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

import java.io.IOException;

import java.util.List;

import java.util.Map;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.apache.lucene.facet.FacetResult;

import org.apache.lucene.facet.Facets;

import org.apache.lucene.facet.FacetsCollector;

import org.apache.lucene.facet.FacetsCollector.MatchingDocs;

import org.apache.lucene.util.BitDocIdSet;

import org.apache.lucene.util.BitSet;

import com.twitter.search.common.facets.FacetSearchParam;

import com.twitter.search.common.facets.thriftjava.FacetFieldRequest;

import com.twitter.search.core.earlybird.index.EarlybirdIndexSegmentAtomicReader;

/\*\*

\* Lucene accumulator implementation that counts on our facet counting array data structure.

\*

\*/

public class EarlybirdFacets extends Facets {

private final AbstractFacetCountingArray countingArray;

private final FacetCountAggregator aggregator;

private final EarlybirdIndexSegmentAtomicReader reader;

private final MatchingDocs matchingDocs;

private final Map<FacetFieldRequest, FacetResult> resultMapping;

/\*\*

\* Constructs an EarlybirdFacets accumulator.

\*/

public EarlybirdFacets(

List<FacetSearchParam> facetSearchParams,

FacetsCollector facetsCollector,

EarlybirdIndexSegmentAtomicReader reader) throws IOException {

Preconditions.checkArgument(facetSearchParams != null && !facetSearchParams.isEmpty());

Preconditions.checkArgument(

facetsCollector != null

&& facetsCollector.getMatchingDocs() != null

&& facetsCollector.getMatchingDocs().size() == 1);

Preconditions.checkNotNull(reader);

this.countingArray = reader.getSegmentData().getFacetCountingArray();

this.reader = reader;

this.aggregator = new FacetCountAggregator(facetSearchParams,

reader.getSegmentData().getSchema(),

reader.getFacetIDMap(),

reader.getSegmentData().getPerFieldMap());

this.matchingDocs = facetsCollector.getMatchingDocs().get(0);

this.resultMapping = count();

}

private Map<FacetFieldRequest, FacetResult> count() throws IOException {

Preconditions.checkState(matchingDocs.bits instanceof BitDocIdSet,

"Assuming BitDocIdSet");

final BitSet bits = ((BitDocIdSet) matchingDocs.bits).bits();

final int length = bits.length();

int doc = reader.getSmallestDocID();

if (doc != -1) {

while (doc < length && (doc = bits.nextSetBit(doc)) != -1) {

countingArray.collectForDocId(doc, aggregator);

doc++;

}

}

return aggregator.getTop();

}

@Override

public FacetResult getTopChildren(int topN, String dim, String... path) throws IOException {

FacetFieldRequest facetFieldRequest = new FacetFieldRequest(dim, topN);

if (path.length > 0) {

facetFieldRequest.setPath(Lists.newArrayList(path));

}

FacetResult result = resultMapping.get(facetFieldRequest);

Preconditions.checkNotNull(

result,

"Illegal facet field request: %s, supported requests are: %s",

facetFieldRequest,

resultMapping.keySet());

return result;

}

@Override

public Number getSpecificValue(String dim, String... path) {

throw new UnsupportedOperationException("Not supported");

}

@Override

public List<FacetResult> getAllDims(int topN) throws IOException {

throw new UnsupportedOperationException("Not supported");

}

}