package com.twitter.search.ingester.pipeline.twitter;

import java.util.List;

import java.util.Map;

import javax.annotation.Nonnull;

import javax.annotation.Nullable;

import javax.naming.NamingException;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import org.apache.commons.pipeline.StageException;

import org.apache.commons.pipeline.validation.ConsumedTypes;

import org.apache.commons.pipeline.validation.ProducedTypes;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common\_internal.text.version.PenguinVersion;

import com.twitter.search.common.debug.thriftjava.DebugEvents;

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

import com.twitter.search.common.relevance.entities.TwitterMessage;

import com.twitter.search.common.schema.earlybird.EarlybirdCluster;

import com.twitter.search.ingester.model.IngesterTweetEvent;

import com.twitter.search.ingester.model.IngesterTwitterMessage;

import com.twitter.search.ingester.pipeline.twitter.thriftparse.ThriftTweetParsingException;

import com.twitter.search.ingester.pipeline.twitter.thriftparse.TweetEventParseHelper;

import com.twitter.tweetypie.thriftjava.TweetCreateEvent;

import com.twitter.tweetypie.thriftjava.TweetDeleteEvent;

import com.twitter.tweetypie.thriftjava.TweetEventData;

@ConsumedTypes(IngesterTweetEvent.class)

@ProducedTypes(IngesterTwitterMessage.class)

public class ThriftTweetParserStage extends TwitterBaseStage<IngesterTweetEvent, TwitterMessage> {

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

// TweetEventData is a union of all possible tweet event types. TweetEventData.\_Fields is an enum

// that corresponds to the fields in that union. So essentially, TweetEventData.\_Fields tells us

// which tweet event we're getting inside TweetEventData. We want to keep track of how many tweet

// events of each type we're getting.

private final Map<TweetEventData.\_Fields, SearchCounter> tweetEventCounters =

Maps.newEnumMap(TweetEventData.\_Fields.class);

private final List<String> tweetCreateEventBranches = Lists.newArrayList();

private final List<String> tweetDeleteEventBranches = Lists.newArrayList();

private boolean shouldIndexProtectedTweets;

private SearchCounter totalEventsCount;

private SearchCounter thriftParsingErrorsCount;

private List<PenguinVersion> supportedPenguinVersions;

@Override

protected void initStats() {

super.initStats();

for (TweetEventData.\_Fields field : TweetEventData.\_Fields.values()) {

tweetEventCounters.put(

field,

this.makeStageCounter(field.name().toLowerCase() + "\_count"));

}

totalEventsCount = this.makeStageCounter("total\_events\_count");

thriftParsingErrorsCount = this.makeStageCounter("thrift\_parsing\_errors\_count");

}

@Override

protected void doInnerPreprocess() throws StageException, NamingException {

supportedPenguinVersions = wireModule.getPenguinVersions();

LOG.info("Supported penguin versions: {}", supportedPenguinVersions);

shouldIndexProtectedTweets = earlybirdCluster == EarlybirdCluster.PROTECTED

|| earlybirdCluster == EarlybirdCluster.REALTIME\_CG;

Preconditions.checkState(!tweetDeleteEventBranches.isEmpty(),

"At least one delete branch must be specified.");

}

@Override

public void innerProcess(Object obj) throws StageException {

if (!(obj instanceof TweetEventData || obj instanceof IngesterTweetEvent)) {

LOG.error("Object is not a TweetEventData or IngesterTweetEvent: {}", obj);

throw new StageException(this, "Object is not a TweetEventData or IngesterTweetEvent");

}

supportedPenguinVersions = wireModule.getCurrentlyEnabledPenguinVersions();

try {

IngesterTweetEvent ingesterTweetEvent = (IngesterTweetEvent) obj;

TweetEventData tweetEventData = ingesterTweetEvent.getData();

DebugEvents debugEvents = ingesterTweetEvent.getDebugEvents();

// Determine if the message is a tweet delete event before the next stages mutate it.

IngesterTwitterMessage message = getTwitterMessage(tweetEventData, debugEvents);

boolean shouldEmitMessage = message != null

&& message.isIndexable(shouldIndexProtectedTweets);

if (shouldEmitMessage) {

if (!message.isDeleted()) {

emitAndCount(message);

for (String tweetCreateEventBranch : tweetCreateEventBranches) {

// If we need to send the message to another branch, we need to make a copy.

// Otherwise, we'll have multiple stages mutating the same object in parallel.

IngesterTwitterMessage tweetCreateEventBranchMessage =

getTwitterMessage(tweetEventData, debugEvents);

emitToBranchAndCount(tweetCreateEventBranch, tweetCreateEventBranchMessage);

}

} else {

for (String tweetDeleteEventBranch : tweetDeleteEventBranches) {

// If we need to send the message to another branch, we need to make a copy.

// Otherwise, we'll have multiple stages mutating the same object in parallel.

IngesterTwitterMessage tweetDeleteEventBranchMessage =

getTwitterMessage(tweetEventData, debugEvents);

emitToBranchAndCount(tweetDeleteEventBranch, tweetDeleteEventBranchMessage);

}

}

}

} catch (ThriftTweetParsingException e) {

thriftParsingErrorsCount.increment();

LOG.error("Failed to parse Thrift tweet event: " + obj, e);

throw new StageException(this, e);

}

}

@Nullable

private IngesterTwitterMessage getTwitterMessage(

@Nonnull TweetEventData tweetEventData,

@Nullable DebugEvents debugEvents)

throws ThriftTweetParsingException {

totalEventsCount.increment();

// TweetEventData is a union of all possible tweet event types. TweetEventData.\_Fields is an

// enum that corresponds to all TweetEventData fields. By calling TweetEventData.getSetField(),

// we can determine which field is set.

TweetEventData.\_Fields tweetEventDataField = tweetEventData.getSetField();

Preconditions.checkNotNull(tweetEventDataField);

tweetEventCounters.get(tweetEventDataField).increment();

if (tweetEventDataField == TweetEventData.\_Fields.TWEET\_CREATE\_EVENT) {

TweetCreateEvent tweetCreateEvent = tweetEventData.getTweet\_create\_event();

return TweetEventParseHelper.getTwitterMessageFromCreationEvent(

tweetCreateEvent, supportedPenguinVersions, debugEvents);

}

if (tweetEventDataField == TweetEventData.\_Fields.TWEET\_DELETE\_EVENT) {

TweetDeleteEvent tweetDeleteEvent = tweetEventData.getTweet\_delete\_event();

return TweetEventParseHelper.getTwitterMessageFromDeletionEvent(

tweetDeleteEvent, supportedPenguinVersions, debugEvents);

}

return null;

}

/\*\*

\* Sets the branches to which all TweetDeleteEvents should be emitted.

\*

\* @param tweetDeleteEventBranchNames A comma-separated list of branches.

\*/

public void setTweetDeleteEventBranchNames(String tweetDeleteEventBranchNames) {

parseBranches(tweetDeleteEventBranchNames, tweetDeleteEventBranches);

}

/\*\*

\* Sets the additional branches to which all TweetCreateEvents should be emitted.

\*

\* @param tweetCreateEventBranchNames A comma-separated list of branches.

\*/

public void setTweetCreateEventBranchNames(String tweetCreateEventBranchNames) {

parseBranches(tweetCreateEventBranchNames, tweetCreateEventBranches);

}

private void parseBranches(String branchNames, List<String> branches) {

branches.clear();

for (String branch : branchNames.split(",")) {

String trimmedBranch = branch.trim();

Preconditions.checkState(!trimmedBranch.isEmpty(), "Branches cannot be empty strings.");

branches.add(trimmedBranch);

}

}

}