package com.twitter.search.core.earlybird.facets;

import java.util.HashMap;

import java.util.Map;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import com.twitter.search.common.schema.thriftjava.ThriftNumericType;

import com.twitter.search.core.earlybird.index.inverted.InvertedIndex;

/\*\*

\* A utility class for selecting iterators and label providers

\* for facets.

\*

\*/

public abstract class FacetUtil {

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

private FacetUtil() {

// unused

}

/\*\*

\* A utility method for choosing the right facet label provider based on the EarlybirdFieldType.

\* Takes in a InvertedIndex since some facet label providers are or depend on the inverted

\* index.

\* Should never return null.

\*

\* @param fieldType A FieldType for the facet

\* @param invertedField The inverted index associated with the facet. May be null.

\* @return A non-null FacetLabelProvider

\*/

public static FacetLabelProvider chooseFacetLabelProvider(

EarlybirdFieldType fieldType,

InvertedIndex invertedField) {

Preconditions.checkNotNull(fieldType);

// In the case neither inverted index existing nor using CSF,

// return FacetLabelProvider.InaccessibleFacetLabelProvider to throw exception

// more meaningfully and explicitly.

if (invertedField == null && !fieldType.isUseCSFForFacetCounting()) {

return new FacetLabelProvider.InaccessibleFacetLabelProvider(fieldType.getFacetName());

}

if (fieldType.isUseCSFForFacetCounting()) {

return new FacetLabelProvider.IdentityFacetLabelProvider();

}

IndexedNumericFieldSettings numericSettings = fieldType.getNumericFieldSettings();

if (numericSettings != null && numericSettings.isUseTwitterFormat()) {

if (numericSettings.getNumericType() == ThriftNumericType.INT) {

return new FacetLabelProvider.IntTermFacetLabelProvider(invertedField);

} else if (numericSettings.getNumericType() == ThriftNumericType.LONG) {

return numericSettings.isUseSortableEncoding()

? new FacetLabelProvider.SortedLongTermFacetLabelProvider(invertedField)

: new FacetLabelProvider.LongTermFacetLabelProvider(invertedField);

} else {

Preconditions.checkState(false,

"Should never be reached, indicates incomplete handling of different kinds of facets");

return null;

}

} else {

return invertedField;

}

}

/\*\*

\* Get segment-specific facet label providers based on the schema

\* and on the fieldToInvertedIndexMapping for the segment.

\* These will be used by facet accumulators to get the text of the termIDs

\*

\* @param schema the schema, for info on fields and facets

\* @param fieldToInvertedIndexMapping map of fields to their inverted indices

\* @return facet label provider map

\*/

public static Map<String, FacetLabelProvider> getFacetLabelProviders(

Schema schema,

Map<String, InvertedIndex> fieldToInvertedIndexMapping) {

HashMap<String, FacetLabelProvider> facetLabelProviderBuilder

= new HashMap<>();

for (Schema.FieldInfo fieldInfo : schema.getFacetFields()) {

EarlybirdFieldType fieldType = fieldInfo.getFieldType();

Preconditions.checkNotNull(fieldType);

String fieldName = fieldInfo.getName();

String facetName = fieldType.getFacetName();

InvertedIndex invertedIndex = fieldToInvertedIndexMapping.get(fieldName);

if (invertedIndex == null && !fieldType.isUseCSFForFacetCounting()) {

LOG.warn("No docs in segment had field " + fieldName

+ " indexed for facet " + facetName

+ " so InaccessibleFacetLabelProvider will be provided."

);

}

facetLabelProviderBuilder.put(facetName, Preconditions.checkNotNull(

chooseFacetLabelProvider(fieldType, invertedIndex)));

}

return facetLabelProviderBuilder;

}

}