

LUCENE - STOPANALYZER

http://www.tutorialspoint.com/lucene/lucene_stopanalyzer.htm

Copyright © tutorialspoint.com

Introduction

This analyzer works similar to SimpleAnalyzer and remove the common words like 'a','an','the' etc.

Class declaration

Following is the declaration for **org.apache.lucene.analysis.StopAnalyzer** class:

```
public final class StopAnalyzer
    extends StopwordAnalyzerBase
```

Fields

- **static Set<?> ENGLISH_STOP_WORDS_SET** - An unmodifiable set containing some common English words that are not usually useful for searching.

Class constructors

S.N.	Constructor & Description
1	StopAnalyzerVersionmatchVersion Builds an analyzer which removes words in ENGLISH_STOP_WORDS_SET.
2	StopAnalyzerVersionmatchVersion, FilestopwordsFile Builds an analyzer with the stop words from the given file.
3	StopAnalyzerVersionmatchVersion, Readerstopwords Builds an analyzer with the stop words from the given reader.
4	StopAnalyzerVersionmatchVersion, Set < ? > stopWords Builds an analyzer with the stop words from the given set.

Class methods

S.N.	Method & Description
1	protected Reusable Analyzer Base. Token Stream Components create Components StringfieldName, Readerreader Creates a new ReusableAnalyzerBase.TokenStreamComponents used to tokenize all the text in the provided Reader.

Methods inherited

This class inherits methods from the following classes:

- org.apache.lucene.analysis.StopwordAnalyzerBase
- org.apache.lucene.analysis.ReusableAnalyzerBase
- org.apache.lucene.analysis.Analyzer
- java.lang.Object

Usage

```
private void displayTokenUsingStopAnalyzer() throws IOException{
    String text
        = "Lucene is simple yet powerful java based search library.";
    Analyzer analyzer = new StopAnalyzer(Version.LUCENE_36);
    TokenStream tokenStream
        = analyzer.tokenStream(LuceneConstants.CONTENTS,
            new StringReader(text));
    TermAttribute term = tokenStream.addAttribute(TermAttribute.class);
    while(tokenStream.incrementToken()) {
        System.out.print "[" + term.term() + " ] ";
    }
}
```

Example Application

Let us create a test Lucene application to test search using BooleanQuery.

Step	Description
1	Create a project with a name <i>LuceneFirstApplication</i> under a package <i>com.tutorialspoint.lucene</i> as explained in the <i>Lucene - First Application</i> chapter. You can also use the project created in <i>Lucene - First Application</i> chapter as such for this chapter to understand searching process.
2	Create <i>LuceneConstants.java</i> as explained in the <i>Lucene - First Application</i> chapter. Keep rest of the files unchanged.
3	Create <i>LuceneTester.java</i> as mentioned below.
4	Clean and Build the application to make sure business logic is working as per the requirements.

LuceneConstants.java

This class is used to provide various constants to be used across the sample application.

```
package com.tutorialspoint.lucene;

public class LuceneConstants {
    public static final String CONTENTS="contents";
    public static final String FILE_NAME="filename";
    public static final String FILE_PATH="filepath";
    public static final int MAX_SEARCH = 10;
}
```

LuceneTester.java

This class is used to test the searching capability of lucene library.

```
package com.tutorialspoint.lucene;
```

```

import java.io.IOException;
import java.io.StringReader;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.StopAnalyzer;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.tokenattributes.TermAttribute;
import org.apache.lucene.util.Version;

public class LuceneTester {

    public static void main(String[] args) {
        LuceneTester tester;

        tester = new LuceneTester();

        try {
            tester.displayTokenUsingStopAnalyzer();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void displayTokenUsingStopAnalyzer() throws IOException{
        String text
            = "Lucene is simple yet powerful java based search library.";
        Analyzer analyzer = new StopAnalyzer(Version.LUCENE_36);
        TokenStream tokenStream = analyzer.tokenStream(
            LuceneConstants.CONTENTS, new StringReader(text));
        TermAttribute term = tokenStream.addAttribute(TermAttribute.class);
        while(tokenStream.incrementToken()) {
            System.out.print "[" + term.term() + " ] ";
        }
    }
}

```

Running the Program:

Once you are done with creating source, you are ready for this step which is compiling and running your program. To do this, Keep LuceneTester.java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **LuceneTester** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

```

[Lucene] [simple] [yet] [powerful] [java] [based] [search] [library]
Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

```