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

import com.google.common.base.Preconditions;

import org.apache.lucene.search.Query;

import com.twitter.search.common.search.TerminationTracker;

import com.twitter.search.earlybird.QualityFactor;

import com.twitter.search.earlybird.search.SearchRequestInfo;

import com.twitter.search.earlybird.thrift.ThriftSearchQuery;

import com.twitter.search.earlybird.thrift.ThriftSearchRelevanceOptions;

import com.twitter.search.earlybird.thrift.ThriftSearchResultMetadataOptions;

public class RelevanceSearchRequestInfo extends SearchRequestInfo {

private final ThriftSearchRelevanceOptions relevanceOptions;

public RelevanceSearchRequestInfo(

ThriftSearchQuery searchQuery, Query query,

TerminationTracker terminationTracker, QualityFactor qualityFactor) {

super(addResultMetadataOptionsIfUnset(searchQuery), query, terminationTracker, qualityFactor);

this.relevanceOptions = searchQuery.getRelevanceOptions();

}

private static ThriftSearchQuery addResultMetadataOptionsIfUnset(ThriftSearchQuery searchQuery) {

if (!searchQuery.isSetResultMetadataOptions()) {

searchQuery.setResultMetadataOptions(new ThriftSearchResultMetadataOptions());

}

return searchQuery;

}

@Override

protected int calculateMaxHitsToProcess(ThriftSearchQuery thriftSearchQuery) {

ThriftSearchRelevanceOptions searchRelevanceOptions = thriftSearchQuery.getRelevanceOptions();

// Don't use the value from the ThriftSearchQuery object if one is provided in the

// relevance options

int requestedMaxHitsToProcess = searchRelevanceOptions.isSetMaxHitsToProcess()

? searchRelevanceOptions.getMaxHitsToProcess()

: super.calculateMaxHitsToProcess(thriftSearchQuery);

return qualityFactorMaxHitsToProcess(getNumResultsRequested(), requestedMaxHitsToProcess);

}

public ThriftSearchRelevanceOptions getRelevanceOptions() {

return this.relevanceOptions;

}

/\*\*

\* Reduces maxHitsToProcess based on quality factor. Never reduces it beyond

\* numResults.

\* @param numResults

\* @param maxHitsToProcess

\* @return Reduced maxHitsToProcess.

\*/

public int qualityFactorMaxHitsToProcess(int numResults, int maxHitsToProcess) {

Preconditions.checkNotNull(qualityFactor);

// Do not quality factor if there is no lower bound on maxHitsToProcess.

if (numResults > maxHitsToProcess) {

return maxHitsToProcess;

}

double currentQualityFactor = qualityFactor.get();

return Math.max(numResults, (int) (currentQualityFactor \* maxHitsToProcess));

}

}