package com.twitter.search.earlybird.queryparser;

import javax.annotation.Nullable;

import com.google.common.base.Optional;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.constants.QueryCacheConstants;

import com.twitter.search.common.query.HitAttributeCollector;

import com.twitter.search.common.query.HitAttributeHelper;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.search.termination.QueryTimeout;

import com.twitter.search.common.search.termination.TerminationQuery;

import com.twitter.search.earlybird.querycache.QueryCacheManager;

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

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

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

import com.twitter.search.queryparser.query.annotation.Annotation;

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

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

public abstract class EarlybirdQueryHelper {

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

/\*\*

\* Wraps the given query and some clauses to exclude antisocial tweets into a conjunction.

\*/

public static Query requireExcludeAntisocial(

Query basicQuery,

QueryCacheManager queryCacheManager) throws QueryParserException {

// Do not set exclude antisocial if they have any other antisocial filters set

Query query = basicQuery;

DetectAntisocialVisitor detectAntisocialVisitor = new DetectAntisocialVisitor();

query.accept(detectAntisocialVisitor);

if (detectAntisocialVisitor.hasAnyAntisocialOperator()) {

return query;

}

// No operator found, force antisocial filter.

if (queryCacheManager.enabled()) {

SearchOperator filter =

new SearchOperator(SearchOperator.Type.CACHED\_FILTER,

QueryCacheConstants.EXCLUDE\_ANTISOCIAL);

query = QueryNodeUtils.appendAsConjunction(query, filter);

} else {

SearchOperator filter = new SearchOperator(SearchOperator.Type.EXCLUDE,

SearchOperatorConstants.ANTISOCIAL);

query = QueryNodeUtils.appendAsConjunction(query, filter);

}

return query;

}

/\*\*

\* Wraps the given query into an equivalent query that will also collect hit attribution data.

\*

\* @param query The original query.

\* @param node The query parser node storing this query.

\* @param fieldInfo The field in which the given query will be searching.

\* @param hitAttributeHelper The helper that will collect all hit attribution data.

\* @return An equivalent query that will also collect hit attribution data.

\*/

public static final org.apache.lucene.search.Query maybeWrapWithHitAttributionCollector(

org.apache.lucene.search.Query query,

@Nullable com.twitter.search.queryparser.query.Query node,

Schema.FieldInfo fieldInfo,

@Nullable HitAttributeHelper hitAttributeHelper) {

// Prevents lint error for assigning to a function parameter.

org.apache.lucene.search.Query luceneQuery = query;

if (hitAttributeHelper != null && node != null) {

Optional<Annotation> annotation = node.getAnnotationOf(Annotation.Type.NODE\_RANK);

if (annotation.isPresent()) {

Integer nodeRank = (Integer) annotation.get().getValue();

luceneQuery = wrapWithHitAttributionCollector(

luceneQuery,

fieldInfo,

nodeRank,

hitAttributeHelper.getFieldRankHitAttributeCollector());

}

}

return luceneQuery;

}

/\*\*

\* Wraps the given query into an equivalent query that will also collect hit attribution data.

\*

\* @param query The original query.

\* @param nodeRank The rank of the given query in the overall request query.

\* @param fieldInfo The field in which the given query will be searching.

\* @param hitAttributeHelper The helper that will collect all hit attribution data.

\* @return An equivalent query that will also collect hit attribution data.

\*/

public static final org.apache.lucene.search.Query maybeWrapWithHitAttributionCollector(

org.apache.lucene.search.Query query,

int nodeRank,

Schema.FieldInfo fieldInfo,

@Nullable HitAttributeHelper hitAttributeHelper) {

org.apache.lucene.search.Query luceneQuery = query;

if (hitAttributeHelper != null && nodeRank != -1) {

Preconditions.checkArgument(nodeRank > 0);

luceneQuery = wrapWithHitAttributionCollector(

luceneQuery, fieldInfo, nodeRank, hitAttributeHelper.getFieldRankHitAttributeCollector());

}

return luceneQuery;

}

private static final org.apache.lucene.search.Query wrapWithHitAttributionCollector(

org.apache.lucene.search.Query luceneQuery,

Schema.FieldInfo fieldInfo,

int nodeRank,

HitAttributeCollector hitAttributeCollector) {

Preconditions.checkNotNull(fieldInfo,

"Tried collecting hit attribution for unknown field: " + fieldInfo.getName()

+ " luceneQuery: " + luceneQuery);

return hitAttributeCollector.newIdentifiableQuery(

luceneQuery, fieldInfo.getFieldId(), nodeRank);

}

/\*\*

\* Returns a query equivalent to the given query, and with the given timeout enforced.

\*/

public static org.apache.lucene.search.Query maybeWrapWithTimeout(

org.apache.lucene.search.Query query,

QueryTimeout timeout) {

if (timeout != null) {

return new TerminationQuery(query, timeout);

}

return query;

}

/\*\*

\* Returns a query equivalent to the given query, and with the given timeout enforced. If the

\* given query is negated, it is returned without any modifications.

\*/

public static org.apache.lucene.search.Query maybeWrapWithTimeout(

org.apache.lucene.search.Query query,

@Nullable com.twitter.search.queryparser.query.Query node,

QueryTimeout timeout) {

// If the node is looking for negation of something, we don't want to include it in node-level

// timeout checks. In general, nodes keep track of the last doc seen, but non-matching docs

// encountered by "must not occur" node do not reflect overall progress in the index.

if (node != null && node.mustNotOccur()) {

return query;

}

return maybeWrapWithTimeout(query, timeout);

}

}