package com.twitter.search.common.schema.base;

import javax.annotation.Nullable;

import org.apache.commons.lang.StringUtils;

import org.apache.lucene.document.FieldType;

import org.apache.lucene.index.DocValuesType;

import org.apache.lucene.index.IndexOptions;

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

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

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

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

/\*\*

\* An extension of Lucene's {@link FieldType} that contains additional Earlybird-specific settings.

\* Lucene IndexingChains can downcast the FieldType object to access these additional settings.

\*/

public class EarlybirdFieldType extends FieldType {

public static final EarlybirdFieldType LONG\_CSF\_FIELD\_TYPE = new EarlybirdFieldType();

public static final EarlybirdFieldType INT\_CSF\_FIELD\_TYPE = new EarlybirdFieldType();

public static final EarlybirdFieldType BYTE\_CSF\_FIELD\_TYPE = new EarlybirdFieldType();

static {

LONG\_CSF\_FIELD\_TYPE.setCsfType(ThriftCSFType.LONG);

LONG\_CSF\_FIELD\_TYPE.setDocValuesType(DocValuesType.NUMERIC);

LONG\_CSF\_FIELD\_TYPE.setCsfLoadIntoRam(true);

LONG\_CSF\_FIELD\_TYPE.freeze();

INT\_CSF\_FIELD\_TYPE.setCsfType(ThriftCSFType.INT);

INT\_CSF\_FIELD\_TYPE.setDocValuesType(DocValuesType.NUMERIC);

INT\_CSF\_FIELD\_TYPE.setCsfLoadIntoRam(true);

INT\_CSF\_FIELD\_TYPE.freeze();

BYTE\_CSF\_FIELD\_TYPE.setCsfType(ThriftCSFType.BYTE);

BYTE\_CSF\_FIELD\_TYPE.setDocValuesType(DocValuesType.NUMERIC);

BYTE\_CSF\_FIELD\_TYPE.setCsfLoadIntoRam(true);

BYTE\_CSF\_FIELD\_TYPE.freeze();

}

private boolean storePerPositionPayloads;

private int defaultPayloadLength;

// This is true for fields that become immutable after optimization

private boolean becomesImmutable = true;

private boolean supportOrderedTerms;

private boolean supportTermTextLookup;

private boolean indexHFTermPairs;

/\*\*

\* This flag turns on tweet specific normalizations.

\* This turns on the following two token processors:

\* {@link com.twitter.search.common.util.text.splitter.HashtagMentionPunctuationSplitter}

\* {@link com.twitter.search.common.util.text.filter.NormalizedTokenFilter}

\*

\* HashtagMentionPunctuationSplitter would break a mention or hashtag like @ab\_cd or #ab\_cd into

\* tokens {ab, cd}.

\* NormalizedTokenFilter strips out the # @ $ from the tokens.

\*

\*

\* @deprecated we should remove this flag. It is confusing to have Earlybird apply additional

\* tokenization on top of what ingester produced.

\*/

@Deprecated

private boolean useTweetSpecificNormalization;

@Nullable

private TokenStreamSerializer.Builder tokenStreamSerializerProvider = null;

// csf type settings

private ThriftCSFType csfType;

private boolean csfVariableLength;

private int csfFixedLengthNumValuesPerDoc;

private boolean csfFixedLengthUpdateable;

private boolean csfLoadIntoRam;

private boolean csfDefaultValueSet;

private long csfDefaultValue;

// True if this is a CSF field which is a view on top of a different CSF field

private boolean csfViewField;

// If this field is a csf view, this is the ID of the CSF field backing the view

private int csfViewBaseFieldId;

private FeatureConfiguration csfViewFeatureConfiguration;

// facet field settings

private String facetName;

private boolean storeFacetSkiplist;

private boolean storeFacetOffensiveCounters;

private boolean useCSFForFacetCounting;

// Determines if this field is indexed

private boolean indexedField = false;

// search field settings

// whether a field should be searched by default

private boolean textSearchableByDefault = false;

private float textSearchableFieldWeight = 1.0f;

// For indexed numerical fields

private IndexedNumericFieldSettings numericFieldSettings = null;

public boolean isStorePerPositionPayloads() {

return storePerPositionPayloads;

}

public void setStorePerPositionPayloads(boolean storePerPositionPayloads) {

checkIfFrozen();

this.storePerPositionPayloads = storePerPositionPayloads;

}

public int getDefaultPayloadLength() {

return defaultPayloadLength;

}

public void setDefaultPayloadLength(int defaultPayloadLength) {

checkIfFrozen();

this.defaultPayloadLength = defaultPayloadLength;

}

public boolean becomesImmutable() {

return becomesImmutable;

}

public void setBecomesImmutable(boolean becomesImmutable) {

checkIfFrozen();

this.becomesImmutable = becomesImmutable;

}

public boolean isSupportOrderedTerms() {

return supportOrderedTerms;

}

public void setSupportOrderedTerms(boolean supportOrderedTerms) {

checkIfFrozen();

this.supportOrderedTerms = supportOrderedTerms;

}

public boolean isSupportTermTextLookup() {

return supportTermTextLookup;

}

public void setSupportTermTextLookup(boolean supportTermTextLookup) {

this.supportTermTextLookup = supportTermTextLookup;

}

@Nullable

public TokenStreamSerializer getTokenStreamSerializer() {

return tokenStreamSerializerProvider == null ? null : tokenStreamSerializerProvider.safeBuild();

}

public void setTokenStreamSerializerBuilder(TokenStreamSerializer.Builder provider) {

checkIfFrozen();

this.tokenStreamSerializerProvider = provider;

}

public ThriftCSFType getCsfType() {

return csfType;

}

public void setCsfType(ThriftCSFType csfType) {

checkIfFrozen();

this.csfType = csfType;

}

public boolean isCsfVariableLength() {

return csfVariableLength;

}

public int getCsfFixedLengthNumValuesPerDoc() {

return csfFixedLengthNumValuesPerDoc;

}

public void setCsfVariableLength() {

checkIfFrozen();

this.csfVariableLength = true;

}

/\*\*

\* Make the field a fixed length CSF, with the given length.

\*/

public void setCsfFixedLengthSettings(int csfFixedLengthNumValuesPerDocument,

boolean isCsfFixedLengthUpdateable) {

checkIfFrozen();

this.csfVariableLength = false;

this.csfFixedLengthNumValuesPerDoc = csfFixedLengthNumValuesPerDocument;

this.csfFixedLengthUpdateable = isCsfFixedLengthUpdateable;

}

public boolean isCsfFixedLengthUpdateable() {

return csfFixedLengthUpdateable;

}

public boolean isCsfLoadIntoRam() {

return csfLoadIntoRam;

}

public void setCsfLoadIntoRam(boolean csfLoadIntoRam) {

checkIfFrozen();

this.csfLoadIntoRam = csfLoadIntoRam;

}

public void setCsfDefaultValue(long defaultValue) {

checkIfFrozen();

this.csfDefaultValue = defaultValue;

this.csfDefaultValueSet = true;

}

public long getCsfDefaultValue() {

return csfDefaultValue;

}

public boolean isCsfDefaultValueSet() {

return csfDefaultValueSet;

}

public String getFacetName() {

return facetName;

}

public void setFacetName(String facetName) {

checkIfFrozen();

this.facetName = facetName;

}

public boolean isStoreFacetSkiplist() {

return storeFacetSkiplist;

}

public void setStoreFacetSkiplist(boolean storeFacetSkiplist) {

checkIfFrozen();

this.storeFacetSkiplist = storeFacetSkiplist;

}

public boolean isStoreFacetOffensiveCounters() {

return storeFacetOffensiveCounters;

}

public void setStoreFacetOffensiveCounters(boolean storeFacetOffensiveCounters) {

checkIfFrozen();

this.storeFacetOffensiveCounters = storeFacetOffensiveCounters;

}

public boolean isUseCSFForFacetCounting() {

return useCSFForFacetCounting;

}

public void setUseCSFForFacetCounting(boolean useCSFForFacetCounting) {

checkIfFrozen();

this.useCSFForFacetCounting = useCSFForFacetCounting;

}

public boolean isFacetField() {

return facetName != null && !StringUtils.isEmpty(facetName);

}

public boolean isIndexHFTermPairs() {

return indexHFTermPairs;

}

public void setIndexHFTermPairs(boolean indexHFTermPairs) {

checkIfFrozen();

this.indexHFTermPairs = indexHFTermPairs;

}

public boolean acceptPretokenizedField() {

return tokenStreamSerializerProvider != null;

}

/\*\*

\* set this field to use additional twitter specific tokenization.

\* @deprecated should avoid doing additional tokenizations on top of what ingester produced.

\*/

@Deprecated

public boolean useTweetSpecificNormalization() {

return useTweetSpecificNormalization;

}

/\*\*

\* test whether this field uses additional twitter specific tokenization.

\* @deprecated should avoid doing additional tokenizations on top of what ingester produced.

\*/

@Deprecated

public void setUseTweetSpecificNormalization(boolean useTweetSpecificNormalization) {

checkIfFrozen();

this.useTweetSpecificNormalization = useTweetSpecificNormalization;

}

public boolean isIndexedField() {

return indexedField;

}

public void setIndexedField(boolean indexedField) {

this.indexedField = indexedField;

}

public boolean isTextSearchableByDefault() {

return textSearchableByDefault;

}

public void setTextSearchableByDefault(boolean textSearchableByDefault) {

checkIfFrozen();

this.textSearchableByDefault = textSearchableByDefault;

}

public float getTextSearchableFieldWeight() {

return textSearchableFieldWeight;

}

public void setTextSearchableFieldWeight(float textSearchableFieldWeight) {

checkIfFrozen();

this.textSearchableFieldWeight = textSearchableFieldWeight;

}

/\*\*

\* Convenience method to find out if this field stores positions. {@link #indexOptions()} can also

\* be used to determine the index options for this field.

\*/

public final boolean hasPositions() {

return indexOptions() == IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS

|| indexOptions() == IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS;

}

public boolean isCsfViewField() {

return csfViewField;

}

public int getCsfViewBaseFieldId() {

return csfViewBaseFieldId;

}

public FeatureConfiguration getCsfViewFeatureConfiguration() {

return csfViewFeatureConfiguration;

}

/\*\*

\* Set the CSF view settings. A CSF view is a portion of an another CSF.

\*/

public void setCsfViewSettings(String fieldName,

ThriftCSFViewSettings csfViewSettings,

Schema.FieldInfo baseField) {

checkIfFrozen();

this.csfViewField = true;

this.csfViewBaseFieldId = csfViewSettings.getBaseFieldConfigId();

FeatureConfiguration.Builder builder = FeatureConfiguration.builder()

.withName(fieldName)

.withType(csfViewSettings.csfType)

.withBitRange(csfViewSettings.getValueIndex(),

csfViewSettings.getBitStartPosition(),

csfViewSettings.getBitLength())

.withBaseField(baseField.getName());

if (csfViewSettings.isSetOutputCSFType()) {

builder.withOutputType(csfViewSettings.getOutputCSFType());

}

if (csfViewSettings.isSetNormalizationType()) {

builder.withFeatureNormalizationType(csfViewSettings.getNormalizationType());

}

if (csfViewSettings.isSetFeatureUpdateConstraints()) {

for (ThriftFeatureUpdateConstraint c : csfViewSettings.getFeatureUpdateConstraints()) {

builder.withFeatureUpdateConstraint(c);

}

}

this.csfViewFeatureConfiguration = builder.build();

}

public IndexedNumericFieldSettings getNumericFieldSettings() {

return numericFieldSettings;

}

public void setNumericFieldSettings(IndexedNumericFieldSettings numericFieldSettings) {

checkIfFrozen();

this.numericFieldSettings = numericFieldSettings;

}

}