package com.twitter.search.common.query;

import java.util.Map;

import java.util.Set;

import com.google.common.collect.Maps;

import com.twitter.search.queryparser.query.BooleanQuery;

import com.twitter.search.queryparser.query.Conjunction;

import com.twitter.search.queryparser.query.Disjunction;

import com.twitter.search.queryparser.query.Operator;

import com.twitter.search.queryparser.query.Phrase;

import com.twitter.search.queryparser.query.Query;

import com.twitter.search.queryparser.query.QueryParserException;

import com.twitter.search.queryparser.query.QueryVisitor;

import com.twitter.search.queryparser.query.SpecialTerm;

import com.twitter.search.queryparser.query.Term;

/\*\*

\* Collects the nodes with a specified query type in the given query.

\*/

public class CollectQueryTypeVisitor extends QueryVisitor<Boolean> {

protected final Query.QueryType queryType;

protected final Map<Query, Boolean> nodeToTypeMap = Maps.newIdentityHashMap();

public CollectQueryTypeVisitor(Query.QueryType queryType) {

this.queryType = queryType;

}

@Override

public Boolean visit(Disjunction disjunction) throws QueryParserException {

return visitBooleanQuery(disjunction);

}

@Override

public Boolean visit(Conjunction conjunction) throws QueryParserException {

return visitBooleanQuery(conjunction);

}

@Override

public Boolean visit(Phrase phrase) throws QueryParserException {

return visitQuery(phrase);

}

@Override

public Boolean visit(Term term) throws QueryParserException {

return visitQuery(term);

}

@Override

public Boolean visit(Operator operator) throws QueryParserException {

return visitQuery(operator);

}

@Override

public Boolean visit(SpecialTerm special) throws QueryParserException {

return visitQuery(special);

}

public Set<Query> getCollectedNodes() {

return nodeToTypeMap.keySet();

}

protected boolean visitQuery(Query query) throws QueryParserException {

if (query.isTypeOf(queryType)) {

collectNode(query);

return true;

}

return false;

}

protected void collectNode(Query query) {

nodeToTypeMap.put(query, true);

}

protected boolean visitBooleanQuery(BooleanQuery query) throws QueryParserException {

boolean found = false;

if (query.isTypeOf(queryType)) {

collectNode(query);

found = true;

}

for (Query child : query.getChildren()) {

found |= child.accept(this);

}

return found;

}

}