package com.twitter.search.common.schema;

import java.util.Collection;

import java.util.Map;

import java.util.concurrent.atomic.AtomicReference;

import javax.annotation.Nullable;

import com.google.common.base.Preconditions;

import com.google.common.base.Predicate;

import com.google.common.collect.ImmutableCollection;

import com.google.common.collect.ImmutableMap;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.facet.FacetsConfig;

import org.apache.lucene.index.FieldInfos;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchema;

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

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

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

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

import com.twitter.search.common.schema.thriftjava.ThriftAnalyzer;

import com.twitter.search.common.schema.thriftjava.ThriftCSFType;

import com.twitter.search.common.schema.thriftjava.ThriftFieldConfiguration;

/\*\*

\* A schema implementation that allow minor version increments at run time.

\*/

public class DynamicSchema implements Schema {

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

private final AtomicReference<ImmutableSchema> schema;

public DynamicSchema(ImmutableSchema schema) {

this.schema = new AtomicReference<>(schema);

}

public ImmutableSchemaInterface getSchemaSnapshot() {

return schema.get();

}

/\*\*

\* Update the schema reference inside this DynamicSchema.

\*/

public synchronized void updateSchema(ImmutableSchema newSchema) throws SchemaUpdateException {

ImmutableSchema oldSchema = schema.get();

if (newSchema.getMajorVersionNumber() != oldSchema.getMajorVersionNumber()) {

throw new SchemaUpdateException("Dynamic major version update is not supported.");

} else {

if (newSchema.getMinorVersionNumber() <= oldSchema.getMinorVersionNumber()) {

throw new SchemaUpdateException("Dynamic backward minor version update is not supported.");

} else {

LOG.info("DynamicSchema accepted update. Old version is {}.{}; new version is {}.{}",

oldSchema.getMajorVersionNumber(),

oldSchema.getMinorVersionNumber(),

newSchema.getMajorVersionNumber(),

newSchema.getMinorVersionNumber());

schema.set(newSchema);

}

}

}

public static class SchemaUpdateException extends Exception {

public SchemaUpdateException(String message) {

super(message);

}

}

// The below are all methods in the Schema interface delegated to the underlying ImmutableSchema.

// The below is generated by IntelliJ, and reviewers can stop reviewing this file here.

// If you are adding logic into this class, please do so above this line.

@Override

public FieldInfos getLuceneFieldInfos(

Predicate<String> acceptedFields) {

return schema.get().getLuceneFieldInfos(acceptedFields);

}

@Override

public FacetsConfig getFacetsConfig() {

return schema.get().getFacetsConfig();

}

@Override

public Analyzer getDefaultAnalyzer(

ThriftAnalyzer override) {

return schema.get().getDefaultAnalyzer(override);

}

@Override

public ImmutableCollection<FieldInfo> getFieldInfos() {

return schema.get().getFieldInfos();

}

@Override

public boolean hasField(int fieldConfigId) {

return schema.get().hasField(fieldConfigId);

}

@Override

public boolean hasField(String fieldName) {

return schema.get().hasField(fieldName);

}

@Override

@Nullable

public FieldInfo getFieldInfo(int fieldConfigId) {

return schema.get().getFieldInfo(fieldConfigId);

}

@Override

@Nullable

public FieldInfo getFieldInfo(String fieldName) {

return schema.get().getFieldInfo(fieldName);

}

@Override

public String getFieldName(int fieldConfigId) {

return schema.get().getFieldName(fieldConfigId);

}

@Override

public FieldInfo getFieldInfo(int fieldConfigId,

ThriftFieldConfiguration override) {

return schema.get().getFieldInfo(fieldConfigId, override);

}

@Override

public int getNumFacetFields() {

return schema.get().getNumFacetFields();

}

@Override

public FieldInfo getFacetFieldByFacetName(

String facetName) {

return schema.get().getFacetFieldByFacetName(facetName);

}

@Override

public FieldInfo getFacetFieldByFieldName(

String fieldName) {

return schema.get().getFacetFieldByFieldName(fieldName);

}

@Override

public Collection<FieldInfo> getFacetFields() {

return schema.get().getFacetFields();

}

@Override

public Collection<FieldInfo> getCsfFacetFields() {

return schema.get().getCsfFacetFields();

}

@Override

public String getVersionDescription() {

return schema.get().getVersionDescription();

}

@Override

public int getMajorVersionNumber() {

return schema.get().getMajorVersionNumber();

}

@Override

public int getMinorVersionNumber() {

return schema.get().getMinorVersionNumber();

}

@Override

public boolean isVersionOfficial() {

return schema.get().isVersionOfficial();

}

@Override

public Map<String, FieldWeightDefault> getFieldWeightMap() {

return schema.get().getFieldWeightMap();

}

@Override

public FeatureConfiguration getFeatureConfigurationByName(

String featureName) {

return schema.get().getFeatureConfigurationByName(featureName);

}

@Override

public FeatureConfiguration getFeatureConfigurationById(int featureFieldId) {

return Preconditions.checkNotNull(schema.get().getFeatureConfigurationById(featureFieldId));

}

@Override

@Nullable

public ThriftCSFType getCSFFieldType(

String fieldName) {

return schema.get().getCSFFieldType(fieldName);

}

@Override

public ThriftSearchFeatureSchema getSearchFeatureSchema() {

return schema.get().getSearchFeatureSchema();

}

@Override

public ImmutableMap<Integer, FeatureConfiguration> getFeatureIdToFeatureConfig() {

return schema.get().getFeatureIdToFeatureConfig();

}

@Override

public ImmutableMap<String, FeatureConfiguration> getFeatureNameToFeatureConfig() {

return schema.get().getFeatureNameToFeatureConfig();

}

}