package com.twitter.search.common.query;

import java.util.List;

import java.util.Map;

import java.util.function.Function;

import com.google.common.collect.Maps;

import com.twitter.search.queryparser.query.Query;

import static com.twitter.search.common.query.FieldRankHitInfo.UNSET\_DOC\_ID;

/\*\*

\* Generic helper class containing the data needed to set up and collect field hit attributions.

\*/

public class HitAttributeHelper implements HitAttributeProvider {

private final HitAttributeCollector collector;

private final Function<Integer, String> fieldIdsToFieldNames;

// This is a mapping of type T query nodes to rank id

private final Map<Query, Integer> nodeToRankMap;

// This is meant to expand individual Query nodes into multiple ranks,

// for example, expanding a multi\_term\_disjunction to include a rank for each disjunction value.

private final Map<Query, List<Integer>> expandedNodeToRankMap;

// A single-entry cache for hit attribution, so we can reuse the immediate result. Will be used

// only when lastDocId matches

private ThreadLocal<Map<Integer, List<String>>> lastHitAttrHolder = new ThreadLocal<>();

private ThreadLocal<Integer> lastDocIdHolder = ThreadLocal.withInitial(() -> UNSET\_DOC\_ID);

protected HitAttributeHelper(

HitAttributeCollector collector,

Function<Integer, String> fieldIdsToFieldNames,

Map<Query, Integer> nodeToRankMap,

Map<Query, List<Integer>> expandedNodeToRankMap) {

this.collector = collector;

this.fieldIdsToFieldNames = fieldIdsToFieldNames;

this.nodeToRankMap = nodeToRankMap;

this.expandedNodeToRankMap = expandedNodeToRankMap;

}

/\*\*

\* Constructs a new {@code HitAttributeHelper} with the specified {@code HitAttributeCollector}

\* instance and fields.

\*

\* @param collector a collector instance

\* @param fieldIdsToFieldNames a list of field names indexed by id

\*/

public HitAttributeHelper(HitAttributeCollector collector, String[] fieldIdsToFieldNames) {

this(collector,

(fieldId) -> fieldIdsToFieldNames[fieldId],

Maps.newHashMap(),

Maps.newHashMap());

}

public HitAttributeCollector getFieldRankHitAttributeCollector() {

return collector;

}

/\*\*

\* Returns hit attribution information indexed by node rank

\*

\* @param docId the document id

\* @return a mapping from the query's node rank to a list of field names that were hit.

\*/

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

// check cache first so we don't have to recompute the same thing.

if (lastDocIdHolder.get() == docId) {

return lastHitAttrHolder.get();

}

lastDocIdHolder.set(docId);

Map<Integer, List<String>> hitAttribution =

collector.getHitAttribution(docId, fieldIdsToFieldNames);

lastHitAttrHolder.set(hitAttribution);

return hitAttribution;

}

/\*\*

\* Adds a new node and its respective rank to the helper's node-to-rank map

\* Will throw an exception if attempting to add/update an existing node

\*

\* @param node the query node

\* @param rank the rank associated with the node

\*/

public void addNodeRank(Query node, int rank) {

// if there are two of the same terms, just map them to the first rank, they should get the same

// hits back

if (!nodeToRankMap.containsKey(node)) {

nodeToRankMap.put(node, rank);

}

}

public Map<Query, Integer> getNodeToRankMap() {

return nodeToRankMap;

}

public Map<Query, List<Integer>> getExpandedNodeToRankMap() {

return expandedNodeToRankMap;

}

}