package com.twitter.search.common.schema;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.util.Collection;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.SortedMap;

import java.util.TreeMap;

import java.util.concurrent.atomic.AtomicLong;

import javax.annotation.Nullable;

import javax.annotation.concurrent.Immutable;

import javax.annotation.concurrent.ThreadSafe;

import com.google.common.annotations.VisibleForTesting;

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 com.google.common.collect.ImmutableSet;

import com.google.common.collect.ImmutableSortedMap;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.facet.FacetsConfig;

import org.apache.lucene.index.DocValuesType;

import org.apache.lucene.index.FieldInfos;

import org.apache.lucene.index.IndexOptions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.collections.Pair;

import com.twitter.common.text.util.TokenStreamSerializer;

import com.twitter.search.common.features.ExternalTweetFeature;

import com.twitter.search.common.features.SearchResultFeature;

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

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

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

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

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

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

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

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.IndexedNumericFieldSettings;

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

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

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

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

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

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

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

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

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

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

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

/\*\*

\* A schema instance that does not change at run time.

\*/

@Immutable @ThreadSafe

public class ImmutableSchema implements ImmutableSchemaInterface {

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

private static final ImmutableSet<ThriftCSFType> CAN\_FACET\_ON\_CSF\_TYPES =

ImmutableSet.<ThriftCSFType>builder()

.add(ThriftCSFType.BYTE)

.add(ThriftCSFType.INT)

.add(ThriftCSFType.LONG)

.build();

private static final SearchCounter FEATURES\_EXISTED\_IN\_OLD\_SCHEMA =

SearchCounter.export("features\_existed\_in\_old\_schema");

// Currently our index uses 4 bits to store the facet field id.

public static final int MAX\_FACET\_FIELD\_ID = 15;

public static final String HF\_TERM\_PAIRS\_FIELD = "hf\_term\_pairs";

public static final String HF\_PHRASE\_PAIRS\_FIELD = "hf\_phrase\_pairs";

private final ImmutableMap<Integer, FieldInfo> fieldSettingsMapById;

private final ImmutableMap<String, FieldInfo> fieldSettingsMapByName;

private final ImmutableMap<String, FeatureConfiguration> featureConfigMapByName;

private final ImmutableMap<Integer, FeatureConfiguration> featureConfigMapById;

@Nullable

private final ThriftAnalyzer defaultAnalyzer;

private final AnalyzerFactory analyzerFactory;

private final ImmutableMap<String, FieldWeightDefault> fieldWeightMap;

private final Map<String, FieldInfo> facetNameToFieldMap = Maps.newHashMap();

private final int numFacetFields;

private final ImmutableSet<FieldInfo> csfFacetFields;

// This is the search result feature schema - it has the definition for all the column stride

// view fields.

private final ThriftSearchFeatureSchema searchFeatureSchema;

private final int majorVersionNumber;

private final int minorVersionNumber;

private final String versionDesc;

private final boolean isVersionOfficial;

/\*\*

\* Construct a Schema instance with the given ThriftSchema and AnalyzerFactory.

\*/

public ImmutableSchema(ThriftSchema thriftSchema,

AnalyzerFactory analyzerFactory,

String featureSchemaVersionPrefix) throws SchemaValidationException {

Pair<Integer, String> versionPair = parseVersionString(thriftSchema.getVersion());

this.majorVersionNumber = thriftSchema.getMajorVersionNumber();

this.minorVersionNumber = thriftSchema.getMinorVersionNumber();

this.versionDesc = versionPair.getSecond();

this.isVersionOfficial = thriftSchema.isVersionIsOfficial();

this.analyzerFactory = analyzerFactory;

Map<Integer, FieldInfo> tmpMap = Maps.newLinkedHashMap();

Set<FieldInfo> tmpSet = Sets.newHashSet();

if (thriftSchema.isSetDefaultAnalyzer()) {

this.defaultAnalyzer = thriftSchema.getDefaultAnalyzer().deepCopy();

} else {

this.defaultAnalyzer = null;

}

Map<Integer, ThriftFieldConfiguration> configs = thriftSchema.getFieldConfigs();

// Collect all the CSF Views, so that we can later verify that they are appropriately

// configured once we've processed all the other field settings.

Map<Integer, ThriftFieldConfiguration> csfViewFields = Maps.newHashMap();

boolean requiresHfPairFields = false;

boolean hasHfTermPairField = false;

boolean hasHfPhrasePairField = false;

int numFacets = 0;

for (Map.Entry<Integer, ThriftFieldConfiguration> entry : configs.entrySet()) {

int fieldId = entry.getKey();

if (tmpMap.containsKey(fieldId)) {

throw new SchemaValidationException("Duplicate field id " + fieldId);

}

ThriftFieldConfiguration config = entry.getValue();

FieldInfo fieldInfo = parseThriftFieldSettings(fieldId, config, csfViewFields);

validate(fieldInfo);

if (fieldInfo.getFieldType().isFacetField()) {

if (numFacets > MAX\_FACET\_FIELD\_ID) {

throw new SchemaValidationException(

"Maximum supported facet field ID is: " + MAX\_FACET\_FIELD\_ID);

}

numFacets++;

facetNameToFieldMap.put(fieldInfo.getFieldType().getFacetName(), fieldInfo);

if (fieldInfo.getFieldType().isUseCSFForFacetCounting()) {

tmpSet.add(fieldInfo);

}

}

tmpMap.put(fieldId, fieldInfo);

if (fieldInfo.getFieldType().isIndexHFTermPairs()) {

requiresHfPairFields = true;

}

if (fieldInfo.getName().equals(HF\_TERM\_PAIRS\_FIELD)) {

hasHfTermPairField = true;

}

if (fieldInfo.getName().equals(HF\_PHRASE\_PAIRS\_FIELD)) {

hasHfPhrasePairField = true;

}

}

this.numFacetFields = numFacets;

this.csfFacetFields = ImmutableSet.copyOf(tmpSet);

// If any field requires high frequency term/phrase pair fields, make sure they exist

if (requiresHfPairFields) {

if (!hasHfTermPairField || !hasHfPhrasePairField) {

throw new SchemaValidationException(

"High frequency term/phrase pair fields do not exist in the schema.");

}

}

this.fieldSettingsMapById = ImmutableMap.copyOf(tmpMap);

Pair<ImmutableMap<String, FeatureConfiguration>, ImmutableMap<Integer, FeatureConfiguration>>

featureConfigMapPair = buildFeatureMaps(csfViewFields);

this.featureConfigMapByName = featureConfigMapPair.getFirst();

this.featureConfigMapById = featureConfigMapPair.getSecond();

for (ThriftFieldConfiguration csfViewField : csfViewFields.values()) {

SchemaBuilder.verifyCSFViewSettings(configs, csfViewField);

}

ImmutableMap.Builder<String, FieldInfo> builder = ImmutableMap.builder();

for (FieldInfo info : fieldSettingsMapById.values()) {

info.getFieldType().freeze();

builder.put(info.getName(), info);

}

this.fieldSettingsMapByName = builder.build();

ImmutableMap.Builder<String, FieldWeightDefault> fieldWeightMapBuilder = ImmutableMap.builder();

for (FieldInfo fi : getFieldInfos()) {

// CSF fields are not searchable. All other fields are.

if (fi.getFieldType().isIndexedField()) {

fieldWeightMapBuilder.put(

fi.getName(),

new FieldWeightDefault(

fi.getFieldType().isTextSearchableByDefault(),

fi.getFieldType().getTextSearchableFieldWeight()));

}

}

this.fieldWeightMap = fieldWeightMapBuilder.build();

// Create features with extra Earlybird derived fields, extra fields won't change the version

// but they do change the checksum.

this.searchFeatureSchema = createSearchResultFeatureSchema(

featureSchemaVersionPrefix, fieldSettingsMapByName, featureConfigMapByName);

}

/\*\*

\* Add a set of features to a schema if they don't exist yet, and update the schema checksum.

\* if there's conflict, RuntimeException will be thrown.

\* Old map won't be touched, a new map will be returned will old and new data combined.

\*/

public static Map<Integer, ThriftSearchFeatureSchemaEntry> appendToFeatureSchema(

Map<Integer, ThriftSearchFeatureSchemaEntry> oldEntryMap,

Set<? extends SearchResultFeature> features) throws SchemaValidationException {

if (oldEntryMap == null) {

throw new SchemaValidationException(

"Cannot append features to schema, the entryMap is null");

}

// make a copy of the existing map

ImmutableMap.Builder<Integer, ThriftSearchFeatureSchemaEntry> builder =

ImmutableSortedMap.<Integer, ThriftSearchFeatureSchemaEntry>naturalOrder()

.putAll(oldEntryMap);

for (SearchResultFeature feature : features) {

if (oldEntryMap.containsKey(feature.getId())) {

FEATURES\_EXISTED\_IN\_OLD\_SCHEMA.increment();

} else {

builder.put(feature.getId(), new ThriftSearchFeatureSchemaEntry()

.setFeatureName(feature.getName())

.setFeatureType(feature.getType()));

}

}

return builder.build();

}

/\*\*

\* Append external features to create a new schema.

\* @param oldSchema The old schema to build on top of

\* @param features a list of features to be appended to the schema

\* @param versionSuffix the version suffix, if not-null, it will be attached to the end of

\* original schema's version.

\* @return A new schema object with the appended fields

\* @throws SchemaValidationException thrown when the checksum cannot be computed

\*/

public static ThriftSearchFeatureSchema appendToCreateNewFeatureSchema(

ThriftSearchFeatureSchema oldSchema,

Set<ExternalTweetFeature> features,

@Nullable String versionSuffix) throws SchemaValidationException {

ThriftSearchFeatureSchema newSchema = new ThriftSearchFeatureSchema();

// copy over all the entries plus the new ones

newSchema.setEntries(appendToFeatureSchema(oldSchema.getEntries(), features));

ThriftSearchFeatureSchemaSpecifier spec = new ThriftSearchFeatureSchemaSpecifier();

// the version is directly inherited or with a suffix

Preconditions.checkArgument(versionSuffix == null || !versionSuffix.isEmpty());

spec.setVersion(versionSuffix == null

? oldSchema.getSchemaSpecifier().getVersion()

: oldSchema.getSchemaSpecifier().getVersion() + versionSuffix);

spec.setChecksum(getChecksum(newSchema.getEntries()));

newSchema.setSchemaSpecifier(spec);

return newSchema;

}

@Override

public FieldInfos getLuceneFieldInfos(Predicate<String> acceptedFields) {

List<org.apache.lucene.index.FieldInfo> acceptedFieldInfos = Lists.newArrayList();

for (FieldInfo fi : getFieldInfos()) {

if (acceptedFields == null || acceptedFields.apply(fi.getName())) {

acceptedFieldInfos.add(convert(fi.getName(), fi.getFieldId(), fi.getFieldType()));

}

}

return new FieldInfos(acceptedFieldInfos.toArray(

new org.apache.lucene.index.FieldInfo[acceptedFieldInfos.size()]));

}

private FieldInfo parseThriftFieldSettings(int fieldId, ThriftFieldConfiguration fieldConfig,

Map<Integer, ThriftFieldConfiguration> csfViewFields)

throws SchemaValidationException {

FieldInfo fieldInfo

= new FieldInfo(fieldId, fieldConfig.getFieldName(), new EarlybirdFieldType());

ThriftFieldSettings fieldSettings = fieldConfig.getSettings();

boolean settingFound = false;

if (fieldSettings.isSetIndexedFieldSettings()) {

if (fieldSettings.isSetCsfFieldSettings() || fieldSettings.isSetCsfViewSettings()) {

throw new SchemaValidationException("ThriftFieldSettings: Only one of "

+ "'indexedFieldSettings', 'csfFieldSettings', 'csfViewSettings' can be set.");

}

applyIndexedFieldSettings(fieldInfo, fieldSettings.getIndexedFieldSettings());

settingFound = true;

}

if (fieldSettings.isSetCsfFieldSettings()) {

if (fieldSettings.isSetIndexedFieldSettings() || fieldSettings.isSetCsfViewSettings()) {

throw new SchemaValidationException("ThriftFieldSettings: Only one of "

+ "'indexedFieldSettings', 'csfFieldSettings', 'csfViewSettings' can be set.");

}

applyCsfFieldSettings(fieldInfo, fieldSettings.getCsfFieldSettings());

settingFound = true;

}

if (fieldSettings.isSetFacetFieldSettings()) {

if (!fieldSettings.isSetIndexedFieldSettings() && !(fieldSettings.isSetCsfFieldSettings()

&& fieldSettings.getFacetFieldSettings().isUseCSFForFacetCounting()

&& CAN\_FACET\_ON\_CSF\_TYPES.contains(fieldSettings.getCsfFieldSettings().getCsfType()))) {

throw new SchemaValidationException("ThriftFieldSettings: 'facetFieldSettings' can only be "

+ "used in combination with 'indexedFieldSettings' or with 'csfFieldSettings' "

+ "where 'isUseCSFForFacetCounting' was set to true and ThriftCSFType is a type that "

+ "can be faceted on.");

}

applyFacetFieldSettings(fieldInfo, fieldSettings.getFacetFieldSettings());

settingFound = true;

}

if (fieldSettings.isSetCsfViewSettings()) {

if (fieldSettings.isSetIndexedFieldSettings() || fieldSettings.isSetCsfFieldSettings()) {

throw new SchemaValidationException("ThriftFieldSettings: Only one of "

+ "'indexedFieldSettings', 'csfFieldSettings', 'csfViewSettings' can be set.");

}

// add this field now, but apply settings later to make sure the base field was added properly

// before

csfViewFields.put(fieldId, fieldConfig);

settingFound = true;

}

if (!settingFound) {

throw new SchemaValidationException("ThriftFieldSettings: One of 'indexedFieldSettings', "

+ "'csfFieldSettings' or 'facetFieldSettings' must be set.");

}

// search field settings are optional

if (fieldSettings.isSetSearchFieldSettings()) {

if (!fieldSettings.isSetIndexedFieldSettings()) {

throw new SchemaValidationException(

"ThriftFieldSettings: 'searchFieldSettings' can only be "

+ "used in combination with 'indexedFieldSettings'");

}

applySearchFieldSettings(fieldInfo, fieldSettings.getSearchFieldSettings());

}

return fieldInfo;

}

private void applyCsfFieldSettings(FieldInfo fieldInfo, ThriftCSFFieldSettings settings)

throws SchemaValidationException {

// csfType is required - no need to check if it's set

fieldInfo.getFieldType().setDocValuesType(DocValuesType.NUMERIC);

fieldInfo.getFieldType().setCsfType(settings.getCsfType());

if (settings.isVariableLength()) {

fieldInfo.getFieldType().setDocValuesType(DocValuesType.BINARY);

fieldInfo.getFieldType().setCsfVariableLength();

} else {

if (settings.isSetFixedLengthSettings()) {

fieldInfo.getFieldType().setCsfFixedLengthSettings(

settings.getFixedLengthSettings().getNumValuesPerDoc(),

settings.getFixedLengthSettings().isUpdateable());

if (settings.getFixedLengthSettings().getNumValuesPerDoc() > 1) {

fieldInfo.getFieldType().setDocValuesType(DocValuesType.BINARY);

}

} else {

throw new SchemaValidationException(

"ThriftCSFFieldSettings: Either variableLength should be set to 'true', "

+ "or fixedLengthSettings should be set.");

}

}

fieldInfo.getFieldType().setCsfLoadIntoRam(settings.isLoadIntoRAM());

if (settings.isSetDefaultValue()) {

fieldInfo.getFieldType().setCsfDefaultValue(settings.getDefaultValue());

}

}

private void applyCsfViewFieldSettings(FieldInfo fieldInfo, FieldInfo baseField,

ThriftCSFViewSettings settings)

throws SchemaValidationException {

// csfType is required - no need to check if it's set

fieldInfo.getFieldType().setDocValuesType(DocValuesType.NUMERIC);

fieldInfo.getFieldType().setCsfType(settings.getCsfType());

fieldInfo.getFieldType().setCsfFixedLengthSettings(1 /\* numValuesPerDoc\*/,

false /\* updateable\*/);

fieldInfo.getFieldType().setCsfViewSettings(fieldInfo.getName(), settings, baseField);

}

private void applyFacetFieldSettings(FieldInfo fieldInfo, ThriftFacetFieldSettings settings) {

if (settings.isSetFacetName()) {

fieldInfo.getFieldType().setFacetName(settings.getFacetName());

} else {

// fall back to field name if no facet name is explicitly provided

fieldInfo.getFieldType().setFacetName(fieldInfo.getName());

}

fieldInfo.getFieldType().setStoreFacetSkiplist(settings.isStoreSkiplist());

fieldInfo.getFieldType().setStoreFacetOffensiveCounters(settings.isStoreOffensiveCounters());

fieldInfo.getFieldType().setUseCSFForFacetCounting(settings.isUseCSFForFacetCounting());

}

private void applyIndexedFieldSettings(FieldInfo fieldInfo, ThriftIndexedFieldSettings settings)

throws SchemaValidationException {

fieldInfo.getFieldType().setIndexedField(true);

fieldInfo.getFieldType().setStored(settings.isStored());

fieldInfo.getFieldType().setTokenized(settings.isTokenized());

fieldInfo.getFieldType().setStoreTermVectors(settings.isStoreTermVectors());

fieldInfo.getFieldType().setStoreTermVectorOffsets(settings.isStoreTermVectorOffsets());

fieldInfo.getFieldType().setStoreTermVectorPositions(settings.isStoreTermVectorPositions());

fieldInfo.getFieldType().setStoreTermVectorPayloads(settings.isStoreTermVectorPayloads());

fieldInfo.getFieldType().setOmitNorms(settings.isOmitNorms());

fieldInfo.getFieldType().setIndexHFTermPairs(settings.isIndexHighFreqTermPairs());

fieldInfo.getFieldType().setUseTweetSpecificNormalization(

settings.deprecated\_performTweetSpecificNormalizations);

if (settings.isSetIndexOptions()) {

switch (settings.getIndexOptions()) {

case DOCS\_ONLY :

fieldInfo.getFieldType().setIndexOptions(IndexOptions.DOCS);

break;

case DOCS\_AND\_FREQS :

fieldInfo.getFieldType().setIndexOptions(IndexOptions.DOCS\_AND\_FREQS);

break;

case DOCS\_AND\_FREQS\_AND\_POSITIONS :

fieldInfo.getFieldType().setIndexOptions(IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS);

break;

case DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS :

fieldInfo.getFieldType().setIndexOptions(

IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS);

break;

default:

throw new SchemaValidationException("Unknown value for IndexOptions: "

+ settings.getIndexOptions());

}

} else if (settings.isIndexed()) {

// default for backward-compatibility

fieldInfo.getFieldType().setIndexOptions(IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS);

}

fieldInfo.getFieldType().setStorePerPositionPayloads(settings.isStorePerPositionPayloads());

fieldInfo.getFieldType().setDefaultPayloadLength(

settings.getDefaultPerPositionPayloadLength());

fieldInfo.getFieldType().setBecomesImmutable(!settings.isSupportOutOfOrderAppends());

fieldInfo.getFieldType().setSupportOrderedTerms(settings.isSupportOrderedTerms());

fieldInfo.getFieldType().setSupportTermTextLookup(settings.isSupportTermTextLookup());

if (settings.isSetNumericFieldSettings()) {

fieldInfo.getFieldType().setNumericFieldSettings(

new IndexedNumericFieldSettings(settings.getNumericFieldSettings()));

}

if (settings.isSetTokenStreamSerializer()) {

fieldInfo.getFieldType().setTokenStreamSerializerBuilder(

buildTokenStreamSerializerProvider(settings.getTokenStreamSerializer()));

}

}

private void applySearchFieldSettings(FieldInfo fieldInfo, ThriftSearchFieldSettings settings)

throws SchemaValidationException {

fieldInfo.getFieldType().setTextSearchableFieldWeight(

(float) settings.getTextSearchableFieldWeight());

fieldInfo.getFieldType().setTextSearchableByDefault(settings.isTextDefaultSearchable());

}

private void validate(FieldInfo fieldInfo) throws SchemaValidationException {

}

private TokenStreamSerializer.Builder buildTokenStreamSerializerProvider(

final ThriftTokenStreamSerializer settings) {

TokenStreamSerializer.Builder builder = TokenStreamSerializer.builder();

for (String serializerName : settings.getAttributeSerializerClassNames()) {

try {

builder.add((TokenStreamSerializer.AttributeSerializer) Class.forName(serializerName)

.newInstance());

} catch (InstantiationException e) {

throw new RuntimeException(

"Unable to instantiate AttributeSerializer for name " + serializerName);

} catch (IllegalAccessException e) {

throw new RuntimeException(

"Unable to instantiate AttributeSerializer for name " + serializerName);

} catch (ClassNotFoundException e) {

throw new RuntimeException(

"Unable to instantiate AttributeSerializer for name " + serializerName);

}

}

return builder;

}

@Override

public FacetsConfig getFacetsConfig() {

FacetsConfig facetsConfig = new FacetsConfig();

for (String facetName : facetNameToFieldMap.keySet()) {

// set multiValued = true as default, since we're using SortedSetDocValues facet, in which,

// there is no difference between multiValued true or false for the real facet, but only the

// checking of the values.

facetsConfig.setMultiValued(facetName, true);

}

return facetsConfig;

}

@Override

public Analyzer getDefaultAnalyzer(ThriftAnalyzer override) {

if (override != null) {

return analyzerFactory.getAnalyzer(override);

}

if (defaultAnalyzer != null) {

return analyzerFactory.getAnalyzer(defaultAnalyzer);

}

return new SearchWhitespaceAnalyzer();

}

@Override

public ImmutableCollection<FieldInfo> getFieldInfos() {

return fieldSettingsMapById.values();

}

/\*\*

\* This is the preferred method to check whether a field configuration is in schema.

\* One can also use getFieldInfo and do null checks, but should be careful about excessive

\* warning logging resulting from looking up fields not in schema.

\*/

@Override

public boolean hasField(int fieldConfigId) {

return fieldSettingsMapById.containsKey(fieldConfigId);

}

/\*\*

\* This is the preferred method to check whether a field configuration is in schema.

\* One can also use getFieldInfo and do null checks, but should be careful about excessive

\* warning logging resulting from looking up fields not in schema.

\*/

@Override

public boolean hasField(String fieldName) {

return fieldSettingsMapByName.containsKey(fieldName);

}

/\*\*

\* Get FieldInfo for the given field id.

\* If the goal is to check whether a field is in the schema, use {@link #hasField(int)} instead.

\* This method logs a warning whenever it returns null.

\*/

@Override

@Nullable

public FieldInfo getFieldInfo(int fieldConfigId) {

return getFieldInfo(fieldConfigId, null);

}

private org.apache.lucene.index.FieldInfo convert(String fieldName,

int index,

EarlybirdFieldType type) {

return new org.apache.lucene.index.FieldInfo(

fieldName, // String name

index, // int number

type.storeTermVectors(), // boolean storeTermVector

type.omitNorms(), // boolean omitNorms

type.isStorePerPositionPayloads(), // boolean storePayloads

type.indexOptions(), // IndexOptions indexOptions

type.docValuesType(), // DocValuesType docValues

-1, // long dvGen

Maps.<String, String>newHashMap(), // Map<String, String> attributes

0, // int pointDataDimensionCount

0, // int pointIndexDimensionCount

0, // int pointNumBytes

false); // boolean softDeletesField

}

/\*\*

\* Get FieldInfo for the given field name, or null if the field does not exist.

\*/

@Override

@Nullable

public FieldInfo getFieldInfo(String fieldName) {

return fieldSettingsMapByName.get(fieldName);

}

@Override

public String getFieldName(int fieldConfigId) {

FieldInfo fieldInfo = fieldSettingsMapById.get(fieldConfigId);

return fieldInfo != null ? fieldInfo.getName() : null;

}

@Override

public FieldInfo getFieldInfo(int fieldConfigId, ThriftFieldConfiguration override) {

FieldInfo fieldInfo = fieldSettingsMapById.get(fieldConfigId);

if (fieldInfo == null) {

// This method is used to check the availability of fields by IDs,

// so no warning is logged here (would be too verbose otherwise).

return null;

}

if (override != null) {

try {

return merge(fieldConfigId, fieldInfo, override);

} catch (SchemaValidationException e) {

throw new RuntimeException(e);

}

}

return fieldInfo;

}

@Override

public int getNumFacetFields() {

return numFacetFields;

}

@Override

public FieldInfo getFacetFieldByFacetName(String facetName) {

return facetNameToFieldMap.get(facetName);

}

@Override

public FieldInfo getFacetFieldByFieldName(String fieldName) {

FieldInfo fieldInfo = getFieldInfo(fieldName);

return fieldInfo != null && fieldInfo.getFieldType().isFacetField() ? fieldInfo : null;

}

@Override

public Collection<FieldInfo> getFacetFields() {

return facetNameToFieldMap.values();

}

@Override

public Collection<FieldInfo> getCsfFacetFields() {

return csfFacetFields;

}

@Override

public String getVersionDescription() {

return versionDesc;

}

@Override

public int getMajorVersionNumber() {

return majorVersionNumber;

}

@Override

public int getMinorVersionNumber() {

return minorVersionNumber;

}

@Override

public boolean isVersionOfficial() {

return isVersionOfficial;

}

/\*\*

\* Parses a version string like "16: renamed field x into y" into a version number and

\* a string description.

\* @return a Pair of the version number and the description

\*/

private static Pair<Integer, String> parseVersionString(String version)

throws SchemaValidationException {

Preconditions.checkNotNull(version, "Schema must have a version number and description.");

int colonIndex = version.indexOf(':');

if (colonIndex == -1) {

throw new SchemaValidationException("Malformed version string: " + version);

}

try {

int versionNumber = Integer.parseInt(version.substring(0, colonIndex));

String versionDesc = version.substring(colonIndex + 1);

return Pair.of(versionNumber, versionDesc);

} catch (Exception e) {

throw new SchemaValidationException("Malformed version string: " + version, e);

}

}

@Override

public Map<String, FieldWeightDefault> getFieldWeightMap() {

return fieldWeightMap;

}

/\*\*

\* Build the feature maps so that we can use feature name to get the feature configuration.

\* @return: an immutable map keyed on fieldName.

\*/

private Pair<ImmutableMap<String, FeatureConfiguration>,

ImmutableMap<Integer, FeatureConfiguration>> buildFeatureMaps(

final Map<Integer, ThriftFieldConfiguration> csvViewFields)

throws SchemaValidationException {

final ImmutableMap.Builder<String, FeatureConfiguration> featureConfigMapByNameBuilder =

ImmutableMap.builder();

final ImmutableMap.Builder<Integer, FeatureConfiguration> featureConfigMapByIdBuilder =

ImmutableMap.builder();

for (final Map.Entry<Integer, ThriftFieldConfiguration> entry : csvViewFields.entrySet()) {

ThriftFieldSettings fieldSettings = entry.getValue().getSettings();

FieldInfo fieldInfo = getFieldInfo(entry.getKey());

FieldInfo baseFieldInfo =

getFieldInfo(fieldSettings.getCsfViewSettings().getBaseFieldConfigId());

if (baseFieldInfo == null) {

throw new SchemaValidationException("Base field (id="

+ fieldSettings.getCsfViewSettings().getBaseFieldConfigId() + ") not found.");

}

applyCsfViewFieldSettings(fieldInfo, baseFieldInfo, fieldSettings.getCsfViewSettings());

FeatureConfiguration featureConfig = fieldInfo.getFieldType()

.getCsfViewFeatureConfiguration();

if (featureConfig != null) {

featureConfigMapByNameBuilder.put(fieldInfo.getName(), featureConfig);

featureConfigMapByIdBuilder.put(fieldInfo.getFieldId(), featureConfig);

}

}

return Pair.of(featureConfigMapByNameBuilder.build(), featureConfigMapByIdBuilder.build());

}

@Override

public FeatureConfiguration getFeatureConfigurationByName(String featureName) {

return featureConfigMapByName.get(featureName);

}

@Override

public FeatureConfiguration getFeatureConfigurationById(int featureFieldId) {

return Preconditions.checkNotNull(featureConfigMapById.get(featureFieldId),

"Field ID: " + featureFieldId);

}

@Override

@Nullable

public ThriftCSFType getCSFFieldType(String fieldName) {

FieldInfo fieldInfo = getFieldInfo(fieldName);

if (fieldInfo == null) {

return null;

}

EarlybirdFieldType fieldType = fieldInfo.getFieldType();

if (fieldType.docValuesType() != org.apache.lucene.index.DocValuesType.NUMERIC) {

return null;

}

return fieldType.getCsfType();

}

@Override

public ImmutableSchemaInterface getSchemaSnapshot() {

return this;

}

private FieldInfo merge(int fieldConfigId,

FieldInfo fieldInfo,

ThriftFieldConfiguration overrideConfig)

throws SchemaValidationException {

throw new UnsupportedOperationException("Field override config not supported");

}

@Override

public ThriftSearchFeatureSchema getSearchFeatureSchema() {

return searchFeatureSchema;

}

@Override

public ImmutableMap<Integer, FeatureConfiguration> getFeatureIdToFeatureConfig() {

return featureConfigMapById;

}

@Override

public ImmutableMap<String, FeatureConfiguration> getFeatureNameToFeatureConfig() {

return featureConfigMapByName;

}

private ThriftSearchFeatureSchema createSearchResultFeatureSchema(

String featureSchemaVersionPrefix,

Map<String, FieldInfo> allFieldSettings,

Map<String, FeatureConfiguration> featureConfigurations) throws SchemaValidationException {

final ImmutableMap.Builder<Integer, ThriftSearchFeatureSchemaEntry> builder =

new ImmutableMap.Builder<>();

for (Map.Entry<String, FieldInfo> field : allFieldSettings.entrySet()) {

FeatureConfiguration featureConfig = featureConfigurations.get(field.getKey());

if (featureConfig == null) {

// This is either a not csf related field or a csf field.

continue;

}

// This is a csfView field.

if (featureConfig.getOutputType() == null) {

LOG.info("Skip unused fieldschemas: {} for search feature schema.", field.getKey());

continue;

}

ThriftSearchFeatureType featureType = getResultFeatureType(featureConfig.getOutputType());

if (featureType != null) {

builder.put(

field.getValue().getFieldId(),

new ThriftSearchFeatureSchemaEntry(field.getKey(), featureType));

} else {

LOG.error("Invalid CSFType encountered for csf field: {}", field.getKey());

}

}

Map<Integer, ThriftSearchFeatureSchemaEntry> indexOnlySchemaEntries = builder.build();

// Add earlybird derived features, they are defined in ExternalTweetFeatures and used in the

// scoring function. They are no different from those auto-generated index-based features

// viewed from outside Earlybird.

Map<Integer, ThriftSearchFeatureSchemaEntry> entriesWithEBFeatures =

appendToFeatureSchema(

indexOnlySchemaEntries, ExternalTweetFeature.EARLYBIRD\_DERIVED\_FEATURES);

// Add other features needed for tweet ranking from EarlybirdRankingDerivedFeature.

Map<Integer, ThriftSearchFeatureSchemaEntry> allSchemaEntries = appendToFeatureSchema(

entriesWithEBFeatures, ExternalTweetFeature.EARLYBIRD\_RANKING\_DERIVED\_FEATURES);

long schemaEntriesChecksum = getChecksum(allSchemaEntries);

SearchLongGauge.export("feature\_schema\_checksum", new AtomicLong(schemaEntriesChecksum));

String schemaVersion = String.format(

"%s.%d.%d", featureSchemaVersionPrefix, majorVersionNumber, minorVersionNumber);

ThriftSearchFeatureSchemaSpecifier schemaSpecifier =

new ThriftSearchFeatureSchemaSpecifier(schemaVersion, schemaEntriesChecksum);

ThriftSearchFeatureSchema schema = new ThriftSearchFeatureSchema();

schema.setSchemaSpecifier(schemaSpecifier);

schema.setEntries(allSchemaEntries);

return schema;

}

// Serializes schemaEntries to a byte array, and computes a CRC32 checksum of the array.

// The serialization needs to be stable: if schemaEntries1.equals(schemaEntries2), we want

// this method to produce the same checksum for schemaEntrie1 and schemaEntrie2, even if

// the checksums are computed in different JVMs, etc.

private static long getChecksum(Map<Integer, ThriftSearchFeatureSchemaEntry> schemaEntries)

throws SchemaValidationException {

SortedMap<Integer, ThriftSearchFeatureSchemaEntry> sortedSchemaEntries =

new TreeMap<Integer, ThriftSearchFeatureSchemaEntry>(schemaEntries);

CRC32OutputStream crc32OutputStream = new CRC32OutputStream();

ObjectOutputStream objectOutputStream = null;

try {

objectOutputStream = new ObjectOutputStream(crc32OutputStream);

for (Integer fieldId : sortedSchemaEntries.keySet()) {

objectOutputStream.writeObject(fieldId);

ThriftSearchFeatureSchemaEntry schemaEntry = sortedSchemaEntries.get(fieldId);

objectOutputStream.writeObject(schemaEntry.getFeatureName());

objectOutputStream.writeObject(schemaEntry.getFeatureType());

}

objectOutputStream.flush();

return crc32OutputStream.getValue();

} catch (IOException e) {

throw new SchemaValidationException("Could not serialize feature schema entries.", e);

} finally {

Preconditions.checkNotNull(objectOutputStream);

try {

objectOutputStream.close();

} catch (IOException e) {

throw new SchemaValidationException("Could not close ObjectOutputStream.", e);

}

}

}

/\*\*

\* Get the search feature type based on the csf type.

\* @param csfType the column stride field type for the data

\* @return the corresponding search feature type

\*/

@VisibleForTesting

public static ThriftSearchFeatureType getResultFeatureType(ThriftCSFType csfType) {

switch (csfType) {

case INT:

case BYTE:

return ThriftSearchFeatureType.INT32\_VALUE;

case BOOLEAN:

return ThriftSearchFeatureType.BOOLEAN\_VALUE;

case FLOAT:

case DOUBLE:

return ThriftSearchFeatureType.DOUBLE\_VALUE;

case LONG:

return ThriftSearchFeatureType.LONG\_VALUE;

default:

return null;

}

}

}