1. 信息检索与全文检索引擎 Lucene 简介

1.1. 信息检索的概念

息检索就是从信息集合中找出与用户需求相关的信息。被检索的 息除了文本外，还有图像、音频、视频等多媒体信息，我们只讨论文本的检索。

1.2. 信息检索技术的分类

目前信息检索技术可分为 3 类：

- 全文检索：把用户的查询请求和全文中的每一个词进行比较，不考虑查询请求与文本语义上的匹配。在信息检索工具中，全文检索是最具通用性和实用性的。
- 数据检索：查询要求和信息系统中的数据都遵循一定的格式，具有一定的结构，允许对特定的字段检索。其性能与使用有很大的局限性，并且支持语义匹配的能 较差。
- 知识检索：强调的是基于知识的、语义上的匹配。

我们要学习的就是全文检索技术。为应用程序添加全文检索的功能。

批注: 是指计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的检索方式。

批注: 数据检索查询要求和信息系统中的数据都遵循一定的格式，具有一定的结构，许对特定的字段检索。
例如，数据均按“时间、人物、地点、事件”的形式存储，查询可以为：地点=“北京”。
数据检索的性能取决于所使用的标识字段的方法和用户对这种方法的理解，因此具有很大的局限性。

1.3. 信息检索的基本流程

以下以在百度中搜索为例，说明信息检索的基本流程

步骤 ：

1， 在搜索框中输入一段文本或几个关键字（用户需求)
2， 搜索引擎（ 息检索系统）从互联网（ 息集合）中找出包含这些关键词的若干网页（相关信息），并按照一定的准则（相关度）排序，然后将部分结果返回（分页）

说明：

1， 搜索的速度很快，是毫秒级的。如 2009-9-21 搜索“传智播客”时，有如下提示：“百度一下，找到相关网页约 96,900 篇，用时 0.002 秒”
2， 结果列 中越靠前的，就是最符合我们所预期的结果，一般在第 1 页中就可以找到想要的文章，越往后翻就越离题。这是因为结果列 是按照相关度排过序的
3， 在返回的结果中页面中，每条结果只显示网页的部分内容，并且把搜索的关键字以不同的颜色显示（高亮）。

批注: 只是知道一下就可以了，不应进行对比

1.4. 信息检索与数据库的搜索对比

数据库的搜索不能实现我们的全文检索的要求，原因如下：

1， 匹配效果：如搜索 ant，在 sql 中使用 like %ant%’是会搜索出 planting 的，但他不应出现。

2， 查出的结果没有相关度排序，不知道有用的结果在哪一页。
3， 搜索速度太慢，达不到毫秒级的要求。

1.5. 全文检索中的建立索引与进行检索的流程

搜索的目的是为了在大量的信息中发现自己感兴趣的信息。但是，当有了足够的资料（比如网页、Word 文档、Pdf 文档，或数据库中的资料等）之后，并不能立即开始搜索，在此之前，必须先对信息建立索引。

就好比在图书馆里存放着大量书籍，当我们想查找关于某个主题的书籍时，不能一本一本的翻阅，而是要先查阅书籍的索引卡，根据索引卡上的信息找到相应的书籍。所以，当一本书进入到图书馆后，最重要的工作就是要马上建立索引卡。同样，对于我们所拥用的信息，也需要为其建立索引。

建立索引，就是对待搜索的信息进行一定的分析，并将分析结果按照一定的组织方式存储起来，通常将这些结果存储在文件中。存储分析结果的文件的集合就是索引。在查询时，先从索引中查找，由于索引是按照一定的结构组织的，所以查询的速度非常快。

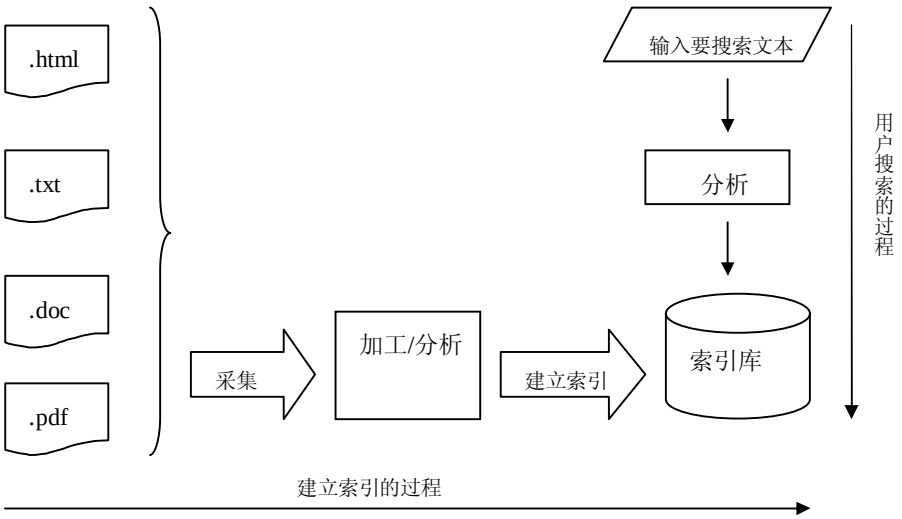
为了提供检索的功能，信息检索系统会事先做一些准备工作：信息的采集与加工。

- 信息采集：把信息源的信息拷贝到本地，构成待检索的信息集合。（信息源可以是互联网中的网页、电脑中的 txt、doc、ppt、pdf 等格式的电子文档，或是文件系统上的文件等等）。
- 信息加工：为采集到本地的信息编排索引，为查询做好准备。

流程如下：

批注: 如果信息检索系统在用户发出了检索请求后再去互联网上找答案，根本无法在有限的时间返回结果。

批注: 这里使用图书管理的例子



说明：

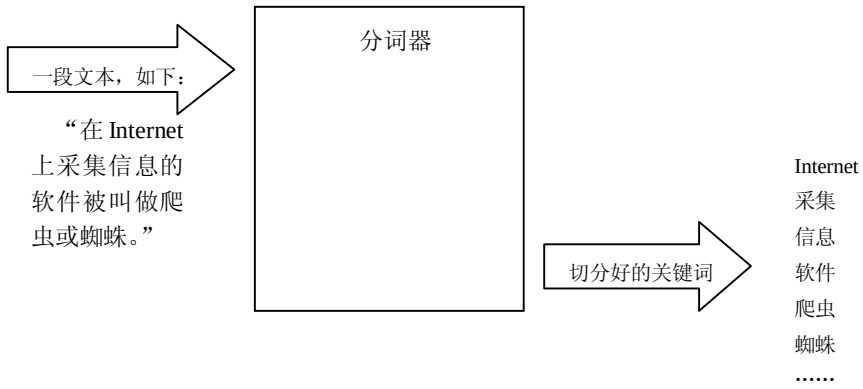
在 Internet 上采集信息的软件被称为爬虫或蜘蛛，或称做网络机器人。爬虫在 Internet 上访问每一个网页，每访问一个网页就把其中的内容传回本地服务器。

信息加工最主要的任务就是为采集到本地的信息编排索引，为查询做好准备。就好比在传统的图书编目工作中，图书管理员需要对书籍进行分类、标引、并撰写摘要。信息加工的过程的过程和图书编目过程类似，但全部由计算机自动完成。

2. 分词器

2.1. 分词器的作用

分词器，对文本资源进行切分，将文本按规则切分为一个个可以进行索引的最小单位（关键词）。



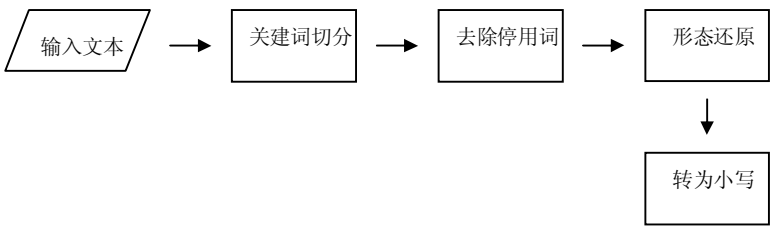
某文档中的一段文本，经过分词后如下：

1	2	3	4	5	6	7	8	9	10	11	12	13
在	Internet	上	采集	信息	的	软件	被	叫做	爬虫	或	蜘蛛	。

建立索引和进行搜索时都要用到分词器。为了保证能正确的搜索到结果，在建立索引与进行搜索时使用的分词器应是同一个。

2.2. 英文分词（分词器的工作流程）

英文分词过程大致如下：



说明：

形态还原，是去除单词词尾的形态变化，将其还原为词的原形。如下所示：

worked → work

批注: 或叫做词干抽取，比如将 running 还原为 run

working → work
studies → study
……

如对“IndexWriter addDocument's a javadoc.txt”进行分词：

1) 切分词
“IndexWriter”、“ addDocument's”、“a”、“ javadoc.txt”
2) 排除停用词
“IndexWriter”、“ addDocument's”、“ javadoc.txt”
3) 形态还原
“IndexWriter”、“ addDocument”、“ javadoc.txt”
4) 转为小写
“IndexWriter”、“ addDocument”、“ javadoc.txt”

2.3. 中文分词

中文的文词比较复杂，因为不是一个字就是一个词，而且一个词在另外一个地方就可能不是一个词，如在“帽子和服装”中，“和服”就不是一个词。对于中文分词，通常有三种方式：单字分词、二分法分词、词典分词。

- 单字分词：就是按照中文一个字一个字地进行分词。如：“我们是中国人”，
效果：“我”、“们”、“是”、“中”、“国”、“人”。（StandardAnalyzer 就是这样）。
- 二分法分词：按两个字进行切分。如：“我们是中国人”，效果：“我们”、“们是”、“是中”、“中国”、“国人”。（CJKAnalyzer 就是这样）。
- 词典分词：按某种算法构造词,然后去匹配已建好的词库集合，如果匹配到就切分出来成为词语。通常词库分词被认为是最理想的中文分词算法。如：“我们是中国人”，效果为：“我们”、“中国人”。（使用极易分词的 MMAnalyzer）。可以使用“极易分词”，或者是“庖丁分词”分词器。

2.4. 停用词

有些词在文本中出现的频率非常高，而且对文本所携带的信息基本不产生影响，例如英文的“a、an、the、of”，或中文的“的、了、着”，以及各种标点符号等，这样的词称为停用词（stop word）。文本经过分词之后，停用词通常被过滤掉，不会被进行索引。在检索的时候，用户的查询中如果含有停用词，检索系统也会将其过滤掉（因为用户输入的查询字符串也要进行分词处理）。排除停用词可以加快建立索引的速度，还可以减小索引文件的大小。

测试分词器

```
public static void printKeywords(Analyzer analyzer, String content)
throws Exception {
    System.out.println("-------> 分词器: " + analyzer.getClass());

    Reader reader = new StringReader(content);
    TokenStream tokenStream = analyzer.tokenStream("content", reader);

    Token token = new Token();
    while ((token = tokenStream.next(token)) != null) {
        System.out.println(token);
    }
}
```

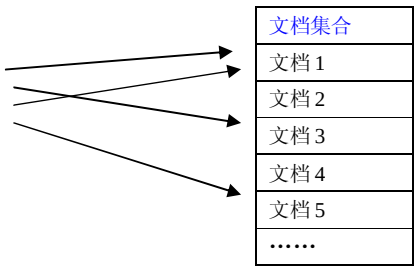
3. 倒排序索引结构

3.1. 索引文件结构

倒排索引，索引对象是文档中的单词等，用来存储这些单词在一个文档中的位置。例如，有些书在最后提供的索引（单词——页码 的对应列表），就可以看成是一种倒排序索引。可以通过一些关键字，在全书中检索出与之相关的部分。

词汇表

关键词	所在的文档编号
采集	1，3
信息	1，5
软件	1，2，5 ……
……	



以上只是用于说明倒排序索引的结构，最终的索引结构要复杂的多，还要存储关键词在文本中的编号位置（或是首字母的字符位置）等信息。

3.2. 索引文件的检索与维护

词汇表规模相对较小，文档集合规模较大。进行检索时，先从检索词汇表开始，然后找到相对应的文档。如果查询中仅包含一个关键词，则在词汇表中找到该单词，并取出他对应的文档就可以了。如果查询中包含多个关键词，则需要将各个单词检索出的记录进行合并。

维护倒排索引有三个操作：插入、删除和更新文档。但是更新操作需要较高的代价。因为

文档修改后（即使是很小的修改），就可能会造成文档中的很多的关键词的位置都发生了变化，这就需要频繁的读取和修改记录，这种代价是相当高的。因此，一般不进行更新操作，而是使用“先删除，后创建”的方式代替更新操作。

4. Lucene 介绍与初步使用（HelloWorld）

4.1. Lucene 介绍

Lucene 是一个高性能、可伸缩的全文检索工具包。可以使用他为你的应用程序添加索引和搜索能力。

Lucene 的作者 Doug Cutting 是资深的全文检索专家。最初，Lucene 以开源的形式出现在 SourceForget 上。2001 年，Lucene 加入了 Apache 旗下的 Jakarta 项目。2005 年，Lucene 正式脱离 Jakarta 成为 Apache 旗下的顶级项目。现在，Lucene 的主页为：http://lucene.apache.org/。

有很多应用程序使用 Lucene 来提供全文检索的功能，如我们经常使用的 Eclipse 的帮助子系统，就是使用 Lucene 实现的。（在第一次使用的时候，会有一个进度条，那是在创建索引）。

批注: http://wiki.apache.org/jakarta-lucene/PoweredBy

4.2. HelloWorld 程序（索引与搜索）

- 添加 jar 包
  - lucene-core-2.4.0.jar（核心）；
  - lucene-analyzers-2.4.0.jar（分词器）；
  - lucene-highlighter-2.4.0.jar（高亮器）；

4.2.1. 建立索引

```
/** 创建索引*/
@Test
public void createIndex() throws Exception {
    // file --> doc
    Document doc = File2DocumentUtils.file2Document(filePath);

    // 建立索引
    IndexWriter indexWriter =
           new IndexWriter(indexPath, analyzer, true, MaxFieldLength.LIMITED);
    indexWriter.addDocument(doc);
    indexWriter.close();
}
```

4.2.2. 搜索

```
/**  搜索 */
@Test
public void search() throws Exception {
    String queryString = "document";

    // 1，把要搜索的文本解析为 Query
    String[] fields = { "name", "content" };
    QueryParser queryParser = new MultiFieldQueryParser(fields, analyzer);
    Query query = queryParser.parse(queryString);

    // 2，进行查询
    IndexSearcher indexSearcher = new IndexSearcher(indexPath);
    Filter filter = null;
    TopDocs topDocs = indexSearcher.search(query, filter, 10000);
    System.out.println("总共有【"+topDocs.totalHits+"】条匹配结果");

    // 3，打印结果
    for(ScoreDoc scoreDoc : topDocs.scoreDocs){
        int docSn = scoreDoc.doc; // 文档内部编号
        Document doc = indexSearcher.doc(docSn); // 根据编号取出相应的文档
        File2DocumentUtils.printDocumentInfo(doc); // 打印出文档信息
    }
}
```

4.2.3. 练习：使用 Lucene 进行索引与搜索

完成建立索引与搜索的两个方法。

5. LuceneIndexDao

完成对索引的增删改查。

5.1. 创建索引的方案

- 定时重新创建索引
- 同步创建索引

Ø 异步任务创建索引

6. Lucene 使用

6.1. 工具类 DateTools、NumberTools

在对属性值进行大小比较时（通常是使用范围查询，比较数字类型的时候），是按照字符串的比较规则，因为他们都是转成字符串后存储的。这样就会出现问题，如“20”是大于“100”的（根据字符串的比较规则）。解决的方法是：让数字转成的字符串具有相同的位数，如位数不足，就在前面补相应数量个“0”。数字到符串、字符串到数字的转换可以使用 Lucene 提供的工具类 NumberTools 完成。对于日期与字符串的双向转换，则可以使用 Lucene 提供的 DateTools 完成。

- DateTools：转换日期的工具类，第二个 Resolution 参数可以指定日期的精度。指定 Resolution 中的常量值。
- NumberTools：转换数字的工具类，把数字转为 36 进制的字符串。

6.1. Highligher

可以截取一段文本（生成摘要），并且让关键字高亮显示（通过指定前缀与后缀实现，因为是在网页中显示，指定<“”，“”>，就会在网页中显示为红色）。

6.2. 查询

查询对象

- TermQuery
- RangeQuery
- WildcardQuery

PhraseQuery

短语查询

```
public void add(Term term, int position)
public void setSlop(int s)
```

例：add(new Term(“name”, “lucene”, 1);
add(new Term(“name”, “教程”, 3);
代表搜索的是“Lucene ？ 教程”，？表示中间隔一个词。

setSlop(2);
代表这两个词中间可以最多隔 2 个词

批注: 如果指定了多个词，则是各个词之间的隔的数量的的和

BooleanQuery

```
public void add(Query query, Occur occur)
```

Occur 用于表示布尔查询子句关系的类，包括：
Occur.MUST，Occur.MUST_NOT，Occur.SHOULD。

1， MUST 和 MUST：取得连个查询子句的交集。
2， MUST 和 MUST_NOT：包含 MUST 并且查询结果中不包含 MUST_NOT 的检索结果。
3， SHOULD 与 SHOULD，表示“或”关系，最终检索结果为所有检索子句的并集。

使用时注意：
1， MUST 和 SHOULD：此时 SHOULD 无意义，结果为 MUST 子句的检索结果。
2， MUST_NOT 和 MUST_NOT：无意义，检索无结果。
3， MUST_NOT 和 SHOULD： 此时 SHOULD 相当于 MUST，结果同 MUST 和 MUST NOT。

4， 单独使用 SHOULD：结果相当于 MUST。
5， 单独使用 MUST_NOT：无意义，检索无结果。

查询语法

要使用 QueryParser 与 MultiFieldQueryParser 进行解析。
指定默认的组合关系。

“+”与“-”：表示后面的条件是“MUST”，还是“MUST_NOT”
“AND”：
“OR”：
“NOT”：
“+”、“-”与“AND”、“OR”同时使用时：
可以使用括号“（”和“）”，对逻辑进行组合

6.3. 排序

- 相关度排序

得分
Document 的 boost 属性：Document. setBoost(float boost)
Field 的 boost 属性：Field. setBoost(float boost)
查询时指定： MultiFieldQueryParser(String[] fields, Analyzer analyzer, Map boosts)

- 按指定字段排序

```
public Sort(SortField field)
public Sort(SortField[] fields)
```

6.4. 过滤器

RangeFilter，可以对搜索出来的结果进行过滤。

练习：Web 文件搜索

a) 功能 1：建立索引

b) 功能 2：进行搜索，分页显示，关键字高亮，点击可以打开小窗口查看文件内容

7. Compass

7.1. 介绍

Compass 是对 Lucene 进行的封装。Compass 对于 Lucene，就好像 Hibernate 对于 Jdbc。在使用时用到的类（除分词器外），都是使用的 Compass 的。

7.2. 使用

```
<compass-core-config
xmlns="http://www.compass-project.org/schema/core-config"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.compass-project.org/schema/core-config

http://www.compass-project.org/schema/compass-core-config-2.2.xsd">

    <compass name="default">

        <!-- 指定索引位置 -->
        <connection>
            <file path="./compassIndex" />
        </connection>

        <!-- 添加可索引类（使用注解） -->
        <mappings>
            <class name="cn.itcast.compass.indexdao.Article" />
        </mappings>

        <!-- 添加其它设置 -->
        <settings>
            <setting name="compass.engine.analyzer.default.type"
value="jeasy.analysis.MMAnalyzer" />
            <setting
name="compass.engine.highlighter.default.formatter.simple.pre"
value="&lt;font color='red'&gt;" />
            <setting
name="compass.engine.highlighter.default.formatter.simple.post"
value="&lt;/font&gt;" />
        </settings>

    </compass>
</compass-core-config>
```

1， 添加 jar 包
2， 定义可搜索的对象
   a) 应使用一个单独的类，专门用于搜索的，只需定义在搜索结果页面中需要显示(用到)的属性。
   b) 在类上声明 @searchable，说明这是可以被搜索的类
   c) 在有唯一标识作用的属性上声明@searchableId，这个必须要有。
   d) 在属性（getter）上声明@searchProperty，并指定 Store 与 Index 策略，还可以指定 boost
3， 初始化 Compass

a) CompassConfiguration cfg = new CompassConfiguration();
b) cfg.setConnection("./index/"); // 设置索引目录
c) cfg.addClass(Post.class);
d) compass = cfg.buildCompass();

4，用 Compass 实现 IndexDao 中的增删改查

a) 创建索引：void create(Object obj)
b) 删除索引：void delete(Object obj)
searchableId 可以重复，删除时会删除所有 searchableId 匹配的记录。
c) 更新索引：void save(Object obj)
d) 查询：
CompassSession.get 和 CompassSession.load 的区别:如果不存在指定的 id 的资源,get 会返回 null,而 load 会抛一个异常.
（不能指定在某个 Field 中查询，默认是在所有的 Field 中查询？）
e) CompassTemplate
f) 排序：Query. addSort(String propertyName, SortDirection direction)
g) 配置分词器
使用配置：compass.engine.analyzer.[analyzer name].type
指定内置的名称，或者是指定分词器的全限定类名
h) 配置高亮器的前缀与后缀
高亮就是在关键字的前后分别加上前缀和后缀.
代码:String ht = hits.highlighter(i).fragment("title");
如果进行高亮的属性中没有出现相应关键字，则返回 null
i) 前缀和后缀的默认值为和,也可以进行修改,配置的 key 在 compass 文档 A.1.12. Search Engine Highlighters 中
设置默认 highlighter 的前缀:"compass.engine.highlighter.default.formatter.simple.pre"
设置默认 highlighter 的后缀:"compass.engine.highlighter.default.formatter.simple.post"

备忘

Lucene 的 API 文档，chm 格式的。