package com.twitter.search.earlybird.search.relevance.scoring;

import java.io.IOException;

import java.util.Objects;

import java.util.Set;

import javax.annotation.Nullable;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.index.Term;

import org.apache.lucene.search.Explanation;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.Scorer;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.results.thriftjava.FieldHitAttribution;

/\*\*

\* A wrapper for a Lucene query which first computes Lucene's query score

\* and then delegates to a {@link ScoringFunction} for final score computation.

\*/

public class RelevanceQuery extends Query {

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

protected final Query luceneQuery;

protected final ScoringFunction scoringFunction;

// True when the lucene query's score should be ignored for debug explanations.

protected final boolean ignoreLuceneQueryScoreExplanation;

public RelevanceQuery(Query luceneQuery, ScoringFunction scoringFunction) {

this(luceneQuery, scoringFunction, false);

}

public RelevanceQuery(Query luceneQuery,

ScoringFunction scoringFunction,

boolean ignoreLuceneQueryScoreExplanation) {

this.luceneQuery = luceneQuery;

this.scoringFunction = scoringFunction;

this.ignoreLuceneQueryScoreExplanation = ignoreLuceneQueryScoreExplanation;

}

public ScoringFunction getScoringFunction() {

return scoringFunction;

}

public Query getLuceneQuery() {

return luceneQuery;

}

@Override

public Query rewrite(IndexReader reader) throws IOException {

Query rewritten = luceneQuery.rewrite(reader);

if (rewritten == luceneQuery) {

return this;

}

return new RelevanceQuery(rewritten, scoringFunction, ignoreLuceneQueryScoreExplanation);

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost)

throws IOException {

Weight luceneWeight = luceneQuery.createWeight(searcher, scoreMode, boost);

if (luceneWeight == null) {

return null;

}

return new RelevanceWeight(searcher, luceneWeight);

}

public class RelevanceWeight extends Weight {

private final Weight luceneWeight;

public RelevanceWeight(IndexSearcher searcher, Weight luceneWeight) {

super(RelevanceQuery.this);

this.luceneWeight = luceneWeight;

}

@Override

public void extractTerms(Set<Term> terms) {

this.luceneWeight.extractTerms(terms);

}

@Override

public Explanation explain(LeafReaderContext context, int doc) throws IOException {

return explain(context, doc, null);

}

/\*\*

\* Returns an explanation of the scoring for the given document.

\*

\* @param context The context of the reader that returned this document.

\* @param doc The document.

\* @param fieldHitAttribution Per-hit field attribution information.

\* @return An explanation of the scoring for the given document.

\*/

public Explanation explain(LeafReaderContext context, int doc,

@Nullable FieldHitAttribution fieldHitAttribution) throws IOException {

Explanation luceneExplanation = Explanation.noMatch("LuceneQuery explain skipped");

if (!ignoreLuceneQueryScoreExplanation) {

// get Lucene score

try {

luceneExplanation = luceneWeight.explain(context, doc);

} catch (Exception e) {

// We sometimes see exceptions resulting from term queries that do not store

// utf8-text, which TermQuery.toString() assumes. Catch here and allow at least

// scoring function explanations to be returned.

LOG.error("Exception in explain", e);

luceneExplanation = Explanation.noMatch("LuceneQuery explain failed");

}

}

Explanation scoringFunctionExplanation;

scoringFunction.setFieldHitAttribution(fieldHitAttribution);

scoringFunctionExplanation = scoringFunction.explain(

context.reader(), doc, luceneExplanation.getValue().floatValue());

// just add a wrapper for a better structure of the final explanation

Explanation luceneExplanationWrapper = Explanation.match(

luceneExplanation.getValue(), "LuceneQuery", luceneExplanation);

return Explanation.match(scoringFunctionExplanation.getValue(), "RelevanceQuery",

scoringFunctionExplanation, luceneExplanationWrapper);

}

@Override

public Scorer scorer(LeafReaderContext context) throws IOException {

return luceneWeight.scorer(context);

}

@Override

public boolean isCacheable(LeafReaderContext ctx) {

return luceneWeight.isCacheable(ctx);

}

}

@Override

public int hashCode() {

return (luceneQuery == null ? 0 : luceneQuery.hashCode())

+ (scoringFunction == null ? 0 : scoringFunction.hashCode()) \* 13;

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof RelevanceQuery)) {

return false;

}

RelevanceQuery query = RelevanceQuery.class.cast(obj);

return Objects.equals(luceneQuery, query.luceneQuery)

&& Objects.equals(scoringFunction, query.scoringFunction);

}

@Override

public String toString(String field) {

return "RelevanceQuery[q=" + luceneQuery.toString(field) + "]";

}

}