package com.twitter.search.common.util.earlybird;

import java.util.List;

import java.util.Map;

import javax.annotation.Nullable;

import com.google.common.base.Function;

import com.google.common.base.Predicate;

import com.google.common.base.Predicates;

import com.google.common.collect.Iterables;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

import com.twitter.search.common.relevance.ranking.ActionChain;

import com.twitter.search.common.relevance.ranking.filters.ExactDuplicateFilter;

import com.twitter.search.common.relevance.text.VisibleTokenRatioNormalizer;

import com.twitter.search.common.runtime.ActionChainDebugManager;

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

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

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

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

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

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

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

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

/\*\*

\* ThriftSearchResultUtil contains some simple static methods for constructing

\* ThriftSearchResult objects.

\*/

public final class ThriftSearchResultUtil {

private ThriftSearchResultUtil() { }

private static final VisibleTokenRatioNormalizer NORMALIZER =

VisibleTokenRatioNormalizer.createInstance();

public static final Function<ThriftSearchResults, Map<ThriftLanguage, Integer>> LANG\_MAP\_GETTER =

searchResults -> searchResults.getLanguageHistogram();

public static final Function<ThriftSearchResults, Map<Long, Integer>> HIT\_COUNTS\_MAP\_GETTER =

searchResults -> searchResults.getHitCounts();

// Some useful Predicates

public static final Predicate<ThriftSearchResult> IS\_OFFENSIVE\_TWEET =

result -> {

if (result != null && result.isSetMetadata()) {

ThriftSearchResultMetadata metadata = result.getMetadata();

return metadata.isIsOffensive();

} else {

return false;

}

};

public static final Predicate<ThriftSearchResult> IS\_TOP\_TWEET =

result -> result != null

&& result.isSetMetadata()

&& result.getMetadata().isSetResultType()

&& result.getMetadata().getResultType() == ThriftSearchResultType.POPULAR;

public static final Predicate<ThriftSearchResult> FROM\_FULL\_ARCHIVE =

result -> result != null

&& result.isSetTweetSource()

&& result.getTweetSource() == ThriftTweetSource.FULL\_ARCHIVE\_CLUSTER;

public static final Predicate<ThriftSearchResult> IS\_FULL\_ARCHIVE\_TOP\_TWEET =

Predicates.and(FROM\_FULL\_ARCHIVE, IS\_TOP\_TWEET);

public static final Predicate<ThriftSearchResult> IS\_NSFW\_BY\_ANY\_MEANS\_TWEET =

result -> {

if (result != null && result.isSetMetadata()) {

ThriftSearchResultMetadata metadata = result.getMetadata();

return metadata.isIsUserNSFW()

|| metadata.isIsOffensive()

|| metadata.getExtraMetadata().isIsSensitiveContent();

} else {

return false;

}

};

/\*\*

\* Returns the number of underlying ThriftSearchResult results.

\*/

public static int numResults(ThriftSearchResults results) {

if (results == null || !results.isSetResults()) {

return 0;

} else {

return results.getResultsSize();

}

}

/\*\*

\* Returns the list of tweet IDs in ThriftSearchResults.

\* Returns null if there's no results.

\*/

@Nullable

public static List<Long> getTweetIds(ThriftSearchResults results) {

if (numResults(results) > 0) {

return getTweetIds(results.getResults());

} else {

return null;

}

}

/\*\*

\* Returns the list of tweet IDs in a list of ThriftSearchResult.

\* Returns null if there's no results.

\*/

public static List<Long> getTweetIds(@Nullable List<ThriftSearchResult> results) {

if (results != null && results.size() > 0) {

return Lists.newArrayList(Iterables.transform(

results,

searchResult -> searchResult.getId()

));

}

return null;

}

/\*\*

\* Given ThriftSearchResults, build a map from tweet ID to the tweets metadata.

\*/

public static Map<Long, ThriftSearchResultMetadata> getTweetMetadataMap(

Schema schema, ThriftSearchResults results) {

Map<Long, ThriftSearchResultMetadata> resultMap = Maps.newHashMap();

if (results == null || results.getResultsSize() == 0) {

return resultMap;

}

for (ThriftSearchResult searchResult : results.getResults()) {

resultMap.put(searchResult.getId(), searchResult.getMetadata());

}

return resultMap;

}

/\*\*

\* Return the total number of facet results in ThriftFacetResults, by summing up the number

\* of facet results in each field.

\*/

public static int numFacetResults(ThriftFacetResults results) {

if (results == null || !results.isSetFacetFields()) {

return 0;

} else {

int numResults = 0;

for (ThriftFacetFieldResults field : results.getFacetFields().values()) {

if (field.isSetTopFacets()) {

numResults += field.topFacets.size();

}

}

return numResults;

}

}

/\*\*

\* Updates the search statistics on base, by adding the corresponding stats from delta.

\*/

public static void incrementCounts(ThriftSearchResults base,

ThriftSearchResults delta) {

if (delta.isSetNumHitsProcessed()) {

base.setNumHitsProcessed(base.getNumHitsProcessed() + delta.getNumHitsProcessed());

}

if (delta.isSetNumPartitionsEarlyTerminated() && delta.getNumPartitionsEarlyTerminated() > 0) {

// This currently used for merging results on a single earlybird, so we don't sum up all the

// counts, just set it to 1 if we see one that was early terminated.

base.setNumPartitionsEarlyTerminated(1);

}

if (delta.isSetMaxSearchedStatusID()) {

long deltaMax = delta.getMaxSearchedStatusID();

if (!base.isSetMaxSearchedStatusID() || deltaMax > base.getMaxSearchedStatusID()) {

base.setMaxSearchedStatusID(deltaMax);

}

}

if (delta.isSetMinSearchedStatusID()) {

long deltaMin = delta.getMinSearchedStatusID();

if (!base.isSetMinSearchedStatusID() || deltaMin < base.getMinSearchedStatusID()) {

base.setMinSearchedStatusID(deltaMin);

}

}

if (delta.isSetScore()) {

if (base.isSetScore()) {

base.setScore(base.getScore() + delta.getScore());

} else {

base.setScore(delta.getScore());

}

}

}

/\*\*

\* Removes the duplicates from the given list of results.

\*

\* @param results The list of ThriftSearchResults.

\* @return The given list with duplicates removed.

\*/

public static List<ThriftSearchResult> removeDuplicates(List<ThriftSearchResult> results) {

ActionChain<ThriftSearchResult> filterChain =

ActionChainDebugManager

.<ThriftSearchResult>createActionChainBuilder("RemoveDuplicatesFilters")

.appendActions(new ExactDuplicateFilter())

.build();

return filterChain.apply(results);

}

/\*\*

\* Returns ranking score from Earlybird shard-based ranking models if any, and 0 otherwise.

\*/

public static double getTweetScore(@Nullable ThriftSearchResult result) {

if (result == null || !result.isSetMetadata() || !result.getMetadata().isSetScore()) {

return 0.0;

}

return result.getMetadata().getScore();

}

}