

IKAnalyzer 中文分词器

V3.X 使用手册

目录

1.IK Analyzer 3.0 介绍.....	2
2.使用指南.....	4
3.词表扩展.....	11
4 针对 solr 的分词器应用扩展.....	13
5.关于作者.....	14

1.IK Analyzer 3.0 介绍

IK Analyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包。从 2006 年 12 月推出 1.0 版开始， IKAnalyzer 已经推出了 3 个大版本。最初，它是以开源项目 Luence 为应用主体的，结合词典分词和文法分析算法的中文分词组件。新版本的 IK Analyzer 3.0 则发展为面向 Java 的公用分词组件，独立于 Lucene 项目，同时提供了对 Lucene 的默认优化实现。

1.1 IK Analyzer 3.0 结构设计



1.2 IK Analyzer 3.0 特性

- 采用了特有的“正向迭代最细粒度切分算法”，具有 60 万字/秒的高速处理能力。
- 采用了多子处理器分析模式，支持：英文字母（IP 地址、Email、URL）、数字（日期，常用中文数量词，罗马数字，科学计数法），中文词汇（姓名、地名处理）等分词处理。
- 优化的词典存储，更小的内存占用。支持用户词典扩展定义
- 针对 Lucene 全文检索优化的查询分析器 IKQueryParser(作者吐血推荐)；采用歧义分析算法优化查询关键字的搜索排列组合，能极大的提高 Lucene 检索的命中率。

1.3 分词效果示例

文本原文 1:

IKAnalyzer 是一个开源的，基于 java 语言开发的轻量级的中文分词工具包。从 2006 年 12 月推出 1.0 版开始， IKAnalyzer 已经推出了 3 个大版本。

分词结果:

ikanalyzer | 是 | 一个 | 一 | 个 | 开源 | 的 | 基于 | java | 语言 | 开发 | 的 | 轻量
级 | 量级 | 的 | 中文 | 分词 | 工具包 | 工具 | 从 | 2006 | 年 | 12 | 月 | 推出 | 1.0
| 版 | 开始 | ikanalyzer | 已经 | 推出 | 出了 | 3 | 个大 | 个 | 版本

文本原文 2:

永和服装饰品有限公司

分词结果:

永和 | 和服 | 服装 | 装饰品 | 装饰 | 饰品 | 有限 | 公司

文本原文 3:

作者博客 : linliangyi2007.javaeye.com 电子邮件 : linliangyi2005@gmail.com

分词结果:

作者 | 博客 | linliangyi2007.javaeye.com | [linliangyi](#) | 2007 | [javaeye](#) | [com](#) | 电子邮件 | 邮件地址 | linliangyi2005@gmail.com | [linliangyi](#) | 2005 | [gmail](#) | [com](#)

2.使用指南

2.1 下载地址

GoogleCode 开源项目 : <http://code.google.com/p/ik-analyzer/>

GoogleCode SVN 下载 : <http://ik-analyzer.googlecode.com/svn/trunk/>

2.2 安装部署

IK Analyzer 安装包包含 :

1. 《IKAnalyzer 中文分词器 V3.X 使用手册》(即本文档)
2. IKAnalyzer3.X.jar
3. IKAnalyzer.cfg.xml

它的安装部署十分简单 ,将IKAnalyzer3.X.jar 部署于项目的 lib 目录中 ,IKAnalyzer.cfg.xml 文件放置在代码根目录(对于web项目 ,通常是 WEB-INF/classes 目录 ,同 hibernate、log4j 等配置文件相同) 下即可。

2.3 Lucene 用户快速入门

代码样例

IKAnalyzerDemo

```
/**
 * IK Analyzer Demo
 * @param args
 */
import java.io.IOException;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.LockObtainFailedException;
import org.apache.lucene.store.RAMDirectory;
//引用IKAnalyzer3.0的类
import org.wltea.analyzer.lucene.IKAnalyzer;
import org.wltea.analyzer.lucene.IKQueryParser;
import org.wltea.analyzer.lucene.IKSimilarity;

/**
 * @author linly
 */
public class IKAnalyzerDemo {

    public static void main(String[] args) {
        //Lucene Document的域名
        String fieldName = "text";
        //检索内容
        String text = "IK Analyzer是一个结合词典分词和语法分词的中文分词开源工具包。它使用了全新的正向迭代最细粒度切分算法。";

        //实例化IKAnalyzer分词器
        Analyzer analyzer = new IKAnalyzer();

        Directory directory = null;
        IndexWriter iwriter = null;
```

```

IndexSearcher isearcher = null;
try {
    //建立内存索引对象
    directory = new RAMDirectory();
    iwriter = new IndexWriter(directory, analyzer, true ,
IndexWriter.MaxFieldLength.LIMITED);
    Document doc = new Document();
    doc.add(new Field(fieldName, text, Field.Store.YES,
Field.Index.ANALYZED));
    iwriter.addDocument(doc);
    iwriter.close();

    //实例化搜索器
    isearcher = new IndexSearcher(directory);
    //在索引器中使用IKSimilarity相似度评估器
    isearcher.setSimilarity(new IKSimilarity());

    String keyword = "中文分词工具包";

    //使用IKQueryParser查询分析器构造Query对象
    Query query = IKQueryParser.parse(fieldName, keyword);

    //搜索相似度最高的5条记录
    TopDocs topDocs = isearcher.search(query , 5);
    System.out.println("命中: " + topDocs.totalHits);
    //输出结果
    ScoreDoc[] scoreDocs = topDocs.scoreDocs;
    for (int i = 0; i < topDocs.totalHits; i++){
        Document targetDoc = isearcher.doc(scoreDocs[i].doc);
        System.out.println("内容: " + targetDoc.toString());
    }

} catch (CorruptIndexException e) {
    e.printStackTrace();
} catch (LockObtainFailedException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} finally{
    if(isearcher != null){
        try {
            isearcher.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
        }
    }
    if(directory != null){
        try {
            directory.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}
```

执行结果：

命中：1

内容：Document<stored/uncompressed,indexed,tokenized<text:IK Analyzer是一个结合词典分词和文法分词的中文分词开源工具包。它使用了全新的正向迭代最细粒度切分算法。>>

2.4 关键 API 说明

● 类 org.wltea.analyzer.lucene.IKAnalyzer

说明：IK 分词器的主类，是 IK 分词器的 Lucene Analyzer 类实现。

该类使用方法请参考 “代码样例” 章节

■ public IKAnalyzer()

说明：构造函数，默认实现最细粒度切分算法

■ public IKAnalyzer(boolean isMaxWordLength)

说明：新构造函数，从版本 V3.1.1 起

参数 1 ：boolean isMaxWordLength ， 当为 true 时，分词器进行最大词长切

分 ；当为 false 时，分词器进行最细粒度切分。

- 类 org.wltea.analyzer.lucene.IKQueryParser

- public static Query parse(String field , String query) throws IOException

说明：单条件,单 Field 查询分析

参数 1 ：String field, 查询的目标域名称

参数 2 ：String query , 查询的关键字

返回值：构造一个单条件,单 Field 查询器

- public static Query parseMultiField(String[] fields , String query) throws IOException

说明：多 Field,单条件查询分析

参数 1 ：String[] fields, 多个查询的目标域名称的数组

参数 2 ：String query , 查询的关键字

返回值：构造一个多 Field,单条件的查询器

- public static Query parseMultiField(String[] fields , String query , BooleanClause.Occur[] flags) throws IOException

说明：多 Field,单条件,多 Occur 查询分析

参数 1 ：String[] fields, 多个查询的目标域名称的数组

参数 2 ：String query , 查询的关键字

参数 3 ：BooleanClause.Occur[] flags , 查询条件的组合方式 (Or/And)

返回值：构造一个多 Field,单条件,多 Occur 的查询器

- public static Query parseMultiField(String[] fields , String[] queries) throws IOException

说明：多 Field , 多条件查询分析

参数 1 : String[] fields, 多个查询的目标域名称的数组

参数 2 : String[] queries, 对应多个查询域的关键字数组

返回值 : 构造一个多 Field, 多条件的查询器

- public static Query parseMultiField(String[] fields , String[] queries , BooleanClause.Occur[] flags) throws IOException

说明 : 多 Field,多条件,多 Occur 查询

参数 1 : String[] fields, 多个查询的目标域名称的数组

参数 2 : String[] queries, 对应多个查询域的关键字数组

参数 3 : BooleanClause.Occur[] flags , 查询条件的组合方式 (Or/And)

返回值 : 构造一个多 Field, 多条件, 多 Occur 的查询器

- 类 org.wltea.analyzer.lucene.IKSimilarity

说明 : IKAnalyzer 的相似度评估器。该类重载了 DefaultSimilarity 的 coord 方法 , 提高词元命中个数在相似度比较中的权重影响 , 即 , 当有多个词元得到匹配时 , 文档的相似度将提高。

该类使用方法请参考 “代码样例” 章节

- 类 org.wltea.analyzer.IKSegmentation

说明 : 这是 IK 分词器的核心类。它是真正意义上的分词器实现。IKAnalyzer 的 3.0 版本有别于之前的版本 , 它是一个可以独立于 Lucene 的 Java 分词器实现。当您需要 在 Lucene 以外的环境中单独使用 IK 中文分词 组件时 , IKSegmentation 正是您要找的。

■ `public IKSegmentation(Reader input)`

说明：IK 主分词器构造函数，默认实现最细粒度切分

参数 1：Reader input，字符输入读取

■ `public IKSegmentation(Reader input, boolean isMaxWordLength)`

说明：IK 主分词器新构造函数，从版本 V3.1.1 起

参数 1：Reader input，字符输入读取

参数 2：boolean isMaxWordLength，当为 true 时，分词器进行最大词长切分；

当为 false 时，分词器进行最细粒度切分。

■ `public Lexeme next() throws IOException`

说明：读取分词器切分出的下一个语义单元，如果返回 null，表示分词器已经结束。

返回值：Lexeme 语义单元对象，即相当于 Lucene 的词元对象 Token

● 类 `org.wltea.analyzer.Lexeme`

说明：这是 IK 分词器的语义单元对象，相当于 Lucene 中的 Token 词元对象。由于 3.0 版本被设计为独立于 Lucene 的 Java 分词器实现，因此它需要 Lexeme 来代表分词的结果。

■ `public int getBeginPosition()`

说明：获取语义单元的起始字符在文本中的位置

返回值：int，语义单元相对于文本的绝对起始位置

■ `public int getEndPosition()`

说明：获取语义单元的结束字符的下一个位置

返回值：int，语义单元相对于文本的绝对终止位置的下一个字符位置

- `public int getLength()`

说明：获取语义单元包含字符串的长度

返回值：int ， 语义单元长度 = `getEndPosition - getBeginPosition`

- `public String getLexemeText()`

说明：获取语义单元包含字符串内容

返回值：String ， 语义单元的实际内容，即分词的结果

3.词表扩展

目前,IK 分词器自带的主词典拥有 27 万左右的汉语单词量。由于作者个人的精力有限，并没有对搜集到的词库进行全范围的筛选、清理。此外，对于分词组件应用场景所涉及的领域的不同，也需要各类专业词库的支持。为此，IK 分词器提供了对词典的扩展支持。

在 IK 分词器 3.1.3 以上版本，同时提供了对用户自定义的停止词（过滤词）的扩展支持。

3.1 基于 API 的词典扩充

IK 分词器支持使用 API 编程模型扩充您的词典和停止词典。如果您的个性化词典是存储于数据库中，这个方式应该对您适用。API 如下：

- 类 `org.wltea.analyzer.dic.Dictionary`

说明：IK 分词器的词典对象。它负责中文词汇的加载，内存管理和匹配检索。

- `public static void loadExtendWords(List<String> extWords)`

说明：加载用户扩展的词汇列表到 IK 的主词典中，增加分词器的可识别词语。

参数 1：List<String> extWords ， 扩展的词汇列表

返回值：无

- `public static void loadExtendStopWords(List<String> extStopWords)`

说明：加载用户扩展的停止词列表，从版本 V3.1.3 起

参数 1：List<String> extStopWords，扩展的停止词列表

返回值：无

3.2 基于配置的词典扩充

IK 分词器还支持通过配置 IKAnalyzer.cfg.xml 文件来扩充您的专有词典以及停止词典（过滤词典）。

1. 部署 IKAnalyzer.cfg.xml

IKAnalyzer.cfg.xml 部署在代码根目录下（对于 web 项目，通常是 WEB-INF/classes 目录）同 hibernate、log4j 等配置文件相同。

2. 词典文件的编辑与部署

分词器的词典文件格式是无 BOM 的 UTF-8 编码的中文文本文件，文件扩展名不限。词典中，每个中文词汇独立占一行，使用\r\n 的 DOS 方式换行。（[注，如果您不了解什么是无 BOM 的 UTF-8 格式，请保证您的词典使用 UTF-8 存储，并在文件的头部添加一空行](#)）。您可以参考分词器源码 org.wltea.analyzer.dic 包下的.dic 文件。

词典文件应部署在 Java 的资源路径下，即 ClassLoader 能够加载的路径中。（[推荐同 IKAnalyzer.cfg.xml 放在一起](#)）

3. IKAnalyzer.cfg.xml 文件的配置

```
<?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">/mydict.dic;
/com/mycompany/dic/mydict2.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">/ext_stopword.dic</entry>
</properties>
```

在配置文件中，用户可一次配置多个词典文件。文件名使用 “;” 号分隔。文件路径为相对 java 包的起始根路径。

4 针对 solr 的分词器应用扩展

IK 分词器 3.1.5 以上版本从 API 层面提供了对 solr 项目扩展。

4.1 TokenizerFactory 接口实现

- 类 org.wltea.analyzer.solr.IKTokenizerFactory

说明：该类继承与 solr1.3 的 BaseTokenizerFactory，是 IK 分词器对 solr1.3 项目 TokenizerFactory 接口的扩展实现。从版本 V3.1.5 起。

属性：isMaxWordLength。该属性决定分词器是否采用最大词语切分。

- 类 org.wltea.analyzer.solr.IKTokenizerFactory14

说明：该类继承与 solr1.4 的 BaseTokenizerFactory，是 IK 分词器对 solr1.4 项目 TokenizerFactory 接口的扩展实现。从版本 V3.1.6 起。

属性：isMaxWordLength。该属性决定分词器是否采用最大词语切分。

4.2 solr1.3 配置样例

使用 IAnalyzer 的配置

```
<schema name="example" version="1.1">

    .....

    <fieldType name="text" class="solr.TextField">
        <analyzer class="org.wltea.analyzer.lucene.IAnalyzer"/>
    </fieldType>

    .....

</schema>
```

使用IKTokenizerFactory的配置

```
<fieldType name="text" class="solr.TextField" >
    <analyzer type="index">
        <tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory"
isMaxWordLength="false"/>

        .....

    </analyzer>
    <analyzer type="query">
        <tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory"
isMaxWordLength="true"/>

        .....

    </analyzer>
</fieldType>
```

4.2 solr1.4 配置样例

在 solr1.4 中，org.apache.solr.analysis.BaseTokenizerFactory 接口做了修改，因此配置有了相应的变化。

使用 IAnalyzer 的配置

使用<analyzer>元素的配置与 solr1.3 相同

```
<schema name="example" version="1.1">
```

```

.....

<fieldType name="text" class="solr.TextField">
  <analyzer class="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>

.....

</schema>

```

使用IKTokenizerFactory的配置

使用< tokenizer>元素的配置与solr1.4是不同的

```

<fieldType name="text" class="solr.TextField" >
  <analyzer type="index">
    <tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory14"
isMaxWordLength="false"/>

    .....

  </analyzer>
  <analyzer type="query">
    <tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory14"
isMaxWordLength="true"/>

    .....

  </analyzer>
</fieldType>

```

5.关于作者

Blog : linliangyi2007.javaeye.com

Email : linliangyi2005@gmail.com

(全文终)