 5 倍以上的膨胀率。

5.1.3 中文分词的原理

分词的两类方法：机械匹配的方法还有统计的方法。

机械匹配的方法，最常见的方法是：利用正向或反向最大匹配的方法来分词。

例如，假设词典包括如下的词语：

大

大学

大学生

生活

活动

中

心

中心

输入：大学生活动中心

最后分词结果：大学生/活动/中心

最大长度分词的匹配窗口为 12 的匹配代码：

```
public void wordSegment(String Sentence)//传入一个字符串作为要处理的对象。

{

    int senLen = Sentence.length();//首先计算出传入的这句话的字符长度

    int i=0, j=0;//i 是用来控制截取的起始位置的变量，j 是用来控制截取的结束位置的
    变量

    int M=12;//所取得的词组的最大值不超过 12。

    String word;//我们需要与词库中做对比的词。

    boolean bFind = false;//用来判断是否是词库中的词的变量。

    while(i < senLen)//如果 i 的大小，小于此句话的长度就进入循环

    {

        int N= i+M<senLen ? i+M : senLen;//如果 i+M<senLen 为真就执行 i+M，如果其
        为假就执行 senLen 把结果赋值给 N

        //以上这句话就是为了界定我们所要取的字或者是词组的大小。

        bFind=false;//首先假设这个我们取出来的词组不是词库中的词
```

```

for(j=N; j>i; j--)//正向最大匹配

{

    word = Sentence.substring(i, j);//截取我们所需要的字符串。

    //如果没有匹配到合适的词的话就会一直的在这里循环直到出循环。

    if(dic.Find(word))

    {

        System.out.print(word + " ");//如果这个词是词库中的那么就打印出来

        bFind=true;

        i=j;//如果你在词库中找到了这个词那么就截取的末尾的位置 j 赋给 i

        //System.out.println(i);

        break;//跳出 for 循环

    }

}

if(bFind == false)//如果不是词库中的词按照以下这中方式处理。

{

    word = Sentence.substring(i, i+1);

    System.out.print(word + " ");

    ++i;//这句话其实就是控制读的顺序

}

}

System.out.println();

}

```


有意见分歧

正向结果：有意/见/分歧

反向结果：有/意见/分歧.

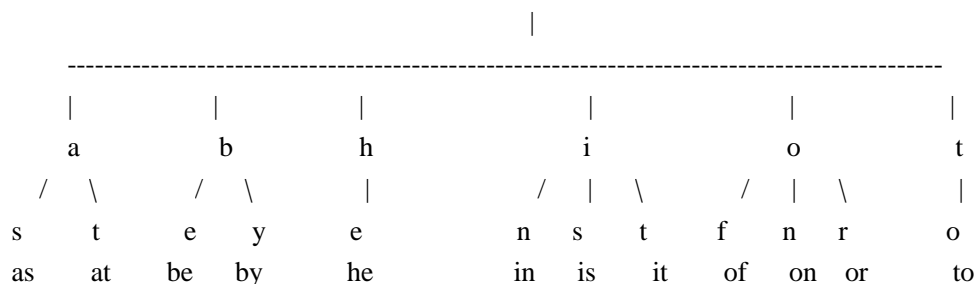
另外一种最少切分的方法是使每一句中切出的词数最小。

5.1.4 查找词典算法

中文分词使用的词典规模往往会越来越大。需要选择一个好的查找词典算法来提高分词性能

。数字搜索树

一个数字搜索Trie树(retrieve 树)的一个节点只保留一个字符。如果一个单词比一个字符长，则包含第一个字符的节点有指针指向下一个字符的节点，依次类推。这样组成一个层次结构的树，树的第一层包括所有单词的第一个字符，树的第二层包括所有单词的第二个字符，依次类推，数字搜索树的最大高度是词典中最长单词的长度。例如，如下单词序列组成的词典(as at be by he in is it of on or to) 会生成下面这个数字搜索树：



数字搜索树的结构独立于生成数时单词进入的顺序。这里，trie树的高度是2。因为数的高度很小，在数字搜索Trie树中搜索一个单词的速度很快。但是，这是以内存消耗为代价的，树中的每一个节点都需要很多内存。假设每个词都是由26个小写英文字母中的一个组成的，这个节点中会有26个指针。所以不太可能直接用这样的数字搜索树来存储中文这样的大字符集。

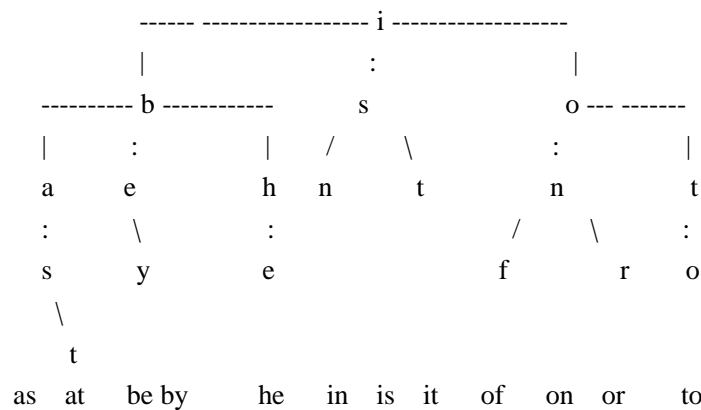
Tire 树

在一个三叉搜索树中，每一个节点包括一个字符，但只有三个指针，一个指向左边的树，一个指向右边的树，还有一个向下，指向单词的下一个数据单元。三叉搜索树是二叉搜索树和数字搜索树的混合体。它有和数字搜索树差不多的速度但是和二叉搜索树一样只需要相对较少的内存空间。单词的读入顺序对于创建平衡的三叉搜索树很重要，但对于二叉搜索树就不是太重要了。

通过选择一个数据单元集合的中间值，并把它作为开始节点，我们可以创建一个平

衡的三叉树，避免了随机过程。我们再次以有序的数据单元为例(as at be by he in

is it of on or to)。首先我们把关键字“is”作为中间值并且构建一个包含字母“i”的根节点。它的直接后继结点包含字母“s”并且可以存储任何与“is.” 有关联的数据。对于“i,” 的左树，我们选择“be”作为中间值并且创建一个带有关键字“b”的结点，关键字“b”的直接后继结点包含“e”。该数据存储在“e” 结点。对于“i,” 的右树，按照逻辑，我们选择“on”作为中间值，并且创建“o”结点以及它的直接后继结点“n”,如此等等。最终的三叉树如下所示：



其中，水平虚线和一个垂直实线合起来代表一条长斜线。如果水平虚线从左侧开始并且沿一垂直实线向下（例如在“i”与“b”之间），它代表一条向左下方倾斜的长斜线。垂直的虚线代表一个父节点的下面的直接的后继结点。只有父节点和它的直接后继节点才能形成一个数据单元的关键字；“i”和“s”形成关键字“is”，但是“i”和“b”不能形成关键字，因为它们之间仅用一条实线相连，不具有直接后继关系。

TernarySearchTrie 本身可以当作 HashMap 对象来使用。Key -> Value 的对应关系。Key 按照 Char 拆分成了 Link，以 Char Link 存在。Value 存储在。

Search 过程中，从树的根节点匹配 Key。Key 按 Char 从前往后匹配。charIndex 表示 Key 的当前要比较的 Char 的位置。currentNode 表述树的当前节点的位置。匹配过程如下：

```

protected TSTNode getNode(String key, TSTNode startNode) {
    if (key == null) {
        return null;
    }
    int len = key.length();
    if (len == 0)
        return null;
    TSTNode currentNode = startNode;
    int charIndex = 0;
    char cmpChar = key.charAt(charIndex);
  
```

```

    int charComp;
    while (true) {
        if (currentNode == null) {
            return null;
        }
        charComp = cmpChar - currentNode.splitchar;
        if (charComp == 0) {
            charIndex++;
            if (charIndex == len) {
                return currentNode;
            }
            else
            {
                cmpChar = key.charAt(charIndex);
            }

            currentNode = currentNode.eqKID;
        } else if (charComp < 0) {
            currentNode = currentNode.loKID;
        } else {
            currentNode = currentNode.hiKID;
        }
    }
}

```

树的创建过程如下:

```

TSTNode currentNode = root;
int charIndex = 0;
while (true) {
    int charComp = (
        key.charAt(charIndex) -
        currentNode.splitchar);
    if (charComp == 0) {
        charIndex++;
        if (charIndex == key.length()) {
            return currentNode;
        }
        if (currentNode.eqKID == null) {
            currentNode.eqKID =
                new TSTNode(key.charAt(charIndex));
        }
        currentNode = currentNode.eqKID;
    } else if (charComp < 0) {

```

```

    if (currentNode.loKID == null) {
        currentNode.loKID =
            new TSTNode(key.charAt(charIndex));
    }
    currentNode = currentNode.loKID;
} else {
    if (currentNode.hiKID == null) {
        currentNode.hiKID =
            new TSTNode(key.charAt(charIndex));
    }
    currentNode = currentNode.hiKID;
}
}

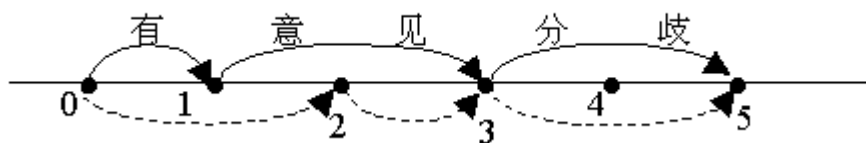
```

相对于查找过程，创建过程在搜索的过程中判断链接的空值，而不是一开始就判断。

5.1.5 最大概率分词方法

基本思想是：

- (1) 一个待切分的汉字串可能包含多种分词结果
- (2) 将其中概率最大的那个作为该字串的分词结果



路径 1: 0—1—3—5

路径 2: 0—2—3—5

该走哪条路呢？

数学描述：

S: 有意见分歧

W1: 有/ 意见/ 分歧/

W2: 有意/ 见/ 分歧/

这里 S 表示待切分的句子， $\text{Max}(P(W1|S), P(W2|S))$ ？

$$P(W|S) = \frac{P(S|W) \times P(W)}{P(S)} \approx P(W)$$

$$P(W) = P(w_1, w_2, \dots, w_l) \approx P(w_1) \times P(w_2) \times \dots \times P(w_l)$$

约等于的假设：每个词之间的概率是上下文无关的。

$$P(w_i) = \frac{w_i \text{在语料库中的出现次数 } n}{\text{语料库中的总词数 } N}$$

词语	概率
...	...
有	0.0180
有意	0.0005
意见	0.0010
见	0.0002
分歧	0.0001
...	...

$$P(W1) = P(\text{有}) * P(\text{意见}) * P(\text{分歧})$$

$$= 1.8 \times 10^{-9}$$

$$P(W2) = P(\text{有意}) * P(\text{见}) * P(\text{分歧})$$

$$= 1 \times 10^{-11}$$

$$P(W1) > P(W2)$$

如何尽快找到概率最大的词串（路径）？

$$P'(w_i) = P'(w_{i-1}) \times P(w_i)$$

$$P'(\text{意见}) = P'(\text{有}) \times P(\text{意见})$$

$$P'(\text{有}) = P(\text{有})$$

左邻词

假定对字串从左到右进行扫描，可以得到

， ， … ， ， ， … 等若干候选词，如果 的尾字 跟 的首字邻接，就称 为 的左邻词。比如上面例中，候选词“有”就是候选词“意见”的左邻词，“意见”和“见”都是“分歧”的左邻词。字串最左边的词没有左邻词。

最佳左邻词

如果某个候选词 有若干个左邻词 ， ， … 等等，其中累计概率最大的候选词称为 的最佳左邻词。比如候选词“意见”只有一个左邻词“有”，因此，“有”同时也是“意见”的最佳左邻词；候选词“分歧”有两个左邻词“意见”和“见”，其中“意见”的累计概率大于“见”累计概率，因此“意见”是“分歧”的最佳左邻词

最大概率分词算法：

1. 对一个待分词的字串 S ，按照从左到右的顺序取出全部候选词 $w_1, w_2, \dots, w_i, \dots, w_n$ ；
2. 到词典中查出每个候选词 的概率值 $P(w_i)$ ，并记录每个候选词的全部左邻词；
3. 按照公式 1 计算每个候选词的累计概率，同时比较得到每个候选词的最佳左邻词；
4. 如果当前词 w_n 是字串 S 的尾词，且累计概率 $P'(w_n)$ 最大，则 w_n 就是 S 的终点词；
5. 从 w_n 开始，按照从右到左顺序，依次将每个词的最佳左邻词输出，即为 S 的分词结果。

算法运行示例：

- (1) 对“有意见分歧”，从左到右进行一遍扫描，得到全部候选词：

“有”，“有意”，“意见”，“见”，“分歧”；

(2) 对每个候选词，记录下它的概率值，并将累计概率赋初值为 0；

(3) 顺次计算各个候选词的累计概率值，同时记录每个候选词的最佳左邻词：

$$P'(\text{有})=P(\text{有}),$$

$$P'(\text{有意})=P(\text{有意}),$$

$$P'(\text{意见})=P'(\text{有}) \times P(\text{意见}), (\text{“意见”的最佳左邻词为“有”})$$

$$P'(\text{见})=P'(\text{有意}) \times P(\text{见}), (\text{“见”的最佳左邻词为“有意”})$$

$$P'(\text{意见})>P'(\text{见})$$

(4) “分歧”是尾词，“意见”是“分歧”的最佳左邻词，分词过程结束，
输出结果：有/ 意见/ 分歧/

5.1.6 新词发现

词典中没有的，但是结合紧密的字或词有可能组成一个新词。

判断词的结合紧密程度的方法，信息熵：

$$I(X,Y)=\log_2 \frac{P(X,Y)}{P(X)P(Y)}$$

如果 x 和 y 的出现相互独立,那么 $p(x,y)$ 的值和 $p(x)p(y)$ 的值相等, $I(x,y)$ 为 0。
如果 x 和 y 密切相关, $p(x,y)$ 将比 $p(x)p(y)$ 大很多, $I(x,y)$ 值也就远大于 0。如果 x 和 y 的几乎不会相邻出现,而它们各自出现的概率又比较大,那么 $I(x,y)$ 将取负值。

$$P(C_1, C_2) = P(C_1) * P(C_2 | C_1) = \frac{f(C_1)}{N} * \frac{f(C_1 C_2)}{f(C_1)} = \frac{f(C_1 C_2)}{N}$$

$$I(C_1, C_2) = \log_2 N + \log_2 \frac{f(C_1 C_2)}{f(C_1) f(C_2)}$$

筛选新词的过程：

从互联网的连接中发现新词。

5.1.7 隐马尔可夫模型

描述:

start: go(cow,1.0)

cow: emit(moo,0.9) emit(hello,0.1) go(cow,0.5) go(duck,0.3) go(end,0.2)

duck: emit(quack,0.6) emit(hello,0.4) go(duck,0.5) go(cow,0.3) go(end,0.2)

转移概率矩阵:

	cow	duck	end
start	1.0	0	0
cow	0.5	0.3	0.2
duck	0.3	0.5	0.2

发射概率（混淆矩阵）:

	moo	hello	quack
cow	0.9	0.1	0
duck	0	0.4	0.6

输入: moo hello quack end

输出: COW DUCK DUCK END

转移概率矩阵:

	a	m	q	v	n	w
a	12					
m						
q						
v						
n					101	
w						

发射概率:

	春光
n	0.9
nr	0.1

语料库中 n 总共有 1000 个, nr 有 100 个

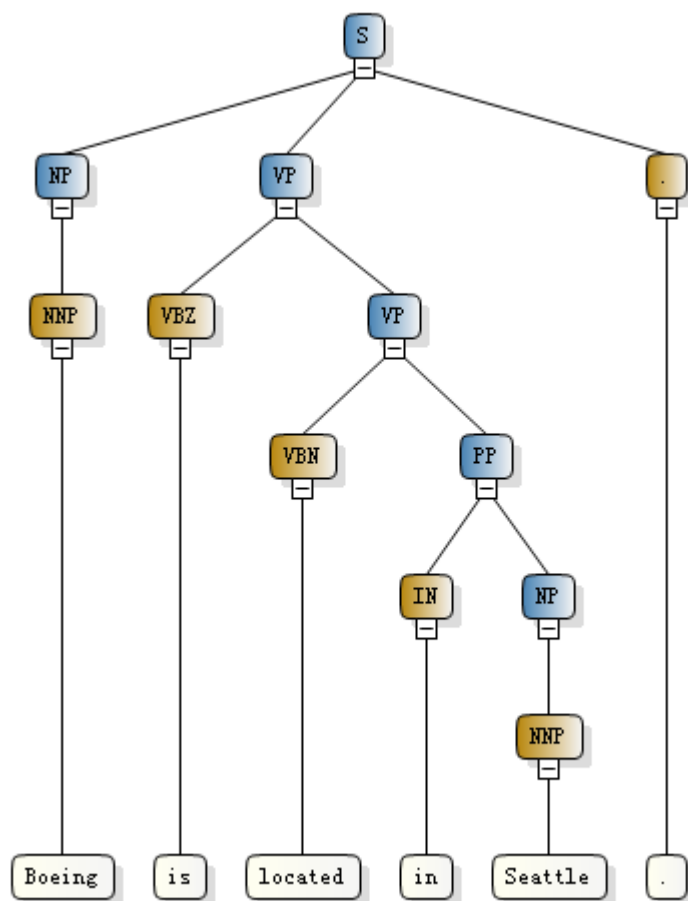
春光:n:9 发射概率 9/1000

春光:nr:1 发射概率 1/100

Start/桂/春光/说/./End

5.2 语法解析树

语法解析树一般用在机器翻译中,但是对于搜索引擎理解文本,更准确的返回也有帮助。比如有用户输入:“肩宽的人适合穿什么衣服”。如果返回结果中包括“肩宽的人穿什么衣服好?”,或者“肩膀宽宽的女孩子穿什么衣服好看?”可能是用户想要的结果。



咬/死/了/猎人/的/狗

Java 版本的分析工具

<http://opennlp.sourceforge.net/>

5.3 文档排重

在做索引的过程中实现网页去重。目前大规模搜索引擎流行的消重算法:

对于每张网页，从净化后的网页中，选取最有代表性的一组关键词，并使用该关键词组生成一个语义指纹。通过比较两个网页的指纹是否相同来判断两个网页是否相似。

语义指纹生成算法如下：

第一步：将每个网页分词表示成基于词的特征向量，使用 **TF*IDF** 作为每个特征项的权值。地名，专有名词等，名词性的词汇往往有更高的语义权重。

第二步：将特征项按照词权值排序。

第三步：选取前 **n** 个特征项，然后重新按照字符排序（否则找不到对应关系了）。

第四步：调用 **MD5** 算法，将每个特征项串转化为一个 128 比特的串，作为该网页的指纹。

通过 **BDB** 判断该语义指纹是否已经存在。

Bloom filter。简单的说是这样一种方法：在内存中开辟一块区域，对其中所有位置 0，然后对数据做是 **C(10,2)**，即 45 次种不同的 **hash**，每个 **hash** 值对内存 **bit** 数求模，求模得到的数在内存对应的位上置 1。置位之前会先判断是否已经置位，每次插入一个 **URL**，只有当全部 45 个位都已经置 1 了才认为是重复的。

如果对上述这段话不太理解，可以换个简单的比喻：有 10 个桶，一个桶只能容纳 1 个球，每次往这些桶中扔两个球，如果两个桶都已经球，才认为是重复，问问为了不重复总共能扔多少次球？

10 次，是这个答案吧？每次扔 1 个球的话，也是 10 次。表面上看一次扔几个球没有区别，事实上一次两个球的情况下，重复概率比一次一个球要低。**Bloom filter** 算法正式借助这一点，仅仅用少量的空间就可以进行大量 **URL** 的排重，并且使误判率极低。

如下是一个简单的使用例子：

// 创建一个 100 位的 bloom filter 容纳字符串，优化成包含 4 个项目

```
SimpleBloomFilter mammals = new SimpleBloomFilter(100, 4);
```

// 增加内容到 bloom filter

```
mammals.add("dog");
```

```
mammals.add("cat");
```

```
mammals.add("mouse");
```

```
mammals.add("dolphin");
```

// 测试是否 bloom filter 记得这个项目

```
if (mammals.contains("dog")) {
```

```
    System.out.println("A dog is a mammal with probability "
```

```
        + (1 - mammals.expectedFalsePositiveProbability()));
```

```
} else {
```

```

        System.out.println("A dog is definitely not a mammal");
    }

```

如果想知道需要使用多少位，如下是一个给定项目和位数比率的误判率的表。

Ratio (items:bits)	False-positive probability
1:1	0.63212055882856
1:2	0.39957640089373
1:4	0.14689159766038
1:8	0.02157714146322
1:16	0.00046557303372
1:32	0.00000021167340
1:64	0.00000000000004

为每个 URL 分配两个字节就可以达到千分之几的冲突。比较保守的实现，为每个 URL 分配了 4 个字节，对于 5000 万的数量级，它只占用了 100 多 M 的空间，并且排重速度超快，一遍下来不到两分钟。

5.4 中文关键词提取

5.4.1 关键词提取的基本方法

- 首先建立关键词提取训练库：训练文件(X.txt)和对应的关键词文件(x.key)。
- 需要去除 StopWords
- 利用 TF*IDF 公式，计算每个可能的关键词的 TF*IDF：统计词频和词在所有文档中出现的总次数。TF 是 Term Frequency 的简称，IDF 是 Invert Document Frequency 的简称。比如说“的”在 100 文档中的 40 篇文档中出现过，则 DF 是 40，IDF 是 1/40。“的”在第一篇文档中出现了 15 次。TF*IDF(的) = 15 * 1/40。另外一个词 TF*IDF(反腐败) = 5 * 1/5。

- 利用位置信息：开始和结束位置的词往往更可能是关键词。比如，利用下面的经验公式：

```
double position = t.startOffset() / content.length();
```

```
position = position * position - position + 2;
```

或者利用一个分段函数，首段或者末段的词的权重更大。

- 标题中出现的词比内容中的词往往更重要。
- 利用词性信息：关键词往往是名词或者名词结尾的词，而介词，副词，动词结尾的词一

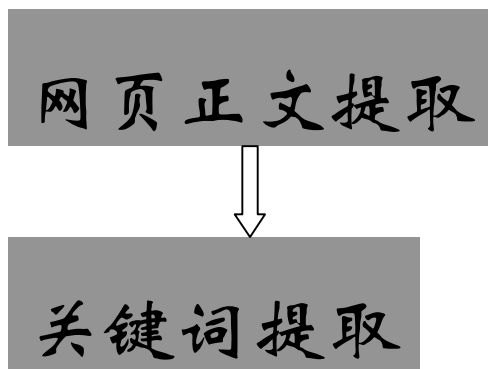
般不能组成词组。

- 利用词或者字的互信息： $I(x,y) = \log_2(P(x,y) / (P(x)P(y)))$ 。比如说 $I(\text{福}, \text{娃}) = \log_2(P(\text{福}, \text{娃}) / (P(\text{福}) P(\text{娃})))$
- 利用标点符号：《 》 和 “ ” 之间的文字。例如：“汉芯一号”造假案。

5.4.2 关键词提取的设计

5.4.3 从网页提取关键词

处理流程如下：



H1 标签中词，黑体加粗的词可能更重要，更有可能是关键词。

Meta 中的 KeyWords 描述也有可能真实的反映了该网页的关键词。

```
<meta name="keywords" content="公判" />
```

5.5 相关搜索

一种方法是从搜索日志中挖掘字面相似的词作为相关搜索词列表。首先从一个给定的词语挖掘多个相关搜索词，可以用编辑距离为主的方法查找一个词的字面相似词，如果候选的相关搜索词很多，就要筛选出最相关的 10 个词。下面是利用 Lucene 筛选最相关词的方法。

```
private static final String TEXT_FIELD = "text";

/**
 *
 * @param words 候选相关词列表
```

```
* @param word 要找相关搜索词的种子词
* @return
* @throws IOException
* @throws ParseException
*/
static String[] filterRelated(HashSet<String> words, String word)
throws IOException, ParseException
{
    StringBuilder sb = new StringBuilder();

    for(int i=0;i<word.length();++i)
    {
        sb.append(word.charAt(i));
        sb.append(" ");
    }

    RAMDirectory store = new RAMDirectory();
    IndexWriter writer = null;
    writer = new IndexWriter(store, new StandardAnalyzer(), true);

    for(String text:words)
    {
        Document document = new Document();
        Field textField = new Field(TEXT_FIELD, text, Field.Store.YES,
Field.Index.TOKENIZED);
        document.add(textField);
        writer.addDocument(document);
    }
    writer.close();

    IndexSearcher searcher = new IndexSearcher(store);

    QueryParser queryParser = new QueryParser(TEXT_FIELD, new
StandardAnalyzer());
    Query query = queryParser.parse(sb.toString());

    Hits hits = searcher.search(query);
    int maxRet = Math.min(10, hits.length());

    String[] relatedWords = new String[maxRet];
    for (int i = 0; i < maxRet ; i++) {
        Document document = hits.doc(i);
        String text = document.get(TEXT_FIELD);
        System.out.println(text);
    }
}
```

```

        relatedWords[i]=text;
    }
    searcher.close();
    store.close();

    return relatedWords;
}

```

整理出这样的相关词表，第一列是关键词，后续是 10 个以内的相关搜索词：

集福轩婚礼%集福轩

手机定位跟踪系统%手机定位系统%手机定位%手机定位仪器

喷绘材料卖店电话%我要喷绘材料卖店电话

厦门房产%厦门租房%厦门新闻%厦门桑拿%房产%青岛房产%厦门%恒雄房产

送水果%送水%水果

三星传真机%三星手机

另外一种方法，可以把多个用户共同查询的词看成相关搜索词，需要有记录用户 IP 的搜索日志才能实现。

然后通过 `RelatedEngine` 类查找某个关键词的相关词。

```

public static void main(String[] args) throws Exception {
    RelatedEngine re =new RelatedEngine(new
File("D:/lg/work/xiaoxishu/dic/relatedwords.txt"));
    String word = "徐家汇";
    String[] relatedWords = re.getRelated(word);
    for(String w : relatedWords)
    {
        System.out.println(w);
    }
}

```

输出如下：

上海徐家汇

徐汇

徐家汇价格是

上房徐家汇路附近有吗

当我们需要为新建立的搜索引擎开发相关搜索时,如果没有搜索日志而用户文本很多的时候,我们可以:

1. 首先运行 **IndexMaker** 从待搜索的文档中提取关键词并生成索引;
2. 然后运行 **RelatedWords** 从索引生成相关词表。

5.6 拼写检查

拼写检查是针对用户可能的输入错误给出对应正确的词提示。实现拼写检查往往需要相关的词库和算法支持。不存在万能的词表,垂直(网站)搜索引擎往往需要整理和自己行业(网站)相关的词库才能达到好的匹配效果。

5.6.1 英文拼写检查

拼写检查就是对错误的词给出正确的提示。如果有个正确的词和用户输入的词很近似,用户的输入可能是错误的,系统提示“您是不是要找”这个正确的词。正确词的词典格式如下:

biogeochemistry : 1

repairer : 3

wastefulness : 3

battier : 2

awl : 3

preadapts : 1

surprisingly : 3

stuffiest : 3

每行一个词,分别是词本身和词频。因为互联网中的新词不断出现,正确的词并不是来

源于固定的词典，而是来源于搜索的文本本身。下面直接从文本内容提取英文单词。不从索引库中提取的原因是 **Term** 可能经过词干化处理过了，所以我们用 **StandardAnalysis** 再次处理。

```
java.io.StringReader input = new java.io.StringReader(content);

TokenStream tokenizer = new StandardTokenizer(input);

for (Token t = tokenizer.next(); t != null; t = tokenizer.next())
{
    if( isAllLetter(t.termText()) &&
        (t.termText().length()>=3) &&
        (t.termText().length()<=30) )
    {
        System.out.println(t.termText());
        fpSource.write(t.termText().toLowerCase());
        fpSource.write(" : 1\n");
    }
}
```

5.6.2 中文拼写检查

和英文拼写检查不一样，中文的用户输入的搜索词串的长度更短，从错误的词猜测可能的正确输入更加困难。这时候需要更多的借助正误词表，词典文本格式如下：

代款:贷款

阿地达是:阿迪达斯

诺基压:诺基亚

飞利浦:飞利浦

寂么沙洲冷:寂寞沙洲冷

欧米加:欧米茄

欧米枷:欧米茄

爱立信:爱立信

西铁成:西铁城

瑞新:瑞星

登心绒:灯心绒

这里，前面一个词是错误的词条，后面是对应的错误词条。为了方便维护，我们还可以把这个词库存放在数据库中：

```
CREATE TABLE [dbo].[CommonMisspellings] (

    [misword] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL, --错误词

    [rightword] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL --正确词

) ON [PRIMARY]

GO
```

除了人工整理，还可以从搜索日志中挖掘相似字符串来找出一些可能的正误词对。比较常用的方法是采用编辑距离(Levenshtein Distance)来衡量两个字符串是否相似。编辑距离就是用来计算从原串(s)转换到目标串(t)所需要的最少的插入，删除和替换的数目。例如源串是“诺基压”，目标串是“诺基亚”，则编辑距离是1。

Levenshtein Distance 算法可以看作动态规划。它的思路就是从两个字符串的左边开始比较，记录已经比较过的子串距离，然后进一步得到下一个字符位置时的距离。

```
/**
 *
 * @param s 输入源串
 * @param t 输入目标串
 * @return 源串和目标串之间的编辑距离
 */
public static int LD(String s, String t) {
    int d[][]; // matrix
    int n; // length of s
    int m; // length of t
    int i; // iterates through s
    int j; // iterates through t
    char s_i; // ith character of s
    char t_j; // jth character of t
    int cost; // cost

    // Step 1 初始化
    n = s.length();
    m = t.length();
    if (n == 0) {
        return m;
    }
}
```

```

    if (m == 0) {
        return n;
    }
    d = new int[n + 1][m + 1];

    // Step 2 Initialize the first row to 0..n.
    for (i = 0; i <= n; i++) {
        d[i][0] = i;
    }

    //Initialize the first column to 0..m.
    for (j = 0; j <= m; j++) {
        d[0][j] = j;
    }

    // Step 3 Examine each character of s (i from 1 to n).
    for (i = 1; i <= n; i++) {
        s_i = s.charAt(i - 1);

        // Step 4 Examine each character of t (j from 1 to m).
        for (j = 1; j <= m; j++) {
            t_j = t.charAt(j - 1);

            // Step 5
            // If s[i] equals t[j], the cost is 0.
            // If s[i] doesn't equal t[j], the cost is 1.
            if (s_i == t_j) {
                cost = 0;
            } else {
                cost = 1;
            }

            // Step 6
            //Set cell d[i,j] of the matrix equal to the minimum of:
            //a. The cell immediately above plus 1: d[i-1,j] + 1.
            //b. The cell immediately to the left plus 1: d[i,j-1] +
1.
            //c. The cell diagonally above and to the left plus the cost:
d[i-1,j-1] + cost.
            d[i][j] = Minimum(d[i - 1][j] + 1, d[i][j - 1] + 1,
                d[i - 1][j - 1] + cost);
        }
    }
}

```

```

        // Step 7
        // After the iteration steps (3, 4, 5, 6) are complete, the distance
        is found in cell d[n,m].
        return d[n][m];
    }

```

日志中有这样的记录:

```
2007-05-24 00:41:41.0781|DEBUG|221.221.167.147||喀尔喀蒙古|2
```

```
...
```

```
2007-05-24 00:43:45.7031|DEBUG|221.221.167.147||喀爾喀蒙古|0
```

挖掘日志的程序如下:

//存放挖掘的词及搜索出的结果数

```

HashMap<String,Integer> searchWords = new HashMap<String,Integer>();

while((readline=br.readLine())!=null)
{
    StringTokenizer st = new StringTokenizer(readline,"|");

    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens()) continue;

    //存放搜索词
    String key = st.nextToken();
    if(key.indexOf(":")>=0)
    {
        continue;
    }
}

```

```

//如果已经处理过这个词就不再处理
if(searchWords.containsKey(key))
{
    continue;
}
if(!st.hasMoreTokens())
{
    continue;
}
String results = st.nextToken();
int resultCount = 0;
try
{
    resultCount = Integer.parseInt(results);
}
catch(Exception e)
{
    continue;
}
for(Entry<String,Integer> e : searchWords.entrySet())
{
    int diff= Distance.LD(key, e.getKey()) ;
    if(diff ==1 && key.length()>2)
    {
        if( resultCount == 0 && e.getValue()>0 )
        {
            System.out.println(key + ":" + e.getKey());
            bw.write(key + ":" + e.getKey()+"\r\n");
        }
        else if(e.getValue()==0 && resultCount>0)
        {
            System.out.println(e.getKey() + ":" + key);
            bw.write(e.getKey() + ":" + key+"\r\n");
        }
    }
}
searchWords.put(key, resultCount);
}

```

可以挖掘出如下一些错误、正确词对:

瑜伽服:瑜伽服

落丽塔:洛丽塔

巴甫洛:巴甫洛夫

hello kiitty:hello kitty

...

5.7 自动摘要

减少原文的长度而保留文章的主要意思叫做摘要。

摘要要有各种形式。和搜索关键词相关的摘要，叫做动态摘要；只和文本内容相关的摘要叫做静态摘要。搜索引擎中显示的搜索结果就是关键词相关摘要的例子。按照信息来源来分有来源于单个文档的摘要和合并多个相关文档意思的摘要。单文档摘要精简一篇文章的主要意思，多文档摘要同时可以过滤掉出现在多篇文档中的重复内容。

5.7.1 自动摘要技术

摘要的实现方法有摘取性的和概括性的。摘取性的方法相对容易实现，通常的实现方法是摘取文章中的主要句子。

Sourceforge.net 中的一个项目 Classifier4J 通过抽取指定文本中的重要句子形成摘要。使用它的例子如下：

```
String input = "Classifier4J is a java package for working with text. Classifier4J includes a summariser.";
```

```
//输入文章内容及摘要中需要返回的句子个数。
```

```
String result = summariser.summarise(input, 1);
```

返回结果是： "Classifier4J is a java package for working with text.";

5.7.2 自动摘要的设计

Classifier4J 通过统计高频词和句子分析来实现。

1. 取得高频词；
2. 把内容拆分成句子；
3. 取得包含高频词的前 k 个句子。可以把意思完整的句子分值加高。

4. 将句子按照在文中出现的顺序重新排列，添加适当的分隔符后输出。

为了快速的统计高频词，这里用到的一个重要的方法是从数组中快速的选取最大的 k 个数。我们可以设计接口如下：

`selectRandom(ArrayList<WordFreq> a, int size, int k, int offset)`

输入是待选取的数组，它的长度， k 值和偏移量。根据快速排序的原理，具体实现方法如下：

```
static void selectRandom(ArrayList<WordFreq> a, int size, int k, int
offset) {
    if (size < 5) {
        for (int i = offset; i < (size + offset); i++)
            for (int j = i + 1; j < (size + offset); j++)
                if (a.get(j).compareTo(a.get(i)) < 0)
                    swap(a, i, j);
        return;
    }

    Random rand = new Random();
    int pivotIdx = partition(a, size, rand.nextInt(size) + offset,
offset);
    if (k != pivotIdx) {
        if (k < pivotIdx) {
            selectRandom(a, pivotIdx - offset, k, offset);
        } else
        {
            selectRandom(a, size - pivotIdx - 1 + offset, k, pivotIdx
+ 1);
        }
    }
}
```

```
static int partition(ArrayList<WordFreq> a, int size, int pivot, int
offset) {
    WordFreq pivotValue = a.get(pivot);

    swap(a, pivot, size - 1 + offset);
    int storePos = offset;
    for (int loadPos = offset; loadPos < (size - 1 + offset); loadPos++)
    {
```

```
        if (a.get(loadPos).compareTo(pivotValue) < 0) {
            swap(a, loadPos, storePos);
            storePos++;
        }
    }
    swap(a, storePos, size - 1 + offset);
    return (storePos);
}
```

//对数组中的5个值排序。

```
static int median5(ArrayList<WordFreq> a, int offset) {
    WordFreq a0 = a.get(0 + offset);
    WordFreq a1 = a.get(1 + offset);
    WordFreq a2 = a.get(2 + offset);
    WordFreq a3 = a.get(3 + offset);
    WordFreq a4 = a.get(4 + offset);

    if (a1.compareTo(a0) < 0) {
        WordFreq tmp = a0;
        a0 = a1;
        a1 = tmp;
    }
    if (a2.compareTo(a0) < 0) {
        WordFreq tmp = a0;
        a0 = a2;
        a2 = tmp;
    }
    if (a3.compareTo(a0) < 0) {
        WordFreq tmp = a0;
        a0 = a3;
        a3 = tmp;
    }
    if (a4.compareTo(a0) < 0) {
        WordFreq tmp = a0;
        a0 = a4;
        a4 = tmp;
    }
    if (a2.compareTo(a1) < 0) {
        WordFreq tmp = a1;
        a1 = a2;
        a2 = tmp;
    }
}
```



```
    if (a3.compareTo(a1) < 0) {
        WordFreq tmp = a1;
        a1 = a3;
        a3 = tmp;
    }

    if (a4.compareTo(a1) < 0) {
        WordFreq tmp = a1;
        a1 = a4;
        a4 = tmp;
    }

    if (a3.compareTo(a2) < 0) {
        WordFreq tmp = a3;
        a3 = a2;
        a2 = tmp;
    }

    if (a4.compareTo(a2) < 0) {
        WordFreq tmp = a4;
        a4 = a2;
        a2 = tmp;
    }

    if (a2 == a.get(0 + offset))
        return 0;
    if (a2 == a.get(1 + offset))
        return 1;
    if (a2 == a.get(2 + offset))
        return 2;
    if (a2 == a.get(3 + offset))
        return 3;

    return 4;
}
```

参考 Classifier4J 的实现方法，中文自动摘要的基本实现方法如下 5 个步骤：

- 通过中文分词，统计词频和词性等信息，抽取出关键词。
- 把文章划分成一个个的句子。
- 通过各句中关键词出现的情况定义出句子的重要度。

- 确定前 K 个最重要的句子为文摘句。
- 把文摘句按照在原文中出现的顺序输出成摘要。

主体程序如下：

```

ArrayList<CnToken> pItem = Tagger.getFormatSegResult(rouseStr);
WordFreq[] charArray = new WordFreq[10];

WordCounter wordCounter = new WordCounter();

for (int i = 0; i < pItem.size(); ++i) {
    CnToken t = pItem.get(i);
    if (t.type().startsWith("n")) {
        wordCounter.ProNChar(t.termText());
    } else if (t.type().startsWith("v")) {
        wordCounter.ProVChar(t.termText());
    }
}

//取得出现的频率最高的五个名词
WordFreq[] charNArray =
wordCounter.getWords(wordCounter.CharNCount);

for (int mn = 0; mn < 5; mn++) {
    charArray[mn] = charNArray[mn];
}

//取得出现的频率最高的五个动词
WordFreq[] charVArray =
wordCounter.getWords(wordCounter.CharVCount);
for (int mn = 5; mn < 10; mn++) {
    charArray[mn] = charVArray[mn - 5];
}

//抽取句子
SentenceExtractor senCou = new SentenceExtractor();
ArrayList<SentenceScore> sentenceArray =
senCou.getSentences(rouseStr);

int q = 0;
int sumCount = 1;
//计算句子权重
while ( q<sentenceArray.size() ) {

```

```

String sentenCompare = sentenceArray.get(q).sentence;

for (int j = 0; j < 10 ; j++) {
    //System.out.println("w:" +j);
    String charCompare = charArray[j].word;
    if(charCompare == null)
    {
        break;
    }
    //System.out.println("比较的词语依次为:"+charCompare);
    int k = sentenCompare.indexOf(charCompare) + 1;
    //System.out.println("词语在数组中的位置为:"+k);
    if (k >= 1) {
        sumCount = sumCount * charArray[j].freq;
    } else {
        sumCount = sumCount * 1;
    }
}
sentenceArray.get(q).score = sumCount;
sumCount = 1;
q++;
}

ArrayList<SentenceScore> copySenArr = new
ArrayList<SentenceScore>();
for(SentenceScore sc:sentenceArray)
{
    copySenArr.add(sc);
}

int minSize = Math.min(sentenceArray.size(), 3);
(new Select<SentenceScore>()).selectRandom(copySenArr,
copySenArr.size(), minSize,0);
//System.out.println("");
for(int i=0;i<minSize;i++)
{
    System.out.println("权值最大的三个句子为:"+
        copySenArr.get(i).sentence+
        "该句子的权值为:"+
        copySenArr.get(i).score);
}

//句子在原文中出现的顺序输出
String summary = "";

```

```

        for (int i = 0; i < minSize; i++) {
            for (int j = 0; j < minSize; j++) {
                if
(sentenceArray.get(i).sentence.equalsIgnoreCase(copySenArr.get(j).sen
tence)) {
                    summary =
summary.concat(sentenceArray.get(i).sentence);
                    //System.out.println(summary);
                }
            }
        }

return summary;

```

这样的一个机械方式的摘要程序还比较简单。每个过程都可以优化。

其中第一步，在提取关键词阶段，可以去掉停用词表，然后再统计关键词。也可以考虑利用同义词信息更准确的统计词频。

在第二步，划分句子阶段，可以记录句子在段落中出现的位置，在段落开始或结束出现的句子更有可能是关键句。同时可以考虑句型，陈述句比疑问句或感叹句更有可能是关键句。

```

public static enum SentenceType {declare, //陈述句
                                question, //疑问句
                                exclamation //感叹句
                                }

```

句子权重统计阶段考虑句型来打分。

```

//判断句子类型
if(sc.type == SentenceScore.SentenceType.question)
{
    sc.score *= 0.1;
}
else if(sc.type == SentenceScore.SentenceType.exclamation)
{
    sc.score *= 0.5;
}

```

为了使输出的摘要意义连续性更好，有必要划分段落。识别自然段和更大的意义段。自然段一般段首缩进两个或四个空格。

在对句子打分时，除了关键词，还可以查看事先编制好的线索词表。表示线索词的权值，有正面的和负面的两种。文摘正线索词就是类似“总而言之”、“总之”、“本文”、“综上所述”

等词汇，含有这些词的句子权重有加分。文摘负线索词可以是“比如”，“例如”等。如果句中包括这些词权重就会降低。

最后为了输出的摘要通顺，还需要处理句子间的关联关系。例如下面的关联句子：

“这个节目，需要的是接班人，而不是变革者。”换言之，一个节目的“心”的意义是大于“脸”的意义的；“换脸”未必就是“换心”；但《新闻联播》目前还只能接班性地“换脸”而不能变革性地“换心”。

处理关联句子的方法有三种：

- 调整关联句的权重，使更重要的句子优先成为摘要句。
- 调整关联句的权重，将关联的两个句子都成为或都不成为摘要句。
- 输出摘要时，如果不能完整的保持相关联的句子，则删除句前的关联词。

句子间的关联通过关联性的词语来表示。处理关联句可以根据关联性词语的类型分别处理。

关联类型	关联词	处理方式
转折	虽然…但是…	对于这类偏正关系的，调整后部分的关键句的权重，保证其大于前面部分的权重。当只有一句是摘要句时，删除该句前的关联词。
因果	因为…所以…/因此	
递进	不但…而且… 尤其	
并列	一方面…另一方面…	对于这类并列关系，使关键句的权重都一样。找不到对应的关联句的删除该句子前面的关连词。
承接	接着 然后	
选择	或者…或者	
分述	首先…其次…	
总述	总而言之 综上所述 总之	这类可承前省略的，如果与前面的句子都是摘要句，则保持不变。否则，如果前面的句子不是摘要句则删除该句子前面的关连词。
等价	也就是说 即 换言之	
话题转移	另外	
对比	相对而言	

关联类型	关联词	处理方式
举例说明	比如 例如	这类可承前省略的，如果与前面的句子都是摘要句，则保持不变。否则删除后面的句子。

5.7.3 Lucene 中的动态摘要

Lucene 扩展中有一个 Highlighter 自动摘要的包。

通过调用：`getBestFragments` 返回一个或多个和搜索关键词最相关的段落。

- `Fragmenter` 把文本分成多个段落。
- `QueryScorer` 计算每个段落的分值。`QueryScorer` 只应该包含需要做高量显示的 `Term`。
- `TokenStream` 通过 `IndexReader` 中的位置信息来返回一个 `TokenStream`，而不要再次分词处理原文后返回 `TokenStream`。如果 `IndexReader` 中的 `Token` 位置有重叠，为了把冗余的 `Token` 去掉，`TokenStream` 计算起来会麻烦一些。

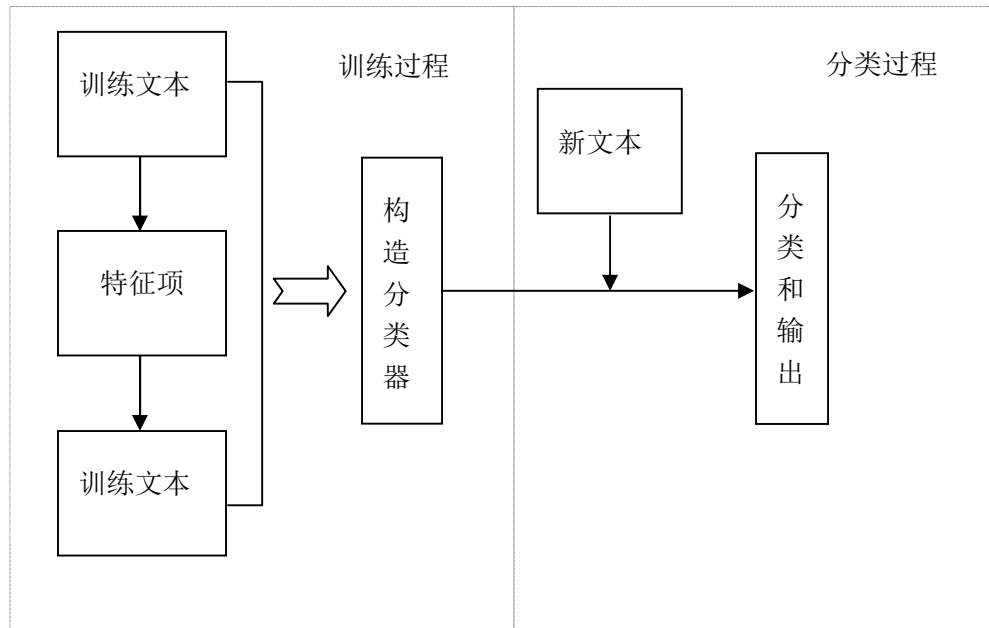
```

Highlighter highlighter =new Highlighter(this,new
QueryScorer(query));
highlighter.setTextFragmenter(new SimpleFragmenter(40));
for (int i = 0; i < hits.length(); i++)
{
    String text = hits.doc(i).get(FIELD_NAME);
    int maxNumFragmentsRequired = 2;
    String fragmentSeparator = "...";
    TermPositionVector tpv =
(TermPositionVector)reader.getTermFreqVector(hits.id(i),FIELD_NAME);
    TokenStream
tokenStream=TokenSources.getSingleTokenStream(tpv,true);
    String result
=highlighter.getBestFragment(tokenStream,text);
    System.out.println("\t" + result);
}

```

5.8 自动分类

自动分类程序把一个未见过的文档分成已知类别中的一个或多个。分成一个类别叫做单类分类，分成多个类别叫做多类分类。一个典型的自动分类程序框架如下：



常见的分类方法有支持向量机(SVM), K 个最近的邻居(KNN)和贝叶斯(bayers)等。这里用 SVM 方法实现文本分类。

5.8.1 Classifier4J

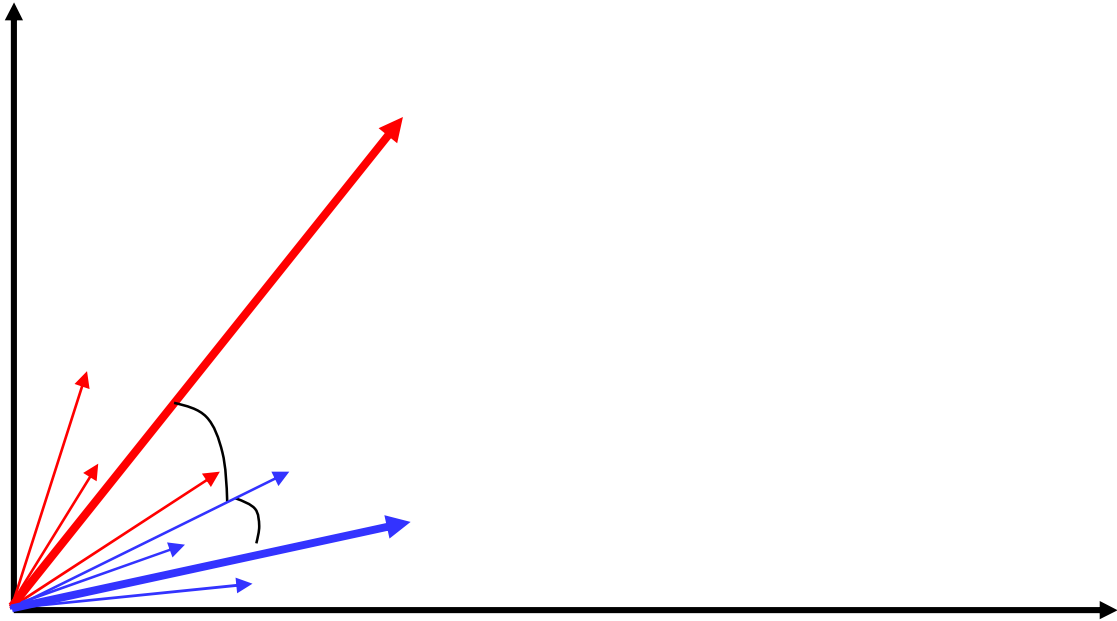
中心向量法和贝叶斯方法。

中心向量法又叫 Rocchio 方法。

使用标准的 TF/IDF 带权重的向量表示文档 (normalized by maximum term frequency)。

通过把训练集文档中的向量加在一起对每一个类别计算一个原型向量。

把测试文档和每一个类别的原型向量计算相似度,把该文档归为相似度最近的原型向量所代表的类。



```
//定义存储向量的变量
TermVectorStorage storage = new HashMapTermVectorStorage();
//新建一个向量分类器
VectorClassifier vc = new VectorClassifier(storage);
try {
    //定义一个叫做"test"的类别
    String category = "test";
    //训练一个句子属于"test"类别
    vc.teachMatch(category, sentence1);

    //距离是cos夹角 相似度在[0,1]之间 0.852 代表比较相似 "hello blah"
    //属于这个类
    assertEquals(0.852d, vc.classify(category, "hello blah"),
0.001);

    //距离是cos夹角 相似度在[0,1]之间 0.301 代表不太相似 "sentence" 不
    //属于这个类
    assertEquals(0.301d, vc.classify(category, "sentence"),
0.001);

    //0 代表不相似 "bye" 不属于这个类
    assertEquals(0.0d, vc.classify(category, "bye"), 0.001);
    //0 代表不相似 "bye" 不属于 "does not exist" 这个类
    assertEquals(0.0d, vc.classify("does not exist", "bye"),
0.001);
} catch (ClassifierException e) {
    e.printStackTrace();
    fail(e.getLocalizedMessage());
}
```


5.8.2 自动分类的接口定义

准备训练文本，例如文本路径在：

D:/train

类别路径是：

D:\train\zippo 收藏乐器性用品

D:\train\宠物玩具音像书

D:\train\电脑数码产品办公设备

D:\train\房屋

D:\train\服装箱包首饰钟表眼镜

D:\train\家电家具居家

D:\train\交友婚介祝福寻人寻物

D:\train\教育培训

D:\train\美容按摩医药

D:\train\汽车交通

D:\train\人才职场

D:\train\生活服务

D:\train\手机电话卡号

D:\train\玩运动出游

训练过程的接口如下：

```
//创建一个分类器
Classifier svmClassifier = new Classifier();

//设置训练文本的路径
svmClassifier.setTrainSet("D:/train");
//设置训练输出模型的路径
```

```
svmClassifier.setModel("D:/model");

//执行训练

svmClassifier.train();
```

执行分类的接口如下：

```
Classifier theClassifier = new
Classifier("D:/xiaoxishu/model/model.prj");

String content="我要买把吉他，希望是二手的，价格 2000 元以下";

System.out.println("SVM分类开始");

String catName = theClassifier.getCategoryName(content);

System.out.println("类别名称:"+catName);
```

5.8.3 自动分类的 SVM 方法实现

可以按词性过滤，只选择某些词性的，文本分类的精度随词的个数持续提高。一般至少可以到 2000 个词。

5.8.4 多级分类

以二级分类为例，在路径：

D:\train\zippo 收藏乐器性用品

下建立子路径：

D:\train\zippo 收藏乐器性用品\打火机 zippo 烟具

D:\train\zippo 收藏乐器性用品\古董收藏

D:\train\zippo 收藏乐器性用品\乐器乐谱

D:\train\zippo 收藏乐器性用品\性用品

D:\train\zippo 收藏乐器性用品\邮币卡字画

下面是训练二级分类的程序：

```

//训练主分类
String strPath = "D:/lg/work/xiaoxishu/train";
String modelPath = "D:/lg/work/xiaoxishu/model";
Classifier svmClassifier = new Classifier();

svmClassifier.setTrainSet(strPath);
svmClassifier.setModel(modelPath);
svmClassifier.train();

File dir = new File(strPath);
File[] files = dir.listFiles();
for (int i = 0; i < files.length; i++)
{
    File f = files[i];

    if (f.isDirectory())
    {
        //训练子分类
        System.out.println(f.getAbsolutePath().getName());
        String subTrain = strPath + "/" +
f.getAbsolutePath().getName();
        Classifier subClassifier = new Classifier();
        //
        subClassifier.setTrainSet(subTrain);

        String subModelPath = modelPath + "/" +
f.getAbsolutePath().getName();
        subClassifier.setModel(subModelPath);
        subClassifier.train();
    }
}

```

下面是分类的执行过程

```

String modelPath = "D:/lg/work/xiaoxishu/model";
Classifier theClassifier = new
Classifier(modelPath+"/model.prj");

String content = "我要买把吉他，希望是二手的，价格2000元以下"; //分类
文本内容
System.out.println("分类开始");

```

```

String catName = theClassifier.getCategoryName(content);

System.out.println("catName:"+catName);
if(catName == null)
{
    //如果没有主分类则返回
    return;
}
String subModelPath = modelPath+"/"+catName+"/model.prj";
Classifier subClassifier = new Classifier(subModelPath);

String subCatName = subClassifier.getCategoryName(content);

System.out.println("subCatName:"+subCatName);

```

上面的执行结果将打印出：

分类开始

catName:zippo收藏乐器性用品

subCatName:乐器乐谱

也就是把 "我要买把吉他，希望是二手的，价格 2000 元以下" 分类成：大类是“zippo 收藏乐器性用品”，子类是“乐器乐谱”。

特征选择方法：

CHI 方法

IG (Information Gain: IG)

5.9 自动聚类

5.9.1 聚类的定义

将一个数据对象的集合分组成为类似的对象组成的多个类的过程称为聚类。每一个类称为簇，同簇中的对象彼此相似，不同簇中的对象相异。聚类不同于前面提到的分类，它不需要训练集合。

文档聚类就是对文档集合进行划分，使得同类间的文档相似度比较大，不同类的文档相似度较小，不需要预先对文档标记类别，具有较高的自动化能力，已经成为文本信息进行有

效地组织、摘要和导航的重要手段。

5.9.2 K 均值聚类方法

目前存在着大量的聚类方法。算法的选择取决于数据的类型、聚类的目的和应用。在众多的方法中，K 均值方法是一种比较流行的方法且其聚类的效果也比较好。

K 均值方法是把含有 n 个对象的集合划分成 k 个簇。每一个簇中对象的平均值称为该簇的聚点（中心）。两个簇的相似度就是根据两个聚点而计算出来的。假设聚点 x 、 y 都有 m

个属性，取值分别为 x_1, x_2, \dots, x_m 和 y_1, y_2, \dots, y_m ，则 x 和 y 的距离 $d_{xy} = \left(\sum_{k=1}^m |x_k - y_k|^2 \right)^{\frac{1}{2}}$ 。

K 均值算法有如下 5 个步骤：

任意从 n 个对象中选择 k 个对象做为初始簇的中心

根据簇中对象的平均值，即簇的聚点，将每个对象（重新）付给最类似的簇

重新计算每个簇的平均值，即更改簇的聚点。

若某些簇的聚点发生了变化，转步骤 2)；若所有的簇的聚点无变化，转步骤 5)

输出划分结果

K 均值算法的流程图如图 1 所示。

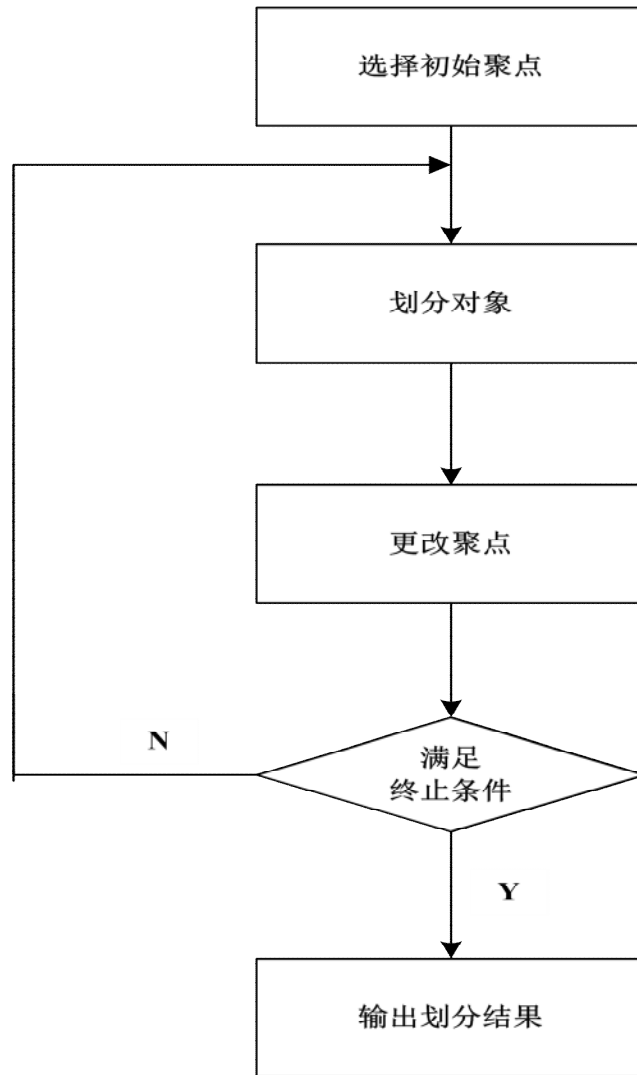


图 1 K 均值算法流程图

对于图 1 流程中的前三个步骤都有各种方法，通过组合可以得到不同的划分方法。下面在 K 均值算法的基础上，许多改进算法在如何选择初始聚点、如何划分对象及如何修改聚点等方面提出了不同的方法。

选择初始聚点的方法

初始聚点的选择对最终的划分有很大的影响。选择的初始聚点不同，算法求的解也不同。选择适当的初始聚点可以加快算法的收敛速度并改善解的质量。

选择初始聚点的方法有以下几种。

- 1) 随机选择法。随机的选择 K 个对象作为初始聚点。
- 2) 最小最大法。现选择所有对象中相距最远的两个对象作为聚点，然后选择第三个聚

点,使得它与已确定的聚点的最小距离是其余对象与已确定的聚点的较小距离中最大的,然后按同样的原则选择以后的聚点。

3) 最小距离法。选择一个正数 r ,把所有对象的中心作为第一个聚点,然后依次输入对象,如果当前输入对象与已确定的聚点的距离都大于 r ,则该对象作为一个新的聚点。

划分方法

划分方法就是决定当前对象应该分到哪一个簇中。划分方法中最为流行的是最近归类法,即将当前对象归类于距离最近的聚点。

修改聚点的方法

1) 按批修改法。把全部对象输入后再修改聚点和划分。步骤如下:①选择一组聚点;②根据聚点划分对象;③计算每个簇的中心作为新的聚点,如果划分合理则停止,否则转②。

2) 逐个修改法。每输入一个对象的同时就改变该对象所归簇的聚点。步骤如下:①选择一组聚点;②将余下的对象依次输入,每输入一个对象,按当前的聚点将其归类,并重新计算中心,代替原先的聚点;③如果划分合理则停止,否则转②。

5.9.3 K 均值实现

```
public class KMeans {
    //to computer the EuclideanDistance
    private static double EuDistance(double array1[], double array2[])
    {
        double Dist = 0.0;
        if (array1.length != array2.length) {
            System.out.println("the number of the array is ineql");
        } else {
            for (int i = 0; i < array2.length; i++) {
                Dist = Dist + (array1[i] - array2[i]) * (array1[i] -
array2[i]);
            }
        }
        return Math.sqrt(Dist);
    }

    //to print the int Array
    private static void printArray(int array[]) {
        System.out.print('[ ');
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i]);
        }
    }
}
```

```

        if ((i + 1) < array.length) {
            System.out.print(", ");
        }
    }

    System.out.println(']');
}

//to print the Matrix
private static void printMatrix(double Matrix[][], int row, int col){
    System.out.println("Matrix is:");
    System.out.println('{');
    for(int i=0; i<row; i++){
        for (int j = 0; j < col; j++) {
            //Matrix[i][j]=-1.0; for test
            System.out.print(FORMAT.format(Matrix[i][j]));
            if ((j + 1) < col) {
                System.out.print(", ");
            }
        }
        System.out.println();
    }
    System.out.println('}');
}

private static DecimalFormat FORMAT = new DecimalFormat("000.000");

//Randperm the ?M form the randpern(N)
private static int[] Randperm(int N,int M){
    double[] PermF=new double[N];
    int[] PermI=new int[N];
    int[] RetArray=new int[M];
    double tempF;
    int tempI;
    for(int i=0; i<N; i++){
        PermF[i]=Math.random();
        PermI[i]=i;
    }
    //sort choosing the big to forward
    for(int i=0; i<N-1; i++){
        for(int j=i+1; j<N; j++){
            if(PermF[i]<PermF[j]){
                tempF=PermF[i];
                tempI=PermI[i];
            }
        }
        PermF[i]=PermF[j];
        PermI[i]=PermI[j];
    }
    for(int i=0; i<M; i++){
        RetArray[i]=PermI[PermI[i]];
    }
}

```



```
        PermF[i]=PermF[j];
        PermI[i]=PermI[j];
        PermF[j]=tempF;
        PermI[j]=tempI;
    }
}

for(int i=0; i<M; i++){
    RetArray[i]=PermI[i];
}
return RetArray;
}

//the judge the equal two Array
private static boolean IsEqual(int Array1[],int Array2[]){
    for(int i=0; i<Array1.length; i++){
        if(Array1[i]!=Array2[i]){
            return false;
        }
    }
    return true;
}

//get the location of min element from the Array
private static int MinLocation(double Array[]){
    int Location;
    double Min;
    //initial
    Min=Array[0];
    Location=0;
    //Iteration
    for(int i=1; i<Array.length; i++){
        if(Array[i]<Min){
            Location=i;
            Min=Array[i];
        }
    }
    return Location;
}

//to clustering the data Matrix
private static int[] KMeansCluster(double Matrix[][], int row, int
col,int ClusterNum){
```

```

int[] CenterId=new int[ClusterNum];
int[] Cid=new int[row];
int[] oldCid=new int[row];
int[] NumOfEveryCluster=new int[ClusterNum];
double[][] ClusterCenter=new double[ClusterNum][col];
double[] CenterDist=new double[ClusterNum];
//initial the ClusterCenter
//random get the ClusterCenter
CenterId=Randperm(row,ClusterNum);
for(int i=0; i<ClusterNum; i++){
    for(int j=0; j<col; j++){
        ClusterCenter[i][j]=Matrix[ CenterId[i] ][j];
    }
}
//initial the oldCide
for(int i=0; i<row; i++){
    oldCid[i]=1;
}
int MaxIter=100;
int Iter=1;

while( !IsEqual(Cid,oldCid) && Iter<MaxIter){
    for(int i=0;i<row;i++){
        oldCid[i]=Cid[i];
    }

    //Implement the hmeans algorithm
    //For each Point, find the distance
    //to all cluster centers
    for(int i=0;i<row;i++){
        for(int j=0; j<ClusterNum;j++){
            CenterDist[j]=EuDistance(Matrix[i], ClusterCenter[j] );
        }
        Cid[i]=MinLocation(CenterDist);
    }

    //to get the number of every cluster
    for(int j=0; j<ClusterNum; j++){
        NumOfEveryCluster[j]=0;
        for(int i=0; i<row; i++){
            if(Cid[i]==j){
                NumOfEveryCluster[j]=NumOfEveryCluster[j]+1;
            }
        }
    }
}

```

```

        }
    }
}

//Find the new cluster centers
//sum the .....
for(int j=0; j<ClusterNum; j++){
    for(int k=0; k<col; k++){
        ClusterCenter[j][k]=0.0;
        for(int i=0; i<row; i++){
            if(Cid[i]==j){

ClusterCenter[j][k]=ClusterCenter[j][k]+Matrix[i][k];

            }
        }
    }
}

//to means the sum...
for(int j=0; j<ClusterNum; j++){
    for(int k=0; k<col; k++){

ClusterCenter[j][k]=ClusterCenter[j][k]/NumOfEveryCluster[j];

    }
}
++Iter;
}

return Cid;
}

//main to test the KMeans Cluster
public static void main(String[] args) {
    int Matrix_row;
    int Matrix_col;
    int ClusterNum;
    Matrix_col=5;
    Matrix_row=20;
    ClusterNum=3;

    double[][] Matrix = new double[Matrix_row][Matrix_col];

    int[] List=new int[Matrix_row];

    for(int i=0; i<Matrix_row; i++){
        for(int j=0; j<Matrix_col; j++){

```

```

        Matrix[i][j]=10*Math.random();
    }
}
//for test the code
double[][] DistMatrix=new double[Matrix_row][Matrix_row];
for(int i=0; i<Matrix_row; i++){
    for(int j=0; j<Matrix_row; j++){
        DistMatrix[i][j]=EuDistance(Matrix[i],Matrix[j]);
    }
}

printMatrix(Matrix,Matrix_row,Matrix_col);
System.out.println("The Distance Matrix is:");
printMatrix(DistMatrix,Matrix_row,Matrix_row);

//System.out.print(IsEqual(List,List1 ));
List=KMeansCluster(Matrix, Matrix_row, Matrix_col,ClusterNum);
System.out.println("The result of clustering, value of No.i means
the ith belong to the No.value cluster");
printArray(List);
}
}

```

5.10 拼音转换

从日志中学习 BigramDict.txt 和 TrigramDict.txt。

Converter 拼音转换成汉字。

5.11 语义搜索



Baidu 西红柿牛腩 百度一下

把百度设为首页

[西红柿牛腩的做法,西红柿牛腩怎么做\[图解\]-家常菜谱](#)
 1、牛腩过水去血末,葱姜爆锅炒牛腩,肉紧了喷料酒,放盐,炒下,(不要加酱油)加水或者高汤(水不要多了,刚好过肉一点就可以了),转高压锅上气小火压20分钟。2、压肉的时候炒西红柿,我没用西红柿酱,就用了4个大西红柿,用葱末小...
www.ttmeishi.com/CaiPu/7ded91680da9518e.htm 15K 2008-9-29 - 百度快照
[www.ttmeishi.com 上的更多结果](#)

[番茄炖牛腩 百度百科](#)
 番茄炖牛腩 英文名: Braised Beef Brisket with Tomato 美食配料: 牛肉; 西红柿(4个小的)。花椒、大料少许。美食特色: 味美汤浓, 趁热喝! 可以在吃饭的时候小火炖汤, 保持温度! 美食做法: 1、将锅中做水, 然后等水开后可以一边切一边下肉。...
baike.baidu.com/view/1693674.htm 18K 2008-7-16 - 百度快照
[baike.baidu.com 上的更多结果](#)

[番茄牛腩汤 是怎么做的呀? 百度知道](#)
 牛腩、番茄各约320克,八角1粒,青椒2个,葱1条(切段),蒜1瓣(拍裂),姜1块(切片),油4又1/2汤匙,水溶粟粉1/2汤匙。 配料: 调味料: 生抽2汤匙,盐1/4茶匙,糖1茶匙,胡椒粉适量。 做法:
 1、牛腩洗净切大块,下3/2汤匙油拌匀,...
zhidao.baidu.com/question/3011118.html 13K 2007-5-6 - 百度快照
[zhidao.baidu.com 上的更多结果](#)

非字面匹配的结果

当你输入招商行的时候, google 返回的第一个结果是招商银行的网站。猜一下 google 怎么知道“招商银行=招商行”。

他并不是维护一个词库。而是按照如果有 url link 介绍这个网站是“招商行”,他就知道了。

中国招商银行

招行

招商银行

提取出同义词:

中国招商银行 招行 招商银行

我们这里通过同义词搜索来尝试语义扩展的搜索。我们实现的方法是通过查询扩展的方式实现。当用户输入“计算机”搜索的同时，程序通过查找同义词库，按照“计算机”，“电脑”，“微机”等多个同义词查找。当用户输入“轿车”的同时，也能按照“奥迪”，“奔驰”等进行下位词扩展。

实现这个功能的基本步骤如下：

1. 准备语义词库；
2. 把语义词库转换成同义词索引库；
3. 在 SynonymAnalyzer 中使用同义词索引库。

英文的同义词词库最著名的是 WordNet，它的介绍和下载地址在 <http://wordnet.princeton.edu>。中文的同义词词林比较流行。同义词采用了当前比较流行的同义词词林的格式，因为它正好用树型结构表示了词的同义和上下位关系。下面是同义词的例子：

Bp31B 表

Bp31B01= 表 手表

Bp31B02= 马表 跑表 停表

Bp31B03= 怀表 挂表

Bp31B04= 防水表 游泳表

Bp31B05= 表针 指针

Bp31B06= 表盘 表面

Bp31B07# 夜光表 秒表 电子表 日历表 自动表

Bp31B08= LONGINES 浪琴

Bp31B09= Rema 瑞尔玛

Bp31B10= ROSSINI 罗西尼

Bp31B11= SEIKO 精工

Bp31B12= SUUNTO 松拓

Bp31B13= Tissot 天梭

Bp31B14= CASIO 卡西欧

整理出这样的词表后，我们通过下面的程序转换成 WordNet 的 prolog 同义词数据库 wn_s.pl 格式：

```
s(synset_id,w_num,'word',ss_type,sense_number,tag_count).
```

第一列是语义编码，同一个语义有唯一的语义编码列，第二列是单词编号，第三列是单词本身，第四列是词性，第五列代表该词第几个语义，第六列是该词在语料库中出现的频率。例如：

```
s(100006026,1,'person',n,1,7229).
s(100006026,2,'individual',n,1,51).
s(100006026,3,'someone',n,1,17).
s(100006026,4,'somebody',n,1,0).
s(100006026,5,'mortal',n,1,2).
s(100006026,6,'human',n,1,7).
s(100006026,7,'soul',n,2,6).
```

同义词词林按照该格式整理如下：

```
s(Dk02A12,1,'中学',n,1,0).
s(Dk02A12,2,'国学',n,1,0).
s(Dk02A12,3,'旧学',n,1,0).
```

实现转换的代码如下：

```
public static void CL2WordNet(String sSourceFile,String
save_filename,String dic_file){
    DicCore dic = new DicCore(dic_file);
    BufferedReader fpSource = null;
    BufferedWriter output=null;
    String sLine;
    StringTokenizer st = null;
    String number= null;

    try{
        fpSource= new BufferedReader(new FileReader(sSourceFile));
        output = new BufferedWriter(new FileWriter(save_filename));

        while( (sLine = fpSource.readLine()) != null )
        {

            if(sLine.length(>9 )
            {
                int pos = sLine.indexOf('=');
                if (pos<=0)
```

```

        {
            continue;
        }
        number = sLine.substring(0,pos);
        st = new StringTokenizer(sLine.substring(pos+1).trim(), "\t\n\r");

        String word;
        int count = 0;
        String wordPOS = "n";

        try{
            while(st.hasMoreElements())
            {
                count++;
                word = st.nextToken();
                POSQueue posQ = dic.get(word);

                if(posQ==null)
                {
                    System.out.print(word+": "+wordPOS+"\n");
                }
                else if (posQ.size()==1)
                {
                    wordPOS =
DicCore.gePOSName(posQ.getHead().item.nPOS);
                }

                output.write("s("+number+", "+count+", '"+word+"', "+wordPOS+",1,0).
" +"\n");
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    output.close();
    fpSource.close();
}
catch(Exception e)
{
    e.printStackTrace();
}

```


简体到繁体： 简体文字做分词， 通过词级别的对照转换到繁体。

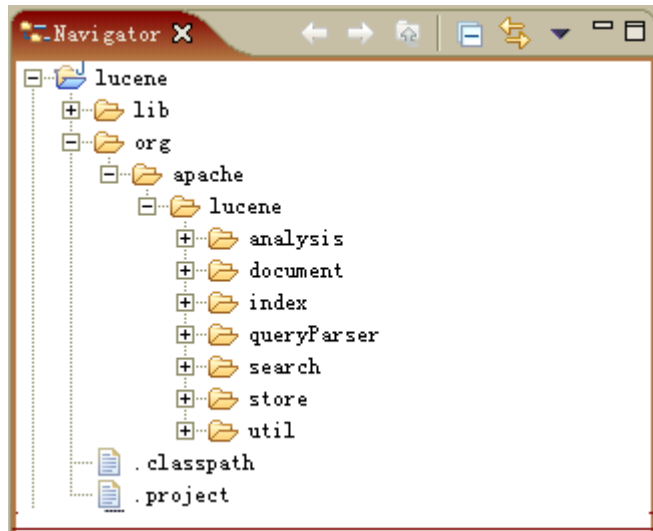
reader 做一个转换 输入繁体，转换成简体。

每个语言储存一列。

5.13 本章小结

第6章 创建索引库

在创建索引库之前，我们对 lucene 的包结构做个详细的介绍，以便读者对 lucene 有更深入的了解。



如上图所示，Lucene 源码中共包括 7 个子包，每个包完成特定的功能：

Lucene 七个子包结构的功能说明	
包名	功能
org.apache.lucene.analysis	语言分析器，主要用于的切词，支持中文主要是扩展此类
org.apache.lucene.document	索引存储时的文档结构管理，类似于关系型数据库的表结构
org.apache.lucene.index	索引管理，包括索引建立、删除等
org.apache.lucene.queryParser	查询分析器，实现查询关键词间的运算，如与、或、非等
org.apache.lucene.search	检索管理，根据查询条件，检索得到结果
org.apache.lucene.store	数据存储管理，主要包括一些底层的 I/O 操作
org.apache.lucene.util	一些公用类

6.1 设计索引库结构

6.1.1 理解 Lucene 的索引库结构

一个索引库类似一个数据库的表结构，但是只能存储字符串。如果是日期或者数字，需要专门的方法转换成字符串后再索引。

数据类型	在 Lucene 中的相关函数
date	dateToString 把日期转换成字符串 stringToDate 把字符串转换成日期
long	longToString 把长整型转换成字符串 stringToLong 把字符串转换成长整型

这里没有针对货币的转换，我们可以自己设计这两个方法。

```
public static String CurrencyToString(double d)
{
    //小数点后两位
    long l = (long)(d * 100);
    return longToString(l);
}

public static double StringToCurrency(String str)
{
    long l = stringToLong(str);
    double d = (double)l / 100;
    return d;
}
```

Lucene 也没有针对小数类型的转换，下面是一个实现：

```
// uses binary representation of an int to build a string of
// chars that will sort correctly. Only char ranges
// less than 0xd800 will be used to avoid UCS-16 surrogates.
// we can use the lowest 15 bits of a char, (or a mask of 0x7fff)
public static int long2sortableStr(long val, char[] out, int offset)
{
    val += Long.MIN_VALUE;
```

```

        out[offset++] = (char)(val >>>60);
        out[offset++] = (char)(val >>>45 & 0x7fff); //取17位 val从45 - 60
        out[offset++] = (char)(val >>>30 & 0x7fff);
        out[offset++] = (char)(val >>>15 & 0x7fff);
        out[offset] = (char)(val & 0x7fff);
        return 5;
    }

    public static long SortableStr2long(String sval, int offset, int len)
    {
        long val = (long)(sval.charAt(offset++)) << 60;
        val |= ((long)sval.charAt(offset++)) << 45;
        val |= ((long)sval.charAt(offset++)) << 30;
        val |= sval.charAt(offset++) << 15;
        val |= sval.charAt(offset);
        val -= Long.MIN_VALUE;
        return val;
    }

    public static String long2sortableStr(long val) {
        char[] arr = new char[5];
        long2sortableStr(val, arr, 0);
        return new String(arr, 0, 5);
    }

    public static String double2sortableStr(double val) {
        long f = Double.doubleToRawLongBits(val);
        if (f<0) f ^= 0x7fffffffffffffffffL;
        return long2sortableStr(f);
    }

    public static String double2sortableStr(String val) {
        return double2sortableStr(Double.parseDouble(val));
    }

    public static double SortableStr2double(String val) {
        long f = SortableStr2long(val, 0, 6);
        if (f<0) f ^= 0x7fffffffffffffffffL;
        return Double.longBitsToDouble(f);
    }

    public static String SortableStr2doubleStr(String val) {
        return Double.toString(SortableStr2double(val));
    }
}

```

名称	描述	在 Lucene 中的定义
title	存储新闻标题，需要分词和关键词加亮	<pre>new Field("title", title , Field.Store.YES, Field.Index.TOKENIZED, Field.TermVector.WITH_POSITIONS_OFFSETS);</pre>
body	存储新闻正文，需要分词和关键词加亮	<pre>new Field("body" , body , Field.Store.YES, Field.Index.TOKENIZED, Field.TermVector.WITH_POSITIONS_OFFSETS);</pre>
url	存储新闻的来源 URL 地址	<pre>new Field("url" , url , Field.Store.YES, Field.Index.UN_TOKENIZED, Field.TermVector.NO);</pre>
date	存储日期字段，在 Lucene 中需要转换成字符串存储	<pre>new Field("date" , DateTools.dateToString(pubDate,Resolution.DAY) , Field.Store.YES, Field.Index.UN_TOKENIZED, Field.TermVector.NO);</pre>

6.2 创建和维护索引库

6.2.1 创建索引库

可以通过 `IndexWriter` 来创建一个新的索引库。相关的参数有：索引路径，分词类，和是否增量索引。

```
index = new IndexWriter(new File(indexDir), new StandardAnalyzer(),
                        !incremental);
```

Lucene 中所有的列都是字符串。对于日期列需要通过 `DateTools` 来转换。下面转换到精度为天的字符串。

```
DateTools.dateToString(rs.getDate("regDate"), Resolution.DAY);
```

转换时使用的时区是 GMT。为了正确地转换，往往需要指定缺省时区，否则会出现时间相差 8 小时的错误。

```
TimeZone.setDefault(TimeZone.getTimeZone("GMT+8"));
```

6.2.2 向索引库中添加索引文档

一般的方法是：

```
Document doc = new Document();
Field f = new Field("url", news.URL ,
                    Field.Store.YES, Field.Index.UN_TOKENIZED,
                    Field.TermVector.NO);
doc.add(f);

f = new Field("title", news.title ,
              Field.Store.YES, Field.Index.TOKENIZED,
              Field.TermVector.WITH_POSITIONS_OFFSETS);
doc.add(f);

f = new Field("body", news.body.toString() ,
              Field.Store.YES, Field.Index.TOKENIZED,
              Field.TermVector.WITH_POSITIONS_OFFSETS);
doc.add(f);

try{
```

```

        index.addDocument(doc);
    }
    catch(Exception e)
    {
        e.printStackTrace();
        System.exit(-1);
    }
}

```

为了提高索引速度，可以重用 `Field`，而不是每次都创建新的。从 `Lucene 2.3` 开始，有新的 `setValue` 方法，可以改变一个 `Field` 的值。这样可以在增加许多 `Document` 的时候重用单个的 `Field` 实例，可以节省许多 GC 消耗的时间。

最好，创建一个独立的 `Document` 实例，然后增加许多 `Field` 实例，但是一直保留这些实例并且增加每个文档的时候都重用这些 `Field`。例如，有一个 `idField` 和 `bodyField`，`nameField` 等。当加入一个 `Document` 后，可以通过 `idField.setValue(...)` 等直接改变 `Field` 值，然后再增加文档实例。下面是一个重用 `Field` 的例子：

```

Field idField = new Field("cid", null ,
    Field.Store.YES, Field.Index.UN_TOKENIZED,
    Field.TermVector.NO);

Field nameField = new Field("cname", null ,
    Field.Store.YES, Field.Index.TOKENIZED,
    Field.TermVector.WITH_POSITIONS_OFFSETS);

while(rs.next())
{
    Document doc = new Document();

    idField.setValue(String.valueOf(rs.getInt("companyID")));
    doc.add(idField);

    nameField.setValue(rs.getString("companyCname"));
    doc.add(nameField);

    try{
        index.addDocument(doc);
    }
    catch(Exception e)
    {
        e.printStackTrace(System.out);
    }
}

```


注意，不能在一个文档中重用单个 Field 实例，不应该改变一个列的值，直到包含这个 Field 的 Document 已经加入到索引库。

当在 Windows 系统下使用的时候。记住最好关闭杀毒软件的自动删除已感染病毒文件的选项。否则当索引带病毒特征的文档时，杀毒软件可能破坏 Lucene 的索引文件。



6.2.3 删除索引库中的索引文档

在 Lucene 2.1.0 之前，删除文档是在 IndexReader 中实现的。从 Lucene2.1.0 开始，可以从 IndexWriter 中删除文档了。

```
indexWriter.delete(new Term("id", "1"));
```

6.2.4 更新索引库中的索引文档

Lucene 早期的版本，先通过 IndexReader 删除文档，然后再通过 IndexWriter 增加文档。在 Lucene2.1.0 以后 IndexWriter 直接提供了更新文档的接口。

```
indexWriter.updateDocument(new Term("id", "1"), document);
```

6.2.5 索引的合并

索引很多文档的过程通常比较慢。为了加快索引速度，可以多台机器同时索引不同内容，然后合并。要合并的几个不同的索引结构要一致。下面的程序可以把多个目录下的索引合并到一个目录下：

```
IndexWriter writer = new IndexWriter(args[0], null, true);
writer.setMergeFactor(50); // 参数越大，用到的内存越多。影响内存的使用。
```

```
//影响索引文件的数量
writer.setUseCompoundFile(false);

Directory[] dirs = new Directory[args.length - 1];
System.out.println("begin :"+args[0]);
for (int i=1 ;i<args.length;i++)
{
    dirs[i-1]= FSDirectory.getDirectory( args[i], false);
    if (dirs[i-1]==null)
        System.out.println("Directory is null:"+i);
    //System.out.println("complete:"+i);
}

writer.addIndexes(dirs);

writer.close();
```

6.2.6 索引的定时更新

为了实现索引库和数据库中的内容同步，需要定时更新索引。可以增量更新或者全量更新。在 windows 平台可以利用计划任务来执行，在 Linux 平台可以有 Contab 或者利用 Java 中的定时程序。这里看一个全量更新的实现。

下面是批处理文件 spider.bat 的内容。

#删除新建索引文件夹

```
call del /Q /S D:\Tomcat5.5\webapps\search\lietu\index_new
```

```
call rmdir /S /Q D:\Tomcat5.5\webapps\search\lietu\index_new
```

#如果该文件夹不存在的话，就重建该新建索引文件夹

```
call mkdir D:\Tomcat5.5\webapps\search\lietu\index_new
```

#实际创建索引文件

```
call java "-Ddic.dir=D:\Tomcat5.5\webapps\search\WEB-INF\classes\dic" -Xmx300m
-classpath ./seg.jar;./search.jar;./lucene-core-2.1.0.jar;./sqljdbc.jar;.
com.bitmechanic.spindle.DbSpider c
```

#如果新索引未成功生成，则结束

```
IF NOT EXIST D:\Tomcat5.5\webapps\search\lietu\index_new\segments.gen GOTO INDEXFAIL
```

#停止 Tomcat 服务，停止前台对当前索引目录的访问

```
call net stop Tomcat5
```

#延迟时间

```
ping -n 20 127.0.0.1
```

#删除索引备份

```
call del /Q /S D:\Tomcat5.5\webapps\search\lietu\index_old
```

```
call rmdir /S /Q D:\Tomcat5.5\webapps\search\lietu\index_old
```

#备份当前索引

```
call rename D:\Tomcat5.5\webapps\search\lietu\index index_old
```

#用新索引替换当前索引

```
call rename D:\Tomcat5.5\webapps\search\lietu\index_new index
```

#索引更新完毕，启动 Tomcat 服务

```
net start Tomcat5
```

#结束

:INDEXFAIL

6.2.7 索引的备份和恢复

当增加、修改索引的时候，Lucene 索引文件一般不会发生太大的变化。优化索引的时候，索引文件才会有大变化。实际中可以通过同步索引文件的方式实现分布式备份。

在 Linux 中，同步文件可以采用 rsync 命令。

在索引主服务器上，为索引做阶段性的检查点。每分钟的时候，关闭 IndexWriter，并且从 Java 中执行 'cp -lr index index.DATE'命令，这里 DATE 是指当前时间。这样通过构建硬连接树，而不是制作完全的备份有效的制作了一个索引的拷贝。如果 Lucene 重写了任何文件(例如 segments 文件)，将会创建新的 inode，而原来的拷贝不变。

在每个搜索从服务器上，定期检查新的检查点。当发现一个新的 index.DATE，使用'cp -lr

index index.DATE'准备一份拷贝，然后使用'`rsync -W --delete master:index.DATE index.DATE`'得到增量的索引改变。然后使用原子性的符号连接操作安装更新的索引 (`ln -fsn index.DATE index`)。

在从服务器上，当索引版本改变的时候重新打开'`index`'。最好是在一个独立的线程中定时检查索引版本。当索引改变时，打开索引新的版本，执行一些热门查询操作，预加载 Lucene 缓存。然后在一个同步块中，替换在线服务的 `Searcher` 变量。

在主索引服务器的一个 `crontab` 中，定时移走最旧的检查点索引。

这样可以实现每分钟的同步，在主索引服务器上的 `mergeFactor` 设为 2 以最小化在产品中的 `segments` 数量。主服务器有一个热备份。

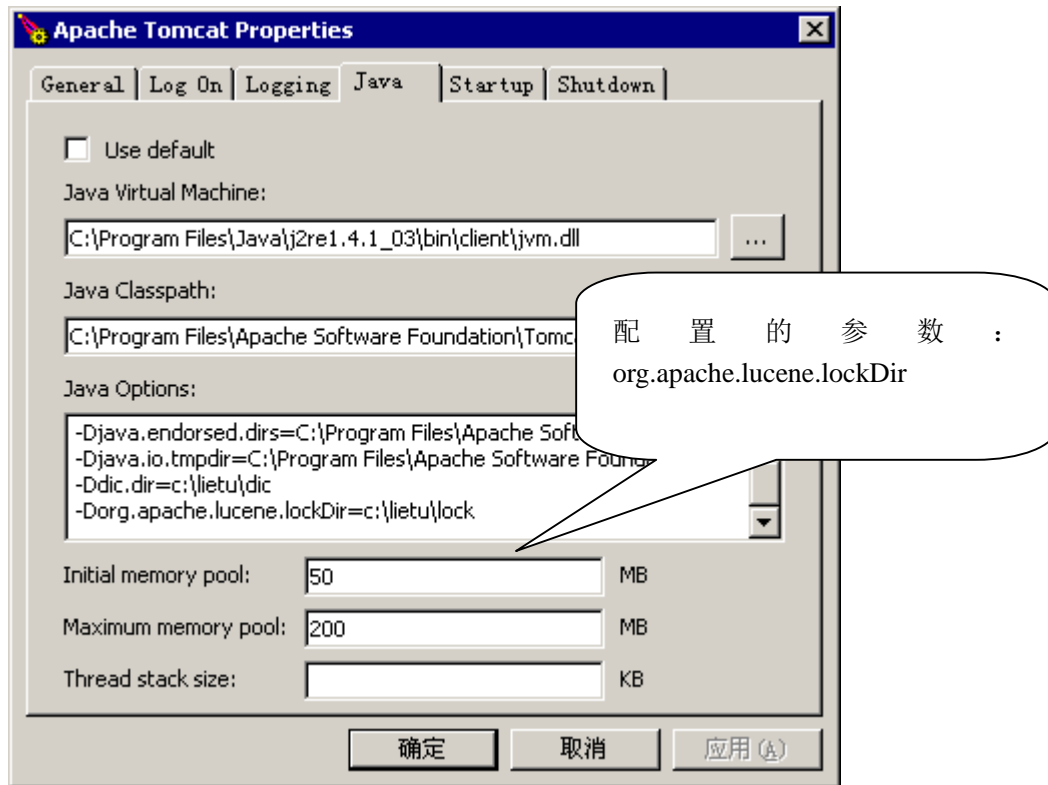
6.2.8 修复索引

使用 `CheckIndex` 来检查索引的完整性。例如：

```
CheckIndex D:\index_test\2
```

6.3 读写并发控制

在 Lucene 2.1 版本以前，可以通过锁文件控制并发访问。锁文件路径可以通过 `org.apache.lucene.lockDir` 修改。



自从 2.1, LOCK_DIR 参数已经没用了, 因为 write.lock 现在默认存储在 index 路径。

当有一个索引库, 需要只设置一个索引只读。这个索引的初始化的设置如下:

Directory indexDir =

```
FSDirectory.getDirectory(indexPath, NoLockFactory.getNoLockFactory());
```

6.4 优化使用 Lucene

6.4.1 索引优化

Lucene 打开文件数量很多的时候, 可能会达到操作系统允许打开文件数量的上限。这时候会出现“Too Many Open Files”的错误。在 Linux 下缺省是 1024。可以通过 `ulimit -a` 命令来查看当前参数。

```
# ulimit -a
```

```
core file size          (blocks, -c) 0
```

```
data seg size          (kbytes, -d) unlimited
```

max nice	(-e) 0
file size	(blocks, -f) unlimited
pending signals	(-i) 81920
max locked memory	(kbytes, -l) 32
max memory size	(kbytes, -m) unlimited
open files	(-n) 1024
pipe size	(512 bytes, -p) 8
POSIX message queues	(bytes, -q) 819200
max rt priority	(-r) 0
stack size	(kbytes, -s) 10240
cpu time	(seconds, -t) unlimited
max user processes	(-u) 81920
virtual memory	(kbytes, -v) unlimited
file locks	(-x) unlimited

可以使用命令 `ulimit -n 65535` 增大最大打开文件数量。

修改配置文件 `/etc/security/limits.conf` 增加如下行：

```
*          -      nofile      65535
```

在 Lcene2.3 版本之前，存入索引的每个 Token 都是新创建的。重复利用 Token 可以加快索引速度。新的 Tokenizer 类可以回收利用 Token。

```
next();
```

```
next(Token result);
```

```
public Token next(Token result) throws IOException
```

6.4.2 查询优化

在英文中，用户的输入中包含空格，中文的用户输入一般不包含空格。英文空格本身既作为单词的边界，在 `QueryParser` 解释查询语法的时候，又当做了 `OR` 来处理。例如，在 Lucene 搜索中输入 “Olympic Torch” 和 “Olympic OR Torch” 返回的结果一样。

因为中文用户的输入往往不包含空格，所以对中文来说，Lucene 的 `QueryParser` 对查询匹配显得太严格了。比如搜索 “日本小仓离合器”，只有连续出现 “日本小仓离合器” 的记录才能匹配上，将无法匹配到下面这段文字：

日本：

MUTOH 编码器、MUTOH 控制器

OGURA（小仓）离合器

这时候，需要把查询约束条件改成 “日本” “小仓” “离合器” 三个词都出现在同一条记录即可匹配。下面的 `BlankAndQueryParser` 重写了 `QueryParser` 类的 `getFieldQuery` 方法。

```
public class BlankAndQueryParser extends QueryParser{
```

```
    /**
```

```
     * Creates a Blank And QueryParser.
```

```
    */
```

```
    public BlankAndQueryParser(String field, Analyzer analyzer) {
```

```
        super(field, analyzer);
```

```
    }
```

```
        protected Query getFieldQuery(String field, String queryText, int slop) throws  
        ParseException {
```

```
TokenStream source = analyzer.tokenStream(field, new StringReader(queryText));
```

```
ArrayList<Token> v = new ArrayList<Token>(10);
```

```
Token t;
```

```
while (true)
```

```
{
```

```
    try
```

```
    {
```

```
        t = source.next();
```

```
    }
```

```
    catch (IOException e)
```

```
    {
```

```
        t = null;
```

```
    }
```

```
    if (t == null)
```

```
        break;
```

```
    v.add(t);
```

```
}
```

```
try
```

```
{
```



```
        source.close();

    }

    catch (IOException e)

    {

        // ignore

    }

    if (v.size() == 0)

        return null;

    else if (v.size() == 1)

        return new TermQuery(new Term(field, ((Token)v.get(0)).termText()));

    else

    {

        PhraseQuery q = new PhraseQuery();

        BooleanQuery b = new BooleanQuery();

        q.setBoost(2048.0f);

        b.setBoost(0.001f);

        for (int i = 0; i < v.size(); i++)

        {

            Token token = v.get(i);

            q.add(new Term(field, token.termText()));

        }

    }

}
```

```
TermQuery tmp = new TermQuery(new Term(field, token.termText()));

//if(! token.Type().Equals("n"))

//{

tmp.setBoost(0.01f);

//}

b.add(tmp, BooleanClause.Occur.SHOULD);

}

BooleanQuery bQuery = new BooleanQuery();

// combine the queries, neither

//requiring or prohibiting matches

bQuery.add(q,BooleanClause.Occur.SHOULD);

bQuery.add(b,BooleanClause.Occur.SHOULD);

//System.out.println("query:"+bQuery);

return bQuery;

}

}

protected Query getFieldQuery(String field, String queryText) throws ParseException {

return getFieldQuery(field, queryText, 0);
```

```

    }

}

```

当用户在搜索框中输入整句话“机床设备的最新退税率是多少？”，直接用上面的 `BlankAndQueryParser` 搜索可能不会返回任何结果。相对来讲，把“机床”和“退税率”当作搜索关键词是更好的选择。可以把“设备”，“的”，“最新”，“是”，“多少”这些意义不大的词从必选词中去掉。这些意义不大的词可以放在一个大的停用词表 `stopSet` 中。这样 `getFieldQuery` 方法中的循环条件改为：

```

        for (int i = 0; i < v.size(); i++)
        {
            Token token = v.get(i);
            q.add(new Term(field, token.termText()));

            if( stopSet.contains(token.termText()) )
            {
                continue;
            }
            TermQuery tmp = new TermQuery(new Term(field,
token.termText()));

            tmp.setBoost(0.01f);
            b.add(tmp, BooleanClause.Occur.MUST);
        }

```

如果用户输入较短的查询串，上面的 `BooleanQuery b` 容易匹配到一些看起来不太相关的长的文本，因为长文本中可能出现各种各样的 `Term` 组合。这时候，可能希望匹配的 `Term` 是分布在文本中较集中的区域，为了计算匹配的位置，可以利用 `Lucene` 中的 `Spans` 对象，它提供本次匹配的文档和位置信息。它的接口定义如下：

```

boolean next() //Move to the next match

int doc()//the doc id of the match

int start()//The match start position

int end() //The match end position

boolean skipTo(int target)//skip to a doc

```

`BlankAndQueryParser` 可以进一步修改为：

```

PhraseQuery q = new PhraseQuery();
q.setBoost(2048.0f);

```

```

ArrayList<SpanQuery> s = new ArrayList<SpanQuery>(v.size());

for (int i = 0; i < v.size(); i++)
{
    Token token = v.get(i);
    q.add(new Term(field, token.termText()));
    if( stopSet.contains(token.termText()) )
    {
        continue;
    }
    SpanTermQuery tmp =
        new SpanTermQuery (new Term(field, token.termText()));
    s.add(tmp);
}

BooleanQuery bQuery = new BooleanQuery();
// combine the queries, neither
//requiring or prohibiting matches
bQuery.add(q, BooleanClause.Occur.SHOULD);
SpanNearQuery nearQuery = new SpanNearQuery(s.toArray(new
SpanQuery[s.size()]), s.size(), false);
nearQuery.setBoost(0.001f);

bQuery.add(nearQuery, BooleanClause.Occur.SHOULD);

```

这样搜索“美丽人生”时，不会匹配“父母对子女的态度会影响他们日后的性格、感情，乃至整个人生。...个性好的人更美丽”。

6.4.3 实现时间加权排序

有时候希望当两个搜索结果的相关度分值 `score` 差不多的时候新的搜索结果显示在前面。这个特性无法用查询语言来实现。也许可以考虑用日期范围来实现类似功能，但是那并不完全是我所需要的。

因此，必须更加深入的研究 `lucene` 结构，并且写出了以下代码的实现：

```
String query="foo"
```

```
QueryParser parser =new QueryParser("name", new StandardAnalyzer());
```

```
Query q = parser.parse(query);
```

```
Sort updatedSort = new Sort();
```

```
FieldScoreQuery dateBooster = new FieldScoreQuery("timestampscore",
FieldScoreQuery.Type.FLOAT);
```

```
CustomScoreQuery customQuery = new CustomScoreQuery(q, dateBooster);
```

```
Hits results = getSearcher().search(customQuery, updatedSort);
```

FieldScoreQuery 是对 lucene 的一个最新的增加类。lucene 2.3 以后才有这个版本，基本上，FieldScoreQuery 的功能是把索引中的一列解释成浮点数并且获得一个分值(score)。然后，CustomScoreQuery 把这个分值与最初查询的分值合并起来。

到目前为止它一直在完美的运行。增加了一个浮点数的列“timestampscore”到索引，这个列的格式是“0.”+时间戳，其中时间戳格式化为 yyyyMMddhhmm 字符串（lucene 索引中只能存储字符串）。结果，最新时间戳有一个稍微高的分数。或者还可以通过权重来进一步调整查询。

Solr 中的实现：

```
queryWord += " AND _val_:\\"linear( recip(rord(timestamp),1,10000,10000),10000000,0)\\"";
```

6.4.4 实现字词混合索引

由于未登录词的识别准确率相对较低，当使用分词索引时，直接搜索商标名称，企业字号等词的时候，可能会漏掉一些错误切分的文档。例如输入：“润泓”，它可能代表一个企业名称，但概率分词的结果难以预料。字词混合索引对于小规模文档的网站搜索往往是必要的。为了在分词的基础上实现字词混合索引，首先增加一个分词后处理的 Filter。

```
public final class SingleFilter extends TokenFilter {
    private Token buff=null;
    private int offset=0;
    private static String tokenType = "1";

    public SingleFilter(TokenStream in)
    {
        super(in);
    }

    public final Token next() throws IOException
    {
        if (buff!=null)
        {
            if(offset == buff.termText().length())

```

```

        {
            Token buff2 = buff;
            buff = null;
            return buff2;
        }
        Token buff2 = new
Token(buff.termText().substring(offset,1+offset),
        buff.startOffset()+offset,
        buff.startOffset()+offset+1,tokenType);
        buff2.setPositionIncrement(0);

        ++offset;
        return buff2;
    }
    Token t = input.next();

    if (t == null)
        return null;
    if (t.termText().length()>1)
    {
        buff = t;
        offset = 0;

        Token buff2 = new
Token(buff.termText().substring(offset,1+offset),
        buff.startOffset()+offset,
        buff.startOffset()+offset+1,tokenType);
        buff2.setPositionIncrement(0);

        ++offset;
        return buff2;
    }

    return t;
}
}

```

对 HighLighter 修改， org.apache.lucene.search.highlight.TokenSources 类 增加 getSingleTokenStream 方法正确处理单字混合索引：

```

public static TokenStream getSingleTokenStream(TermPositionVector
tpv, boolean tokenPositionsGuaranteedContiguous)
{
    //an object used to iterate across an array of tokens

```

```

class StoredTokenStream extends TokenStream
{
    Token tokens[];
    int currentToken=0;
    StoredTokenStream(Token tokens[])
    {
        this.tokens=tokens;
    }
    public Token next()
    {
        if(currentToken>=tokens.length)
        {
            return null;
        }
        return tokens[currentToken++];
    }
}

//code to reconstruct the original sequence of Tokens
String[] terms=tpv.getTerms();
int[] freq=tpv.getTermFrequencies();
int totalTokens=0;
for (int t = 0; t < freq.length; t++)
{
    totalTokens+=freq[t];
}
Token tokensInOriginalOrder[]=new Token[totalTokens];
ArrayList<Token> unsortedTokens = null;
for (int t = 0; t < freq.length; t++)
{
    TermVectorOffsetInfo[] offsets=tpv.getOffsets(t);
    if(offsets==null)
    {
        return null;
    }

    int[] pos=null;
    if(tokenPositionsGuaranteedContiguous)
    {
        //try get the token position info to speed up assembly of
tokens into sorted sequence
        pos=tpv.getTermPositions(t);
    }
    if(pos==null)
    {

```

```

        //tokens NOT stored with positions or not guaranteed
        contiguous - must add to list and sort later
        if(unsortedTokens==null)
        {
            unsortedTokens=new ArrayList<Token>();
        }
        for (int tp = 0; tp < offsets.length; tp++)
        {
            unsortedTokens.add(new Token(terms[t],
                offsets[tp].getStartOffset(),
                offsets[tp].getEndOffset()));
        }
    }
    else
    {
        //We have positions stored and a guarantee that the token
        position information is contiguous

        // This may be fast BUT wont work if Tokenizers used which
        create >1 token in same position or
        // creates jumps in position numbers - this code would fail
        under those circumstances

        //tokens stored with positions - can use this to index straight
        into sorted array
        for (int tp = 0; tp < pos.length; tp++)
        {
            tokensInOriginalOrder[pos[tp]]=new Token(terms[t],
                offsets[tp].getStartOffset(),
                offsets[tp].getEndOffset());
        }
    }
}
//If the field has been stored without position data we must perform
a sort
if(unsortedTokens!=null)
{
    Collections.sort(unsortedTokens,new Comparator<Token>(){

        public int compare(Token t1, Token t2) {
            if(t1.startOffset()>t2.startOffset())
                return 1;
            if(t1.startOffset()<t2.startOffset())
                return -1;
        }
    });
}

```



```

        if(t1.endOffset()>t2.endOffset())
            return 1;
        return 0;
    }));
    for(int i=0;i<(unsortedTokens.size() - 1);)
    {
        Token t1 = unsortedTokens.get(i) ;
        //System.out.println("get token:"+t1.termText());
        Token t2 = unsortedTokens.get(i+1) ;
        if(t1.startOffset() == t2.startOffset())
        {
            if(t1.endOffset()< t2.endOffset())
            {
                unsortedTokens.remove(i);
            }
            else
            {
                ++i;
            }
        }
        else if(t1.startOffset() < t2.startOffset())
        {
            if(t1.endOffset()>=t2.endOffset())
            {
                unsortedTokens.remove(i+1);
            }
            else
            {
                ++i;
            }
        }
        else
        {
            ++i;
        }
    }
    tokensInOriginalOrder=(Token[]) unsortedTokens.toArray(new
Token[unsortedTokens.size()]);
}
    return new StoredTokenStream(tokensInOriginalOrder);
}

```

最后在 Lucene 中增加 SingleQueryParser,形成字词混合的查询对象:

```
protected Query getFieldQuery(String field, String queryText) throws
```

```

ParseException {
    TokenStream source = analyzer.tokenStream(field, new
StringReader(queryText));

    ArrayList<Token> v = new ArrayList<Token>(10);
    Token t;

    while (true)
    {
        try
        {
            t = source.next();
        }
        catch (IOException e)
        {
            t = null;
        }
        if (t == null)
            break;
        v.add(t);
    }
    try
    {
        source.close();
    }
    catch (IOException e)
    {
        // ignore
    }

    if (v.size() == 0)
        return null;
    else if (v.size() == 1)
        return new TermQuery(new Term(field,
((Token)v.get(0)).termText()));
    else
    {
        PhraseQuery q = new PhraseQuery();
        q.setBoost(2048.0f);

        ArrayList<SpanQuery> s = new ArrayList<SpanQuery>(v.size());

        for (int i = 0; i < v.size(); i++)
        {

```

```

        Token token = v.get(i);
        if(token.getPositionIncrement()>0)
        {
            q.add(new Term(field, token.termText()));
        }
        if(token.termText().length()==1)
        {
            SpanTermQuery tmp = new SpanTermQuery (new Term(field,
token.termText()));
            s.add(tmp);
        }
    }

    BooleanQuery bQuery = new BooleanQuery();
    // combine the queries, neither
    //requiring or prohibiting matches
    bQuery.add(q, BooleanClause.Occur.SHOULD);
    SpanNearQuery nearQuery = new SpanNearQuery(s.toArray(new
SpanQuery[s.size()]),s.size(),true);
    nearQuery.setBoost(0.001f);
    bQuery.add(nearQuery, BooleanClause.Occur.SHOULD);

    return bQuery;
}
}

```

这里返回的是有序的 `SpanNearQuery`。为了更加宽松的匹配环境，可以修改 `NearSpansOrdered` 判断两个 `Span` 是否有序的方法 `docSpansOrdered`:

```
return (start1 == start2)?(spans1.end() <= spans2.end()): (start1 < start2);
```

在 `spans1.end() <= spans2.end()` 之间多加了一个等于。这样当两个 `span` 都在同一位置时也算这两个 `Span` 是有序的。

6.4.5 定制 Similarity

Similarity 的计算公式:

score(q,d) =

$$\sum \text{tf}(t \text{ in } d) * \text{idf}(t) * \text{getBoost}(t.\text{field in } d) * \text{lengthNorm}(t.\text{field in } d) * \text{coord}(q,d) * \text{queryNorm}(q)$$

有个问题是, Lucene 的缺省的长度归一化公式把长的文档惩罚太多。SweetSpotSimilarity 可以改进这个问题。SweetSpotSimilarity 实现了一个 lengthNorm 把一段区间内的文档长度看成同样好。可以定义一个全局的 min/max, 在这段区间内的 lengthNorm 都是 1.0。低于最小值或高于最大值的 lengthNorm 以一个平方的函数下降。这样的结果是, 在此区间内稍微长点的文档就不会有罚分了。

也可以对每一个 field 设置不同的 min/max, 这样它们有不同的 sweet spots。

选择 min/max 的依据, 可以根据文档的平均的长度。你必须提前知道和推测出这个平均长度, 并且通过方法 SweetSpotSimilarity.setLengthNormFactors(yourAvg,yourAvg,steepness) 把它明确的设定为优选区。

做好这个 SweetSpotSimilarity 之后, 用它来替换缺省的 Similarity。在 org.apache.lucene.search.Searcher 中有个方法:

```
public void setSimilarity(Similarity similarity) {
    this.similarity = similarity;
}
```

6.4.6 定制 Tokenizer

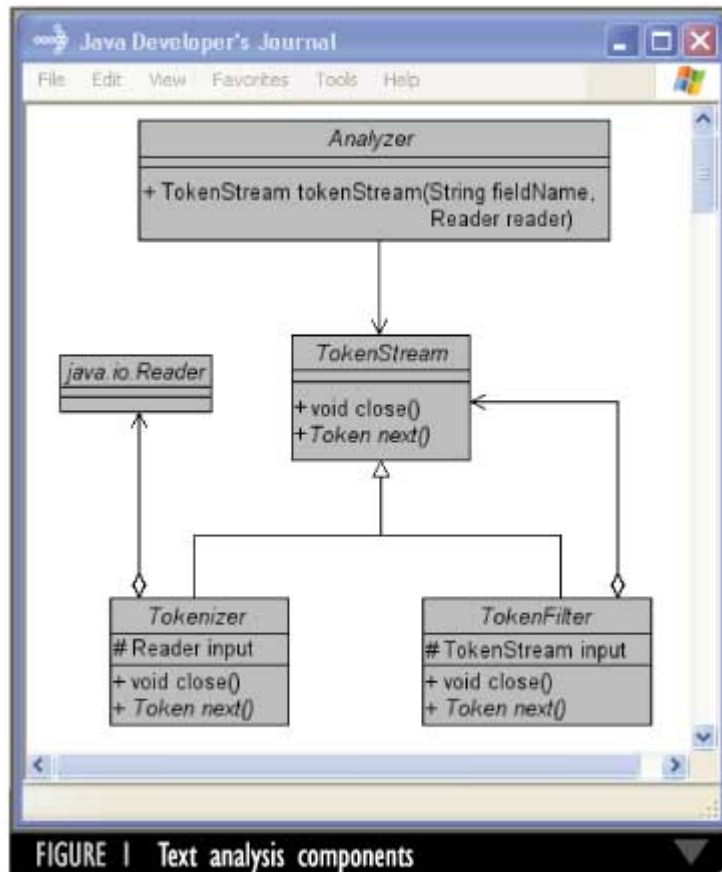


FIGURE 1 Text analysis components

Lucene 通过 `Tokenizer` 类切分文本。最通常使用的是 `StandardTokenizer`，`StandardTokenizer` 虽然处理中文仍然是单字的方式，但是在 `Lucene2.3` 版本中已经可以区分出像 `lsdfs@sina.com` 这样的邮件地址，或者是 `AT&T` 这样的公司名字以及 `U.S.A.` 这样的缩写。下面用个简单的测试类验证：

```
public static void testStandardAnalyzer() throws Exception {
    StandardAnalyzer cna = new StandardAnalyzer();

    String input = "公司:P&G 信箱:webmaster@ertt.com 网址:
http://www.xssdfff.com ";

    TokenStream ts = cna.tokenStream("dummy", new
StringReader(input));

    for (Token t = ts.next(); t != null; t = ts.next())
    {
        System.out.println(String.valueOf(t.termBuffer(),0,t.termLength()) + "
" + t.startOffset() + " "
                                + t.endOffset() + " "+t.type());
    }
}
```

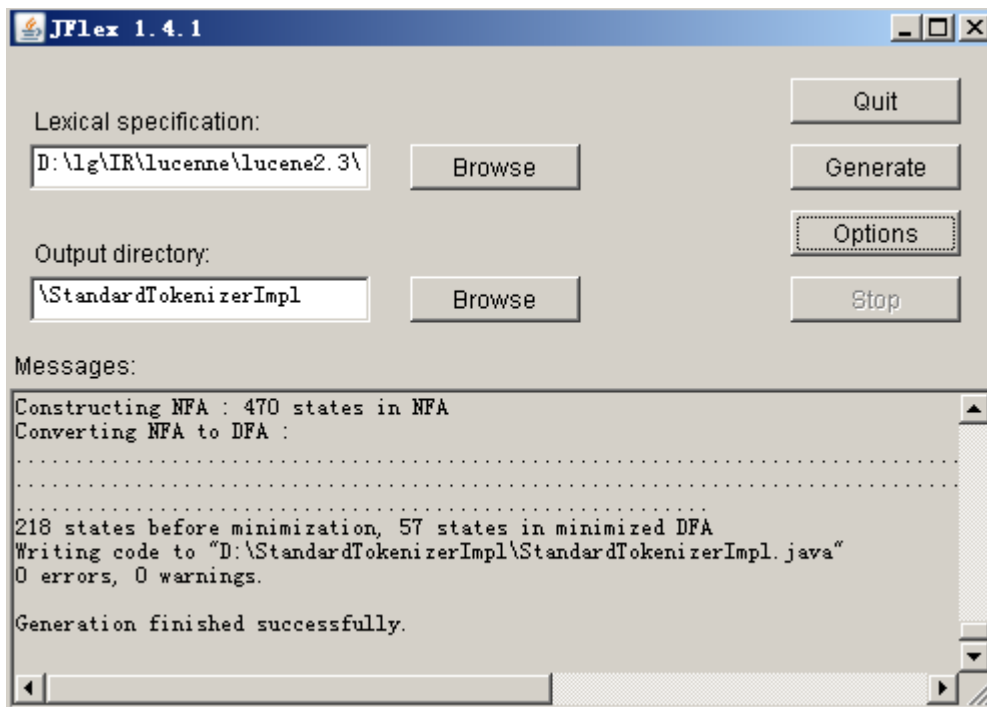
输出结果是：

```
公 0 1 <CJ>
司 1 2 <CJ>
p&g 3 6 <COMPANY>
信 7 8 <CJ>
箱 8 9 <CJ>
webmaster@ertt.com 10 28 <EMAIL>
网 29 30 <CJ>
址 30 31 <CJ>
http 33 37 <ALPHANUM>

www.xssdfff.com 40 54 <HOST>
```

我们可以根据需要修改 `StandardTokenizer`，它的实现是根据编译原理中的状态机理论使用工具自动生成的。早期版本采用 `JavaCC` 生成，`Lucene2.3` 的版本采用了 `JFlex` 生成，`JFlex` 生成的代码性能更好。

`JFlex` 把词法声明规范转换成实际的 Java 源代码。可以从<http://www.jflex.de/>下载 `JFlex` 的最新版本。解压后，执行 `C:\JFLEX\bin\jflex.bat` 批处理文件。输入声明规范 `StandardTokenizerImpl.jflex`，输出 `StandardTokenizerImpl.java`。如下是图示：



比如我们希望增加对电话号码的识别，就需要修改 `StandardTokenizerImpl.jflex`。`StandardTokenizerImpl.jflex` 由%%分开的三部分组成，分别是用户代码段，选项声明段和语法规则段。比如我们定义电话号码是由三位区号和八位号码以及中间的“-”组成的。

```
// telephone number

//Telephone number
TEL      = ( {DIGIT} ){3} "-" ( {DIGIT} ){8}
          | ( {DIGIT} ){4} "-" ( {DIGIT} ){7}
```

这样当碰到 010-51667560 这样的电话号码时，就可以正确识别成<TEL> 类型的Token。

6.5 查询大容量索引

一般情况下，索引达到 10 多个 G 以上，搜索的响应时间可能会增加到秒级。可以使用并发多索引查询来改进性能。

```
//并发多索引查询 String[] index: 索引目录
```

```
public static Hits Multisearch(String[] index,String q) throws Exception {
```

```
    int length = index.length;
```

```
    IndexSearcher[] is = new IndexSearcher[length];
```

```
for ( int i = 0 ; i < length ; i ++){  
  
    is[i] = new IndexSearcher(index[i]);  
  
}  
  
Searcher searcher = new ParallelMultiSearcher(is);  
  
Query query = QueryParser.parse(q, "temp",  
  
    new StandardAnalyzer());  
  
Hits hits = searcher.search(query);  
  
return hits;  
  
}
```

6.6 本章小结

第7章 用户界面设计与实现

7.1 Lucene 搜索接口(search 代码)

通过一个控制台程序测试一下搜索：

```

Searcher searcher = new IndexSearcher(indexPath);
IndexReader reader = IndexReader.open(indexPath);
System.out.println("doc number in the index: " + reader.numDocs());

Query bodyQuery = null, titleQuery = null, query = null;

QueryParser parser = new QueryParser("intro", analyzer);
bodyQuery = parser.parse(queryString);
parser = new QueryParser("cname", analyzer);
titleQuery = parser.parse(queryString);

System.out.println("Searching for: " +
bodyQuery.toString("body"));

BooleanQuery bodyOrTitle = new BooleanQuery();
bodyOrTitle.add(bodyQuery, BooleanClause.Occur.SHOULD);
bodyOrTitle.add(titleQuery, BooleanClause.Occur.SHOULD);

query = bodyOrTitle.rewrite(reader); //需要这行语句来扩展搜索词

Hits hits = null;

//设置排序方式，比如说高优先级的先显示
SortField classSortField = new
SortField("class", SortField.INT, true);
Sort classSort = new Sort(new SortField[] {classSortField});
hits = searcher.search(query, classSort);

System.out.println("find place " + hits.length());

Highlighter highlighter = new Highlighter(new
SimpleFormatter(), new QueryScorer(query));
String text;
TokenStream tokenStream;

for (int i = 0; i < hits.length(); i++)

```



```

    {
        text = hits.doc(i).get("intro");

        //内容的高量显示
        TermPositionVector tpv =
(TermPositionVector)reader.getTermFreqVector(hits.id(i),"intro");
        tokenStream=TokenSources.getTokenStream(tpv,false);
        String result =
highlighter.getBestFragment(tokenStream,text);
        System.out.println("intro:"+result);

        //标题的高量显示
        text = TextHtml.text2html(hits.doc(i).get("cname"));
        tokenStream=analyzer.tokenStream("cname",new
StringReader(text));

        result = highlighter.getBestFragment(tokenStream,text);
        if (result == null)
            System.out.println(hits.doc(i).get("cname"));
        else
            System.out.println("cname:"+result);
    }

    searcher.close();

```

Query 的 rewrite 方法把复杂的查询条件重写成简单的查询条件。例如, RangeQuery 会扩展成很多的 TermQuery。

7.2 搜索页面设计

7.2.1 用于显示搜索结果的 taglib

搜索结果页是一个表格型的数据。Listlib 实现了对数据的封装和抽象, 可以通过它来控制显示的结果数量, 比如可以指定每页显示 20 条记录或 10 条记录路。实际执行 Lucene 搜索的类继承 ListCreator 接口, 并把搜索结果通过 ListContainer 类的实例返回即可。

init

listlib 的起始 tag。创建一个 ListCreator 对象，并且运行该对象的 execute() 方法，并且把它存储在 HttpServletRequest 属性中。这是一个容器 tag，所以在 jsp 页面使用时，其他的 tag 都必须嵌套在这个 tag 中间。

它的主要属性有：通过 name 指定一个名字，因为需要通过这个名字来把 ListCreator 对象存储在 HttpServletRequest 属性。通过 listCreator 来指定创建 ListCreator 的对象。通过 max 来声明每页必须显示的记录条数。

hasResults

如果 list 有结果，就会执行这个 tag，否则会跳过。

hasNoResults

如果 list 没有结果就会执行这个 tag，否则会跳过。

prop

返回 list 中的属性值。和搜索结果总体相关的信息可以通过它来显示，例如搜索提示词，搜索结果分类统计等等。

hasPrev

如果还可以继续往回遍历，就会显示这个 tag 中的内容，否则就跳过。

hasNext

如果还可以继续向下遍历，就会显示这个 tag 中的内容，否则就跳过。

iterate

遍历 ListContainer 中的元素。

iterateProp

从 Iterator 的当前对象返回属性，比如返回标题通过<list:iterateProp property="title"/>。

实现搜索的主要代码如下：

```
<!--创建一个 Lucene 搜索对象 - 它实现了 ListCreator 的 execute 方法 -->
```

```

<jsp:useBean          id="searchInf"          class="com.bitmechanic.spindle.SearchInfo"
scope="application">

    <!--指定存储 Lucene 索引的路径-->

    <% searchInf.init("d:/tomcat5.5/webapps/search/lietu/index/info/"); %>

</jsp:useBean>

<!--把查询从 http 的 get 参数设置到搜索对象中去-->

<jsp:setProperty name="searchInf" property="query" value="<%=query%>"/>

<!--执行搜索并把返回结果封装到 ListContainer -->

<%long start = System.currentTimeMillis();%>

<list:init name="information" listCreator="searchInf" max="20">

<% long end = System.currentTimeMillis();%>

<!--记录搜索执行的时间 -->

```

7.2.2 用于搜索结果分页的 taglib

Pager Tag Library 是一个广泛使用的分页 taglib，其出处是 <http://jsptags.com/tags/navigation/pager/index.jsp>。上面的例子正好是模拟 Google 和 Yahoo! 等搜索引擎分页的效果。

```

<pg:pager            url="/Search.jsp"

    items="<%=Integer.parseInt(listSize)%>"

    maxPageItems="20"

    maxIndexPages="10"

    export="currentPageNumber=pageNumber"

    scope="request">

```

items 记录了当前页面实际有多少行，maxPageItems 记录一个页面显示的最大行数是 20 行，maxIndexPages 记录最多显示多少个页面的 link。

```
<pg:param name="query" value="<%=query%>"/>
```

```
<pg:param name="cat" value="<%=cat%>"/>
```

```
<pg:param name="s" value="<%=order%>"/>
```

记录分页的 link 要保存的参数值，分别是 query 查询词，cat 类别，和 s 排序方式。

和其他很多开源组件一样，这个组件也会碰到乱码问题。生成的页面 link 中的汉字没有正常编码时，需要调用 `java.net.URLEncoder.encode` 对 query 等参数编码。可以修改 `com.jsptags.navigation.pager` 的 `PagerTag` 类的 `addParam` 函数。

```
final void addParam(String name, String value) {
    try{
        if (value != null) {
            name = java.net.URLEncoder.encode(name, charset);
            value = java.net.URLEncoder.encode(value, charset);

            uri.append(params == 0 ? "?" : "&")
                .append(name).append('=').append(value);

            params++;
        } else {
            String[] values =
pageContext.getRequest().getParameterValues(name);

            if (values != null) {
                name = java.net.URLEncoder.encode(name, charset);
                for (int i = 0, l = values.length; i < l; i++) {
                    value =
java.net.URLEncoder.encode(values[i], charset);
                    uri.append(params == 0 ? "?" : "&")
                        .append(name).append('=').append(value);

                    params++;
                }
            }
        }
    } catch (Exception e)
    {
        e.printStackTrace(System.err);
    }
}
```



```

        <td width=66%><a href="<list:iterateProp property="url"/>"
target="_blank" ><B><FONT style="FONT-SIZE: 14px"><list:iterateProp
property="title"/></FONT></B>&nbsp;&nbsp;&nbsp;<FONT size=-1 color=#6f6f6f></FONT></td>

        <td width=18%><FONT size=-1 color=#6f6f6f><list:iterateProp
property="accessDate"/></FONT></td>

        <td width=16%></td>

</TR>

<TR vAlign=top <% if((count%2)==0){ %> bgcolor=#f6f6f6 <% } else { %>
<% } %>>

        <td colspan=3> <FONT size=-1><list:iterateProp property="desc"/></FONT>

        </td>

</TR>

<TR vAlign=top <% if((count%2)==0){ %> bgcolor=#f6f6f6 <% } else { %>
<% } %>>

        <td colspan=3> <span class=tailurl><a class=tail href="<list:iterateProp
property="url"/>" target="_blank" ><list:iterateProp property="url"/></a></span>

        </td>

</TR>

</list:iterate>

</TABLE>

<pg:index export="totalItems=itemCount">

<div class="rnav"> <span class="rnavLabel">&nbsp;&nbsp;&nbsp;结果 : </span>&nbsp;&nbsp;&
<pg:prev export="pageUrl">

        <a href="<%= pageUrl %>"class="rnavLink">&#171;&nbsp;&nbsp;&上一页</a>&nbsp;&nbsp;&

</pg:prev> <pg:pages>

<%

```


7.3 实现搜索接口

7.3.1 布尔搜索

用布尔查询来实现多个查询条件的合并，最常见的例子是搜索标题或正文。

```
BooleanQuery bodyOrTitle = new BooleanQuery();
bodyOrTitle.add(bodyQuery, BooleanClause.Occur.SHOULD);

bodyOrTitle.add(titleQuery, BooleanClause.Occur.SHOULD);
```

这里 `BooleanClause.Occur.SHOULD` 代表 “或者” 的关系，如果要 “并且” 就用 `BooleanClause.Occur.MUST`。

也可以使用 `MultiFieldQueryParser` 来合并对多个列的搜索，比如下面实现对 “body” 和 “title” 两列的查找。

```
Query query = MultiFieldQueryParser.Parse(queryWord, new
string[]{"body","title"}, analyzer);
```

7.3.2 指定范围搜索

在商品搜索中，经常需要指定按时间条件或价格等数值条件查找。如下图所示：

可以通过 `RangeQuery` 来实现这样的时间条件区间条件查找：

```
java.util.Calendar upper = GregorianCalendar.getInstance();
upper.add(java.util.Calendar.YEAR, +100);
String t2 = formatter.format(upper.getTime());
```



```

        if ("1".equals(dateRange))
        {
            //一周内
            now.add(java.util.Calendar.DATE, -7);
            String t1 = formatter.format(now.getTime());
            ConstantScoreRangeQuery dateQuery = new
ConstantScoreRangeQuery("time", t1, t2, true, true);
        }
        else if ("2".equals(dateRange))
        {
            //一月内
            now.add(java.util.Calendar.MONTH, -1);
            String t1 = formatter.format(now.getTime());
            ConstantScoreRangeQuery dateQuery = new
ConstantScoreRangeQuery("time", t1, t2, true, true);
        }
        else if ("3".equals(dateRange))
        {
            //三月内
            now.add(java.util.Calendar.MONTH, -3);
            String t1 = formatter.format(now.getTime());
            ConstantScoreRangeQuery dateQuery = new
ConstantScoreRangeQuery("time", t1, t2, true, true);
        }
        else if ("4".equals(dateRange))
        {
            //六月内
            now.add(java.util.Calendar.MONTH, -6);
            String t1 = formatter.format(now.getTime());
            ConstantScoreRangeQuery dateQuery = new
ConstantScoreRangeQuery("time", t1, t2, true, true);
        }
    }

```

如果是在界面中，区间条件的查询语法例子如下：

+汽车 +expiretime:[2007-08-13T00:00:00Z TO 2008-08-13T00:00:00Z]

Lucene 是通过 BooleanQuery 的扩展实现区间查找的。索引库在该区间有多少个不同的值就会扩展出多少个 TermQuery。而 BooleanQuery 有个 MaxClause 限制，缺省是 1024。因此，为了不超过上限，我们经常需要扩大这个值。

```
BooleanQuery.setMaxClauseCount(2000);
```

为了实现像价格这样数字的查找，因为 Lucene 索引库不直接支持像 Long 或者 Double

这样的数值类型的存储，最终这些值都要存储成字符串。所有的数值都是通过 `org.apache.lucene.document.NumberTools` 类实现的数值到字符串的转换。lucene 中的 `NumberTools` 不提供 `Double` 和 `String` 直接的转换。而 solr 提供了一个支持 `Double` 和 `float` 类型的 `NumberTools` 实现。一个 4 个字节的 `int` 和 `float` 用 3 个 java 字符就可以表示。一个 8 个字节的 `double` 用 5 个 java 字符就可以表示。和 `org.apache.lucene.document.NumberTools` 用了 14 个 java 字符相比，这种方法能够减小索引大小。

```
public class NumberUtils {
    public static String int2sortableStr(int val) {
        char[] arr = new char[3];
        int2sortableStr(val, arr, 0);
        return new String(arr, 0, 3);
    }

    public static String int2sortableStr(String val) {
        return int2sortableStr(Integer.parseInt(val));
    }

    public static String SortableStr2int(String val) {
        int ival = SortableStr2int(val, 0, 3);
        return Integer.toString(ival);
    }

    public static String long2sortableStr(long val) {
        char[] arr = new char[5];
        long2sortableStr(val, arr, 0);
        return new String(arr, 0, 5);
    }

    public static String long2sortableStr(String val) {
        return long2sortableStr(Long.parseLong(val));
    }

    public static String SortableStr2long(String val) {
        long ival = SortableStr2long(val, 0, 5);
        return Long.toString(ival);
    }

    //
    // IEEE floating point format is defined so that it sorts correctly
    // when interpreted as a signed integer (or signed long in the
case
    // of a double) for positive values. For negative values, all
```

```

the bits except
    // the sign bit must be inverted.
    // This correctly handles all possible float values including
    -Infinity and +Infinity.
    // Note that in float-space, NaN<x is false, NaN>x is false, NaN==x
    is false, NaN!=x is true
    // for all x (including NaN itself). Internal to Solr, NaN==NaN
    is true and NaN
    // sorts higher than Infinity, so a range query of [-Infinity
    TO +Infinity] will
    // exclude NaN values, but a query of "NaN" will find all NaN
    values.

    // Also, -0==0 in float-space but -0<0 after this transformation.
    //
    public static String float2sortableStr(float val) {
        int f = Float.floatToRawIntBits(val);
        if (f<0) f ^= 0x7fffffff;
        return int2sortableStr(f);
    }

    public static String float2sortableStr(String val) {
        return float2sortableStr(Float.parseFloat(val));
    }

    public static float SortableStr2float(String val) {
        int f = SortableStr2int(val,0,3);
        if (f<0) f ^= 0x7fffffff;
        return Float.intBitsToFloat(f);
    }

    public static String SortableStr2floatStr(String val) {
        return Float.toString(SortableStr2float(val));
    }

    public static String double2sortableStr(double val) {
        long f = Double.doubleToRawLongBits(val);
        if (f<0) f ^= 0xffffffffffffffffL;
        return long2sortableStr(f);
    }

    public static String double2sortableStr(String val) {
        return double2sortableStr(Double.parseDouble(val));
    }

```

```

public static double SortableStr2double(String val) {
    long f = SortableStr2long(val,0,6);
    if (f<0) f ^= 0x7fffffffffffffffL;
    return Double.longBitsToDouble(f);
}

public static String SortableStr2doubleStr(String val) {
    return Double.toString(SortableStr2double(val));
}

// uses binary representation of an int to build a string of
// chars that will sort correctly. Only char ranges
// less than 0xd800 will be used to avoid UCS-16 surrogates.
public static int int2sortableStr(int val, char[] out, int offset)
{
    val += Integer.MIN_VALUE;
    out[offset++] = (char)(val >>> 24);
    out[offset++] = (char)((val >>> 12) & 0x0fff);
    out[offset++] = (char)(val & 0x0fff);
    return 3;
}

public static int SortableStr2int(String sval, int offset, int
len) {
    int val = sval.charAt(offset++) << 24;
    val |= sval.charAt(offset++) << 12;
    val |= sval.charAt(offset++);
    val -= Integer.MIN_VALUE;
    return val;
}

// uses binary representation of an int to build a string of
// chars that will sort correctly. Only char ranges
// less than 0xd800 will be used to avoid UCS-16 surrogates.
// we can use the lowest 15 bits of a char, (or a mask of 0x7fff)
public static int long2sortableStr(long val, char[] out, int
offset) {
    val += Long.MIN_VALUE;
    out[offset++] = (char)(val >>>60);
    out[offset++] = (char)(val >>>45 & 0x7fff);
    out[offset++] = (char)(val >>>30 & 0x7fff);
}

```

```

        out[offset++] = (char)(val >>>15 & 0x7fff);
        out[offset] = (char)(val & 0x7fff);
        return 5;
    }

    public static long SortableStr2long(String sval, int offset, int
len) {
        long val = (long)(sval.charAt(offset++)) << 60;
        val |= ((long)sval.charAt(offset++)) << 45;
        val |= ((long)sval.charAt(offset++)) << 30;
        val |= sval.charAt(offset++) << 15;
        val |= sval.charAt(offset);
        val -= Long.MIN_VALUE;
        return val;
    }
}

```

7.3.3 搜索结果排序

可以对多个字段排序。比如先按地区，然后按类别排序：

```
Sort sort= new Sort(new SortField[]{new SortField("area"),new SortField("type")});
```

```
hits = search.search(query,sort);
```

也可以通过 `SortComparatorSource` 自定义排序方法。

下面我们先按匹配相似度，然后按时间排序：

```

Lucene.Net.Search.SortField[] sortF =
    new SortField[]{ Lucene.Net.Search.SortField.FIELD_SCORE,
        new Lucene.Net.Search.SortField("postdate",
Lucene.Net.Search.SortField.STRING, true) };
    Lucene.Net.Search.Sort scoreSort = new
Lucene.Net.Search.Sort(sortF);

    hits = searcher.Search(query, scoreSort);

```

可以扩展 `Similarity`，一个例子：

```
private class SimilarityOne extends DefaultSimilarity {
    public float lengthNorm(String fieldName, int numTerms) {
        return 1;
    }
}

...

```

在 IndexWriter 引用 SimilarityOne。

```
IndexWriter iw = new IndexWriter(dir, analyzer, true);
iw.setMaxBufferedDocs(5);
iw.setMergeFactor(3);
iw.setSimilarity(similarityOne);
iw.setUseCompoundFile(true);

iw.close();

...

```

也可以在 Search 里面指定自定义的 Similarity。

```
String indexDir = "...";
IndexReader reader = IndexReader.open(indexDir);

IndexSearcher searcher = new IndexSearcher(reader);

searcher.setSimilarity(similarityOne);

```

7.3.4 搜索页面的索引缓存与更新

索引一般是一个比较大的文件。一般从几百 M 到几个 G 不等。页面执行搜索的时候打开大的索引往往是一个非常耗时的过程。一般情况下需要缓存 IndexReader 和 Searcher。

```
private static IndexReader reader = null;

private static Searcher searcher = null;

public void init(String indexPath) throws Exception
{
    if(searcher != null)
        searcher.close();
}

```

```

if(reader != null)
    reader.close();

_dir = indexPath;

searcher = new IndexSearcher(indexPath);
reader = IndexReader.open(indexPath);
}

```

因为后台在更新，前台的缓存会导致索引更新不及时，不能搜到已经更新的内容，这时候就需要重新装载索引库。

```

private void refreshIndexReader() {
    try {
        //如果检查时间已经到了，就检查当前索引的版本号
        if ((LastCheckTime+interval) < System.currentTimeMillis())
        {
            long newIndexVersion;
            newIndexVersion = IndexReader.getCurrentVersion(_dir);

            //如果索引已经是最新的，就重新设置检查时间
            if (newIndexVersion == currentIndexVersion)
            {
                LastCheckTime = System.currentTimeMillis();
                return;
            }

            synchronized (this)
            {
                LastCheckTime = System.currentTimeMillis();
                searcher.close();
                reader.close();
                reader = IndexReader.open(_dir);
                searcher = new IndexSearcher(reader);
                currentIndexVersion = newIndexVersion;
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

7.4 实现关键词高亮显示

在搜索结果中一般都有和用户搜索关键词相关的摘要。关键词一般都会高亮显示出来。Lucene 的 highlighter 包可以做到这一点。

```
doSearching("汽车");
//使用一个查询初始化Highlighter对象
Highlighter highlighter = new Highlighter(new
QueryScorer(query));
//设置分段显示的文本长度
highlighter.setTextFragmenter(new SimpleFragmenter(40));
//设置最多显示的段落数量
int maxNumFragmentsRequired = 2;
for (int i = 0; i < hits.length(); i++)
{
    //取得索引库中存储的原始文本
    String text = hits.doc(i).get(FIELD_NAME);
    TokenStream tokenStream=analyzer.tokenStream(FIELD_NAME,new
StringReader(text));

    //取得关键词加亮后的结果
    String result =

    highlighter.getBestFragments(tokenStream,text,maxNumFragmentsRequired, "...");
    System.out.println("\t" + result);
}
```

QueryScorer 设置查询的 query，这里还可以加上对字段列的限制，比如只对 body 条件的 Term 加量，可以使用：new QueryScorer(query," body")。

为了实现关键词高亮，必须知道关键词在文本中的位置。对英文来说，可以在搜索的时候实时切分出位置。但是中文分词的速度一般相对来说慢很多。在 Lucene1.4.3 以后的版本中，Term Vector 支持保存 Token.getPositionIncrement() 和 Token.startOffset() 以及 Token.endOffset() 信息。利用 Lucene 中新增加的 Token 信息的保存结果以后，就不需要为了高亮显示而在运行时解析每篇文档。为了实现一系列的高亮显示，索引的时候通过 Field 对象保存该位置信息。

//增加文档时保存 Term 位置信息。

```
private void addDoc(IndexWriter writer, String text) throws IOException
{
    Document d = new Document();

    Field f = new Field(FIELD_NAME, text ,
```



```

        Field.Store.YES, Field.Index.TOKENIZED,
        Field.TermVector.WITH_POSITIONS_OFFSETS);
    d.add(f);
    writer.addDocument(d);
}

```

//利用 Term 位置信息节省 Highlight 时间。

```

void doStandardHighlights() throws Exception

{

    Highlighter highlighter =new Highlighter(this,new QueryScorer(query));

    highlighter.setTextFragmenter(new SimpleFragmenter(20));

    for (int i = 0; i < hits.length(); i++)

    {

        String text = hits.doc(i).get(FIELD_NAME);

        int maxNumFragmentsRequired = 2;

        String fragmentSeparator = "...";

        TermPositionVector tpv =
(TermPositionVector)reader.getTermFreqVector(hits.id(i),FIELD_NAME);

        // 如果该列没有去除 stop words 还可以改成
TokenSources.getTokenStream(tpv,true); 进一步提速。

        TokenStream tokenStream=TokenSources.getTokenStream(tpv);

        //analyzer.tokenStream(FIELD_NAME,new StringReader(text));

        String result =

            highlighter.getBestFragments(

                tokenStream,

```

```

        text,

        maxNumFragmentsRequired,

        fragmentSeparator);

    System.out.println("\t" + result);

}

}

```

最后把 `highlight` 包中的一个额外的判断去掉。对于中文来说没有明显的单词界限，所以下面这个判断是错误的：

```
tokenGroup.isDistinct(token)
```

注意上面的 `highlighter.setTextFragmenter(new SimpleFragmenter(20));` 这句话。`SimpleFragmenter` 是一个最简单的段落分割器。它把文章分成 20 个字的一个段落。这种方式简单易行，但显得比较初步。有时候会出现一些没意义的符号出现在摘要开始这样的现象。

营销宣传策划：企业如何借媒体之力打开市场

，无不如群雄逐鹿中原。初是跨国公司如宝洁、联合利华进军中国市场发动兼并，欧莱雅收购著名品牌小护士等形成合纵之势。而后是我国各汽车企业开始大规模的整合，第一汽车集团和天津汽车集团重组，一汽控股天汽的优良资产——夏利

`RegexFragmenter` 是一个改进版本的段落分割器。它通过一个正则表达式匹配可能的热点区域。但它是为英文定制的。我们可以让它认识中文的字符段。

```
protected static final Pattern textRE = Pattern.compile("[\\w\\u4e00-\\u9fa5]+");
```

这样使用 `highlighter` 就变成了：

```
highlighter.setTextFragmenter(new RegexFragmenter(descLenth));
```

7.5 实现多维视图

`cnet` 是有关高端 IT 电子产品（如手机，数码相机等）比价、评价等综合网站，相关的产品搜索功能是由 `lucene` 支持的。该系统实现的最大特点是产品从各个角度(维)进行分类并

进行了相关统计。分类是多层次的，用户可以沿着某一类继续细化，这有点儿象 OLAP 应用模式，但它不是用数据库而是用 lucene 完成的，因此，这也是 lucene 的一个很有特色的应用案例。



lucene 是如何做到这一点的呢？Chris Hostetter 在 lucene maillist 中有过详细的讨论，这里根据他的论述，概要说明一下：

系统主要是利用 QueryFilter 来实现，QueryFilter 有个 bits 方法返回一个 BitSet 集，这个 BitSet 的大小是所针对的 lucene 库的大小（也就是 new BitSet(reader.maxDoc())），凡符合 filter 条件的 document 在集中相应位置上置布尔 1（true）。这个 BitSet 集对特定 QueryFilter 对象来说是 cache 保存的，下次调用不会重新计算。cnet 正是利用了这个集，将满足各个基本属性值的 BitSet 值计算出，根据特定用户需要进行相关的 BitSet 与（交集）操作，最后利用 BitSet 集的 cardinality() 方法就可计算出满足该类的总数。

下面是参考代码：

```
String[] mfgs = {"105", "93", "125"}; //类别数组

QueryParser parser = new QueryParser("title", analyzer);

Query q = parser.parse(qString);

Searcher searcher = new IndexSearcher(indexPath);
IndexReader reader = IndexReader.open(indexPath);

Hits results = searcher.search(q);
BitSet all = (new QueryFilter(q)).bits(reader);
```

```
int[] mfg_counts = new int[mfgs.length]; //类别计数

for (int i=0;i<mfgs.length;++i) {
    BitSet these = (new QueryFilter(new TermQuery(new Term("type",
mfgs[i])))).bits(reader);
    these.and(all);
    mfg_counts[i] = these.cardinality();
}
```

在 Apache 的另外一个企业搜索项目 Solr 中，通过优化后的 BitSet 实现了一个 DocSetHitCollector来做分组求和。这个优化后的 BitSet 叫做 OpenBitSet。但是这个 OpenBitSet 只是在 64 位的机器上，和当返回的结果数量很多的时候才比 java 内部的 BitSet 类更快。

计算一个二进制的数组的 1 的个数在整个计算中对性能有比较重要的影响：

对照表：

数值	1	2	3	4	5 ... 255
1 的个数	1	1	2	1	2 ..8

```
private static int[] _bitsSetArray65536 = null;

static {

    _bitsSetArray65536 = new int[65536];

    byte[] _bitsSetArray256 = { 0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2,
                                3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3,
4, 4, 5, 1,
                                2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5, 2, 3, 3,
4, 3, 4, 4,
                                5, 3, 4, 4, 5, 4, 5, 5, 6, 1, 2, 2, 3, 2, 3, 3, 4, 2,
3, 3, 4, 3,
```

```

        4, 4, 5, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5,
6, 2, 3, 3,

        4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4,
5, 5, 6, 4,

        5, 5, 6, 5, 6, 6, 7, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3,
4, 3, 4, 4,

        5, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 2,
3, 3, 4, 3,

        4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5,
6, 4, 5, 5,

        6, 5, 6, 6, 7, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4,
5, 5, 6, 3,

        4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7, 3, 4, 4,
5, 4, 5, 5,

        6, 4, 5, 5, 6, 5, 6, 6, 7, 4, 5, 5, 6, 5, 6, 6, 7, 5,
6, 6, 7, 6,

        7, 7, 8 };

```

```

for (int j = 0; j < 65536; j++) {

    _bitsSetArray65536[j] = _bitsSetArray256[j & 0xff]

        + _bitsSetArray256[(j>> 8 )& 0xff];

}

}

```

计算 A 中的 1 的个数:

```

public static long pop_array2(long A[],int wlen) {

    long _count = 0;

```

```

        for (int i = 0; i < wlen; i++) {

            _count += _bitsSetArray65536[(int) (A[i]& 0xffff)]

                + _bitsSetArray65536[(int) ((A[i] >>> 16) & 0xffff)]

                + _bitsSetArray65536[(int) ((A[i] >>> 32) & 0xffff)]

                + _bitsSetArray65536[(int) ((A[i] >>> 48) &
0xffff)];

        }

        return _count;

    }

```

鉴于 Facetedsearch 已经变得如此普及和重要，对分组统计小的优化也许是值得的。下面的实现通过减少计算步骤又比上面的实现至少快了百分之几。

```

System.String[] mfgs = new System.String[] { "105", "93", "125" }; //类别
数组

```

```

DocSetHitCollector all = new DocSetHitCollector(reader.MaxDoc());

```

```

searcher.Search(q, all);

```

```

DocSet allDocSet = all.DocSet;

```

```

int[] mfg_counts = new int[mfgs.Length];

```

```

for (int i = 0; i < mfgs.Length; ++i)

```

```

{

```

```
DocSetHitCollector these = new DocSetHitCollector(reader.MaxDoc());

searcher.Search(new TermQuery(new Term("type", mfgs)), these);

//通过把集合求并和集合大小的计算合并在一个函数中实现来加快计算



mfg_counts = these.DocSet.intersectionSize(allDocSet);

}
```

上面这个实现比起最初的实现，在于合并了以下两个步骤：

```
these.and(all);
mfg_counts[i] = these.cardinality();
```

二级子树展开的效果图：

ipod nano (Best Matching categories for your search)			
Consumer Electronics MP4 Players & Access.. (32780)  Show All	MP4 Players & Access.. MP4 Players (30494) MP4 Accessories (2208)	MP3 Players & Access.. MP3 Players (6869) MP3 Accessories (807)	Ipod Accessories iPod Cases (631) other iPod accessori.. (318)  Show All

二级子树展开的实现：

```
HashMap<Integer, CountNode> cat1Set = new HashMap<Integer,
CountNode>(); //搜索形成的二级子树

for (Count c : facetCounts) {

    Integer cat2Id = Integer.parseInt(c.getName());

    CatNode cat2Node = catMap.get(cat2Id);

    CountNode newParen = cat1Set.get(cat2Node.parent.no);

    if (limitCat > 0) {

        if (cat2Node.parent.no != limitCat) {

            continue;
        }
    }
}
```

```

    }

}

if (newParen == null) {

    newParen = new CountNode(cat2Node.parent.no,

        cat2Node.parent.name, null, false);

    CountNode childNode = new CountNode(cat2Node.no,
cat2Node.name,

        newParen, true);

    childNode.count = c.getCount();

    newParen.children.add(childNode);

    cat1Set.put(newParen.no, newParen);

} else {

    CountNode childNode = new CountNode(cat2Node.no,
cat2Node.name,

        newParen, true);

    childNode.count = c.getCount();

    newParen.children.add(childNode);

}

}

```

7.6 实现相似文档搜索

有时候需要检索与给定文档（例如 BBS 讨论区内某一帖子）相似的文档。

可以修改查询语法，当用户输入 `related:doc_id` 的时候返回索引库中的相关文档。虽然

更加通用的 `related:url` 地址 形式也是可以实现,但对 URL 地址内容的提取毕竟不内部文档准确。下面实现对内部文档查询相似文档。在 Lucene 的外围资源中有个 `MoreLikeThis` 类。

```
MoreLikeThis mlt = new MoreLikeThis(reader);

mlt.setFieldNames(new String[] { "title", "content" });
mlt.setMaxQueryTerms(5);

if(queryString.startsWith("related:"))
{
    int docId = Integer.parseInt(queryString.substring(8));
    query = mlt.like(docId);
}
```

顾名思义,这个类的作用就是找出更多类似于“This”的结果。举个例子说明这个类的作用。比如对一个卖商品的网站来说,当顾客正在浏览一件商品时,如果能把和这件商品性能,作用很相近的商品也同时罗列在网页的左边,万一顾客想要的商品正好就在其中,那么这个网站的营业额肯定会有所提高。

http://lucene.apache.org/java/2_0_0/api/org/apache/lucene/search/similar/MoreLikeThis.html 有 `MoreLikeThis` 的 API 文档。这个类实现的大体思想说明一下。通过 API 文档可以看出 `MoreLikeThis.java` 类有一个主要的方法“`like(int docNum)`”,它被重载,参数可以是 `File`, `InputStream`, `Reader` 和 `URL`, 返回值是一个 `Query.MoreLikeThis` 类的构造函数是 `MoreLikeThis(IndexReader ir)`, 它需要传进一个 `IndexReader`。下面我就小举一例子说明 `like` 方法的用法。就拿第一段的需求为例:

```
public static void main(String[] a) throws Throwable {

    String indexName = "indexpath";

    IndexReader r = IndexReader.open(indexName);

    PrintStream o = System.out;

    o.println("Open index " + indexName + " which has " + r.numDocs()
+ " docs");

    MoreLikeThis mlt = new MoreLikeThis(r);
```

```

mlt.setMaxQueryTerms(5);

o.println("Query generation parameters:");

o.println(mlt.describeParams());

o.println();

String keygoodid = "";

String similarGoodsid = "";

String keygoodName = "";

String similarGoodsName = "";

if(!r.isDeleted(100)){

Document keyDoc = r.document(j);

keygoodid = keyDoc.get("id");

keygoodName = keyDoc.get("name");


Query query = mlt.like(100);

IndexSearcher searcher = new IndexSearcher(indexName);

Hits hits = searcher.search(query);

int len = hits.length();


for (int i = 0; i < Math.min(5, len); i++) {

    Document d = hits.doc(i);

    similarGoodsid += d.get("id") + ",";

```

```

        similarGoodsName += d.get("name") + ",";

    }

    o.println("keygoodid:" + keygoodid + "|" + "similarGoodsid:" +
similarGoodsid);

    o.println("keygoodName:" + keygoodName + "|" + "similarGoodsName:"
+ similarGoodsName);

}

}

```

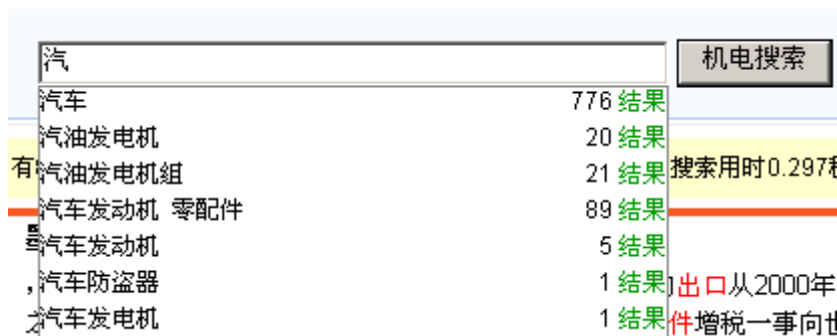
IndexPath 下面存放的是对所有商品的索引，构造一个 MoreLikeThis 的对象 mlt,然后调用 mlt.like(100),100 为 Lucene 内部的 docNum。然后 serach 一下，取前几个结果就是与此 doc 最为相似的。

like(int docNum)方法的 Query 是怎么产生的呢？它首先根据传入的 docNum 找出该 doc 里的高频 terms，然后用这些高频 terms 生成 Queue，最后把 Queue 传进 search 方法得到最后结果。它的主要思想就是认为这些高频 terms 足以表示 doc 信息,然后通过搜索得到最后与此 doc 类似的结果。

缺省的 MoreLikeThis 没有对 StopWords 的定义，以及对中文分词的支持，这都是需要进一步完善的。

7.7 实现 AJAX 自动完成

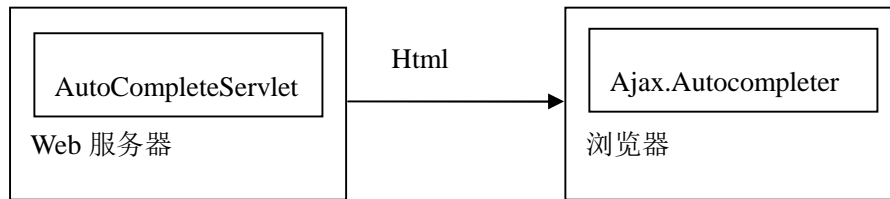
搜索输入框中的下拉提示给用户一个有参考意义的搜索词表。同时也提供用户搜索该词预期的结果数量。这个功能一般是由浏览器端的 Ajax 代码完成的。



搜索词表从用户搜索日志中统计出来。搜索次数多的词排在前面。例如：用户输入“汽”时，“汽车”的搜索次数比“汽油发电机”多，所以排在提示词列表的前面。

7.7.1 总体结构

自动完成功能的总体结构如下：



7.7.2 服务器端处理

当用户输入一个搜索字的同时由 Sevelet 从后台取数。

```

CREATE TABLE [keywordAnalysis] (

    [SearchTerms] [varchar] (50) NOT NULL,    --搜索词

    [AccessCount] [int] NULL,                --用户搜索词数

    [Result] [int] NULL,                    --搜索返回结果数

    [Count] [varchar] (50) NULL             --搜索返回结果数的字符型表示

)
    
```

AutoCompleteServlet 的主要代码如下：

```

StringBuilder message = new StringBuilder("<ul>");
try {
    drv = (Driver)Class.forName(driver).newInstance();
    DriverManager.registerDriver(drv);
    con = DriverManager.getConnection(url,user,password);

    String sql = "SELECT top 9 [searchTerms] , [Count] FROM
[KeywordAnalysis] WHERE ([searchTerms] LIKE '"+val+"%') and
    
```

```
[searchTerms]!=''+val+' ' and result>0";
    PreparedStatement stmt = con.prepareStatement(sql);
    ResultSet rs = stmt.executeQuery();

    while (rs.next())
    {
        String Word = rs.getObject(1).toString();
        String Count = rs.getObject(2).toString();

        message.append("<li
style=\"font-size:12px;padding:0px;height:15px;line-height:15px;overflow:hidden\"><div style=\"text-align:left;float:left\">");
        message.append(Word);
        message.append("</div><div
style=\"text-align:right;float:right\"><span class=\"informal\">");
        message.append(Count);
        message.append("&nbsp;<font color=\"#009900\">结果
</font></span></div></li>");
    }
    rs.close();
    con.close();
    stmt.close();
} catch (Exception e) {
    e.printStackTrace();
}

message.append("</ul>");

response.setContentType("text/html; charset=utf-8");
response.setCharacterEncoding("utf-8");
PrintWriter out = response.getWriter();

out.println(message.toString());
```

AutoCompleteServlet 通过 web.xml 部署到 URL 地址 “/autoComplete”。

```
<servlet>

    <servlet-name>AutoCompleteServlet</servlet-name>
<servlet-class>com.lietu.autocomplete.AutoCompleteServlet</servlet-cl
ass>

</servlet>

<servlet-mapping>
```

```
<servlet-name>AutoCompleteServlet</servlet-name>

<url-pattern>/autoComplete</url-pattern>

</servlet-mapping>
```

7.7.3 浏览器端处理

剩下的就是在前台通过 Ajax 组件库 scriptaculous 中的 Ajax.Autocompleter 组件来完成显示了。

```
<script language="JavaScript" type="text/javascript"
src="scripts/prototype/prototype.js"></script>

<script language="JavaScript" type="text/javascript"
src="scripts/scriptaculous/scriptaculous.js"></script>

<script language="JavaScript" type="text/javascript"
src="scripts/scriptaculous/controls.js"></script>

<script language="JavaScript" type="text/javascript"
src="scripts/scriptaculous/effects.js"></script>

...

<input autocomplete="off" type="text" name="query" id="autocomplete"
style="width:315px;" class="wd" value="<%=query%>" />

<div class="auto_complete" id="autocomplete_choices"></div>

<script type="text/javascript">

new Ajax.Autocompleter(' autocomplete ', ' autocomplete_choices ',

                        'autoComplete', { afterUpdateElement :

getSelectionId});

function getSelectionId(text, li) {

window.location =

"/SearchCompany.jsp?query="+encodeURIComponent(text.value);

}
```

```
</script>
```

其中 `prototype.js`、`scriptaculous.js`、`controls.js` 和 `effects.js` 的包含顺序不能随意颠倒。

7.7.4 服务器端改进

当自动完成的性能需要进一步提高的时候，可以直接在内存中管理查询，而不是访问数据库取数。

我们先设计词典格式。它是三列组成，第一列是词，第二列是搜索返回结果数量，第三列是用户搜索次数，中间用%隔开，例如：

综合教程第一册%34%2

搜索词是“综合教程第一册”，搜索返回结果数量是 34，用户搜索了 2 次。这样把用户搜索次数多的关键词放在前面优先显示。我们构造一个快速的查找树“Ternary Search Trie”来实现这个词典：

```
public class SuggestDic {
    /**
     * An inner class of Ternary Search Trie that represents a node in
the trie.
     */
    public static class TSTNode
    {
        /** The key to the node. */
        public int data = 0;
        public int weight = 0;

        /** The relative nodes. */
        public TSTNode LOKID;
        public TSTNode EQKID;
        public TSTNode HIKID;
        public TSTNode PARENT;

        /** The char used in the split. */
        public char splitchar;

        /**
         * Constructor method.
         */
    }
}
```

```

        @param splitchar The char used in the split.
        @param parent    The parent node.
    */
    public TSTNode(char splitchar, TSTNode p)
    {
        this.splitchar = splitchar;
        PARENT = p;
    }
}

public static class TSTItem implements Comparable
{
    /** The key to the node. */
    public int data = 0;
    public int weight = 0;

    /** The char used in the split. */
    public String key;

    /**
     * Constructor method.
     *
     * @param splitchar The char used in the split.
     * @param parent    The parent node.
     */
    public TSTItem(String k, int d,int w)
    {
        this.key = k;
        this.data = d;
        this.weight = w;
    }

    public int compareTo(Object obj){
        TSTItem that = (TSTItem)obj;

        return (that.weight - weight);
    }
}

/** The base node in the trie. */
private TSTNode root;

public static String getDir()
{

```



```

        String dir = System.getProperty("dic.dir");
        if (dir == null)
            dir = "/dic/";
        else if( !dir.endsWith("/"))
            dir += "/";
        return dir;
    }

    private static SuggestDic dicSug = new SuggestDic();

    /**
     *
     * @return the singleton of the dictionary
     */
    public static SuggestDic getInstance()
    {
        return dicSug;
    }

    private SuggestDic()
    {
        this("suggestDic.txt");
    }

    /**
     * Constructs a Ternary Search Trie and loads data from a
    <code>File</code> into the Trie.
     * The file is a normal text document, where each line is of the form
     * word : integer.
     *
     * @param file The <code>File</code> with the data to load
    into the Trie.
     * @exception IOException A problem occurred while reading the data.
     */
    public SuggestDic(String dic){
        try{
            InputStream file = null;
            if (System.getProperty("dic.dir") == null)
                file =
getClass().getResourceAsStream(SuggestDic.getDir()+dic);
            else
                file = new FileInputStream(new
File(SuggestDic.getDir()+dic));

```

```

BufferedReader in;
in = new BufferedReader(new InputStreamReader(file, "GBK"));

String word;
int occur=0;
int weight = 0;
while ((word = in.readLine()) != null) {
    StringTokenizer st = new StringTokenizer(word, "%");

    String key = st.nextToken();
    occur=Integer.parseInt(st.nextToken());
    weight=Integer.parseInt(st.nextToken());
    if (root == null) {
        root = new TSTNode(key.charAt(0), null);
    }
    TSTNode node = null;
    if (key.length() > 0 && root != null) {
        TSTNode currentNode = root;
        int charIndex = 0;
        while (true) {
            if (currentNode == null)
                break;
            int charComp =
                (key.charAt(charIndex)-
                 currentNode.splitchar);
            if (charComp == 0) {
                charIndex++;
                if (charIndex == key.length()) {
                    node = currentNode;
                    break;
                }
                currentNode = currentNode.EQKID;
            } else if (charComp < 0) {
                currentNode = currentNode.LOKID;
            } else {
                currentNode = currentNode.HIKID;
            }
        }
        int occur2 = 0;
        if (node != null)
        {
            occur2 = node.data;
        }
        if (occur2 != 0) {

```

```

        occur+=occur2;
    }
    currentNode =
        getOrCreateNode(key);

    occur2 = currentNode.data;
    if (occur2 != 0) {
        //System.out.println("add");
        occur+=occur2;
    }
    currentNode.data = occur;
    currentNode.weight = weight;
}
}
in.close();
} catch( IOException e)
{
    e.printStackTrace();
}
}

/**
 * Returns the node indexed by key, creating that node if it doesn't
 * exist,
 * and creating any required intermediate nodes if they don't exist.
 *
 * @param key A <code>String</code> that
 * indexes the node that is returned.
 * @return The node object indexed by key.
 * This object is an
 * instance of an inner class
 * named <code>TernarySearchTrie.TSTNode</code>.
 * @exception NullPointerException If the key is
 * <code>null</code>.
 * @exception IllegalArgumentException If the key is an empty
 * <code>String</code>.
 */
protected TSTNode getOrCreateNode(String key)
    throws NullPointerException, IllegalArgumentException {
    if (key == null) {
        throw new NullPointerException("attempt to get or create node
with null key");
    }
    if ("".equals(key)) {

```

```

        throw new IllegalArgumentException("attempt to get or create
node with key of zero length");
    }
    if (root == null) {
        root = new TSTNode(key.charAt(0), null);
    }
    TSTNode currentNode = root;
    int charIndex = 0;
    while (true) {
        int charComp = (
            key.charAt(charIndex) -
            currentNode.splitchar);
        if (charComp == 0) {
            charIndex++;
            if (charIndex == key.length()) {
                return currentNode;
            }
            if (currentNode.EQKID == null) {
                currentNode.EQKID =
                    new TSTNode(key.charAt(charIndex), currentNode);
            }
            currentNode = currentNode.EQKID;
        } else if (charComp < 0) {
            if (currentNode.LOKID == null) {
                currentNode.LOKID =
                    new TSTNode(key.charAt(charIndex), currentNode);
            }
            currentNode = currentNode.LOKID;
        } else {
            if (currentNode.HIKID == null) {
                currentNode.HIKID =
                    new TSTNode(key.charAt(charIndex), currentNode);
            }
            currentNode = currentNode.HIKID;
        }
    }
}

/**
 * Returns the node indexed by key, or <code>null</code> if that node
doesn't exist.
 * The search begins at root node.
 *
 * @param key2      A <code>String</code> that indexes the node that

```

is returned.

```

    @param startNode The top node defining the subtrie to be searched.
    @return          The node object indexed by key. This object is
    *               an instance of an inner class named
<code>TernarySearchTrie.TSTNode</code>.

```

```

    */
    protected TSTNode getNode(String key, TSTNode startNode) {
        if (key == null || startNode == null || key.length() == 0) {
            return null;
        }
        TSTNode currentNode = startNode;
        int charIndex = 0;
        while (true) {
            if (currentNode == null) {
                return null;
            }
            int charComp = key.charAt(charIndex) - currentNode.splitchar;
            if (charComp == 0) {
                charIndex++;
                if (charIndex == key.length()) {
                    return currentNode;
                }
                currentNode = currentNode.EQKID;
            } else if (charComp < 0) {
                currentNode = currentNode.LOKID;
            } else {
                currentNode = currentNode.HIKID;
            }
        }
    }
}

```

```

/**
 * Returns the node indexed by key, or <code>null</code> if that node
doesn't exist.

```

```

    * Search begins at root node.
    *
    @param key A <code>String</code> that indexes the node that is
returned.

```

```

    @return The node object indexed by key. This object is an
    *       instance of an inner class named
<code>TernarySearchTrie.TSTNode</code>.

```

```

    */
    public TSTNode getNode(String key) {
        return getNode(key, root);
    }

```

```

    }

    /**
     * Returns the key that indexes the node argument.
     *
     * @param node The node whose index is to be calculated.
     * @return The String that indexes the node argument.
     */
    protected String getKey(TSTNode node) {
        StringBuffer getKeyBuffer = new StringBuffer();
        getKeyBuffer.setLength(0);
        getKeyBuffer.append(node.splitchar);
        TSTNode currentNode;
        TSTNode lastNode;
        currentNode = node.PARENT;
        lastNode = node;
        while (currentNode != null) {
            if (currentNode.EQKID == lastNode) {
                getKeyBuffer.append(currentNode.splitchar);
            }
            lastNode = currentNode;
            currentNode = currentNode.PARENT;
        }
        getKeyBuffer.reverse();

        return getKeyBuffer.toString();
    }

    /**
     * Returns an List of all keys in the trie that begin
    with a
     * given prefix. Only keys for nodes having non-null data are included
    in the List.
     *
     * @param prefix Each key returned from this method will begin
    with the characters in prefix.
     * @param numReturnValues The maximum number of values returned from
    this method.
     * @return A List with the results
     */
    public TSTItem[] matchPrefix(String prefix, int numReturnValues) {

        TSTNode startNode = getNode(prefix);
        if (startNode == null) {

```

```

        return null;
    }
    ArrayList<TSTItem> sortKeysResult = new ArrayList<TSTItem>();

    ArrayList<TSTItem> wordTable = sortKeysRecursion(
        startNode.EQKID,
        ((numReturnValues < 0) ? -1 : numReturnValues),
        sortKeysResult);
    int retNum = Math.min(numReturnValues,wordTable.size());

    Select.selectRandom(wordTable,wordTable.size(),retNum,0);
    TSTItem[] fullResults = new TSTItem[retNum];
    for(int i=0;i<retNum;++i)
    {
        fullResults[i] = wordTable.get(i);
    }

    return fullResults;
}

/**
 * Returns keys sorted in alphabetical order. This includes the current
Node and all
 * nodes connected to the current Node.
 * <p>
 * Sorted keys will be appended to the end of the resulting
<code>List</code>. The result may be
 * empty when this method is invoked, but may not be <code>null</code>.
 *
 * @param currentNode          The current node.
 * @param sortKeysNumReturnValues The maximum number of values in the
result.
 * @param sortKeysResult2      The results so far.
 * @return A <code>List</code> with the results.
 */
private ArrayList<TSTItem> sortKeysRecursion(
    TSTNode currentNode,
    int sortKeysNumReturnValues,
    ArrayList<TSTItem> sortKeysResult2) {

    if (currentNode == null) {
        return sortKeysResult2;
    }

```

```

        ArrayList<TSTItem> sortKeysResult =
            sortKeysRecursion(
                currentNode.LOKID,
                sortKeysNumReturnValues,
                sortKeysResult2);

        if (currentNode.data != 0) {
            sortKeysResult.add(
                new TSTItem(getKey(currentNode),
                    currentNode.data,
                    currentNode.weight)
            );
        }

        sortKeysResult =
            sortKeysRecursion(
                currentNode.EQKID,
                sortKeysNumReturnValues,
                sortKeysResult);

        return sortKeysRecursion(
            currentNode.HIKID,
            sortKeysNumReturnValues,
            sortKeysResult);
    }
}

```

我们可以写一个简单的测试代码：

```

public static void main(String[] args) {
    SuggestDic sugDic = SuggestDic.getInstance();
    String prefix = "m";
    TSTItem[] ret = sugDic.matchPrefix(prefix, 10);
    for(TSTItem i:ret )
    {
        System.out.println(i.key+":"+i.data+":"+i.weight);
    }
}

```

这样自动完成的 Servlet 类可以改写成下面这样：

```

String val = request.getParameter("query");

StringBuilder message = new StringBuilder("<ul>");

```



```

try {
    SuggestDic sugDic = SuggestDic.getInstance();

    TSTItem[] ret = sugDic.matchPrefix(val, 10);
    for(TSTItem i:ret )
    {
        String Word = i.key;
        String Count = String.valueOf(i.data);

        message.append("<li
style=\"font-size:12px;padding:0px;height:15px;line-height:15px;overflow:hidden\"><div style=\"text-align:left;float:left\">");
        message.append(Word);
        message.append("</div><div
style=\"text-align:right;float:right\"><span class=\"informal\">");
        message.append(Count);
        message.append("&nbsp;<font color=\"#009900\">结果
</font></span></div></li>");
    }
} catch (Exception e) {
    e.printStackTrace();
}

message.append("</ul>");

response.setContentType("text/html; charset=utf-8");
response.setCharacterEncoding("utf-8");
PrintWriter out = response.getWriter();

out.println(message.toString());

```

7.7.5 部署总结

suggestDic.txt 可以放在 WEB-INF/classes/dic/ 路径。

AutoCompleteServlet 可以放在 WEB-INF/lib/路径。通过 web.xml 发布。

prototype.js 、 scriptaculous.js、 controls.js 和 effects.js 可以放在 ROOT/js 路径。

7.8 jQuery 实现的自动完成

在 jQuery 环境下，也可以使用 jQuery 的 Autocomplete 插件实现自动完成功能。插件的

原址在:

<http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/>

传递的 json 数据格式由两列组成: w 表示搜索词, c 表示搜索次数, 例如:

```
var searches = [

    { w: "北京", c: "1000" },

    { w: "iPod", c: "12" },

    { w: "iPhone", c: "3" },

    { w: "北方", c: "6" }

];
```

为了达到在下拉列表中显示两列的效果:



修改 jquery.autocomplete.js 中的 function fillList():

```
//var li = $("<li/>").html( options.highlight(formatted, term) ).addClass(i%2 == 0 ? "ac_even" :
"ac_odd").appendTo(list)[0];
```

改成:

```
var li = $("<li/>").html( options.highlight(formatted, term) ).addClass(i%2 == 0 ? "ac_even" :
"ac_odd").appendTo(list)[0];
```

为了能解析 JSON 格式的返回结果, 修改 jquery.autocomplete.js 中的 parse:

```
function parse(data) {
```

```

var parsed = [];

//var rows = data.split("\n");

var rows = eval('(' + data + ')');

if(rows)

{

    for (var i=0; i < rows.length; i++) {

        //var row = $.trim(rows[i]);

        var row = rows[i];

        if (row) {

            //row = row.split("|");

            parsed[parsed.length] = {

                data: row,

                value: row.w,

                result: row.w

            };

            //parsed[parsed.length] = {

            //    data: row,

            //    value: row[0],

            //    result: options.formatResult && options.formatResult(row, row[0])

            // row[0]

            //};

        }
    }
}

```

```

        }

    }

    return parsed;

};

```

界面部分代码，首先增加一个输入框到搜索页面：

```
<input autocomplete="off" type="text" name="q" id="suggest" style="width:315px;" class="wd" value=""/>
```

然后调用修改后的 jQuery 的 autocomplete 插件：

```

$.ready(function() {

    //设置成 POST 方式发送数据给 Servlet

    $.ajaxSetup({type: 'POST'});

    $("#suggest").autocomplete("./autoComplete", {

        minChars: 0,

        width: 310,

        matchContains: true,

        autoFill: false,

        formatItem: function(row, i, max) {

            return i + "/" + max;

        },

        formatMatch: function(row, i, max) {

            return row.w;

        },

        formatResult: function(row) {

```

```

        return row.w;

    }

});

});

```

服务器端修改成:

```

public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
    throws IOException {
    String val = request.getParameter("q");
    //System.out.println("value:"+val);

    String message = null;
    SuggestItem[] items = null;
    try {
        SuggestDic sugDic = SuggestDic.getInstance();

        items = sugDic.matchPrefix(val, 10);

        JSONValue lMyValue = JSONMapper.toJSON(items);
        message =
Escape.toUnicodeEscapeString( lMyValue.render(false) );

    } catch (Exception e) {
        e.printStackTrace();
    }

    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    if(message!=null)
    {
        //message like:
        "[{\"c\":\"5\",\"w\":\"pc\"},\n{\"c\":\"2\",\"w\":\"pci\"}]";
        //System.out.println(message);
        out.println(message);
    }
    else
    {
        out.println(" ");
    }
}

```

```
}  
}
```

可以通过 EasyMock 测试这个 Servlet 的返回值:

```
String queryWord = "P";  
HttpServletRequest request =  
createMock(HttpServletRequest.class);  
HttpServletResponse response =  
createMock(HttpServletResponse.class);  
ServletConfig servletConfig = createMock(ServletConfig.class);  
ServletContext servletContext =  
createMock(ServletContext.class);  
  
AutoCompleteServlet instance = new AutoCompleteServlet();  
  
//初始化servlet,一般由容器承担,一般调用servletConfig作为参数初始化,此  
处模拟容器行为  
instance.init(servletConfig);  
  
//在某些方法被调用时设置期望的返回值,如下这样就不会去实际调用  
servletConfig的getServletContext方法,而是直接返回  
//servletContext,由于servletConfig是mock出来的,所以可以完全控制。  
  
expect(servletConfig.getServletContext()).andReturn(servletContext).a  
nyTimes();  
  
expect(request.getParameter("q")).andReturn(queryWord);  
  
PrintWriter pw=new PrintWriter(System.out,true);  
expect(response.getWriter()).andReturn(pw).anyTimes();  
response.setContentType("text/html; charset=utf-8");  
  
//以上均是录制,下面为重放,该种机制为easymock测试机制,要理解请看easymock  
测试的一些资料  
replay(request);  
replay(response);  
replay(servletConfig);  
replay(servletContext);  
  
instance.doPost(request, response);  
  
pw.flush();
```

```

verify(request);
verify(response);
verify(servletConfig);

verify(servletContext);

```

返回一个 JSON 数组格式的数据，其中汉字已经编码：

```

[{"c":1,"w":"PID"}, {"c":23,"w":"PLC"}, {"c":6,"w":"PLC
\u897f\u95e8\u5b50"}, {"c":6,"w":"PLC\u57f9\u8bad"}, {"c":1,"w":"PLC\u5
728\u7535\u529b\u7cfb\u7edf\u4e2d\u7684\u5e94\u7528"}, {"c":9,"w":"PLC
\u7a0b\u5e8f"}, {"c":1,"w":"PLC\u7684\u9009\u578b"}, {"c":1,"w":"PLC5"},
{"c":2,"w":"PLC\u7f16\u7a0b\u624b\u518c"}, {"c":11,"w":"PLC
\u63a7\u5236\u5668"}]

```

这个实现和 scriptaculous 中的 `Ajax.Autocompleter` 组件比较起来，它内部有用户已经输入过词的缓存，而且服务器端不需要返回格式信息，因此总的数据传输量更少。

7.9 集成其他功能

7.9.1 拼写检查

因为用户的查询本身是一个符合查询语法的字符串，所以不能把用户的查询本身直接输入给拼写检查模块，而要通过一个 `didYouMeanParser` 给出这个提示。拼写提示词的代码如下：

```

IndexSearcher is = new
IndexSearcher(originalIndexDirectory);
Query query = didYouMeanParser.parse(queryString);
Hits hits = is.search(query);

String suggestedQueryString = null;

//如果搜索返回结果的数量小于一个值或者匹配第一个结果的分值小于最小值就查找提示词
if (hits.length() < minimumHits || hits.score(0) < minimumScore)
{
    Query didYouMean = didYouMeanParser.suggest(queryString);
    if (didYouMean != null) {
        suggestedQueryString =
didYouMean.toString(defaultField);
    }
}
is.close();

```

7.9.2 分类统计

7.9.3 相关搜索

一种方法是从搜索日志中挖掘字面相似的词作为相关搜索词列表。首先从一个给定的词语挖掘多个相关搜索词，可以用编辑距离为主的方法查找一个词的字面相似词，如果候选的相关搜索词很多，就要筛选出最相关的 10 个词。下面是利用 Lucene 筛选最相关词的方法。

```
private static final String TEXT_FIELD = "text";

/**
 *
 * @param words 候选相关词列表
 * @param word 要找相关搜索词的种子词
 * @return
 * @throws IOException
 * @throws ParseException
 */
static String[] filterRelated(HashSet<String> words, String word)
throws IOException, ParseException
{
    StringBuilder sb = new StringBuilder();

    for(int i=0;i<word.length();++i)
    {
        sb.append(word.charAt(i));
        sb.append(" ");
    }

    RAMDirectory store = new RAMDirectory();
    IndexWriter writer = null;
    writer = new IndexWriter(store, new StandardAnalyzer(), true);

    for(String text:words)
    {
        Document document = new Document();
        Field textField = new Field(TEXT_FIELD, text, Field.Store.YES,
Field.Index.TOKENIZED);
        document.add(textField);
        writer.addDocument(document);
    }
    writer.close();
}
```



```

IndexSearcher searcher = new IndexSearcher(store);

QueryParser queryParser = new QueryParser(TEXT_FIELD, new
StandardAnalyzer());
Query query = queryParser.parse(sb.toString());

Hits hits = searcher.search(query);
int maxRet = Math.min(10, hits.length());

String[] relatedWords = new String[maxRet];
for (int i = 0; i < maxRet ; i++) {
    Document document = hits.doc(i);
    String text = document.get(TEXT_FIELD);
    System.out.println(text);
    relatedWords[i]=text;
}
searcher.close();
store.close();

return relatedWords;
}

```

整理出这样的相关词表，第一列是关键词，后续是 10 个以内的相关搜索词：

集福轩婚礼%集福轩

手机定位跟踪系统%手机定位系统%手机定位%手机定位仪器

喷绘材料卖店电话%我要喷绘材料卖店电话

厦门房产%厦门租房%厦门新闻%厦门桑拿%房产%青岛房产%厦门%恒雄房产

送水果%送水%水果

三星传真机%三星手机

另外一种方法，可以把多个用户共同查询的词看成相关搜索词，需要有记录用户 IP 的搜索日志才能实现。

然后通过 `RelatedEngine` 类查找某个关键词的相关词。

```

public static void main(String[] args) throws Exception {

```

```

        RelatedEngine re =new RelatedEngine(new
File("D:/lg/work/xiaoxishu/dic/relatedwords.txt"));
        String word = "徐家汇";
        String[] relatedWords = re.getRelated(word);
        for(String w : relatedWords)
        {
            System.out.println(w);
        }
    }
}

```

输出如下：

上海徐家汇
 徐汇
 徐家汇价格是
 上房徐家汇路附近有吗

最后通过自定义的Tag标签RelatedTag在Jsp页面显示出相关搜索词。
 在Taglib Library Descriptor 中定义Tag:

```

<tag>
    <name>relatedWords</name>
    <tag-class>com.bitmechanic.listlib.RelatedTag</tag-class>
    <description></description>

    <attribute>
        <name>index</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>

    <attribute>
        <name>url</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>

    <attribute>
        <name>query</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>

</tag>

```

在Jsp页面调用：

```
<list:relatedWords index="D:/zxh/related" url="Search.jsp" query="<%=query%>" />
```

7.9.4 再次查找

经常需要从结果中缩小范围再次查找记录。

通过+连接符连接上次查询和当前查询。例如：`inputstr` 记录了上次查询词，`queryString` 记录当前查询词。

```
if (refind)

    queryString = " + (" + queryString + ") + (" + inputstr + ")";
```

使用这个新的查询词就可以实现再次搜索的功能。

7.9.5 搜索日志

搜索日志是用来分析用户搜索行为和信息需求的重要依据。一般记录如下信息：

- 搜索关键字；
- 用户来源 ip；
- 本次搜索返回结果数量；
- 搜索时间；
- 其他需要记录的应用相关信息。

为了不影响即时搜索的速度，不把搜索日志记录在数据库，而是写在文本中。使用 `log4j` 的日志功能实现。`log4j.properties` 配置如下：

```
log4j.rootCategory=INFO,R
```

```
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
```

```
log4j.appender.R.File=D:/logs/log
```

```
log4j.appender.R.Append=true
```

```
log4j.appender.R.DatePattern=yyyy-MM-dd'.txt'
```

```
log4j.appender.R.Threshold=INFO
```

```
log4j.appender.R.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.R.layout.ConversionPattern=%m|%d{yyyy-MM-dd HH:mm:ss}%n
```

这里把当前日志写到: D:/logs/log 文件中, 旧的日志以每天生成一个文件新的形式存放。

在搜索类中初始化日志类:

```
private static Logger logger =
Logger.getLogger(SearchBbs.class.getName());
```

然后当用户执行一次搜索时, 记录:

```
logger.info(_query+"|"+desc.count+"|"+bbs+"|"+ip);
```

日志文件 log.txt 记录的结果例子如下:

```
什么是新生儿|37|topic|124.1.0.0|2007-11-21 12:25:36
```

```
什么是新生儿|28|bbs|124.1.0.0|2007-11-21 12:25:42
```

```
怀孕|18|topic|124.1.0.0|2007-11-21 12:26:05
```

```
怀孕|2|shangjia|124.1.0.0|2007-11-21 12:26:05
```

```
怀孕|145|bbs|124.1.0.0|2007-11-21 12:26:06
```

```
怀孕|18|topic|124.1.0.0|2007-11-21 12:30:33
```

这里, 第一列是用户搜索词, 第二列是搜索返回结果数量, 第三列是搜索类别, 第四列是 IP 地址, 第五列是搜索的时间。

然后定义搜索日志统计表, 例如我们需要统计搜索最多的词, 可以把搜索最多的词放在 keywordAnalysis 表中:

```
CREATE TABLE [keywordAnalysis] (
```

```
[searchTerms] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL, --搜索词
```

```
[AccessCount] [int] NULL, --搜索计数
```

```
[Result] [int] NULL --该词返回结果数
```

) ON [PRIMARY]

GO

7.10 搜索日志分析

搜索日志中包括大量的 Google 爬虫信息，需要把它和普通用户的搜索区分出来。

可以从 request 信息判断：

baidu 爬虫的 “User-Agent” 信息：

Baiduspider+(+http://help.baidu.jp/system/05.html)

goolge 爬虫的 “User-Agent” 信息：

Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)

```
String userAgent = request.getHeader( "User-Agent" );
```

```
public static String[] getBotName(String userAgent) {
    userAgent = userAgent.toLowerCase();
    int pos=0;
    String res=null;
    if ((pos=userAgent.indexOf("google/"))>-1) {
        res= "Google";
        pos+=7;
    } else
    if ((pos=userAgent.indexOf("msnbot/"))>-1) {
        res= "MSNBot";
        pos+=7;
    }
```

```

    } else

    if ((pos=userAgent.indexOf("googlebot/"))>-1) {

        res= "Google";

        pos+=10;

    } else

    if ((pos=userAgent.indexOf("webcrawler/"))>-1) {

        res= "WebCrawler";

        pos+=11;

    } else

    // The following two bots don't have any version number in their
    User-Agent strings.

    if ((pos=userAgent.indexOf("inktomi"))>-1) {

        res= "Inktomi";

        pos=-1;

    } else

    if ((pos=userAgent.indexOf("teoma"))>-1) {

        res= "Teoma";

        pos=-1;

    }

    if (res==null) return null;

    return getArray(res,res,res + getVersionNumber(userAgent,pos));

}

```

搜索热词的统计:

搜索我们公司排行前二十名的关键字及其访问次数:

SearchTerms	Accesscount	Result
摩托车	3008	181
电子	2312	1224
轴承	1860	745
上海	1632	656
广州	1572	426
计算机	1520	269
浙江	1400	215
轴	1320	508
广交会	1100	7
dvd	1068	95
齿轮	1024	361
国际贸易	968	36
海尔	840	15
包装机	776	307
汽车	700	776
数控机床	688	57
招标	624	4
进出口公司	600	8
出口退税	580	5
进出口	492	123

导出文本	查询条件	2008-07-07	至	2008-07-07	统计
关键词		搜索次数	返回结果数		
下半年运势		4905	7		
内裤		1610	5100		
内衣		1367	10529		
裙子		1176	12171		
测试		1060	20		
凉鞋		929	2569		
拐上床		854	2		
恤		556	9386		
雪纺		525	3721		
12星座		496	1326		
凉拖		456	503		
男人最想娶的十类女人		440	1		
吊带		438	20		
袜子		384	20		
发型		368	12414		
牛仔		309	12957		
女人		292	94979		
性爱		289	9339		
短袖		279	3151		
做爱		273	10976		
小计：		17,006	本页返回结果平均数: 9,459		
总计：		51,954	总共返回结果平均数: 1,577		

按地区来源的搜索日志分析:

地区	搜索次数	总排名
广东省	97716	1
浙江省	49764	2
山东省	47406	3
河北省	43495	4
北京市	41774	5
江苏省	40858	6
辽宁省	36944	7
湖北省	33046	8
河南省	31875	9
四川省	31498	10
湖南省	31243	11
陕西省	27327	12
安徽省	26522	13
上海市	25835	14
山西省	24903	15
广西省	24664	16
黑龙江省	24001	17
福建省	22834	18
江西省	21617	19
吉林省	18048	20
1 2		

从文本到数据库的统计表。

SearchLog.java 类实现：

```

HashMap<String,HashSet<String>> word2IP =

    new HashMap<String,HashSet<String>>();

HashMap<String,Integer> word2ResultNum = new HashMap<String,Integer>();

```

IPCounter.java 类实现：

7.11 本章小结

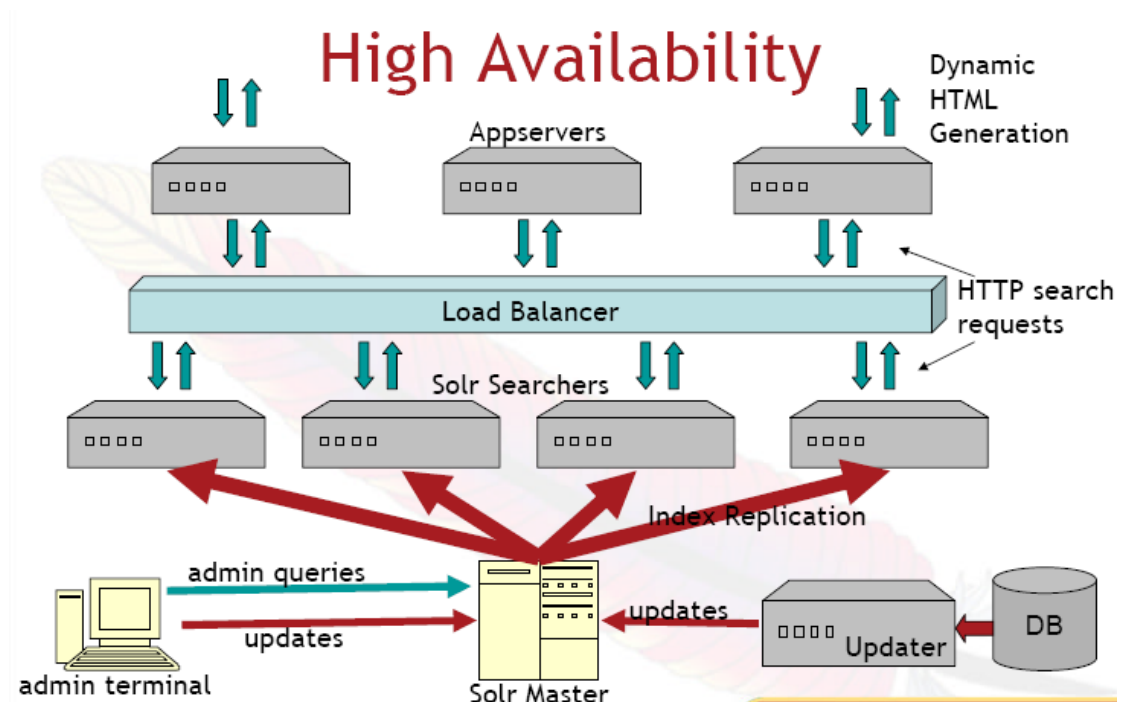
第8章 其他高级主题

8.1 使用 Solr 实现分布式搜索

Solr 把对 Lucene 索引的调用和管理做了一个 REST 风格的封装。它是一个 Web 方式的索引服务器。需要搜索的应用通过 http 请求调用 Solr 索引服务。

单台机器的计算能力有限，可以采用多机集群的分布式搜索来实现高负载和高可用性。Solr 就是一个商用系统演化出来的一个基于 Lucene 的开源项目。关于它的文档介绍在：<http://wiki.apache.org/solr/>。

Solr 自身由客户端和服务端组成。一个完整的分布结构如下：



8.1.1 Solr 服务器端的配置与中文支持

可以把 Solr 的服务器端看成是包装在 Web 应用服务器中的 Lucene 服务器。和 Lucene 索引库相比的好处是可以实现缓存预加载。Web 应用服务器通常可以选用 Resin 或者 Tomcat。

服务器主要需要通过 JNDI 中的 solr/home 来指定 Solr 存储库的路径。

在 Resin 中的 resin.conf 的内容如下：

```
<web-app id="/index" document-directory="webapps/index">
```

```
<env-entry>
```

```
<env-entry-name>solr/home</env-entry-name>
```

```
<env-entry-type>java.lang.String</env-entry-type>
```

```
<env-entry-value>${resin.home}/solr</env-entry-value>
```

```
</env-entry>
```

```
</web-app>
```

Tomcat 的 \Tomcat 5.5\webapps\solr\META-INF\context.xml 配置如下:

```
<Context docBase="${catalina.home}/webapps/solr" debug="0" crossContext="true" >
```

```
<Resource name="/solr/home" type="java.lang.String" value="C:/Program Files/Apache Software  
Foundation/Tomcat 5.5/solr" override="true" />
```

```
</Context>
```

另外也可以通过系统属性 solr.solr.home 来指定存储库路径。

Tomcat 下多个 Solr 的配置:

/usr/local/tomcat55/conf/Catalina/localhost 路径下分别对应两个 web 应用:

gongkong.xml logistics.xml

```
# cat ./logistics.xml
```

```
<Context docBase="" debug="0" crossContext="true" >
```

```
<Environment name="solr/home" type="java.lang.String" value="/usr/local/logistics"  
override="true" />
```

```
</Context>
```

为了支持中文, Tomcat 中的相关配置如下, 在 server.xml 中增加对 UTF-8 的处理, 这样是为了支持 http 的 get 方法:

```
<Connector port="80" useBodyEncodingForURI="true" URIEncoding="UTF-8" />
```

另外在 solr 的 web 项目的 web.xml 配置中增加对 utf-8 的转码:

```
<filter>

    <filter-name>Set Character Encoding</filter-name>

    <filter-class>filters.SetCharacterEncodingFilter</filter-class>

    <init-param>

        <param-name>encoding</param-name>

        <param-value>utf-8</param-value>

    </init-param>

</filter>

<filter-mapping>

    <filter-name>Set Character Encoding</filter-name>

    <url-pattern>/*</url-pattern>

</filter-mapping>
```

这个 filter 是为了支持 http 的 post 方法。

然后需要调整的是 solrconfig.xml。

```
<maxWarmingSearchers>4</maxWarmingSearchers>
```

自动加载缓存的一个线程数，缺省是 4，要调整小一点。否则会使一个双 CPU 双核的机器 CPU 使用率达到近 100%。

通过 Schema.xml 来配置索引库结构，可以把它类比作数据库中的一张表。Schema.xml 使用的基本的数据类型如下：

```
<fieldtype name="sint" class="solr.SortableIntField" />

<fieldtype name="slong" class="solr.SortableLongField"/>
```

```
<fieldtype name="sfloat" class="solr.SortableFloatField"/>
```

```
<fieldtype name="sdouble" class="solr.SortableDoubleField" />
```

```
<fieldtype name="date" class="solr.DateField" />
```

和 Lucene 比较起来, 这些数据类型有更优化的存储方式。

下面定义一个基本的文字搜索列。

```
<fieldtype name="text_ws" class="solr.TextField" positionIncrementGap="100">
```

```
<analyzer>
```

```
<tokenizer class="CnTokenizerFactory"/> CnTokenizerFactory
```

```
</analyzer>
```

这里的 CnTokenizerFactory 是一个分词类, 它扩展了 BaseTokenizerFactory。当然也可以定义自己的做 Tokenizer 的类。所有这样的类都必须是 BaseTokenizerFactory 的子类。

```
package org.apache.solr.analysis;
```

```
public class CnTokenizerFactory extends BaseTokenizerFactory
```

```
{
```

```
    public TokenStream create(Reader input)
```

```
    {
```

```
        return new CnTokenizer(input);
```

```
    }
```

```
}
```

在 Schema.xml 中定义了索引库的结构, 下面定义三列, 分别是唯一 id, 标题和内容。

```
<field name="id" type="string" indexed="true" stored="true" multiValued="false" />
```

```
<field name="title" type="text_ws" indexed="true" stored="true" multiValued="false" />
```

```
<field name="body" type="text_ws" indexed="true" stored="true" multiValued="false" />
```

它的服务器端管理界面包含 Solr 和索引库中的一些宏观的参数。相比较而言，Luke 则显示更细一些的 Document 和 Term 等信息。<http://hot.lietu.com:8080/solr/admin/>是一个已经安装好的 Solr 界面。可以先通过<http://hot.lietu.com:8080/solr/admin/analysis.jsp?highlight=on>来对每个列做基本的测试，看看对定义好的 title 列是否能对中文做正确的处理。

Solr Admin (example)

localhost.localdomain:8080
 cwd=/usr/local/resin-3.1.0 SolrHome=solr/

Field Analysis

Field name

Field value (Index)

verbose output ☒

highlight matches ☒

Field value (Query)

verbose output ☒

Index Analyzer

org.apache.solr.analysis.CnTokenizerFactory {}

term position	1	2	3	4	5
term text	我	爱	北京	天安	门
term type	r	v	ns	ns	ns
source start,end	0,1	1,2	2,4	4,6	6,7

Query Analyzer

org.apache.solr.analysis.CnTokenizerFactory {}

term position	1	2
term text	天安	门
term type	ns	ns
source start,end	0,2	2,3

然后可以通过 post.jar 把数据放到 Solr 存储库。post.jar 读取的是 xml 文件。如果处理中文，这个 xml 文件必须是 utf-8 编码的，否则就会出现乱码了。下面是一个 xml 文件的样例。

```
<add>
```

```
<doc>
```

```
  <field name="id">126788</field>
```

```
  <field name="title">中文内容测试,TEST</field>
```

```
  <field name="body">上海</field>
```

```
</doc>
```

</add>

在 Tomcat 中配置多个 Solr 的方法是，用 JNDI 的方法配置多个 solr.home。例如下面在 \$CATALINA_HOME/conf/Catalina/localhost 下为每个 solr webapp 创建独立的 context：

```
$ cat /tomcat55/conf/Catalina/localhost/solr1.xml
```

```
<Context docBase="" debug="0" crossContext="true" >
```

```
    <Environment      name="solr/home"      type="java.lang.String"      value="f:/solr1home"
    override="true" />
```

```
</Context>
```

```
$ cat /tomcat55/conf/Catalina/localhost/solr2.xml
```

```
<Context docBase="" debug="0" crossContext="true" >
```

```
    <Environment      name="solr/home"      type="java.lang.String"      value="f:/solr2home"
    override="true" />
```

```
</Context>
```

把每个 solr 实例服务器端配置都放在不同的目录，修改 solrconfig.xml 的索引数据存储路径：

```
<dataDir>${solr.data.dir:./solr1/data}</dataDir>
```

有个小问题是 Solr 的日志在缺省情况下增加的很快，可以通过日志的输出级别来控制日志输出，Solr 中的配置界面是：

<http://localhost/solr/admin/logging.jsp>

8.1.2 把数据放进 Solr

我们通程序，把数据库中的数据转换成符合 Solr 格式的 XML 数据流，然后通过 Http 协议发送出去。

```
StringBuilder xmlContent = new StringBuilder();

xmlContent.append("<add>");

xmlContent.append("<doc>");

XML.writeXML(xmlContent, "field", String.valueOf(rs

    .getInt("AnnounceID")), "name", "id");

XML.writeXML(xmlContent, "field", String.valueOf(rs

    .getInt("BoardID")), "name", "boardID");

//把标题列设置更高的权重

XML.writeXML(xmlContent, "field", rs.getString("Topic"),

    "name", "title", "boost", "20.0");


String body = TextHtml.html2text(rs.getString("Body")).trim();

Lexer lexer = new Lexer(body);

body = HtmlParser.parse(lexer);

XML.writeXML(xmlContent, "field", body, "name", "body");


XML.writeXML(xmlContent, "field", rs.getString("UserName"),

    "name", "username");


XML.writeXML(xmlContent, "field", rs.getString("Child"),

    "name", "child");
```

```
XML.writeXML(xmlContent, "field", rs.getString("hits"), "name",
    "hits");
```

```
//set UTC format
```

```
SimpleDateFormat formatter = new SimpleDateFormat(
    "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
```

```
formatter.setTimeZone( TimeZone.getTimeZone("Z") );
```

```
XML.writeXML(xmlContent, "field", formatter.format(rs
    .getTimestamp ("DateAndTime")), "name", "postdate");
```

```
xmlContent.append("</doc>");
```

```
xmlContent.append("</add>");
```

```
//System.out.println("xmlContent:"+xmlContent);
```

```
StringWriter sw = new StringWriter();
```

```
PostData.postData(new StringReader(xmlContent.toString()), sw,
    solrUrl, POST_ENCODING);
```

数据更新以后，如果要在索引库即刻看到更新的数据，需要提交更新：

```
public static void commit(URL solrUrl) {
    StringBuilder xmlContent = new StringBuilder();
    xmlContent.append("<commit waitFlush=\"true\"
waitSearcher=\"true\" />");

    StringWriter sw = new StringWriter();
    try {
        PostData.postData(new StringReader(xmlContent.toString()),
sw,
```



```

        solrUrl, POST_ENCODING);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

8.1.3 删除数据

可以通过查询的方式删除 Solr 的索引数据：

```

#curl      http://192.168.10.30:8080/solr/update      --data-binary
"<delete><query>id:314685</query></delete>" -H "Content-type:text/xml"

```

```

#curl      http://localhost/gongkong/update          --data-binary
"<delete><query>id:http:\\\\www.aljoin.com\\news\\2007-10\\200710101
54644.htm</query></delete>" -H "Content-type:text/xml"

```

或者直接通过 Id 列删除：

```

#curl      http://192.168.10.30:8080/solr/update      --data-binary
"<delete><id>314685</id></delete>" -H "Content-type:text/xml"

```

```

#curl      http://localhost/gongkong/update          --data-binary
"<delete><id>http:\\\\www.aljoin.com\\news\\2007-10\\20071010154644.
htm</id></delete>" -H "Content-type:text/xml"

```

#我要删除所有含有“答案”这个关键字的结果应该怎么写 query?

```

curl      http://192.168.10.30:8080/solr/update      --data-binary
"<delete><query>title: 答案 OR body: 答案 </query></delete>" -H
"Content-type:text/xml"

```

```

public static void TestDelete() throws Exception
{

```

```
String url = "http://localhost/solr/";
SolrServer server = new CommonsHttpSolrServer( url );
String q = "source:\"industrial acs\"";
UpdateResponse res = server.deleteByQuery( q );
//server.deleteById("314685");
System.out.println(res.getStatus());
server.commit( true, true );
}
```

8.1.4 客户端搜索界面

Solr 客户端是对 Http 请求的封装。返回的数据格式通常是 XML, 返回 JSON, Python, Ruby 格式的数据也是支持的。SolrJ 是一个易于使用的客户端。下面是一个简单的使用例子。

```
String url = "http://hot.lietu.com:8080/solr/";
SolrServer server = new CommonsHttpSolrServer( url );

SolrQuery query = new SolrQuery("的");

QueryResponse response = server.query( query );

int numFound = response.getResults().getNumFound();

System.out.println("fount:"+numFound);

for(SolrDocument d : response.getResults())
{
    String id = (String)d.getFieldValue("id");
    System.out.println("id:"+id);
    String title = (String)d.getFieldValue("title");
    System.out.println("title:"+title);
}
```

因为 Solr 内部已经有分页控制, 所以实际显示的记录会比搜索到的结果少。下面增加对高亮显示的支持。

```
String url = "http://hot.lietu.com:8080/solr/";
SolrServer server = new CommonsHttpSolrServer( url );

SolrQuery query = new SolrQuery("的");

HighlightingParams hp = query.getHighlightingParams();
hp.addField("body");
```

```

hp.setSimplePre("<font color=red>");
hp.setSimplePost("</font>");
//限制高亮显示词的范围
hp.setHRequireFieldMatch(true);
QueryResponse response = server.query( query );
System.out.println("found:" + response.getResults().getNumFound());

for(SolrDocument d : response.getResults())
{
    String id = (String)d.getFieldValue("id");
    System.out.println("id:" + id);
    String hl =
response.getHighlighting().get(id).get( "body" ).get(0);//取得高亮显示的第一段
    System.out.println("hl:" + hl);
}

```

下面是对“catlid”这一列的分类统计：

```

String url = "http://hot.lietu.com:8080/solr/";
SolrServer server = new CommonsHttpSolrServer( url );

SolrQuery query = new SolrQuery("的");

query.getFacetParams().addField("catlid");

QueryResponse response = server.query( query );
System.out.println("found:" + response.getResults().getNumFound());

List<Count> facetCounts =
response.getFacetField("catlid").getValues();
for(Count c:facetCounts )
{
    System.out.println(c);
}

```

8.1.5 Solr 索引库的查找

通过 solr 的管理界面，可以直接分析索引库。通过什么方式可以查询，假设 cat 列是分类列，我们可以看一下这个索引分了几类。

http://localhost/solr/select/?q=%3A*&rows=0&facet.field=cat&rows=0&f

```
acet=true&&facet.limit=-1
```

下面是部分返回结果:

```
<result name="response" numFound="1937323" start="0"/>
```

```
-
```

```
<lst name="facet_counts">
```

```
<lst name="facet_queries"/>
```

```
-
```

```
<lst name="facet_fields">
```

```
-
```

```
<lst name="cat">
```

```
<int name="产品">395413</int>
```

```
<int name="供求">151571</int>
```

```
<int name="其他">86733</int>
```

```
<int name="厂商">175977</int>
```

```
<int name="合作">75374</int>
```

```
<int name="图书">8066</int>
```

```
<int name="培训">6053</int>
```

```
<int name="展会">10761</int>
```

```
<int name="技术文摘">75885</int>
```

```
<int name="教程">8516</int>
```

```
<int name="新闻">150630</int>
```

```
<int name="方案及应用">14282</int>
```

```
<int name="特价">360621</int>

<int name="研究">139945</int>

<int name="职位">10565</int>

<int name="论文">265031</int>

<int name="销售">1900</int>

</lst>

</lst>

<lst name="facet_dates"/>

</lst>
```

如果要按 url 地址查找一个网站的数据，可以：

[url:http\:\\\\2packaging*](http://localhost:8983/solr/select/?q=video&fl=name+score&qt=standard)

可以通过 qt 参数来调用不同的 RequestHandler 处理查询请求，下面这个调用 StandardRequestHandler 来处理查询。

<http://localhost:8983/solr/select/?q=video&fl=name+score&qt=standard>

有没有办法把最满足条件的排在最前面，不是有很多个关键词吗？把都满足的或者满足得最多的排在最前？

可以这样，如果有三个条件，那么这三个条件都必须满足，如果超过三个只要满足 75% 的条件

mm= 75%

调用 solr 的 DisMaxRequestHandler

<http://localhost/solr/select/?q=test&rows=0&qt=dismax&mm=75%>

先重新配置 solrconfig.xml，修改 dismax 的缺省参数配置，否则有些列名可能无效。

```
<requestHandler name="dismax" class="solr.DisMaxRequestHandler" >

  <lst name="defaults">
```

```
<str name="echoParams">explicit</str>

<float name="tie">0.01</float>

<str name="qf">

    body^0.5 title^1.2

</str>

<str name="pf">

    body^0.2 title^1.5

</str>

<str name="fl">

    title,body

</str>

<str name="mm">

    75%

</str>

<int name="ps">100</int>

<str name="q.alt">*:*</str>

<!-- example highlighter config, enable per-query with hl=true -->

<str name="hl.fl">text features name</str>

<!-- for this field, we want no fragmenting, just highlighting -->

<str name="f.name.hl.fragsize">0</str>

<!-- instructs Solr to return the field itself if no query terms are

    found -->
```

```

<str name="f.name.hl.alternateField">name</str>

<str name="f.text.hl.fragmenter">regex</str> <!-- defined below -->

</lst>

</requestHandler>

```

8.1.6 索引分发

索引从 rsyncd daemon 发布到

8.1.7 Solr 搜索优化

如果要在做索引时把不同的列设置不同的权重：

```

<field name="myBoostedField" boost="7.0">value1</field>

<field name="myBoostedField" boost="8.0">value2</field>

<field name="myBoostedField" boost="4.0">value3</field>

```

发送数据的写法：

```

XML.writeXML(xmlContent, "field", value1, "name",

               " myBoostedField", "boost", "7.0");

```

也可以在搜索的时候动态加权。对于标准的 request handler, 对标题列加权：

```
q=title:superman^2 subject:superman
```

Using the dismax request handler, one can specify boosts on fields in parameters such as qf:

```
q=superman&qf=title^2 subject
```

使用 Function Query 实现日期和相关度混合排序：

```
queryWord += " AND _val_:\"linear( recip(rord(timestamp),1,10000,10000),10000000,0)\";
```

在索引中出现“头疼 药”和“头疼药”这样的内容，其分词都是“头疼”和“药”，前者有搜索结果，后者无搜索结果。这样的原因是 Solr 的查询词解析类使用的是短语匹配的

方式。“头疼药”只按照“短语匹配”搜索是不恰当的，应该是先短语匹配，后按照分词进行垂直搜索。为了实现这样的效果，我们修改 SolrQueryParser 类。

```

    protected Query getFieldQuery(String field, String queryText) throws
ParseException {
    // intercept magic field name of "_" to use as a hook for our
    // own functions.
    if (field.equals("_val_")) {
        return QueryParsing.parseFunction(queryText, schema);
    }

    // default to a normal field query
    //return super.getFieldQuery(field, queryText);

    TokenStream source = this.getAnalyzer().tokenStream(field, new
StringReader(queryText));

    ArrayList<Token> v = new ArrayList<Token>(10);
    Token t;

    while (true)
    {
        try
        {
            t = source.next();
        }
        catch (IOException e)
        {
            t = null;
        }
        if (t == null)
            break;
        v.add(t);
    }
    try
    {
        source.close();
    }
    catch (IOException e)
    {
        // ignore
    }

    if (v.size() == 0)

```



```

        return null;
    else if (v.size() == 1)
        return new TermQuery(new Term(field,
((Token)v.get(0)).termText()));
    else
    {
        PhraseQuery q = new PhraseQuery();
        BooleanQuery b = new BooleanQuery();
        q.setBoost(2048.0f);
        //q.SetSlop(0);
        b.setBoost(0.001f);
        for (int i = 0; i < v.size(); i++)
        {
            Token token = v.get(i);
            //System.out.println("add token:"+token.termText());
            q.add(new Term(field, token.termText()));
            TermQuery tmp = new TermQuery(new Term(field,
token.termText()));

            //if(! token.Type().Equals("n"))
            //{
            tmp.setBoost(0.01f);
            //}
            b.add(tmp, BooleanClause.Occur.MUST);
        }

        BooleanQuery bQuery = new BooleanQuery();
        // combine the queries, neither
        //requiring or prohibiting matches
        bQuery.add(q, BooleanClause.Occur.SHOULD);
        bQuery.add(b, BooleanClause.Occur.SHOULD);

        //System.out.println("query:"+bQuery);
        return bQuery;
    }
}

```

如果需要限制这里的BooleanQuery bQuery的查询范围还可以:

```

if(!stopwords.contains(token.termText())){
    b.add(tmp, BooleanClause.Occur.MUST);
    System.out.println("add tmp:"+tmp);
}

```

在通常的搜索中用户可以按内容相关度排序，或者按照日期逆序排序。

```
queryWord;createtime desc
```

还可以按 sort 参数排序:

```
inStock desc, price asc
```

SolrJ 中按时间列排序的例子:

```
query.addSortField("timestamp", ORDER.desc);
```

经常通过一个选项来切换这两种排序方式。为了更加简化搜索界面,可以综合内容相关度排序和日期排序。

Solr 函数查询来实现这一点。基本的方法是设置时间加权。

A Function query influences the score by a function of a field's numeric value or ordinal.

```
// OrdFieldSource ord(myfield)
```

```
// ReverseOrdFieldSource rord(myfield)
```

```
// LinearFloatFunction on numeric field value linear(myfield,1,2)
```

```
// MaxFloatFunction of LinearFloatFunction on numeric field value or constant
max(linear(myfield,1,2),100)
```

```
// ReciprocalFloatFunction on numeric field value recip(myfield,1,2,3)
_val_:"linear(recip(rord(broadcast_date),1,1000,1000),11,0)"
```

实际使用的例子:

```
+_val_:"linear\(\recip\(\rord\(\timestamp\),1,10000,10000\),10000000,0\)" a
```

8.1.8 Solr 中字词混合索引

在 Lucene 的介绍中已经提到了实现字词混合索引的方法。在此基础上增加 FilterFactory。

```
public class SingleFilterFactory extends BaseTokenFilterFactory {
    public SingleFilter create(TokenStream input) {
        return new SingleFilter(input);
    }
}
```

在 schema.xml 中定义 text 列类型如下：

```
<fieldType name="text" class="solr.TextField" positionIncrementGap="100">

  <analyzer type="index">

    <tokenizer class="CnTokenizerFactory"/>

    <filter class="solr.SingleFilterFactory"/>

  </analyzer>

  <analyzer type="query">

    <tokenizer class="CnTokenizerFactory"/>

    <filter class="solr.SingleFilterFactory"/>

  </analyzer>

</fieldType>
```

修改 org.apache.solr.search 包中的 SolrQueryParser 类：

```
// default to a normal field query
//return super.getFieldQuery(field, queryText);
//change by luogang
TokenStream source = this.getAnalyzer().tokenStream(field, new
StringReader(queryText));

ArrayList<Token> v = new ArrayList<Token>(10);
Token t;

while (true)
{
    try
    {
        t = source.next();
    }
    catch (IOException e)
    {
        t = null;
    }
}
```

```
    }
    if (t == null)
        break;
    v.add(t);
}
try
{
    source.close();
}
catch (IOException e)
{
    // ignore
}

if (v.size() == 0)
    return null;
else if (v.size() == 1)
    return new TermQuery(new Term(field,
((Token)v.get(0)).termText()));
else
{
    /*PhraseQuery q = new PhraseQuery();
    BooleanQuery b = new BooleanQuery();
    q.setBoost(2048.0f);
    //q.SetSlop(0);
    b.setBoost(0.001f);
    for (int i = 0; i < v.size(); i++)
    {
        Token token = v.get(i);
        //System.out.println("add token:"+token.termText());
        if(token.getPositionIncrement()>0)
        {
            q.add(new Term(field, token.termText()));
        }
        if(token.termText().length()==1)
        {
            TermQuery tmp = new TermQuery(new Term(field,
token.termText()));
            tmp.setBoost(0.01f);
            b.add(tmp, BooleanClause.Occur.MUST);
        }
    }

    BooleanQuery bQuery = new BooleanQuery();
```

```

        // combine the queries, neither
        //requiring or prohibiting matches
        bQuery.add(q, BooleanClause.Occur.SHOULD);
        bQuery.add(b, BooleanClause.Occur.SHOULD);

        */

        PhraseQuery q = new PhraseQuery();
        q.setBoost(2048.0f);

        ArrayList<SpanQuery> s = new ArrayList<SpanQuery>(v.size());

        for (int i = 0; i < v.size(); i++)
        {
            Token token = v.get(i);
            if(token.getPositionIncrement()>0)
            {
                q.add(new Term(field, token.termText()));
            }
            if(token.termText().length()==1)
            {
                SpanTermQuery tmp = new SpanTermQuery (new Term(field,
token.termText()));
                s.add(tmp);
            }
        }

        BooleanQuery bQuery = new BooleanQuery();
        // combine the queries, neither
        //requiring or prohibiting matches
        bQuery.add(q, BooleanClause.Occur.SHOULD);
        if(s.size()>0)
        {
            SpanNearQuery nearQuery = new SpanNearQuery(s.toArray(new
SpanQuery[s.size()]),s.size(),true);
            nearQuery.setBoost(0.001f);
            bQuery.add(nearQuery, BooleanClause.Occur.SHOULD);
        }

        //System.out.println("query:"+bQuery);
        return bQuery;
    }

    //end change

```

8.1.9 相关检索

经常需要实现相似检索的功能。例如输入一篇文章，返回相关的 5 篇文章。Solr 中包括了一个 MoreLikeThis 的处理模块。在 solrconfig.xml 中包括：

```
<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">

  <lst name="defaults">

    <str name="mlt.fl">title,abstract</str>

    <int name="mlt.mindf">1</int>

  </lst>

</requestHandler>
```

mlt.fl 中包括提取关键词的缺省列。

搜索的时候输入类似：

<http://hot.lietu.com:8080/solr/select/?q=%E6%B1%BD%E8%BD%A6&version=2.2&start=0&rows=1&mlt=true&mlt.mindf=1&mlt.mintf=1&mlt.fl=title,abstract>

返回的结果后面包括了相似匹配的结果：

```
<lst name="moreLikeThis">
-
  <result name="521838" numFound="6586103" start="0">
-
    <doc>
-
      <str name="abstract">
```

本文通过分析世界汽车工业的发展趋势，总结出 21 世纪世界汽车工业发展的八大趋势，主要表现在汽车产业组织、汽车生产方式、汽车产品及造型和汽车可持续发展战略上。

```
</str>

<str name="cn_institution">武汉理工大学</str>

<str name="en_abstract"/>

-

    <str name="en_title">

Eight Developmont Trends of The World Automobile Industry in The 21st Century

</str>

<str name="id">5141078</str>

<str name="pin_yin_name">杨瑞海,韩雄辉</str>

<str name="title">21 世纪世界汽车工业发展的八大趋势</str>

<str name="user_real_name">杨瑞海,韩雄辉</str>

</doc>

...

</result>

</lst>
```

这个缺省的相关检索仍然有待改进，因为 `MoreLikeThis` 查询的准确性依赖于提取关键词的准确性，为了提取关键词，首先要做的工作是去掉 `StopWord`。`MoreLikeThis` 类缺省使用的是英文的 `StopWord`。下面我们修改 `MoreLikeThisHandler` 让它从外部读取 `StopWord.txt`。

```
<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">
  <lst name="defaults">
    <str name="mlt.fl">manu,cat</str>
    <int name="mlt.mindf">1</int>
  </lst>
  <str name="stopWordFile">stopwords.txt</str>
</requestHandler>
```

`MoreLikeThisHandler` 类本身加上：

```

private static Set stopWords = null;

public void init(NamedList args) {
    super.init(args);
    SolrParams p = SolrParams.toSolrParams(args);
    String stopWordFile = p.get("stopWordFile");
    if(stopWordFile == null)
    {
        stopWords = StopFilter.makeStopSet(StandardAnalyzer.STOP_WORDS);
        return;
    }

    List<String> wlist;
    try {
        wlist = Config.getLines(stopWordFile);
    } catch (IOException e) {
        throw new SolrException( SolrException.ErrorCode.NOT_FOUND,
            "MoreLikeThis requires a stop word list " );
    }
    stopWords = StopFilter.makeStopSet((String[])wlist.toArray(new
String[0]), true);
}

...

    mlt.setStopWords(stopWords);

...

```

8.1.10 搜索结果去重

折叠对于一个给定列的相同或相似值的搜索结果叫做 *collapsing*。例如对同一站点搜索结果的折叠，经常也会在这个搜索结果上加上“来源于该站点的更多结果”。

AHC2001 No. CP-PC-1377 高性能・高附加值...	2007-10-10 7:51:15
[山武株式会社上海代表处]	
,還有可與小型PLC匹配的邏輯控制功能。可處理多達256點的數字量輸入/輸出,可組成象PLC那樣的在模擬量控制中不可缺少的聯鎖信號的順控邏輯。邏輯控制功能可與小型PLC匹配提供了與PC裝載器連	
山武株式会社上海代表处 中还有8条相关信息>>	
Hamburg汉堡车辙试验机成型机	2007-9-19 13:35:42
[欧美大地仪器设备中国]	
驱动的 一次成型2个板块状试件提高了效率缩短了清洁时间减少了试件质量和温度的偏差 成型机可以编程设定用户指定的不同行程长度 采用了PLC和一台线性传感器控制装有定量阀的液压缸来保证行程的精度 用户可以	
欧美大地仪器设备中国 中还有3条相关信息>>	
一位参展人员的日记	2007-9-14 15:41:40
[正泰集团]	
自动化产品后,向我询问有关设备配套的问题,特别是对PLC产品兴趣非常大。经过了解,原来他们公司一直是使用台湾产的PLC产品,但由于售后服务跟不上,给他们成套设备销售带来了障碍。在详细了解了我们的产品后,	
正泰集团 中还有3条相关信息>>	

这样需要我们给 Solr 增加 collapsing 的功能。首先我们假定折叠的列是未分词的。

<https://issues.apache.org/jira/browse/SOLR-236> 正是关于这个问题的解决方法。先取得这个版本:

```
# svn export -r592129 http://svn.apache.org/repos/asf/lucene/solr/
```

```
# patch -u -p0 < field-collapsing-extended-592129.patch
```

搜索测试:

http://localhost:8080/select/?q=words_t%3AApple&version=2.2&start=0&rows=10&indent=on&collapse.field=t_s&collapse.threshold=1

返回的结果是:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<response>
```

```
<lst name="responseHeader">
```

```
<int name="status">0</int>
```

```
<int name="QTime">0</int>
```

```
<lst name="params">
```

```
<str name="start">0</str>

<str name="collapse.max">1</str>

<str name="indent">on</str>

<str name="q">words_t:Apple</str>

<str name="version">2.2</str>

<str name="rows">10</str>

<str name="collapse.field">t_s</str>

</lst>

</lst>

<result name="response" numFound="2" start="0">

<doc>

  <int name="c_i">3</int>

  <str name="id">1</str>

  <int name="popularity">0</int>

  <str name="sku">1</str>

  <str name="t_s">movie</str>

  <date name="timestamp">2007-12-21T15:22:42.211Z</date>

  <str name="words_t">Apple Orange</str>

</doc>

<doc>

  <int name="c_i">4</int>

  <str name="id">3</str>
```

```
<int name="popularity">0</int>

<str name="sku">3</str>

<str name="t_s">book</str>

<date name="timestamp">2007-12-22T02:01:53.328Z</date>

<str name="words_t">Apple Orange</str>

</doc>

</result>

<lst name="collapse_counts">

  <str name="field">t_s</str>

  <lst name="doc">

    <int name="1">1</int>

  </lst>

  <lst name="count">

    <int name="movie">1</int>

  </lst>

  <str name="debug">HashDocSet(1) Time(ms): 0/0/0/0</str>

</lst>

</response>
```

这样的返回结果表示：对于列“t_s”，还有一个“t_s”列的值是 movie 的。

主要参数介绍：

- collapse.facet = before|after 参数控制 faceting 是在 collapsing 前或后发生。
- collapse.threshold 参数控制在多少重复后开始折叠结果。

- `collapse.maxdocs` 参数控制折叠执行时最多考虑的文档数量。当结果集很大的时候，增加此参数能缩短执行时间。
- `collapse.info.doc` 和 `collapse.info.count` 参数控制 `collapse_counts` 返回结果内容。

实际应用场景设想，一个 `solr` 索引库包含很多新闻故事，这些故事来源于许多报纸或有限电视。

一条新闻故事可能来源于多个不同的报纸，例如《人民日报》或一些地方小报等。每个报纸对同一个新闻起上不同的标题，并截成不同的长度。

需要检测并把这些重复的故事分组在一起显示。假设每个故事都由一个 `Hash` 整数代表，例如叫“`similarity_hash`”，开始字段的头 `X` 个单词。这个值索引和存储起来作为检测重复故事的方法。

我们可以基于这个“`similarity_hash`”来折叠搜索结果，这样同一个故事的多次出现就折叠到了一起。

而且，用户可能更愿意读到更加权威的版本，这个结果优先显示到搜索结果中，当然也会有个重复新闻的计数和连接。权威性可能是 1) 有限电视新闻网，2) 国家级报纸，3) 地区报纸，4) 地方小报，可以索引和存储成一个整数“`authority`”。

然后，可以显示给用户：

"人咬狗"

**日报, 连接可见其它 77 个重复新闻

通过折叠“`similarity_hash`”列实现这个功能，选择基于另外一列“`authority`”的值返回显示的新闻中的一条。

这样我们需要进一步修改这个折叠列的实现，增加一个参数：

`collapse.authority=[field]` //索引列，用来控制折叠后的组中返回哪一个值

以前 `CollapseFilter` 只对一列排序，然后在排序后的结果中发现重复，实现折叠功能。现在改为实现多个列排序，把 `collapse.authority` 也加到排序列中。我们修改它的主要实现 `src/java/org/apache/solr/search/CollapseFilter.java` 文件：

```
if (collapseType == CollapseType.NORMAL)
{
    if(sort!=null)
    {
        SortField[] ofields = sort.getSort();
        SortField[] nfields = new SortField[ofields.length+1];
```

```

    for (int k=0;k<ofields.length;++k)
    {
        nfields[k] = ofields[k];
    }
    nfields[nfields.length-1] = new SortField(collapseField);
    sort.setSort(nfields);
}
else
{
    sort = new Sort(new SortField(collapseField));
}
}

```

8.1.11 分布式搜索

当一个索引的大小超过一个机器的处理能力的时候，就需要用到分布式索引了。Solr 上实现分布式搜索有几种方式，有的采用 RMI 远程过程调用，有的直接用 HTTP 来实现。

分布式搜索要达到如下目标：

客户端应用，例如 SolrJ 对分布式搜索是完全不可知的。分布式搜索的处理和结果合并都在请求 handler 内部处理了。在结果合并以后，保持响应返回的结果格式不变。

从不同的 shard 返回的响应反序列化到了 SolrQueryResponse 对象。SolrQueryResponse 对象合并成一个单独的 SolrQueryResponse 对象。这样能够基本保持 结果输出类不变。

高效的查询处理和高亮处理仅从合并后的文档生成。分两个阶段执行查询。第一阶段，使用排序条件得到唯一的文档关键字。第二阶段得到所有的请求列的信息和高亮信息。当有许多 shard 和高亮信息请求时，这样能节省 CPU。

应该容易配置查询执行。例如，用户可以声明只执行查询的第一阶段，这样当高亮信息不需要和列的数量不需要的时候会更有效。

能够容易的通过合适的插件和请求参数重写缺省的分布式搜索能力。由于分布式搜索通过请求 Handler 执行。多个请求 handler 可以容易的用 solrconfig.xml 中不同的分布式搜索预配置。

全局权重计算可以通过从所有的 shard 查询词的文档频率实现。

分布式搜索通过 Http 网络协议实现，因此可以查询 shard 的虚拟 IP 地址（VIP）。负载均衡和容错可以通过 VIP 来实现。

解析子搜索响应可以通过一个插件接口实现。可以基于 JSON，xml SAX 做出不同的实

现。当前的实现是基于 XML DOM。

具体的实现方法是：

一个新的叫做 `MultiSearchRequestHandler` 的 `RequestHandler` 执行对多个子搜索的分布式搜索（子搜索服务叫做"shard"）。`handleRequestBody` 方法被分成查询构建和执行两个方法。

为了增加分布式搜索功能，所有的搜索请求 handler 都扩展这个 `MultiSearchRequestHandler`。标准的 `StandardRequestHandler` 和 `DisMaxRequestHandler` 改成扩展这个类。

如果"shards"出现在请求参数中，分布式搜索就开始起作用了，否则搜索的是本地索引库。例如，`shards=local,host1:port1,host2:port2` 会搜索本地索引和两个远程索引。从三个 shard 返回的结果合并后返回给客户端。

在 shard 集合上的搜索请求按如下流程处理：

第一步：构建查询，提取搜索词。通过请求所有的 shard 并求和计算全局的文档数量和文档频率。

第二步：(FirstQueryPhase)查询所有的 shard。把全局的文档数量和文档频率作为参数传递。所有的文档列都不请求，仅仅请求文档唯一列和排序列。`MoreLikeThis` 和高亮信息也不请求。

第三步：基于 "sort", "start" 和 "rows" 参数合并从 `FirstQueryPhase` 返回的请求。收集合并的文档唯一列和排序列。同时也合并其他的信息例如 facet 和调试信息。

第四步：(SecondQueryPhase) 合并的文档唯一列和排序列按 shard 分组。向在组中的所有 shard 查询合并的文档唯一列(来自 `FirstQueryPhase`)，高亮信息和 `MoreLikeThis` 信息。

第五步：合并从所有的 shard 的 `SecondQueryPhase` 响应。

第六步：从 `SecondQueryPhase` 得到的文档列，高亮和 `moreLikeThis` 信息合并进 `FirstQueryPhase` 响应。

下面我们实际应用这个 <https://issues.apache.org/jira/browse/SOLR-303> 的分布式搜索实现。

```
# svn export -r574785 http://svn.apache.org/repos/asf/lucene/solr/
```

```
# patch -u -p0 < fedsearch.stu.patch
```

应用这个 patch 之后，`solrconfig.xml` 会增加：

```
<requestHandler name="/federated/collectionstats"
```

```
class="solr.federated.component.GlobalCollectionStatComponent$CollectionStatsHandle  
r"/>
```

我们可以通过下面的 URL 地址访问多个 Solr 实例:

```
http://localhost/select/?q=%E5%8C%BB%E7%94%9F&version=2.2&start=1000&  
rows=20&indent=on&shards=local,localhost:8080
```

返回的结果类似:

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<response>  
  
  <lst name="responseHeader">  
  
    <int name="status">0</int>  
  
    <int name="QTime">188</int>  
  
    <lst name="params">  
  
      <str name="shards">local,localhost:8080</str>  
  
      <str name="indent">on</str>  
  
      <str name="start">1019</str>  
  
      <str name="q">医生</str>  
  
      <str name="rows">2</str>  
  
      <str name="version">2.2</str>  
  
    </lst>  
  
  </lst>  
  
  <result name="response" numFound="28799" start="0">  
  
    <doc>  
  
      <str name="body"> 我 1 9 岁，皮肤很脆弱，太阳一晒就通红，运动完了更红，尤其是在
```

害羞的时候，更更红了。别人为什么没事啊，我的皮肤是不是不正常，还有冬天时候在很热的屋子里也红，为什么啊，我该怎么办，很苦恼的，医生帮帮我，好吗。我听说有光子收缩毛细血管的治疗，这个有效果吗，能治本吗。如果医生有更好的建议，请告诉我 </str>

```
<str name="cat">患者</str>

<str name="id">5720</str>

<str name="isread">0</str>

<str name="postdate">2007-08-14T00:00:00.000</str>

<int name="rank">4</int>

<str name="source">问医生</str>

<date name="timestamp">2005-07-17T15:58:18.682Z</date>

<str name="title">我的皮肤很红，为什么啊</str>

<str
name="url">http://www.wenyisheng.com/ask/askview.asp?id=50929</str>

</doc>

</result>

<lst name="responseHeader">

<lst name="localhost:8080">

<int name="status">0</int>

<int name="QTime">47</int>

<lst name="params">

<str name="fl">id,score,</str>

<str name="fsv">true</str>

<str name="q">医生</str>

<str name="nd">152443</str>
```



```
<str name="tdf">body:医生@29127,</str>

<str name="version">2.2</str>

<str name="rows">1021</str>

</lst>

</lst>

</lst>

<lst name="responseHeader">

  <lst name="localhost:8080">

    <int name="status">0</int>

    <int name="QTime">0</int>

    <lst name="params">

      <str name="start">0</str>

      <str name="dq">id:"5720" </str>

      <str name="q">医生</str>

      <str name="version">2.2</str>

      <str name="rows">2</str>

    </lst>

  </lst>

</lst>

</response>
```

8.1.12 SolrJ 查询分析器

为了支持象 AND（与）、OR（或）、 NOT（非） 这样的高级查询语法。Lucene 使用

JavaCC 生成的 `QueryParser` 类实现用户查询串的解析。`SolrJ` 自身没有这样的实现，下面实现一个和 `Lucene` 查询语法兼容的查询语法解析器。

查询分析一般用两步实现，词法分析和语法分析。

词法分析的功能是从左到右扫描用户输入查询串，从而识别出标识符、保留字、整数、浮点数、算符、界符等单词符号，把识别结果返回到语法分析器，以供语法分析器使用。这一部分的输入是用户查询串，输出是单词符号串的识别结果。例如，对如下的输入片断：

title:car site:http://www.sina.com

词法分析的输出可能是：

TREM title

COLON :

TREM car

TREM site

COLON :

TREM http://www.sina.com

词法分析可以采用 `JFlex` 这样的工具生成，但因为查询词法比较简单，这里采用手工实现一个词法分析。语法分析采用 `YACC` 的 Java 版本 `BYACC/J`。定义 `YACC` 推导的返回类型 `Query`。

```
public interface Query {
    public String getQueryType(); //取得查询类型,对应Solr的Request Handler
    public String getQuery(); //取得查询串
}
```

修改 `ParserVal` 的 `obj` 类型定义。

```
public class ParserVal
{
    ...
    /**
     * object value of this 'union'
     */
    public Query obj;
    ...
}
```

定义 Token 的类型有如下几种:

```
%token AND OR NOT PLUS MINUS LPAREN RPAREN COLON TREM SKIP
RANGEIN_START RANGEIN_TO RANGEIN_END
```

下面是词法分析器的实现:

```
public class Yylex {
    private Parser yyparser; //解析器
    private String buffer; //查询串缓存
    private int tokPos = 0; //Token的当前位置
    private int tokLen = 0; //Token的长度

    /**
     * 词法分析器的构造函数
     *
     * @param r 输入查询串
     * @param yyparser 解析器对象
     */
    public Yylex(String r, Parser yyparser) {
        buffer = r;
        this.yyparser = yyparser;
    }

    /**
     * Resumes scanning until the next regular expression is matched,
     * return 0 if end
     *
     * @return the next token
     */
    public int yylex() {
        tokPos += tokLen;
        if (tokPos >= buffer.length())
        {
            //输入串解析结束
            return 0;
        }

        char ch = buffer.charAt(tokPos);
        switch (ch)
        {
            case '+':
                tokLen = 1;
                return Parser.PLUS;
            case '-':
```

```

        tokLen = 1;
        return Parser.MINUS;
    case '(':
    case ' (':
        tokLen = 1;
        return Parser.LPAREN;
    case ')':
    case ') ':
        tokLen = 1;
        return Parser.RPAREN;
    case '[':
        tokLen = 1;
        return Parser.RANGEIN_START;
    case ']':
        tokLen = 1;
        return Parser.RANGEIN_END;
    case ':':
    case ': ':
        tokLen = 1;
        return Parser.COLON;
    case '|':
        if (tokPos + 1 < buffer.length() && buffer.charAt(tokPos +
1) == '|')
        {
            tokLen = 2;
            return Parser.OR;
        }
        tokLen = 1;
        return Parser.TREM;

    case '&':
        if (tokPos + 1 < buffer.length() && buffer.charAt(tokPos +
1) == '&')
        {
            tokLen = 2;
            return Parser.AND;
        }
        tokLen = 1;
        return Parser.TREM;
    case ' ':
    case '\\t':
    case ' ':
        tokLen = 1;

```

```
        return yylex();
    case ' ':
        tokLen = 1;
        while (tokPos + tokLen < buffer.length())
        {
            char chTerm = buffer.charAt(tokPos + tokLen);
            if (chTerm != ' ')
            {
                tokLen++;
            }
            else
            {
                tokLen++;
                break;
            }
        }
        yyparser.yylval = new ParserVal(buffer.substring(tokPos,
tokPos+tokLen));

        return Parser.TREM;
    default:
        tokLen = 1;
        while (tokPos + tokLen < buffer.length())
        {
            char chTerm = buffer.charAt(tokPos + tokLen);
            if (chTerm != ' ' &&
                chTerm != '\\t' &&
                chTerm != ' ' &&
                chTerm != '(' &&
                chTerm != ')' &&
                chTerm != ')' &&
                chTerm != '(' &&
                chTerm != ';' &&
                chTerm != ']')
            {
                if( chTerm == ':' )
                {
                    if("http".equals(buffer.substring(tokPos,
tokPos+tokLen)))
                    {
                        tokLen++;
                    }
                    else
                    {
```

```

        break;
    }
}
else
{
    tokLen++;
}
}
else
{
    break;
}
}

String cur = buffer.substring(tokPos, tokPos+tokLen);
if (cur.equals("AND"))
{
    return Parser.AND;
}
if (cur.equals("OR"))
{
    return Parser.OR;
}
if (cur.equals("TO"))
{
    return Parser.RANGEIN_TO;
}
yyvsparser.yylval = new ParserVal(cur);
return Parser.TREM;
}
}
}

```

可以简单的测试一下这个词法分析程序，看是否能返回正确的 Token 类型：

```

public static void main(String[] args) {
    String i = "title:car link:http://www.sina.com";
    Parser yyparser = new Parser();
    Yylex lexer = new Yylex(i, yyparser);

    int type = 1;
    while (type!=0)
    {
        type = lexer.yylex();
    }
}

```

```
String result = String.valueOf(type);
if (yyvsparser!=null && yyparser.yy1val!=null )
{
    result+= " "+ yyparser.yy1val.sval;
}
System.out.println(result);
}
}
```

Yacc 语法文件如下:

```
%start querystr
```

```
%token AND OR NOT PLUS MINUS LPAREN RPAREN COLON TREM SKIP RANGEIN_START
RANGEIN_TO RANGEIN_END
```

```
%left OR
```

```
%left AND
```

```
%left NOT
```

```
%left PLUS
```

```
%left MINUS
```

```
%%
```

```
querystr    : query { $$ = $1; }
```

```
;
```

```
query      : /*empty */
```

```
| query clause {  
  
    if ($1 == null)  
  
        {  
  
            $$ = $2;  
  
        }  
  
    else  
  
    {  
  
        BooleanQuery bq = new BooleanQuery();  
  
  
        bq.Add($1.obj, BooleanClause.Occur.SHOULD);  
  
        if ($2.obj instanceof BooleanQuery)  
  
        {  
  
            BooleanQuery clause = (BooleanQuery)$2.obj;  
  
            if (clause.plus)  
  
            {  
  
                bq.Add($2.obj,  
BooleanClause.Occur.MUST);  
  
            }  
  
            else if (clause.minus )  
  
            {  
  
                bq.Add($2.obj,  
BooleanClause.Occur.MUST_NOT);  
  
            }  
  
        }  
  
    }  
  
}
```



```
        else

            {

                bq.Add($2.obj,
BooleanClause.Occur.SHOULD);

            }

        }

        else

        {

            bq.Add($2.obj, BooleanClause.Occur.SHOULD);

        }

        $$ = new ParserVal(bq);

    }

}

;

clause    : clause OR clause {

    BooleanQuery bq = new BooleanQuery();

    bq.Add($1.obj, BooleanClause.Occur.SHOULD);

    bq.Add($3.obj, BooleanClause.Occur.SHOULD);

    $$ = new ParserVal( bq );

}

| clause AND clause {

    BooleanQuery bq = new BooleanQuery();

    bq.Add($1.obj, BooleanClause.Occur.MUST);
```

```
        bq.Add($3.obj, BooleanClause.Occur.MUST);

        $$ = new ParserVal( bq );

    }

    | LPAREN clause RPAREN {

        $$ = new ParserVal( $2.obj );

    }

    | PLUS clause {

        BooleanQuery bq = new BooleanQuery();

        bq.Add($2.obj, BooleanClause.Occur.SHOULD);

        bq.plus = true;

        $$ = new ParserVal( bq );

    }

    | MINUS clause {

        BooleanQuery bq = new BooleanQuery();

        bq.Add($2.obj, BooleanClause.Occur.SHOULD);

        bq.minus = true;

        $$ = new ParserVal( bq );

    }

    | NOT clause {

        BooleanQuery bq = new BooleanQuery();

        bq.Add($2.obj, BooleanClause.Occur.SHOULD);

        bq.minus = true;
```

```
        $$ = new ParserVal( bq );

    }

    |  TREM  {

        String termStr = $1.sval;

        BooleanQuery bq = new BooleanQuery();

        for (int i = 0; i < defaultSearchFields.length; ++i)

        {

            bq.Add(new TermQuery(defaultSearchFields[i],
termStr), BooleanClause.Occur.SHOULD);

        }

        $$ = new ParserVal(bq);

    }

    |  TREM COLON TREM  {

        String fieldStr = $1.sval;

        String termStr = $3.sval;

        $$ = new ParserVal( new TermQuery(fieldStr,
termStr) );

    }

    |  TREM COLON RANGEIN_START TREM RANGEIN_TO TREM RANGEIN_END {

        String fieldStr = $1.sval;

        String fromStr = $4.sval;

        String toStr = $6.sval;

        $$ = new ParserVal( new RangeQuery(fieldStr,
fromStr,toStr) );

    }
```

```
}
```

```
;
```

```
%%
```

```
private Yylex lexer;
```

```
public String[] defaultSearchFields = {"title","body"};
```

```
private int yylex () {
```

```
    int yyl_return = lexer.yylex();
```

```
    return yyl_return;
```

```
}
```

```
public void yyerror (String error) {
```

```
    System.err.println ("Error: " + error);
```

```
}
```

```
public Parser(String i) {
```

```
    lexer = new Yylex(i, this);
```

```
}

public static void main(String args[])

{

    String input = "title:中国 OR 北京";

    Parser parser = new Parser(input);

    parser.yyparse();

    System.out.println(parser.val_peek(0).obj.ToString());

}
```

用 BYACC 执行它：

```
yacc -J queryparser.y
```

8.1.13 扩展 SolrJ

SolrJ 通过请求类和响应类提供了方便的接口用来扩展。例如实现从 URL 地址：

<http://localhost/admin/stats.jsp#core>

读取索引状态。首先定义请求类：

```
public CoreRequest()
{
    super( METHOD.GET, "/admin/stats.jsp#core" );
}
```

然后定义响应类：

```
public CoreResponse(NamedList<Object> res) {
```

```

    super(res);

    indexInfo = res;
}

```

以及返回 XML 格式内容的解析类 CoreResponseParser:

```

protected NamedList<Object> readNamedList( XMLStreamReader parser )
throws XMLStreamException
{
    NamedList<Object> nl = new NamedList<Object>();

    int status;
    while( parser.hasNext() )
    {
        status = parser.next();
        if (status==XMLStreamConstants.START_ELEMENT)
        {
            String n = parser.getLocalName();
            if ("stat".equals(n))
            {
                //取得属性名
                n = parser.getAttributeValue(0);
                parser.next();
                //取得属性值
                String v = parser.getText().trim();
                nl.add(n, v);
                //System.out.println(n+"="+v);
            }
        }
    }
    return nl;
}

```

8.1.14 扩展 Solr

Solr 本身是 REST 风格的, 它通过 Servlet 响应用户请求。以 more like this handler 为例, 在 solrconfig.xml 中包含:

```

<requestHandler name="/mlt" class="solr.MoreLikeThisHandler">

    <lst name="defaults">

```

```
<str name="mlt.fl">manu,cat</str>
```

```
<int name="mlt.mindf">1</int>
```

```
</lst>
```

```
</requestHandler>
```

它对应的请求方法：

<http://localhost:8983/solr/mlt?q=id:UTF8TEST&mlt.fl=manu,cat&mlt.mindf=1&mlt.mintf=1>

编写 solr 的 handler 的方法：

1. 将 solr 源文件解压，并且使用开发工具（如：Eclipse），新建工程，将源文件以及相关的 jar 导入。
2. 在 org.apache.solr.handler 包下（一般在此包下进行扩展），新建 java 类。
3. 新建的类需要继承 RequestHandlerBase 类，并且实现其中的 handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp)方法。同时需要覆盖 getVersion()、getDescription()、getSourceId()、getSource()、getDocs()等方法。
4. handleRequestBody(SolrQueryRequest req, SolrQueryResponse rsp)方法中，req 表示传入的参数对象，rsp 表示经过处理后得到的需要显示的对象。
5. 业务处理根据个人的需要进行编写。
6. 在 solr 中，常量一般在 org.apache.solr.common.params 包下的接口 CommonParams 中定义。在本项目中，需要在 CommonParams 中添加新的常量，如：
7. public static final String URI = "uri";//URI 的值表示访问的参数。
8. 以上工作实现后，对 solr 重新打包。然后加入到 Web 项目目录\WEB-INF\lib 中。
9. 打开 solr 文件夹（对应 resin 文件夹下的 solr 文件夹），打开 conf 文件夹下的 solrconfig.xml 文件，在<config>.....</config>标签中，添加如下内容：
10. <requestHandler name="/urlinfo" class="solr.URIToWordsHandler">
11. </requestHandler>
12. 其中 name 属性决定了访问的路径，class 属性决定了处理类。

13. 打开 IE 浏览器，输入 “http://localhost:8081/index/urlinfo?url=? wt=? ”，其中 “localhost:8081/index/” 表示该 WEB 应用程序的访问地址，“urlinfo” 对应第 8 条中的<requestHandler>标签中的 name 属性的值。url 对应传入的查询网页地址（对应第 6 条中添加的常量的值（uri））。wt 表示要输出的格式，值一般为：xml 和 json，表示输出的格式为 xml 格式或者 json 格式。

以下通过一个例子具体说明：

1. 在合适的包下编写所需要的 java 类，本例中类名为 URIToWordsHandler，该类继承 RequestHandlerBase 类。

```
import java.net.MalformedURLException;

import java.net.URL;

import java.util.ArrayList;

import java.util.List;


import org.apache.solr.common.params.CommonParams;

import org.apache.solr.common.params.SolrParams;

import org.apache.solr.common.util.NamedList;

import org.apache.solr.common.util.SimpleOrderedMap;

import org.apache.solr.request.SolrQueryRequest;

import org.apache.solr.request.SolrQueryResponse;


/**

 * Solr MoreLikeThis --

 *

 * Return similar documents either based on a single document or based
on posted
```



```
* text.

*

* @since solr 1.3

*/

public class URIToWordsHandler extends RequestHandlerBase {

    public void handleRequestBody(SolrQueryRequest req,
        SolrQueryResponse rsp)

        throws Exception {

        //      RequestHandlerUtils.addExperimentalFormatWarning(rsp);

        //

        /**

        * 获取传递的参数集合

        */

        SolrParams params = req.getParams();

        /**

        * 从参数集中得到想要的参数

        */

        String uri = params.get(CommonParams.URI);

        /**

        * 进行业务处理

        */
```

```
ArrayList<String> s = new ArrayList<String>();

s.add("大家新年好");

s.add("今日新闻： 火车票大战");

s.add("生活");

Words words = new Words(s, "娱乐");


String type = words.getType();

NamedList<Object> word = new SimpleOrderedMap<Object>();

for (String o : words.getWord()) {

    word.add("word", o);

}


/**

 * 将业务处理结果添加到rsp, 并且输出

 */

rsp.add("type", type);

rsp.add("words", word);

}


class Words {

    private List<String> word;

    private String type;
```

```
public Words(List<String> word, String type) {  
  
    super();  
  
    this.word = word;  
  
    this.type = type;  
  
}
```

```
public List<String> getWord() {  
  
    return word;  
  
}
```

```
public void setWord(List<String> word) {  
  
    this.word = word;  
  
}
```

```
public String getType() {  
  
    return type;  
  
}
```

```
public void setType(String type) {  
  
    this.type = type;  
  
}
```

```
}

//////////////////////////////// SolrInfoMBeans methods //////////////////////////////////

////////////////////////////////以下覆写父类的方法 //////////////////////////////////

@Override

public String getVersion() {

    return "$Revision: 561904 $";

}

@Override

public String getDescription() {

    return "Solr MoreLikeThis";

}

@Override

public String getSourceId() {

    return "$Id: MoreLikeThisHandler.java 561904 2007-08-01 18:43:02Z  
ryan $";

}

@Override

public String getSource() {
```

```

    return "$URL:
http://svn.apache.org/repos/asf/lucene/solr/trunk/src/java/org/ap
ache/solr/handler/MoreLikeThisHandler.java $";
}

@Override

public URL[] getDocs() {

    try {

        return new URL[] { new URL(

            "http://wiki.apache.org/solr/MoreLikeThis") };

    } catch (MalformedURLException ex) {

        return null;

    }

}

}

```

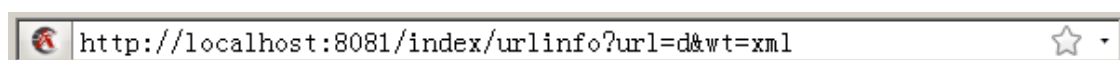
2. 在 solrconfig.xml 中<config>标签下添加相应的配置信息:

```

<!-- zhouwendong 2009-1-7 START!!! 注释, 便于后期修改 -->
<requestHandler name="/urlinfo" class="solr.URItoWordsHandler">
    <!-- /urlinfo 表示访问路径 class 表示对应的处理类 -->
    <!-- 以下为其他说明, 可根据需要修改或删除, 此信息将会在网页显示 -->
    <lst name="其他说明">
        <int name="名称">参数</int>
    </lst>
</requestHandler>
<!-- zhouwendong 2009-1-7 END!!! -->

```

3. 启动服务器, 在浏览器里查看结果:

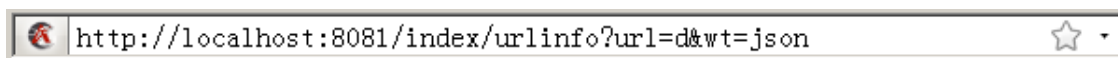


上图为 IE 地址栏。其中, urlinfo 对应 b 项中提到的访问路径, url 表示需要传递的参数, wt 表示返回的结果格式。下图为 wt=xml 时的显示结果:

```

- <response>
  - <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">16</int>
  </lst>
  <str name="type">娱乐</str>
  - <lst name="words">
    <str name="word">大家新年好</str>
    <str name="word">今日新闻：火车票大战</str>
    <str name="word">生活</str>
  </lst>
</response>

```



下图为 wt=json 时的显示结果：

```

{"responseHeader":{"status":0,"QTime":0},"type":"娱乐","words":{"word":"大家新年好","word":"今日新闻：火车票大战","word":"生活"}}

```

8.1.15 Solr 的 .net 客户端

SolrSharp 是 Solr 的客户端 .net 版本。SolrSharp 中存在一些错误，我们需要修改它。比如查询多个词的实现存在问题。SolrSharp 也缺少查找相似文档的接口和折叠搜索结果的接口。在 QueryParameter 类中，错误的使用了 Split 来分割查询串中的多个查找词。

```

//foreach (string s in this.Value.Split(" ".ToCharArray()))
//{
//    listString.Add(this.Field + ":" + "\"" + this.LuceneEscape(s) + "\"");
//}

```

当搜索串中包括连续的空格时，上面的语句会导致错误。可以用一个模拟 Java 版本的 StringTokenizer 来代替分割空格。

```

StringTokenizer st = new StringTokenizer(this.Value, " ");
while (st.HasMoreTokens())
{
    string s = st.NextToken();
    listString.Add(this.Field + ":" + "\"" + LuceneEscape(s) + "\"");
}

```

参考 Lucene 本身的查询方法改了一下构造查询的方法。

```

TermQuery qpBody = new TermQuery("body", "研究");

```

```
BooleanQuery query = new BooleanQuery();

query.Add(qpBody, BooleanClause.Occur.MUST);

queryBuilder.Query = query;
```

布尔查询 BooleanClause.Occur.MUST 表示这个条件是必须的。AND 用 BooleanClause.Occur.MUST，OR 的关系用 BooleanClause.Occur.SHOULD。

增加多个查询条件了，再给一个详细点的例子：

```
BooleanQuery query = new BooleanQuery();

String queryWord = "医药";

qAbstract = new TermQuery("abstract", queryWord);

query.Add(qAbstract, BooleanClause.Occur.MUST);

qYear = new RangeQuery("year_id", "1997", "1998");

query.Add(qYear, BooleanClause.Occur.MUST);
```

可以用嵌套的 BooleanQuery 来实现 AND 又有 OR 的条件查询，例如：

```
query.Add( anotherBooleanQuery, BooleanClause.Occur.MUST );
```

8.1.16 Solr 的 php 客户端

做索引有两种方式：用 php 模拟 curl 的方式做索引和直接发送 POST 的客户端，后一种方法更成熟一些：

```
<?php

require_once( 'Apache/Solr/Service.php' );

//

//

// Try to connect to the named server, port, and url
```

```
//

$solr = new Apache_Solr_Service( 'localhost', '8983', '/solr' );


if ( ! $solr->ping() ) {

    echo 'Solr service not responding.';

    exit;

}


//

//

// Create two documents to represent two auto parts.

// In practice, documents would likely be assembled from a

//     database query.

//

$parts = array(

    'spark_plug' => array(

        'partno' => 1,

        'name' => 'Spark plug',

        'model' => array( 'Boxster', '924' ),

        'year' => array( 1999, 2000 ),

        'price' => 25.00,

        'inStock' => true,
```



```
),

'windshield' => array(

    'partno' => 2,

    'name' => 'Windshield',

    'model' => '911',

    'year' => array( 1999, 2000 ),

    'price' => 15.00,

    'inStock' => false,

)

);

$documents = array();

foreach ( $parts as $item => $fields ) {

    $part = new Apache_Solr_Document();

    foreach ( $fields as $key => $value ) {

        if ( is_array( $value ) ) {

            foreach ( $value as $datum ) {

                $part->setMultiValue( $key, $datum );

            }

        }

    }

}
```

```
        else {

            $part->$key = $value;

        }

    }

    $documents[] = $part;

}

//

//

// Load the documents into the index

//

try {

    $solr->addDocuments( $documents );

    $solr->commit();

    $solr->optimize();

}

catch ( Exception $e ) {

    echo $e->getMessage();

}

//
```

```
//

// Run some queries. Provide the raw path, a starting offset

//   for result documents, and the maximum number of result

//   documents to return. You can also use a fourth parameter

//   to control how results are sorted and highlighted,

//   among other options.

//

$offset = 0;

$limit = 10;

$queries = array(

    'partno: 1 OR partno: 2',

    'model: Boxster',

    'name: plug'

);

foreach ( $queries as $query ) {

    $response = $solr->search( $query, $offset, $limit );

    if ( $response->getHttpStatus() == 200 ) {

        // print_r( $response->getRawResponse() );

    }

}
```

```
if ( $response->response->numFound > 0 ) {

    echo "$query <br />";

    foreach ( $response->response->docs as $doc ) {

        echo "$doc->partno $doc->name <br />";

    }

    echo '<br />';

}

}

else {

    echo $response->getHttpStatusMessage();

}

}

?>
```

8.2 图片搜索

可以把图片的内容转化成文字，然后建立索引。这个就是 OCR 识别的过程。

```
String imgFile = "D:/lg/ocr/sample-images/ESfjydz.png";
OCR ocr = new OCR();
ocr.preload(); //学习的过程
String text = ocr.recognize(imgFile); //识别的过程

System.out.println("\nresult:"+text);
```

8.2.1 图像的 OCR 识别

在 OCR 识别过程中，需要对从图像提取出的数据集进行分类，因此合适的分类器对结果的预测起着非常重要的作用。近年来，支持向量机 (SVM) 作为一个核方法的机器学习技术，不仅有强烈的理论基础而且有极好的成功经验。本例采用 SVM 来训练数据样本。下面的代码就是基于 SVM 的 OCR 识别代码：

(1) CharRange 类：对字符区域定位，把带有字符的区域从图像中分离出一个矩形框出来。

```
public class CharRange {
    int x;
    int y;
    int width;
    int height;
}
```

(2) Entry 类：对图片进行垂直和水平方向的扫描，该类主要成员变量及方法如下：

```
public class Entry
{
    static final int DOWNSAMPLE_WIDTH = 12; // 样本数据宽度
    static final int DOWNSAMPLE_HEIGHT = 18; // 样本数据高度
    protected Image entryImage; // 存储检测的图像
    protected Graphics entryGraphics; // 处理图形图像
    protected int pixelMap[]; // 存储图像像素
    protected boolean hLineClear(int x, int w, int y)
    {
        int totalWidth = entryImage.getWidth(null);
        System.out.println("w:" + w);
        for (int i = x; i <= w; i++)
        {
            if (pixelMap[(y * totalWidth) + i] != -1)
                return false;
        }
        return true;
    } // 水平扫描图像并进行像素检测
    protected boolean vLineClear(int x)
    {
        int w = entryImage.getWidth(null);
        int h = entryImage.getHeight(null);
        for (int i = 0; i < h; i++)
        {
            if (pixelMap[(i * w) + x] != -1)
                return false;
        }
        return true;
    } // 垂直扫描图像并进行像素检测
```

```

protected ArrayList<CharRange> findBounds(int w, int h)
{
    ArrayList<CharRange> bounds = findHBounds(w,h);
    for(CharRange cr:bounds)
    {
        findVBound(cr);
    }
    return bounds;
}

void findVBound(CharRange cr)
{
    for (int i = 0; i < cr.height; i++)
    {
        if (!hLineClear(cr.x,cr.width,i))
        {
            cr.y = i;
            break;
        }
    }
    for (int i = cr.height - 1; i >= 0; i--)
    {
        if (!hLineClear(cr.x,cr.width,i))
        {
            cr.height = i;
            break;
        }
    }
}

//找到水平扫描时的上边界和下边界
protected ArrayList<CharRange> findHBounds(int w,int h)
{
    ArrayList<CharRange> bounds = new ArrayList<CharRange>();
    int begin=0;
    int end=w;
    boolean lastState = false;
    boolean curState = false;
    for (int i = 0; i < w; i++)
    {
        if (vLineClear(i))
        {
            System.out.println("find blank:"+i);
            curState = false;
        }
        else
        {

```

```

        curState = true;
    }
    if(!lastState && curState)
    {
        begin = i;
    }
    else if(lastState && !curState)
    {
        end = (i -1);
        CharRange cr = new CharRange(begin,0,end,h);
        bounds.add(cr);
    }
    lastState = curState;
}
if(curState)
{
    CharRange cr = new CharRange(begin,0,w -1,h);
    bounds.add(cr);
}
return bounds;
} //找到垂直扫描时的左边界和右边界
public ArrayList<SampleData> downSample()
{
    int w = entryImage.getWidth(null);
    int h = entryImage.getHeight(null);
    ArrayList<SampleData> samples = new ArrayList<SampleData>();
    PixelGrabber grabber = new PixelGrabber(entryImage, 0, 0, w, h, true);
    try
    {
        grabber.grabPixels();
        pixelMap = (int[]) grabber.getPixels();
        ArrayList<CharRange> bounds = findBounds(w, h);
        for(CharRange cr:bounds)
        {
            SampleData data = new
SampleData("?",DOWNSAMPLE_WIDTH,DOWNSAMPLE_HEIGHT);
            System.out.println(cr);
            double ratioX = (double) (cr.width - cr.x +1)
                / (double) DOWNSAMPLE_WIDTH;
            double ratioY = (double) (cr.height - cr.y +1)
                / (double) DOWNSAMPLE_HEIGHT;
            for (int y = 0; y < data.getHeight(); y++)
            {
                for (int x = 0; x < data.getWidth(); x++)

```

```

        {
            if (downSampleQuadrant(x, y, ratioX, ratioY, cr.x, cr.y))
                data.setData(x, y, true);
            else
                data.setData(x, y, false);
        }
    }
    data.ratio = (double)(cr.width - cr.x + 1) / (double)(cr.height -
cr.y + 1);
    data.ratio = (data.ratio - 1)/4;
    samples.add(data);
}
} catch (InterruptedException e) { }
return samples;
} //对样本数据进行采样、归一化
protected boolean downSampleQuadrant(double x, double y,
                                     double ratioX, double ratioY,
                                     double downSampleLeft, double
downSampleTop)
{
    int w = entryImage.getWidth(null);
    int startX = (int) (downSampleLeft + (x * ratioX));
    int startY = (int) (downSampleTop + (y * ratioY));
    int endX = (int) (startX + ratioX);
    int endY = (int) (startY + ratioY);
    for (int yy = startY; yy <= endY; yy++)
    {
        for (int xx = startX; xx <= endX; xx++)
        {
            int loc = xx + (yy * w);
            if (pixelMap[loc] != -1)
                return true;
        }
    }
    return false;
} //在样本数据的指定范围内进行像素扫描
}

```

(3) Java2DAPI 提供了一些用于图像表示的类以及一些新的支持图像处理相关操作的类。这些类都包含在 java.awt 包、java.awt.image 包以及六个新增加的包中。这些类大大加强了 Java 的 2D 图像处理能力。因此，可以通过 Java2DAPI 中图像类对要识别的图像区域进行二值化处理。比如一个 b 这样的图像区域变成一个 0 和 1 组成的图像：

```

1100000000000
1100000000000
1100000000000

```



```

110000000000
110011111100
110011111100
111100000011
111100000011
110000000011
110000000011
110000000011
110000000011
110000000011
110000000011
110000000011
111100000011
111100000011
110011111100
110011111100

```

在实际应用过程中图像二值化处理很重要，否则识别出来的可能都是 1 了，也就是说图片需要预处理成 0101 这样的黑白格式。现在的包本身还只能处理黑白格式的 BMP 图像。如果图像是 JPG 等格式的，还需要额外的图像格式转换工作。

(4) svm_train.java 类，采用 svm 方法学习分类模型对图像进行实际采样。其主要代码如下：

```

prob.l = outputNumber;
prob.x = new svm_node[prob.l][];
for(int i=0;i<prob.l;i++)
    prob.x[i] = set.getInputSet(i);
prob.y = new double[prob.l];
for(int i=0;i<prob.l;i++)
    prob.y[i] = set.getOutput(i);
if(param.gamma == 0)
    param.gamma = 1.0/inputNeuron;
model = svm.svm_train(prob,param);

```

(5) 对图像实际采样后用 svm_predict.java 进行分类，其主要代码如下：

```

int best = (int)svm.predict(model,input);

String result = map[best];

```

8.3 竞价排名

大的商业搜索引擎都通过竞价排名的方式来取得收入。最基本的是在右侧或左侧显示广告。竞价排名能够为搜索引擎取得最大效益的同时保证一定的客户满意度。

商业搜索也不想让更多的用户搜索某个关键词时找到没有价值的网站，不让搜索用户失望比多赚几毛钱点击费来的更有意义。

综合排名指数=网站质量度×竞价价格

新开户用户网站质量度默认为 1，浮动范围为：0.5~1.5。当网站质量度为 1 的情况下，竞价价格出价为 1 元，综合指数等于 1。

当客户网站访问量日渐升高，网站质量度也随之升高，也就是说，访问您网站的人越多，网站质量度也就会升高。网站质量度的另外一种计算方法是：关键词在网站相关搜索页的关联度/密度。

当网站质量度升高之后，相对之下，竞价价格出价也就随之会减少。

举例：当网站质量度为 1.5 的情况下，竞价价格出价为 1 元，综合指数等于 1.5，相对于网站质量度为 1 的就可以少出钱就可以比别人的综合排名指数高。

8.4 Web 图分析

挖掘网页之间的相互引用关系可以用来提高搜索结果排序或分析相似网页等。如果把每个网页看成一个节点，几百万以上的网页将会抽象成一个几百万节点以上的图。不能简单的使用邻接表或矩阵完全在内存中处理这样大的图，往往需要通过压缩文件来存储 Web 图。WebGraph 就是这样一个项目，它把图压缩成 BV 格式。

因为 BerkeleyDB 可以用来快速高效的存储海量数据。先采用它实现一个简单的 Web 图。来源 URL 和目的 URL 可以定义成一个多对多的映射。

```
@Entity
public class Link {
    @PrimaryKey public String fromURL;

    @SecondaryKey(related=Relationship.MANY_TO_MANY)
    public HashSet<String> toURL = new HashSet<String>();
}
```

在 WebGraph 类中定义从源 URL 到目标 URL 的主索引和目标 URL 到源 URL 的二级索引。

```
private PrimaryIndex<String,Link> outLinkIndex;

private SecondaryIndex<String,String,Link> inLinkIndex;
```

往WebGraph记录链接功能。

```
public void addLink (String fromLink, String toLink) throws
DatabaseException {
    Link outLinks = new Link();
    outLinks.fromURL = fromLink;
    outLinks.toURL = new HashSet<String>();
```

```

outLinks.toURL.add(toLink);

boolean inserted = outLinkIndex.putNoOverwrite(outLinks);

if(!inserted)
{
    outLinks = outLinkIndex.get(fromLink);
    outLinks.toURL.add(toLink);
    outLinkIndex.put(outLinks);
}
}

```

最后是取得指向一个 URL 地址的 Link 列表：

```

public String[] inLinks ( String URL ) throws DatabaseException {
    EntityIndex<String,Link> subIndex = inLinkIndex.subIndex(URL);
    String[] linkList = new String[(int)subIndex.count()];
    int i=0;
    EntityCursor<Link> cursor = subIndex.entities();
    try {
        for (Link entity : cursor) {
            linkList[i++] = entity.fromURL;
        }
    } finally {
        cursor.close();
    }
    return linkList;
}

```

BerkeleyDB 的 WebGraph 存储方式主要实现结束，下面继续分析 BV 压缩格式的使用。因为 WebGraph 存储的节点是整型，不是直接的 URL 地址，首先需要把 Web 网页映射成 0 到 n 的节点编号。然后形成有向边的图，例如 example.arcs 文件中包括(0,1,2,3,4)5 个节点组成的图：

0 2

1 2

1 4

2 3

3 4

下面这个命令将会产生一个压缩图到 `bvexample` 文件。

```
java it.unimi.dsi.webgraph.BVGraph -g ArcListASCIIGraph example.arcs bvexample
```

生成压缩图以后我们可以统计图的出度：

```
java it.unimi.dsi.webgraph.examples.OutdegreeStats -g BVGraph bvexample
```

执行结果是：

The minimum outdegree is 0, attained by node 4

The maximum outdegree is 2, attained by node 1

The average outdegree is 1.0

为了支持在 Solr 中以 “link:” 语法查找链接网页，写一个 `LinkToThisHandler` 的 Solr 插件调用上面已经实现的 `WebGraph`。

```
public final static String PREFIX = "lt.";
public final static String URL_FIELD = PREFIX + "fl";
private static final String INDEX_DIR = PREFIX + "dbDir";

private WebGraph webGraph = null;

@Override
public void init(NamedList args) {
    super.init(args);
    SolrParams p = SolrParams.toSolrParams(args);
    String dir = p.get(INDEX_DIR);
    try {
        //读入WebGraph
        webGraph = new WebGraph(dir);
    } catch (DatabaseException e) {
        throw new RuntimeException("Cannot open WebGraph database", e);
    }
}

@Override
public void handleRequestBody(SolrQueryRequest req, SolrQueryResponse
rsp) throws Exception
{
    SolrParams params = req.getParams();
    SolrIndexSearcher searcher = req.getSearcher();
    String q = params.get(CommonParams.Q);
```

```
SolrParams required = params.required();
//取得URL列的名称
String field = required.get(LinkToThisHandler.URL_FIELD) ;

// What fields do we need to return
String fl = params.get(CommonParams.FL);
int flags = 0;
if (fl != null) {
    flags |= SolrPluginUtils.setReturnFields(fl, rsp);
}

int start = params.getInt( CommonParams.START, 0 );
int rows = params.getInt( CommonParams.ROWS, 10 );

DocList docList = null;

if( q != null ) {

    // Find the base match
    Query query = QueryParsing.parseQuery(q,
params.get(CommonParams.DF), params, req.getSchema());
    DocList match = searcher.getDocList(query, null, null, 0, 1, flags );
    // only get the first one...

    // This is an iterator, but we only handle the first match
    DocIterator iterator = match.iterator();
    if( iterator.hasNext() ) {
        // do a MoreLikeThis query for each document in results
        int id = iterator.nextDoc();
        IndexReader reader = searcher.getReader();
        Document doc = reader.document(id);
        String toURL = doc.getField(field).stringValue();
        String[] fromURLs = webGraph.inLinks(toURL);
        ArrayList<Integer> docIds = new
ArrayList<Integer>(fromURLs.length);

        for(int i = 0;i<fromURLs.length;++i)
        {
            TermQuery tq = new TermQuery(new Term(field,fromURLs[i]));
            Hits hits = searcher.search(tq);
            if(hits.length()>=1)
            {
                int docId = hits.id(0);
```

```

        docIds.add(docId);
    }
}
int[] ids = new int[docIds.size()];
for(int i=0;i<ids.length;++i)
{
    ids[i]=docIds.get(i);
}
DocSlice result = new
DocSlice(0,ids.length,ids,null,ids.length,0);
    docList = result.subset(start, rows);
}
}
else {
    throw new SolrException( SolrException.ErrorCode.BAD_REQUEST,
        "MoreLikeThis requires either a query (?q=) or text to find similar
documents." );
}
if( docList == null ) {
    docList = new DocSlice(0,0,null,null,0,0); // avoid NPE
}
rsp.add( "response", docList );
}

```

在 SolrConfig.xml 中注册它:

```

<requestHandler name="/lt" class="solr.LinkToThisHandler">

    <lst name="defaults">

        <str name="lt.fl">url</str>

    </lst>

    <!-- Main init params for handler -->

    <!-- The directory where your webgraph db should live.    -->

    <str name="lt.dbDir">webgraph</str>

```

```
</requestHandler>
```

然后我们就可以在浏览器中测试效果了：

<http://localhost/solr/lt/?q=url%3Ahttp%5C%3A%5C%2F%5C%2Fwww.baotron.com%5C%2Findex.asp&version=2.2&start=100&rows=10&indent=on>

查询哪些网页指向了 <http://www.baotron.com/Findex.asp>

8.5 使用并程序分析数据

文档聚类或发现相关搜索词的时间复杂度是 $n*n$ ，例如分析 10m 多的相关搜索词，单线程的计算量可能长达 24 小时。

我们使用 Java 中自带的线程池来实现，子线程类的主要实现：

```
public class FindSimCall implements Callable<String[]> {
    private HashSet<String> words; // 总的搜索词集合
    private String s; // 待发现相关词的词

    public FindSimCall(HashSet<String> w ,String source)
    {
        words = w;
        s = source;
    }

    @Override
    public String[] call() throws Exception {
        System.out.println(s);
        // 形成related words列表
        // ...
        return relatedWords;
    }
}
```

主线程类的实现如下：

```
int threads = 4;
ExecutorService es = Executors.newFixedThreadPool(threads);

Set<Future<String[]>> set = new HashSet<Future<String[]>>();
```

```
for (final String s : words) {
    FindSimCall task = new FindSimCall(words,s);
    Future<String[]> future = es.submit(task);
    set.add(future);
}

FileOutputStream fos = new FileOutputStream(relatedWordsFile);
OutputStreamWriter osw = new OutputStreamWriter(fos, "GBK");
BufferedWriter writer = new BufferedWriter(osw);

for (Future<String[]> future : set) {
    String[] ret = future.get();

    for(String word:ret)
    {
        writer.write("%"+word);
    }
    writer.write( "\r\n" );
}

writer.close();
```

采用线程池可以充分利用多核 CPU 的计算能力。

8.6 RSS 搜索

首先从网站中自动发现 RSS。然后遍历每一个 RSS，必要的时候分析详细页面。

从一个网页发现 Feed 的步骤：

1. 首先，有一个函数来验证 feed 是有效的。
2. 如果这个 URI 已经指向一个 feed，则只是返回它，否则分析这个页面。
3. 看这个页面的头信息是否包含 LINK 标签。
4. <A>链接到同一个服务器上以".rss", ".rdf", ".xml"或".atom"结尾的 Feed。
5. <A>链接到同一个服务器上包含".rss", ".rdf", ".xml"或".atom"的 Feed。
6. <A>链接到以".rss", ".rdf", ".xml"或".atom"结尾的外部服务器 Feed。
7. <A>链接到包含".rss", ".rdf", ".xml"或".atom"的外部服务器 Feed。

8. 尝试猜测一些可能有 Feed 的通用的地方，例如(index.xml, atom.xml, etc.)。

8.7 本章小结

参考资料

书籍

ERIK HATCHER 和 OTIS GOSPODNETIC.”Lucene in Action”

网址

网址	说明
http://lucene.apache.org/	Lucene 主站点。
http://www.dotlucene.com	Lucene 的 .net 移植版本。
http://clucene.sourceforge.net/	Lucene 的 c++ 移植站点。
http://incubator.apache.org/lucene4c/	Lucene 的 c 语言移植版本。

本书中的章节和代码对照表