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

import java.io.IOException;

import java.util.Arrays;

import java.util.Collection;

import java.util.Map;

import com.google.common.collect.Maps;

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

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.DataSerializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.common.util.io.flushable.Flushable;

/\*\*

\* Currently a facet is configured by:

\* - Index field name: The Lucene field name which stores the indexed terms of this facet

\* - Facet name: The name of the facet that the search API specifies to request facet counts.

\* - Facet id: An internal id which is used to store the facet forward mapping in the facet counting

\* data structures.

\*

\* This is a multi-map with two different mappings:

\* Facet name -> Facet id

\* Facet id -> FieldInfo

\*/

public final class FacetIDMap implements Flushable {

private final FacetField[] facetIDToFieldMap;

private final Map<String, Integer> facetNameToIDMap;

private FacetIDMap(FacetField[] facetIDToFieldMap) {

this.facetIDToFieldMap = facetIDToFieldMap;

facetNameToIDMap = Maps.newHashMapWithExpectedSize(facetIDToFieldMap.length);

for (int i = 0; i < facetIDToFieldMap.length; i++) {

facetNameToIDMap.put(facetIDToFieldMap[i].getFacetName(), i);

}

}

public FacetField getFacetField(Schema.FieldInfo fieldInfo) {

return fieldInfo != null && fieldInfo.getFieldType().isFacetField()

? getFacetFieldByFacetName(fieldInfo.getFieldType().getFacetName()) : null;

}

public FacetField getFacetFieldByFacetName(String facetName) {

Integer facetID = facetNameToIDMap.get(facetName);

return facetID != null ? facetIDToFieldMap[facetID] : null;

}

public FacetField getFacetFieldByFacetID(int facetID) {

return facetIDToFieldMap[facetID];

}

public Collection<FacetField> getFacetFields() {

return Arrays.asList(facetIDToFieldMap);

}

public int getNumberOfFacetFields() {

return facetIDToFieldMap.length;

}

/\*\*

\* Builds a new FacetIDMap from the given schema.

\*/

public static FacetIDMap build(Schema schema) {

FacetField[] facetIDToFieldMap = new FacetField[schema.getNumFacetFields()];

int facetId = 0;

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

if (fieldInfo.getFieldType().isFacetField()) {

facetIDToFieldMap[facetId] = new FacetField(facetId, fieldInfo);

facetId++;

}

}

return new FacetIDMap(facetIDToFieldMap);

}

public static final class FacetField {

private final int facetId;

private final Schema.FieldInfo fieldInfo;

private FacetField(int facetId, Schema.FieldInfo fieldInfo) {

this.facetId = facetId;

this.fieldInfo = fieldInfo;

}

public int getFacetId() {

return facetId;

}

public Schema.FieldInfo getFieldInfo() {

return fieldInfo;

}

public String getFacetName() {

return fieldInfo.getFieldType().getFacetName();

}

public String getDescription() {

return String.format(

"(FacetField [facetId: %d, fieldInfo: %s])",

getFacetId(), fieldInfo.getDescription());

}

}

@SuppressWarnings("unchecked")

@Override

public FacetIDMap.FlushHandler getFlushHandler() {

return new FlushHandler(this);

}

public static final class FlushHandler extends Flushable.Handler<FacetIDMap> {

private static final String NUM\_FACET\_FIELDS\_PROP\_NAME = "numFacetFields";

private final Schema schema;

public FlushHandler(Schema schema) {

this.schema = schema;

}

public FlushHandler(FacetIDMap objectToFlush) {

super(objectToFlush);

// schema only needed here for loading, not for flushing

this.schema = null;

}

@Override

public void doFlush(FlushInfo flushInfo, DataSerializer out) throws IOException {

FacetIDMap toFlush = getObjectToFlush();

int[] idMap = new int[toFlush.facetIDToFieldMap.length];

for (int i = 0; i < toFlush.facetIDToFieldMap.length; i++) {

idMap[i] = toFlush.facetIDToFieldMap[i].getFieldInfo().getFieldId();

}

out.writeIntArray(idMap);

flushInfo.addIntProperty(NUM\_FACET\_FIELDS\_PROP\_NAME, idMap.length);

}

@Override

public FacetIDMap doLoad(FlushInfo flushInfo, DataDeserializer in) throws IOException {

int[] idMap = in.readIntArray();

if (idMap.length != schema.getNumFacetFields()) {

throw new IOException("Wrong number of facet fields. Expected by schema: "

+ schema.getNumFacetFields()

+ ", but found in serialized segment: " + idMap.length);

}

FacetField[] facetIDToFieldMap = new FacetField[schema.getNumFacetFields()];

for (int i = 0; i < idMap.length; i++) {

int fieldConfigId = idMap[i];

facetIDToFieldMap[i] = new FacetField(i, schema.getFieldInfo(fieldConfigId));

}

return new FacetIDMap(facetIDToFieldMap);

}

}

}