

一 全文检索技术

1 搜索背景

- 任何一个软件，都提供搜索的功能
 - 直播网站：主播，视频名字，标题等
 - 外卖软件：商铺，食品名
 - 大众点评：很多
- 搜索的数据特点
 - 数据量庞大
 - 要求速度快
 - 要求数据准确

2 传统检索技术的瓶颈

2.A 文件系统检索

word 文档，linux 的 vim 编辑器。低层逻辑是将全部文件加载到内存里，检索时使用内存搜索数据。不能保证大量数据。

2.B 数据库检索

相比于文件内存检索，数据量有所保证，但是为了满足多方位，多功能的搜索需求，搜索速度和准确性不能保证。

3 全文检索技术

3.A 定义

全文数据库是全文检索系统的主要构成部分。所谓全文数据库是将一个完整的信息源的全部内容转化为计算机可以识别、处理的信息单元而形成的数据集合。全文数据库不仅

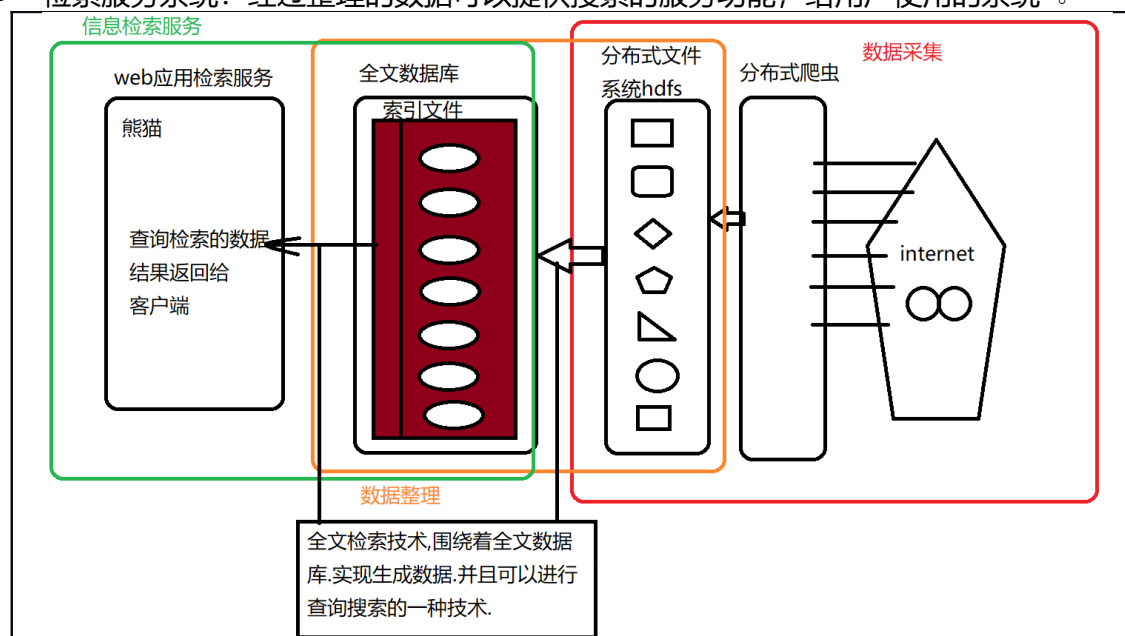
存储了信息，而且还有对全文数据进行词、字、段落等更深层次的编辑、加工的功能，而且所有全文数据库无一不是海量信息数据库。

全文检索技术是什么：是一个围绕这个全文数据展开的技术，可以将数据的非结构化数据整理成全文数据库中有结构的数据，并且配合应用代码实现数据查询工作。

3.B 检索系统架构（百度搜索引擎为例）

3.B.a 一个完整的检索系统由三部分包含的

- 数据采集：数据源是分散的，需要收集到一起。
- 数据整理：采集之后的数据是非结构化的整体，不能直接使用，必须经过整理计算。
- 检索服务系统：经过整理的数据可以提供搜索的服务功能，给用户使用的系统。



二 lucene

1 什么是 lucene

lucene 是一种全文检索技术的**工具包**。可以实现快速方便的开发全文检索技术的所有功能。由美国搜索领军人物 Doug Cutting 2004 年左右创建的一种技术。(还是 hadoop 创始人)

2 Lucene 特点

lucene 原生语言 java，具备如下一些。

- **占用内存少**：程序中 lucene 运行过程不会占用太多的栈内存
- **增量创建索引**：提供了增量(新增一批数据基础上，增加新的数据)索引创建，和批量(新增一批数据)索引创建，性能几乎一样。
- **索引相关信息占比小**：全文检索文件中，数据结构(保证搜索速度的内容)，数据信息(整理的数据信息单元)，索引数据结构的占比不大 20%
- **提供了丰富搜索功能**：根据不同的需求和场景，使用不同的搜索方式。
 - 词项查询
 - 多域查询
 - 布尔查询
 - 范围查询
 - 模糊查询
 - 通配查询
 -

三 倒排索引算法

1 索引的概念

索引--index。

读书---目录(索引)

超市---类别牌子

词典---拼音法，偏旁法，难检字法等

图书馆---书号

索引定义：

一批具备顺序的数据结构，形成文件存储在介质中，帮助快速定位到需要的数据。

全文检索--lucene 所能够创建出来的索引文件是什么样的结构--索引的意义

2 倒排索引计算

2.A 一些概念

分词(analyze):

全文检索技术, 对文本数据做词, 字段落的加工处理逻辑。能够形成的结果是多个词项(term)。

例如: 文本="中华人民共和国"

字的处理: 中, 华, 人, 民, 共, 和, 国

词的处理(最常见, 最广泛): 中华, 华人, 人民, 中华人民, 共和国, 人民共和, 人民共和国

句的处理: 中华人民共和国

每一个计算结果中的字, 词, 句这个整体文本都是分词计算的词项数据(term)

文档对象(document):

lucene 中, 存在一个整理之后的对象数据(信息单元)叫做 document, 表示一个数据源中的完整数据(一个网页, 一行数据库数据), document 最终保存在索引文件里。被查询搜索到后, 可以使用这个对象的数据。

域属性(field):

文档对象虽然是一个整体数据单元, 但是可以由不同的属性构成, 例如

一个网页 document:

title 标题, address 连接地址, content 网页内容

一个数据库商品数据 document

name, price, num, image 等等

2.B 倒排索引算法

lucene 所创建出来的索引文件到底是什么结构。为什么能查询速度快?

lucene 在收集数据源之后, 经过几步计算封装整理能够形成可以被 lucene 自己搜索的索引文件(索引文件包含了数据的 document 对象集合)

2.B.a 数据源和 document 的封装: (域属性封装一个 document 对象具体属性值)

假如有 2 个新闻网页数据源:

document 封装, 定义域属性(根据数据源定义)

文章标题: title

文章内容: content

出版社: publisher

点击次数: click

| | |
|-------|-------|
| web1: | web2: |
|-------|-------|

| | |
|--|--|
| 文章标题：美国耍赖 文章内容：频繁甩锅，控诉中国疫情信息 透露不明确 出版社：新华网 点击次数：58 | 文章标题：美国甩锅 文章内容：美国谎称，援助中国 1 亿美 金，中国外交部称从未收到 出版社：新华网 点击次数：66 |
| doc1: id: uuid/1 title: 美国耍赖 content: 频繁甩锅，控诉中国疫情信息 透露不明确 publisher: 新华网 click: 58 | doc2: id: uuid/2 title: 美国甩锅 content: 美国谎称，援助中国 1 亿美 金，中国外交部称从未收到 publisher: 新华网 click: 66 |

2.B.b 进行分词计算：对 document 中的文本内容进行分词计算

分词计算词项，经过二进制计算，词项总会携带计算之后的很多参数

例如：词项对应属性，词项出现频率，词项的位移，词项的所在 document 的 id 值

doc1: (1 表示出线一次，2 就是出线 2 次...)

美国(1)，耍赖(1)，甩锅(1)，中国(1)，疫情(1)，明确(1)，新华网(1)，58(1)

doc2:

美国(2)，甩锅(2)，谎称(2)，中国(2)，外交部(2)，新华网(2)，66(2)

doc3: doc4: ...docn

2.B.c 形成倒排索引表：合并词项结果和参数-形成倒排索引表

美国(1, 2)，甩锅(1, 2)，中国(1, 2)，新华网(1, 2)，耍赖(1)，谎称(2)

记录形式，以表格方式记录

| 词项 \docid | doc1 | doc2 | doc3 | doc4 | 。。 |
|--------------|------|------|------|------|----|
| 美国 | 1 | 1 | 0 | 0 | 0 |
| 甩锅 | 1 | 1 | 0 | 0 | 0 |
| 中国 | 1 | 1 | 0 | 0 | 0 |
| 新华网 | 1 | 1 | 0 | 0 | 0 |
| 耍赖 | 1 | 0 | 0 | 0 | 0 |
| 谎称 | 0 | 1 | 0 | 0 | 0 |
| 其他 | 0 | 0 | 1 | 1 | 0 |
| 。。 | 。。 | 。。 | 。。 | 。。 | 。。 |

通过读取这个合并表格的数据可以得到对应词项的二进制做搜索计算

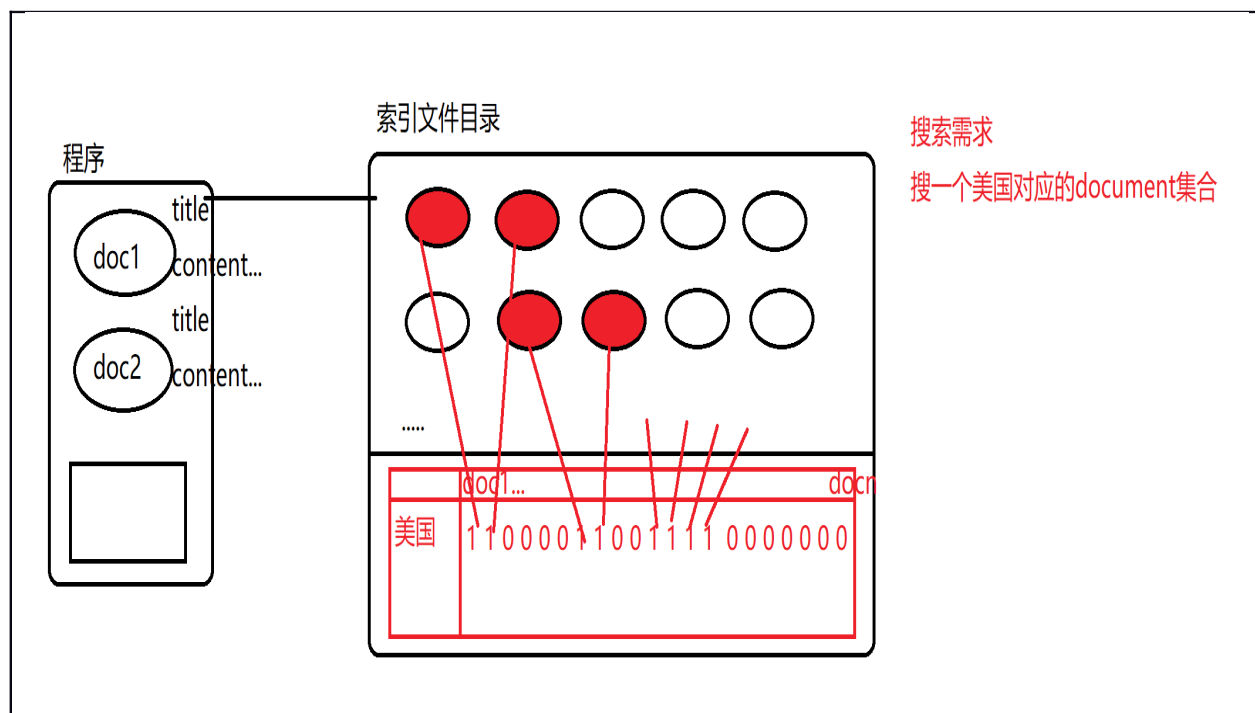
例如：查询一个有美国词语，并且有耍赖词语出现的 document 集合

美国 11000

耍赖 10000

做位的与运算得到结果 美国&耍赖 10000--->同时出现 2 个词语的文档 doc1

2.B.d 输出索引文件：最终 lucene 将所有数据输出到索引文件



两部分组成,

第一部分：海量 document 对象的集合(搜索到的数据)

第二部分：计算过程出现分词合并结果倒排索引表(搜索手段)

四 lucene 分词器

1 分词器概括

分词计算在倒排索引创建的过程中，起着非常重要的作用。lucene 是不是提供了所有分词计算的逻辑实现类呢？

不同分词计算需求，使用分词计算规则算法逻辑底层代码完全不同的。lucene 也不能完成世界上所有对分词器需求的计算。

例如：

中文分词器，计算逻辑，对应底层二进制，一定和俄文不一样，德文，法文，西班牙文，吐火罗文等等

lucene 提供了规范---分词器接口，Analyzer(分词器)想要加入 lucene 的逻辑，就需要实现这个接口。

2 分词器测试

2.A 导入依赖

```
<dependency> <!-- 查询相关 jar 包 -->
  <groupId>org.apache.lucene</groupId>
  <artifactId>lucene-queryparser</artifactId>
  <version>6.0.0</version>
</dependency>
<dependency> <!-- lucene 自带只能中文分词器 jar 包 -->
  <groupId>org.apache.lucene</groupId>
  <artifactId>lucene-analyzers-smartcn</artifactId>
  <version>6.0.0</version>
</dependency>
```

2.B lucene 分词器类型

StandardAnalyzer-->标准分词器

英文：对词做加工，按照空格，标点分开

中文：字做加工

simpleAnalyzer--->简单分词器

对空格/标点符号进行切分计算

英文：词加工

中文：段落，句加工

whitespaceAnalyzer--->空格分词器

按照空格处理数据

smartChineseAnalyzer--->智能中文分词器

处理常用的中文词语

2.C 测试

```
/**
 * @作者 舒新胜
 * @项目 blog
 * @创建时间 2020/6/6 18:24
 */
public class LuceneIndexCreate {
    public static void main(String[] args) throws IOException {
        String msg="今天晚上干嘛呢? ";
        //构造标准分词器、简单分词器、空格分词器、智能中文分词器
        Analyzer a1=new StandardAnalyzer();
        Analyzer a2=new SimpleAnalyzer();
        Analyzer a3=new WhitespaceAnalyzer();
        Analyzer a4=new SmartChineseAnalyzer();
    }
}
```

```

        System.out.println("*****标准*****");
        printTerm(a1,msg);//今,天,晚,上,干,嘛,呢

        System.out.println("*****简单*****");
        printTerm(a2,msg);//今天晚上干嘛呢

        System.out.println("*****空格*****");
        printTerm(a3,msg);//今天晚上干嘛呢?

        System.out.println("*****智能*****");
        printTerm(a4,msg);//今天,晚上,干,嘛,呢

    }
    private static void printTerm(Analyzer a, String msg) throws IOException {

        //调用 a 这个分词器的 api 将需要分词的数据计算成词项
        //分词不能独立存在,需要依托 document 数据, 这里"document"模拟为文档对象
        //实际 “document”就是定义的域属性
        TokenStream tokenStream = a.tokenStream("document", msg);

        //上面分词计算后 指针指向这里需要重置回到开头
        tokenStream.reset();

        //获取分词后词项的文本属性及偏移量属性
        CharTermAttribute charAttr =
            tokenStream.getAttribute(CharTermAttribute.class);
        OffsetAttribute offAttr =
            tokenStream.getAttribute(OffsetAttribute.class);

        //遍历词项
        while(tokenStream.incrementToken()){
            /*
                System.out.println("偏移量起始位置:"+offAttr.startOffset());
                System.out.println("偏移量结束位置:"+offAttr.endOffset());*/
            System.out.print(charAttr.toString()+"");
        }
    }
}

```

五 IK 分词器

对于语言的发展,新的词语的出现,总是改变对分词计算的需求.引入基于词典的分词器---

IKAnalyzer.可以对词典进行扩展,计算分词时,读取到词典的数据就可以计算该词语成为一个词项.

还支持停用词典:在停用词典中的词语,不会计算分词

例如: 敏感词,禁语,无意义词语

1 使用 ik 分词器

1.A 导入依赖的 jar 包

maven 中央库没有资源,手动导入系统.

IKAnalyzer2012_u6.jar-->粘贴到项目,右键jar包-->Add as Libaray

1.B 实现 Analyzer 接口

```
package cn.shu.blog.utils.lucene;

import java.io.IOException;

import org.apache.lucene.analysis.Tokenizer;
import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;
import org.apache.lucene.analysis.tokenattributes.OffsetAttribute;
import org.apache.lucene.analysis.tokenattributes.TypeAttribute;
import org.wltea.analyzer.core.IKSegmenter;
import org.wltea.analyzer.core.Lexeme;

public class IKTokenizer6x extends Tokenizer{
    //ik 分词器实现
    private IKSegmenter _IKImplement;
    //词元文本属性
    private final CharTermAttribute termAtt;
    //词元位移属性
    private final OffsetAttribute offsetAtt;
    //词元分类属性
    private final TypeAttribute typeAtt;
    //记录最后一个词元的结束位置
    private int endPosition;
    //构造函数, 实现最新的 Tokenizer
    public IKTokenizer6x(boolean useSmart){
        super();
        offsetAtt=addAttribute(OffsetAttribute.class);
        termAtt=addAttribute(CharTermAttribute.class);
        typeAtt=addAttribute(TypeAttribute.class);
        _IKImplement=new IKSegmenter(input, useSmart);
    }

    @Override
    public final boolean incrementToken() throws IOException {
        //清除所有的词元属性
        clearAttributes();
        Lexeme nextLexeme=_IKImplement.next();
        if(nextLexeme!=null){
            //将 lexeme 转成 attributes
            termAtt.append(nextLexeme.getLexemeText());
            termAtt.setLength(nextLexeme.getLength());
            offsetAtt.setOffset(nextLexeme.getBeginPosition(),
```

```

        nextLexeme.getPosition());
        //记录分词的最后位置
        endPosition=nextLexeme.getPosition();
        typeAtt.setType(nextLexeme.getLexemeText());
        return true;//告知还有下个词元
    }
    return false;//告知词元输出完毕
}

@Override
public void reset() throws IOException {
    super.reset();
    _IKImplement.reset(input);
}

@Override
public final void end(){
    int finalOffset = correctOffset(this.endPosition);
    offsetAtt.setOffset(finalOffset, finalOffset);
}
}

package cn.shu.blog.utils.lucene;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.Tokenizer;

public class IKAnalyzer6x extends Analyzer{
    private boolean useSmart;
    public boolean useSmart(){
        return useSmart;
    }
    public void setUseSmart(boolean useSmart){
        this.useSmart=useSmart;
    }
    public IKAnalyzer6x(){
        this(false);//IK 分词器 lucene analyzer 接口实现类，默认细粒度切分
        算法
    }
    //重写最新版本 createComponents; 重载 analyzer 接口，构造分词组件
    @Override
    protected TokenStreamComponents createComponents(String
    fileName) {
        Tokenizer _IKTokenizer=new IKTokenizer6x(this.useSmart);
        return new TokenStreamComponents(_IKTokenizer);
    }
    public IKAnalyzer6x(boolean useSmart){
        super();
        this.useSmart=useSmart;
    }
}

```

1.C 代码中使用 IKAnalyzer6x 的对象,观察计算分词结果

```
String msg="今天晚上干嘛呢? ";
```

```
System.out.println("*****IK*****");  
printTerm(ikAnalyzer6x,msg); //今天,晚上,干嘛,呢
```

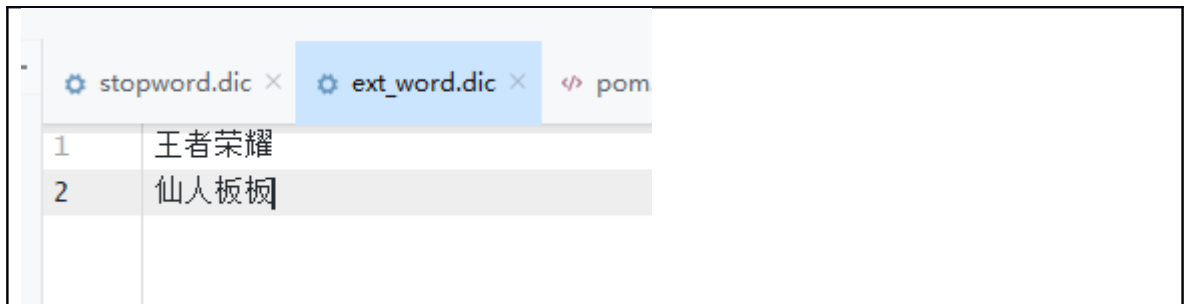
2 IK 分词器的词典配置

- 2 个内容
 - IKAnalyzer.cfg.xml 配置文件 指定扩展和停用词典的名称
 - **.dic 这是词典文件

2.A ik 分词器配置-->IKAnalyzer.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE properties SYSTEM  
"http://java.sun.com/dtd/properties.dtd">  
<properties>  
    <comment>IK Analyzer 扩展配置</comment>  
    <!--用户可以在这里配置自己的扩展字典-->  
    <entry key="ext_dict">ext.dic;</entry>  
    <!--用户可以在这里配置自己的扩展停止词字典-->  
    <entry key="ext_stopwords">stopword.dic;</entry>  
</properties>
```

2.B 扩展词典：分词时就可以分成指定的词语



| | | |
|--------------|--------------|-----|
| stopword.dic | ext_word.dic | pom |
| 1 | 王者荣耀 | |
| 2 | 仙人板板 | |

2.C 停用词典：分词时会过滤掉，不会打印



| | |
|--------------|----------------|
| stopword.dic | pom.xml (blog) |
| 1 | 仙人板板 |
| 2 | 日龙包 |

2.D 说明：dic 后缀的文件

- 要在 IKAnalyzer.cfg.xml 中指定 dic 位置
- 编解码要和代码一致,否则无法准确计算
- 保证这一批配置文件在运行代码时被加载,
 - src/main/resources
 - src/test/resources

2.E 测试

```
String msg="小娃儿,你打个仙人板板的王者荣耀?";
IKAnalyzer6x ikAnalyzer6x = new IKAnalyzer6x();
printTerm(ikAnalyzer6x,msg);//今天,晚上,干嘛,呢
```

结果：当停用词典中的词语（仙人板板）没有打印，
扩展词典（王者荣耀）词语成功打印

```
E:\JAVA\jdk1.8.0_152\bin\java.exe ...
*****IK*****
加载扩展词典：ik_analyzer/ext_word.dic
加载扩展停止词典：ik_analyzer/stopword.dic
小娃儿,小娃,娃儿,你,打个,仙人,板,板,的,王者荣耀,王者,荣耀,
Process finished with exit code 0
|
```

六 lucene 的索引文件

可以利用 lucene 的底层代码,实现全文检索中重要数据组成全文数据库的索引文件的创建功能;

1 创建索引基本逻辑

- 选择一个文件夹,作为保存索引数据的目标
- 创建一个输出流对象 Writer
- 读取数据源,封装 document 对象数据
 - 根据数据源的结构定义 document 的各种域属性的内容
- 通过 writer 输出流,将数据经过倒排索引计算输出到文件形成所以的数据

2 测试代码

```
public void createIndex() {
    IndexWriter indexWriter=null;
    try {
        //1、创建一个路径对象
        Path path = Paths.get("D:/article_index");

        //2、将路径对象交给 lucene 管理
        FSDirectory dir = FSDirectory.open(path);
        //3、创建一个输出流配置对象，需要指定一个分词器，这里使用 IK 分词
器
        IndexWriterConfig writerConfig = new
IndexWriterConfig(new IKAnalyzer6x());
        //配置每次创建索引的模式 每次创建后追加

writerConfig.setOpenMode(IndexWriterConfig.OpenMode.CREATE_OR_
APPEND);
        //4、创建索引输出流
        indexWriter = new IndexWriter(dir, writerConfig);
        //5、封装数据到 document 对象
        Document doc = new Document();
        /*  文章标题:Lucene 学习
        文章内容:lucene 是一种全文检索技术的工具包。可以实现快速方便的开发
        全文检索技术的所有功能。由美国搜索领军人物 Doug Cutting 2004 年左右创建
        的一种技术。(还是 hadoop 创始人)
        作者:新胜科技有限责任公司
        访问次数:58
        */
        doc.add(new TextField("title", "Lucene 学习",
Field.Store.YES));
        doc.add(new StringField("content", "lucene 是一种全文检索
技术的工具包...", Field.Store.YES));
        doc.add(new TextField("author", "新胜科技有限责任公司",
Field.Store.YES));
        doc.add(new IntPoint("visitor", 60));

        indexWriter.addDocument(doc);
        //6、输出到索引文件
        indexWriter.commit();

    }catch (IOException e){
        e.printStackTrace();
    }finally {
        try {
            indexWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

| 此电脑 > 系统工具 (D:) > article_index | | | | |
|---------------------------------|----------------|---------|------|--|
| 名称 | 修改日期 | 类型 | 大小 | |
| _0.cfe | 2020/6/6 20:08 | CFE 文件 | 1 KB | |
| _0.cfs | 2020/6/6 20:08 | CFS 文件 | 2 KB | |
| _0.si | 2020/6/6 20:08 | SI 文件 | 1 KB | |
| segments_1 | 2020/6/6 20:08 | 文件 | 1 KB | |
| write.lock | 2020/6/6 20:08 | LOCK 文件 | 0 KB | |

3 可视化工具查看索引文件

共享 查看

互联网框架\09-Lucene\luke-6.0.0-luke-release

| 名称 | 修改日期 | 类型 | 大小 |
|----------|-----------------|----------------|------|
| target | 2020/5/29 18:13 | 文件夹 | |
| luke.bat | 2016/4/12 13:31 | Windows 批处理... | 1 KB |
| luke.sh | 2016/4/12 13:31 | SH 文件 | 1 KB |

Luke - Lucene Index Toolbox (6.0.0)

File Tools Settings Help

Overview Documents Search Commits Plugins

Index name: ?
 Number of fields: ?
 Number of documents: ?
 Number of terms: ?
 Has deletions? / Optimized? / ? / ?
 Index version: ?
 Index format: ?
 Index full-text: ?
 Directory implementation: ?
 Currently opened core: ?
 Current commit: ?

Select fields from the index:
 Available fields and terms:
 Name Term

Path to Index directory:
 Path: D:\article_index Browse...
 Hint: Luke can open multiple indexes in subdirectories.

☐ Open in Read-Only mode
☐ Force unlock, if locked

Expert options:
 Directory (one of predefined, or full class name): FSDirectory
☐ Load into RAMDirectory
☒ Keep all commit points
☐ Don't open IndexReader (when opening corrupted index)
☐ Slow IO - avoid and track expensive IO operations

OK Cancel

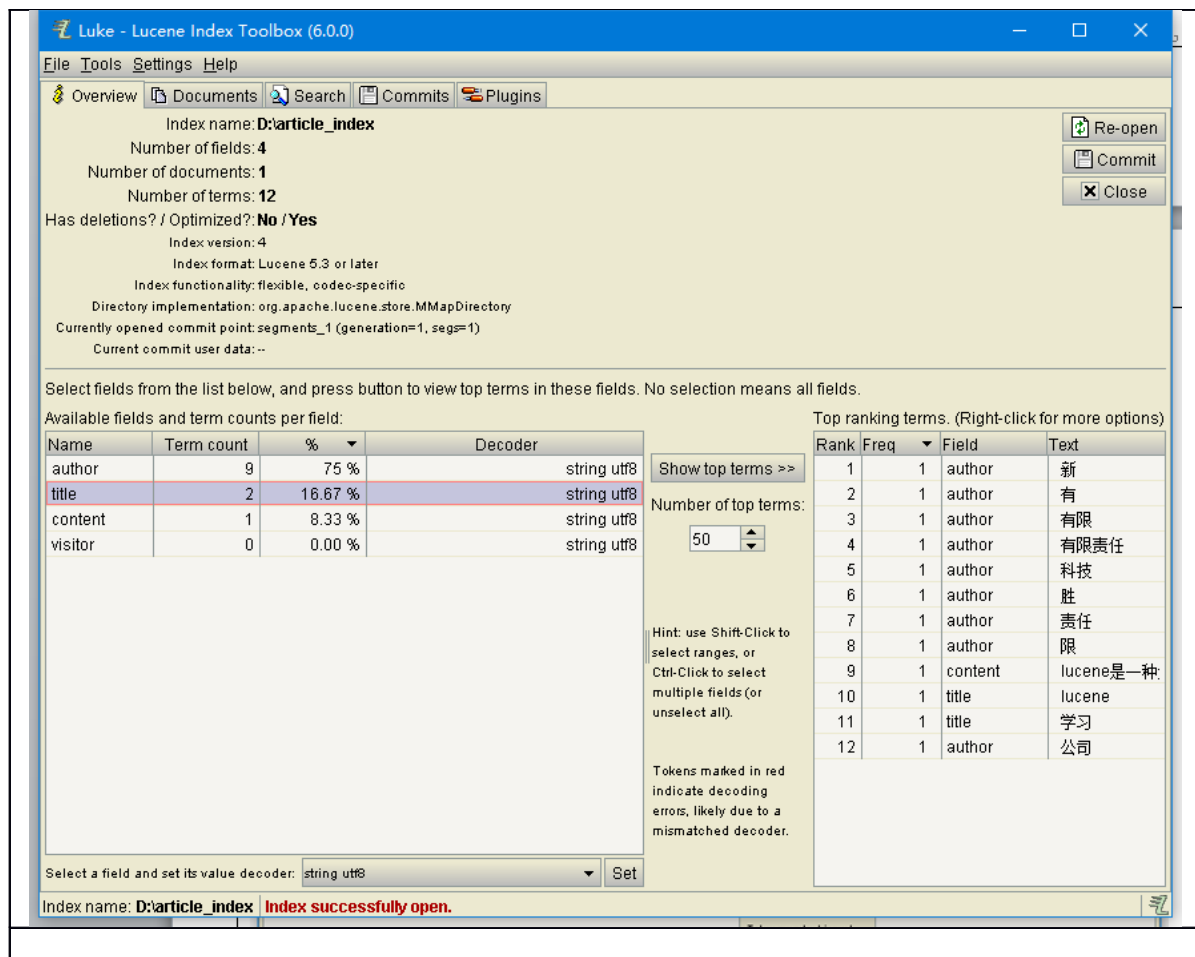
Re-open
 Commit
 Close

Check for more options
 Text

unselect all).
 Tokens marked in red indicate decoding errors, likely due to a mismatched decoder.

Select a field and set its value decoder: string utf8 Set

Index name: ?



4 相关类及方法介绍

域属性有多少个类型？

StringField 文本，TextField 文本区别？

为什么 doc 可以存在索引也可以不存在索引中 Store.YES/NO？

4.A 域属性的类型

- 文本类型
 - TextField StringField
- 数字类型：既不做分词计算，也不做存储，数字特性的数据在索引中单独处理，用来实现范围搜索(价钱范围)

★ 如果某个字段，既需要数字特性进行范围搜索，搜索到 document 之后又要使用这个数字数据--通过一个同名的 StringField 使得该域属性具备 2 个特点。（可范围搜索、可使用）

- IntPoint
- LongPoint
- DoublePoint
- FloatPoint

4.B StringField 文本, TextField 文本区别

- TextField 类型的域属性 value 值会经过分词计算
- StringField 类型的域属性 value 值不会经过分词计算

数据中并不是所有的文本数据, 都需要分词

例如:

uuid: 05e20c1a-0401-4c0a-82ab-6fb0f37db397

address: https://news.sina.com.cn/c/2020-06-05/doc-
iirczymk5454551.shtml /图片 imgurl

4.C Store.YES/NO 的作用

- Store.YES 索引中的文件 document 会保存该属性
- Store.NO 索引中没有这个域值保存在 document

有的数据确实从源数据没有保存到索引数据的必要。因为搜索到数据后没有使用该数据

总结: 存储的域属性, 就是搜索后用到的展示的数据, 不存的域属性就是搜索后用不到的数据

4.D 不存储的数据, 不代表不计算分词 (意义何在? 会建立索引用于搜索)

- doc.add(new TextField("test", "我们都有一个家", Store.NO))

TextField 类型决定计算分词 我们 都有 一个 家

倒排索引表中我们 都有 一个 家指向该 document

但是由于 Store.NO 搜到 document 拿不到“我们都有一个家”这个数据

七 搜索功能

lucene 对于创建的索引可以实现多种搜索功能.在进行搜索时,lucene 总会把搜索条件封装成一个 query 查询对象.通过对查询对象数据的计算最终拿到返回 document 数据.查询对象又可以根据查询条件不同,需求不同使用不同的实现类封装.

1 浅查询与深查询

lucene 支持的这种查询的逻辑,是浅查询/深查询,和查询分页计算有关.

假如查询 page=2 rows=5

1.A 深查询:

直接读取 2*5 条数据,读取分页数据最后一条之前的所有内容,读完了数据在进行分页计算,用到哪些展示哪些,不用的白读了,效率在大数据量时比较低--先读在计算

1.B 浅查询:

利用数据结构(没有类似的数据结构无法实现浅查询),先进行标识,标记,表格的计算,得到最终分页的数据结果,再去读取数据--先计算在读,效率大数据量时高于深查询

2 词项查询

最基本的查询方式,传递的参数就是一个域当中的词语文本,例如 "title":"中国".查询条件进行查询计算,判断倒排表中是否有中国词项(完整匹配),有的话,是否在 title 中存在.最终把匹配结果的 document 集合返回使用.

•搜索的逻辑

- 指向一个索引文件
- 创建一个查询对象 searcher
- 构造一个查询条件 query---TermQuery
- 利用浅查询,获取查询的关键信息 docId 读取数据

```
@Test
public void searchDocument() throws IOException {
    //获取目录
    Path path = Paths.get("D:/article_index");
    FSDirectory fsDirectory=FSDirectory.open(path);
    DirectoryReader reader =
    DirectoryReader.open(fsDirectory);

    //创建查询条件
    Term term=new Term("author","新");
    TermQuery termQuery = new TermQuery(term);
    //创建查询对象
    IndexSearcher searcher = new IndexSearcher(reader);
    TopDocs topDocs = searcher.search(termQuery, 10);
    System.out.println("查询总条数:"+topDocs.totalHits);
    //获取查询的文档数组
    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
    for (ScoreDoc scoreDoc : scoreDocs) {
        //获取文档对象
```

```

        int doc = scoreDoc.doc;
        Document doc1 = searcher.doc(doc);
        System.out.println(doc1.get("title"));
    }
}

```

3 多域查询

如果查询需求,是判断多个域中是否有一些文本词项,可以使用多域查询.例如:

title:美国, content:美国,publisher:美国,title:中国;底层就是一堆 TermQuery (词项查询) 并集查询.MultiFieldQuery;

```

public void searchDocByMultiQuery() throws IOException,
ParseException {
    //获取目录
    Path path = Paths.get("D:/article_index");
    FSDirectory fsDirectory=FSDirectory.open(path);
    DirectoryReader reader =
DirectoryReader.open(fsDirectory);

    //创建多域查询条件
    String[] fields={"title","content"};
    MultiFieldQueryParser queryParser = new
MultiFieldQueryParser(fields, new IKAnalyzer6x());
    Query query = queryParser.parse("中国美国");
    //底层实现逻辑
    //ik 分词器对"中国美国"-->"中国""美国""国美"
    //和 string[] 中的域名做 排列组合 title:中国,title:美国,title:
国美
    //content:中国,content:美国,content:国美 将每个结果按词项查询单
独搜索计算得到一个 document 结果集

    //创建查询对象
    IndexSearcher searcher = new IndexSearcher(reader);
    TopDocs topDocs = searcher.search(query, 10); //10 表示查询
10 条数据
    System.out.println("查询总条数:"+topDocs.totalHits);
    //获取查询的文档数组
    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
    for (ScoreDoc scoreDoc : scoreDocs) {
        //获取文档对象
        int doc = scoreDoc.doc;
        Document doc1 = searcher.doc(doc);
        System.out.println(doc1.get("title"));
    }
}

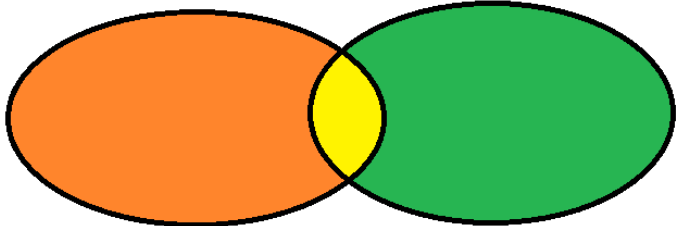
```

4 布尔查询

各种查询方式中,应用最广的一种.组合其他的所有查询条件,包括他自己.可以将任何的 query 对象做成一个布尔查询的子条件,从而决定最终的结果集的逻辑关系.

booleanQuery

query1 query2



```
@Test
public void searchDocByBooleanQuery() throws IOException {
    //获取目录
    Path path = Paths.get("D:/article_index");
    FSDirectory fsDirectory=FSDirectory.open(path);
    DirectoryReader reader = DirectoryReader.open(fsDirectory);

    //布尔查询子条件
    TermQuery termQuery = new TermQuery(new Term("title", "美国"));
    TermQuery termQuery1 = new TermQuery(new Term("content", "中
国"));
    //封装子条件
    BooleanClause booleanClause = new BooleanClause(termQuery,
BooleanClause.Occur.MUST);
    BooleanClause booleanClause1 = new BooleanClause(termQuery1,
BooleanClause.Occur.MUST_NOT);
    Query query = new
BooleanQuery.Builder().add(booleanClause).add(booleanClause1).build
();
    //MUST 布尔查询结果必须是子条件的子集
    //MUST_NOT 布尔查询结果必须不是子条件的子集

    //创建查询对象
    IndexSearcher searcher = new IndexSearcher(reader);
    TopDocs topDocs = searcher.search(query, 10); //10 表示查询 10 条数据
    System.out.println("查询总条数:"+topDocs.totalHits);
    //获取查询的文档数组
    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
    for (ScoreDoc scoreDoc : scoreDocs) {
        //获取文档对象
        int doc = scoreDoc.doc;
        Document doc1 = searcher.doc(doc);
        System.out.println(doc1.get("title"));
    }
}
```

5 范围查询

可以对域属性中,具备**Point 类型 (域属性为数值类型) 的属性值进行范围搜索.

```
@Test
public void searchDocByRangeQuery() throws IOException {
    //获取目录
    Path path = Paths.get("D:/article_index");
    FSDirectory fsDirectory=FSDirectory.open(path);
    DirectoryReader reader = DirectoryReader.open(fsDirectory);

    //范围查询条件访客数量 10-20
    Query query = IntPoint.newRangeQuery("visitor", 10, 20);

    //创建查询对象
    IndexSearcher searcher = new IndexSearcher(reader);
    TopDocs topDocs = searcher.search(query, 10); //10 表示查询 10 条数据
    System.out.println("查询总条数:"+topDocs.totalHits);
    //获取查询的文档数组
    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
    for (ScoreDoc scoreDoc : scoreDocs) {
        //获取文档对象
        int doc = scoreDoc.doc;
        Document doc1 = searcher.doc(doc);
        System.out.println(doc1.get("title"));
    }
}
```

6 模糊查询

```
Query query=new FuzzyQuery(new Term("name","tramp"));
```

查询时,如果词项和搜索字符串"tramp"是类似的,name:trump,name:trimp 等都会查询到.

中文模糊 日/曰 品/晶 日/目

7 通配查询

```
Query query=new WildcartQuery(new Term("name","王?花"));
```

?可以代表任意一个字符数据中 name:王小花,name:王美花,name:王翠花属于匹配范围之内

八 lucene 实现功能面临的问题

- 索引文件不是分布式,只能保存在服务器本地,无法处理海量数据
- 对于非java 语言编程开发者,无法实现通过 lucene 完成这一套结构