package com.twitter.search.common.schema;

import java.io.Reader;

import java.text.ParseException;

import java.util.Map;

import com.google.common.base.Splitter;

import com.google.common.collect.Lists;

import com.google.common.collect.Sets;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.CharArraySet;

import org.apache.lucene.analysis.CharFilter;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.analysis.Tokenizer;

import org.apache.lucene.analysis.charfilter.HTMLStripCharFilter;

import org.apache.lucene.analysis.core.WhitespaceAnalyzer;

import org.apache.lucene.analysis.fa.PersianCharFilter;

import org.apache.lucene.analysis.standard.StandardAnalyzer;

import org.apache.lucene.util.Version;

import com.twitter.search.common.schema.thriftjava.ThriftAnalyzer;

import com.twitter.search.common.schema.thriftjava.ThriftClassInstantiater;

import com.twitter.search.common.schema.thriftjava.ThriftCustomAnalyzer;

public class AnalyzerFactory {

private static final Logger LOG = LoggerFactory.getLogger(AnalyzerFactory.class);

private static final String MATCH\_VERSION\_ARG\_NAME = "matchVersion";

private static final String STANDARD\_ANALYZER = "StandardAnalyzer";

private static final String WHITESPACE\_ANALYZER = "WhitespaceAnalyzer";

private static final String SEARCH\_WHITESPACE\_ANALYZER = "SearchWhitespaceAnalyzer";

private static final String HTML\_STRIP\_CHAR\_FILTER = "HTMLStripCharFilter";

private static final String PERSIAN\_CHAR\_FILTER = "PersianCharFilter";

/\*\*

\* Return a Lucene Analyzer based on the given ThriftAnalyzer.

\*/

public Analyzer getAnalyzer(ThriftAnalyzer analyzer) {

if (analyzer.isSetAnalyzer()) {

return resolveAnalyzerClass(analyzer.getAnalyzer());

} else if (analyzer.isSetCustomAnalyzer()) {

return buildCustomAnalyzer(analyzer.getCustomAnalyzer());

}

return new SearchWhitespaceAnalyzer();

}

private Analyzer resolveAnalyzerClass(ThriftClassInstantiater classDef) {

Map<String, String> params = classDef.getParams();

Version matchVersion = Version.LUCENE\_8\_5\_2;

String matchVersionName = getArg(params, MATCH\_VERSION\_ARG\_NAME);

if (matchVersionName != null) {

try {

matchVersion = Version.parse(matchVersionName);

} catch (ParseException e) {

// ignore and use default version

LOG.warn("Unable to parse match version: " + matchVersionName

+ ". Will use default version of 8.5.2.");

}

}

if (classDef.getClassName().equals(STANDARD\_ANALYZER)) {

String stopwords = getArg(params, "stopwords");

if (stopwords != null) {

CharArraySet stopwordSet = new CharArraySet(

Lists.newLinkedList(Splitter.on(",").split(stopwords)),

false);

return new StandardAnalyzer(stopwordSet);

} else {

return new StandardAnalyzer();

}

} else if (classDef.getClassName().equals(WHITESPACE\_ANALYZER)) {

return new WhitespaceAnalyzer();

} else if (classDef.getClassName().equals(SEARCH\_WHITESPACE\_ANALYZER)) {

return new SearchWhitespaceAnalyzer();

}

return null;

}

private Analyzer buildCustomAnalyzer(final ThriftCustomAnalyzer customAnalyzer) {

return new Analyzer() {

@Override

protected TokenStreamComponents createComponents(String fieldName) {

final Tokenizer tokenizer = resolveTokenizerClass(customAnalyzer.getTokenizer());

TokenStream filter = tokenizer;

if (customAnalyzer.isSetFilters()) {

for (ThriftClassInstantiater filterClass : customAnalyzer.getFilters()) {

filter = resolveTokenFilterClass(filterClass, filter);

}

}

return new TokenStreamComponents(tokenizer, filter);

}

};

}

private Tokenizer resolveTokenizerClass(ThriftClassInstantiater classDef) {

return null;

}

private TokenStream resolveTokenFilterClass(ThriftClassInstantiater classDef, TokenStream input) {

return null;

}

private CharFilter resolveCharFilterClass(ThriftClassInstantiater classDef, Reader input) {

if (classDef.getClassName().equals(HTML\_STRIP\_CHAR\_FILTER)) {

String escapedTags = getArg(classDef.getParams(), "excapedTags");

if (escapedTags != null) {

return new HTMLStripCharFilter(input, Sets.newHashSet(Splitter.on(",").split(escapedTags)));

} else {

return new HTMLStripCharFilter(input);

}

} else if (classDef.getClassName().equals(PERSIAN\_CHAR\_FILTER)) {

return new PersianCharFilter(input);

}

throw new ClassNotSupportedException("CharFilter", classDef);

}

private String getArg(Map<String, String> args, String arg) {

if (args == null) {

return null;

}

return args.get(arg);

}

public final class ClassNotSupportedException extends RuntimeException {

private ClassNotSupportedException(String type, ThriftClassInstantiater classDef) {

super(type + " class with name " + classDef.getClassName() + " currently not supported.");

}

}

}