package com.twitter.search.common.query;

import java.util.List;

import java.util.Map;

import java.util.function.BiFunction;

import java.util.function.Function;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.search.Query;

/\*\*

\* Not threadsafe, but should be reused across different queries unless the size of the existing

\* one is too small for a new huge serialized query.

\*/

public class HitAttributeCollector {

private final List<FieldRankHitInfo> hitInfos = Lists.newArrayList();

private final BiFunction<Integer, Integer, FieldRankHitInfo> hitInfoSupplier;

private int docBase = 0;

public HitAttributeCollector() {

this.hitInfoSupplier = FieldRankHitInfo::new;

}

/\*\*

\* Constructs a new {@code HitAttributionCollector} with the specified {@code FieldRankHitInfo}

\* supplier.

\*

\* @param hitInfoSupplier function to supply a {@code FieldRankHitInfo} instance

\*/

public HitAttributeCollector(BiFunction<Integer, Integer, FieldRankHitInfo> hitInfoSupplier) {

this.hitInfoSupplier = hitInfoSupplier;

}

/\*\*

\* Creates a new IdentifiableQuery for the given query, fieldId and rank, and "registers"

\* the fieldId and the rank with this collector.

\*

\* @param query the query to be wrapped.

\* @param fieldId the ID of the field to be searched.

\* @param rank The rank of this query.

\* @return A new IdentifiableQuery instance for the given query, fieldId and rank.

\*/

public IdentifiableQuery newIdentifiableQuery(Query query, int fieldId, int rank) {

FieldRankHitInfo fieldRankHitInfo = hitInfoSupplier.apply(fieldId, rank);

hitInfos.add(fieldRankHitInfo);

return new IdentifiableQuery(query, fieldRankHitInfo, this);

}

public void clearHitAttributions(LeafReaderContext ctx, FieldRankHitInfo hitInfo) {

docBase = ctx.docBase;

hitInfo.resetDocId();

}

public void collectScorerAttribution(int docId, FieldRankHitInfo hitInfo) {

hitInfo.setDocId(docId + docBase);

}

/\*\*

\* This method should be called when a global hit occurs.

\* This method returns hit attribution summary for the whole query tree.

\* This supports getting hit attribution for only the curDoc.

\*

\* @param docId docId passed in for checking against curDoc.

\* @return Returns a map from node rank to a set of matching field IDs. This map does not contain

\* entries for ranks that did not hit at all.

\*/

public Map<Integer, List<Integer>> getHitAttribution(int docId) {

return getHitAttribution(docId, (fieldId) -> fieldId);

}

/\*\*

\* This method should be called when a global hit occurs.

\* This method returns hit attribution summary for the whole query tree.

\* This supports getting hit attribution for only the curDoc.

\*

\* @param docId docId passed in for checking against curDoc.

\* @param fieldIdFunc The mapping of field IDs to objects of type T.

\* @return Returns a map from node rank to a set of matching objects (usually field IDs or names).

\* This map does not contain entries for ranks that did not hit at all.

\*/

public <T> Map<Integer, List<T>> getHitAttribution(int docId, Function<Integer, T> fieldIdFunc) {

int key = docId + docBase;

Map<Integer, List<T>> hitMap = Maps.newHashMap();

// Manually iterate through all hitInfos elements. It's slightly faster than using an Iterator.

for (FieldRankHitInfo hitInfo : hitInfos) {

if (hitInfo.getDocId() == key) {

int rank = hitInfo.getRank();

List<T> rankHits = hitMap.computeIfAbsent(rank, k -> Lists.newArrayList());

T fieldDescription = fieldIdFunc.apply(hitInfo.getFieldId());

rankHits.add(fieldDescription);

}

}

return hitMap;

}

}