package com.twitter.search.earlybird.search.queries;

import java.io.IOException;

import java.util.Set;

import org.apache.lucene.index.IndexReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.index.Term;

import org.apache.lucene.search.ConstantScoreQuery;

import org.apache.lucene.search.ConstantScoreScorer;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.Explanation;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.Scorer;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.TwoPhaseIterator;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.search.TerminationTracker;

import com.twitter.search.earlybird.common.config.EarlybirdConfig;

public class GeoTwoPhaseQuery extends Query {

private static final boolean ENABLE\_GEO\_EARLY\_TERMINATION =

EarlybirdConfig.getBool("early\_terminate\_geo\_searches", true);

private static final int GEO\_TIMEOUT\_OVERRIDE =

EarlybirdConfig.getInt("early\_terminate\_geo\_searches\_timeout\_override", -1);

// How many geo searches are early terminated due to timeout.

private static final SearchCounter GEO\_SEARCH\_TIMEOUT\_COUNT =

SearchCounter.export("geo\_search\_timeout\_count");

private final SecondPhaseDocAccepter accepter;

private final TerminationTracker terminationTracker;

private final ConstantScoreQuery query;

public GeoTwoPhaseQuery(

Query query, SecondPhaseDocAccepter accepter, TerminationTracker terminationTracker) {

this.accepter = accepter;

this.terminationTracker = terminationTracker;

this.query = new ConstantScoreQuery(query);

}

@Override

public Query rewrite(IndexReader reader) throws IOException {

Query rewritten = query.getQuery().rewrite(reader);

if (rewritten != query.getQuery()) {

return new GeoTwoPhaseQuery(rewritten, accepter, terminationTracker);

}

return this;

}

@Override

public int hashCode() {

return query.hashCode();

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof GeoTwoPhaseQuery)) {

return false;

}

GeoTwoPhaseQuery that = (GeoTwoPhaseQuery) obj;

return query.equals(that.query)

&& accepter.equals(that.accepter)

&& terminationTracker.equals(that.terminationTracker);

}

@Override

public String toString(String field) {

return new StringBuilder("GeoTwoPhaseQuery(")

.append("Accepter(")

.append(accepter.toString())

.append(") Geohashes(")

.append(query.getQuery().toString(field))

.append("))")

.toString();

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost)

throws IOException {

Weight innerWeight = query.createWeight(searcher, scoreMode, boost);

return new GeoTwoPhaseWeight(this, innerWeight, accepter, terminationTracker);

}

private static final class GeoTwoPhaseWeight extends Weight {

private final Weight innerWeight;

private final SecondPhaseDocAccepter accepter;

private final TerminationTracker terminationTracker;

private GeoTwoPhaseWeight(

Query query,

Weight innerWeight,

SecondPhaseDocAccepter accepter,

TerminationTracker terminationTracker) {

super(query);

this.innerWeight = innerWeight;

this.accepter = accepter;

this.terminationTracker = terminationTracker;

}

@Override

public void extractTerms(Set<Term> terms) {

innerWeight.extractTerms(terms);

}

@Override

public Explanation explain(LeafReaderContext context, int doc) throws IOException {

return innerWeight.explain(context, doc);

}

@Override

public Scorer scorer(LeafReaderContext context) throws IOException {

Scorer innerScorer = innerWeight.scorer(context);

if (innerScorer == null) {

return null;

}

if (ENABLE\_GEO\_EARLY\_TERMINATION

&& (terminationTracker == null || !terminationTracker.useLastSearchedDocIdOnTimeout())) {

innerScorer = new ConstantScoreScorer(

this,

0.0f,

ScoreMode.COMPLETE\_NO\_SCORES,

new TimedDocIdSetIterator(innerScorer.iterator(),

terminationTracker,

GEO\_TIMEOUT\_OVERRIDE,

GEO\_SEARCH\_TIMEOUT\_COUNT));

}

accepter.initialize(context);

return new GeoTwoPhaseScorer(this, innerScorer, accepter);

}

@Override

public boolean isCacheable(LeafReaderContext ctx) {

return innerWeight.isCacheable(ctx);

}

}

private static final class GeoTwoPhaseScorer extends Scorer {

private final Scorer innerScorer;

private final SecondPhaseDocAccepter accepter;

private GeoTwoPhaseScorer(Weight weight, Scorer innerScorer, SecondPhaseDocAccepter accepter) {

super(weight);

this.innerScorer = innerScorer;

this.accepter = accepter;

}

@Override

public TwoPhaseIterator twoPhaseIterator() {

return new TwoPhaseIterator(innerScorer.iterator()) {

@Override

public boolean matches() throws IOException {

return checkDocExpensive(innerScorer.docID());

}

@Override

public float matchCost() {

return 0.0f;

}

};

}

@Override

public int docID() {

return iterator().docID();

}

@Override

public float score() throws IOException {

return innerScorer.score();

}

@Override

public DocIdSetIterator iterator() {

return new DocIdSetIterator() {

private int doNext(int startingDocId) throws IOException {

int docId = startingDocId;

while ((docId != NO\_MORE\_DOCS) && !checkDocExpensive(docId)) {

docId = innerScorer.iterator().nextDoc();

}

return docId;

}

@Override

public int docID() {

return innerScorer.iterator().docID();

}

@Override

public int nextDoc() throws IOException {

return doNext(innerScorer.iterator().nextDoc());

}

@Override

public int advance(int target) throws IOException {

return doNext(innerScorer.iterator().advance(target));

}

@Override

public long cost() {

return 2 \* innerScorer.iterator().cost();

}

};

}

@Override

public float getMaxScore(int upTo) throws IOException {

return innerScorer.getMaxScore(upTo);

}

private boolean checkDocExpensive(int doc) throws IOException {

return accepter.accept(doc);

}

}

public abstract static class SecondPhaseDocAccepter {

/\*\*

\* Initializes this accepter with the given reader context.

\*/

public abstract void initialize(LeafReaderContext context) throws IOException;

/\*\*

\* Determines if the given doc ID is accepted by this accepter.

\*/

public abstract boolean accept(int doc) throws IOException;

/\*\*

\* Returns a string description for this SecondPhaseDocAccepter instance.

\*/

public abstract String toString();

}

public static final SecondPhaseDocAccepter ALL\_DOCS\_ACCEPTER = new SecondPhaseDocAccepter() {

@Override

public void initialize(LeafReaderContext context) { }

@Override

public boolean accept(int doc) {

return true;

}

@Override

public String toString() {

return "AllDocsAccepter";

}

};

}