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

import java.io.IOException;

import org.apache.lucene.index.BinaryDocValues;

import org.apache.lucene.index.Fields;

import org.apache.lucene.index.LeafMetaData;

import org.apache.lucene.index.NumericDocValues;

import org.apache.lucene.index.PointValues;

import org.apache.lucene.index.SortedDocValues;

import org.apache.lucene.index.SortedNumericDocValues;

import org.apache.lucene.index.SortedSetDocValues;

import org.apache.lucene.index.StoredFieldVisitor;

import org.apache.lucene.index.Term;

import org.apache.lucene.index.Terms;

import org.apache.lucene.search.Sort;

import org.apache.lucene.util.Bits;

import org.apache.lucene.util.Version;

import com.twitter.search.core.earlybird.facets.EarlybirdFacetDocValueSet;

import com.twitter.search.core.earlybird.index.column.ColumnStrideFieldDocValues;

import com.twitter.search.core.earlybird.index.column.ColumnStrideFieldIndex;

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

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

public final class EarlybirdRealtimeIndexSegmentAtomicReader

extends EarlybirdIndexSegmentAtomicReader {

private final Fields fields;

private final int maxDocId;

private final int numDocs;

/\*\*

\* Creates a new real-time reader for the given segment. Do not add public constructors to this

\* class. EarlybirdRealtimeIndexSegmentAtomicReader instances should be created only by calling

\* EarlybirdRealtimeIndexSegmentData.createAtomicReader(), to make sure everything is set up

\* properly (such as CSF readers).

\*/

EarlybirdRealtimeIndexSegmentAtomicReader(EarlybirdRealtimeIndexSegmentData segmentData) {

super(segmentData);

this.fields = new InMemoryFields(segmentData.getPerFieldMap(), syncData.getIndexPointers());

// We cache the highest doc ID and the number of docs, because the reader must return the same

// values for its entire lifetime, and the segment will get more tweets over time.

// These values could be slightly out of sync with 'fields', because we don't update these

// values atomically with the fields.

this.maxDocId = segmentData.getDocIDToTweetIDMapper().getPreviousDocID(Integer.MAX\_VALUE);

this.numDocs = segmentData.getDocIDToTweetIDMapper().getNumDocs();

}

@Override

public int maxDoc() {

return maxDocId + 1;

}

@Override

public int numDocs() {

return numDocs;

}

@Override

protected void doClose() {

// nothing to do

}

@Override

public void document(int docID, StoredFieldVisitor visitor) {

// not supported

}

@Override

public int getOldestDocID(Term t) throws IOException {

InvertedIndex perField = getSegmentData().getPerFieldMap().get(t.field());

if (perField == null) {

return TERM\_NOT\_FOUND;

}

return perField.getLargestDocIDForTerm(t.bytes());

}

@Override

public int getTermID(Term t) throws IOException {

InvertedIndex perField = getSegmentData().getPerFieldMap().get(t.field());

if (perField == null) {

return TERM\_NOT\_FOUND;

}

return perField.lookupTerm(t.bytes());

}

@Override

public Bits getLiveDocs() {

// liveDocs contains inverted (decreasing) docIDs.

return getDeletesView().getLiveDocs();

}

@Override

public boolean hasDeletions() {

return getDeletesView().hasDeletions();

}

@Override

public Terms terms(String field) throws IOException {

return fields.terms(field);

}

@Override

public NumericDocValues getNumericDocValues(String field) throws IOException {

ColumnStrideFieldIndex csf =

getSegmentData().getDocValuesManager().getColumnStrideFieldIndex(field);

return csf != null ? new ColumnStrideFieldDocValues(csf, this) : null;

}

@Override

public boolean hasDocs() {

// smallestDocID is the smallest document ID that was available when this reader was created.

// So we need to check its value in order to decide if this reader can see any documents,

// because in the meantime other documents might've been added to the tweet ID mapper.

return getSmallestDocID() != Integer.MAX\_VALUE;

}

@Override

public BinaryDocValues getBinaryDocValues(String field) {

return null;

}

@Override

public SortedDocValues getSortedDocValues(String field) {

return null;

}

@Override

public SortedSetDocValues getSortedSetDocValues(String field) {

// special handling for facet field

if (EarlybirdFacetDocValueSet.FIELD\_NAME.equals(field)) {

return ((EarlybirdRealtimeIndexSegmentData) getSegmentData()).getFacetDocValueSet();

}

return null;

}

@Override

public NumericDocValues getNormValues(String field) throws IOException {

ColumnStrideFieldIndex csf = getSegmentData().getNormIndex(field);

return csf != null ? new ColumnStrideFieldDocValues(csf, this) : null;

}

@Override

public SortedNumericDocValues getSortedNumericDocValues(String field) {

return null;

}

@Override

public void checkIntegrity() {

// nothing to do

}

@Override

public PointValues getPointValues(String field) {

return null;

}

@Override

public LeafMetaData getMetaData() {

return new LeafMetaData(Version.LATEST.major, Version.LATEST, Sort.RELEVANCE);

}

@Override

public CacheHelper getCoreCacheHelper() {

return null;

}

@Override

public CacheHelper getReaderCacheHelper() {

return null;

}

}