package com.twitter.search.common.query;

import java.util.Collections;

import java.util.EnumSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import javax.annotation.Nullable;

import com.google.common.base.Enums;

import com.google.common.base.Function;

import com.google.common.base.Functions;

import com.google.common.base.Predicates;

import com.google.common.collect.FluentIterable;

import com.google.common.collect.ImmutableMap;

import com.google.common.collect.Iterables;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import com.twitter.search.queryparser.query.annotation.Annotation;

import com.twitter.search.queryparser.query.annotation.FieldAnnotationUtils;

import com.twitter.search.queryparser.query.annotation.FieldNameWithBoost;

public final class FieldWeightUtil {

private static final Logger LOG = LoggerFactory.getLogger(FieldWeightUtil.class);

private FieldWeightUtil() {

}

/\*\*

\* Combines default field weight configuration with field annotations and returns a

\* field-to-weight map.

\*

\* @param query The query whose annotations we will look into

\* @param defaultFieldWeightMap field-to-FieldWeightDefault map

\* @param enabledFieldWeightMap for optimization, this is the field-to-weight map inferred from

\* the field-to-FieldWeightDefault map

\* @param fieldNameToTyped A function that can turn string field name to typed field

\* @param <T> The typed field

\*/

public static <T> ImmutableMap<T, Float> combineDefaultWithAnnotation(

Query query,

Map<T, FieldWeightDefault> defaultFieldWeightMap,

Map<T, Float> enabledFieldWeightMap,

Function<String, T> fieldNameToTyped) throws QueryParserException {

return combineDefaultWithAnnotation(

query,

defaultFieldWeightMap,

enabledFieldWeightMap,

fieldNameToTyped,

Collections.<MappableField, T>emptyMap(),

Functions.forMap(Collections.<T, String>emptyMap(), ""));

}

/\*\*

\* Combines default field weight configuration with field annotations and returns a

\* field-to-weight map. Also maps generic mappable fields to field weight boosts and resolves them

\*

\* @param query The query whose annotations we will look into

\* @param defaultFieldWeightMap field-to-FieldWeightDefault map

\* @param enabledFieldWeightMap for optimization, this is the field-to-weight map inferred from

\* the field-to-FieldWeightDefault map

\* @param fieldNameToTyped A function that can turn a string field name to typed field

\* @param mappableFieldMap mapping of mappable fields to the corresponding typed fields

\* @param typedToFieldName A function that can turn a typed field into a string field name

\* @param <T> The typed field

\*

\* Note: As a result of discussion on SEARCH-24029, we now allow replace and remove annotations

\* on a single term. See http://go/fieldweight for info on field weight annotations.

\*/

public static <T> ImmutableMap<T, Float> combineDefaultWithAnnotation(

Query query,

Map<T, FieldWeightDefault> defaultFieldWeightMap,

Map<T, Float> enabledFieldWeightMap,

Function<String, T> fieldNameToTyped,

Map<MappableField, T> mappableFieldMap,

Function<T, String> typedToFieldName) throws QueryParserException {

List<Annotation> fieldAnnotations = query.getAllAnnotationsOf(Annotation.Type.FIELD);

List<Annotation> mappableFieldAnnotations =

query.getAllAnnotationsOf(Annotation.Type.MAPPABLE\_FIELD);

if (fieldAnnotations.isEmpty() && mappableFieldAnnotations.isEmpty()) {

return ImmutableMap.copyOf(enabledFieldWeightMap);

}

// Convert mapped fields to field annotations

Iterable<Annotation> fieldAnnotationsForMappedFields =

FluentIterable.from(mappableFieldAnnotations)

.transform(FieldWeightUtil.fieldAnnotationForMappableField(mappableFieldMap,

typedToFieldName))

.filter(Predicates.notNull());

Iterable<Annotation> annotations =

Iterables.concat(fieldAnnotationsForMappedFields, fieldAnnotations);

// Sanitize the field annotations first, remove the ones we don't know

// for REPLACE and REMOVE.

List<FieldNameWithBoost> sanitizedFields = Lists.newArrayList();

Set<FieldNameWithBoost.FieldModifier> seenModifierTypes =

EnumSet.noneOf(FieldNameWithBoost.FieldModifier.class);

for (Annotation annotation : annotations) {

FieldNameWithBoost fieldNameWithBoost = (FieldNameWithBoost) annotation.getValue();

T typedField = fieldNameToTyped.apply(fieldNameWithBoost.getFieldName());

FieldNameWithBoost.FieldModifier modifier = fieldNameWithBoost.getFieldModifier();

if (defaultFieldWeightMap.containsKey(typedField)) {

seenModifierTypes.add(modifier);

sanitizedFields.add(fieldNameWithBoost);

}

}

// Even if there is no mapping for a mapped annotation, if a query is replaced by an unknown

// mapping, it should not map to other fields, so we need to detect a REPLACE annotation

if (seenModifierTypes.isEmpty()

&& FieldAnnotationUtils.hasReplaceAnnotation(mappableFieldAnnotations)) {

seenModifierTypes.add(FieldNameWithBoost.FieldModifier.REPLACE);

}

boolean onlyHasReplace = seenModifierTypes.size() == 1

&& seenModifierTypes.contains(FieldNameWithBoost.FieldModifier.REPLACE);

// If we only have replace, start with an empty map, otherwise, start with all enabled fields.

Map<T, Float> actualMap = onlyHasReplace

? Maps.<T, Float>newLinkedHashMap()

: Maps.newLinkedHashMap(enabledFieldWeightMap);

// Go over all field annotations and apply them.

for (FieldNameWithBoost fieldAnnotation : sanitizedFields) {

T typedField = fieldNameToTyped.apply(fieldAnnotation.getFieldName());

FieldNameWithBoost.FieldModifier modifier = fieldAnnotation.getFieldModifier();

switch (modifier) {

case REMOVE:

actualMap.remove(typedField);

break;

case ADD:

case REPLACE:

if (fieldAnnotation.getBoost().isPresent()) {

actualMap.put(typedField, fieldAnnotation.getBoost().get());

} else {

// When annotation does not specify weight, use default weight

actualMap.put(

typedField,

defaultFieldWeightMap.get(typedField).getWeight());

}

break;

default:

throw new QueryParserException("Unknown field annotation type: " + fieldAnnotation);

}

}

return ImmutableMap.copyOf(actualMap);

}

public static ImmutableMap<String, Float> combineDefaultWithAnnotation(

Query query,

Map<String, FieldWeightDefault> defaultFieldWeightMap,

Map<String, Float> enabledFieldWeightMap) throws QueryParserException {

return combineDefaultWithAnnotation(

query, defaultFieldWeightMap, enabledFieldWeightMap, Functions.<String>identity());

}

/\*\*

\* Create an annotation of the FIELD type from annotations of the MAPPED\_FIELD type

\* @param mappableFieldMap mapping of mappable fields to the corresponding typed fields

\* @param typedToFieldName A function that can turn a typed field into a string field name

\* @param <T> The typed field

\* @return an Annotation with the same modifier and boost for a FIELD as the incoming MAPPED\_FIELD

\* annotation

\*/

private static <T> Function<Annotation, Annotation> fieldAnnotationForMappableField(

final Map<MappableField, T> mappableFieldMap,

final Function<T, String> typedToFieldName) {

return new Function<Annotation, Annotation>() {

@Nullable

@Override

public Annotation apply(Annotation mappableAnnotation) {

FieldNameWithBoost fieldNameWithBoost = (FieldNameWithBoost) mappableAnnotation.getValue();

MappableField mappedField =

Enums.getIfPresent(

MappableField.class,

fieldNameWithBoost.getFieldName().toUpperCase()).orNull();

T typedFieldName = mappableFieldMap.get(mappedField);

Annotation fieldAnnotation = null;

if (typedFieldName != null) {

String fieldName = typedToFieldName.apply(typedFieldName);

FieldNameWithBoost mappedFieldBoost =

new FieldNameWithBoost(

fieldName,

fieldNameWithBoost.getBoost(),

fieldNameWithBoost.getFieldModifier());

fieldAnnotation = Annotation.Type.FIELD.newInstance(mappedFieldBoost);

}

return fieldAnnotation;

}

};

}

}