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

import java.util.Date;

import java.util.List;

import java.util.Optional;

import javax.annotation.Nonnull;

import javax.annotation.Nullable;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.Lists;

import org.apache.commons.lang.StringEscapeUtils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import com.twitter.dataproducts.enrichments.thriftjava.GeoEntity;

import com.twitter.dataproducts.enrichments.thriftjava.PotentialLocation;

import com.twitter.dataproducts.enrichments.thriftjava.ProfileGeoEnrichment;

import com.twitter.escherbird.thriftjava.TweetEntityAnnotation;

import com.twitter.expandodo.thriftjava.Card2;

import com.twitter.gizmoduck.thriftjava.User;

import com.twitter.mediaservices.commons.tweetmedia.thrift\_java.MediaInfo;

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

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

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

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

import com.twitter.search.common.partitioning.snowflakeparser.SnowflakeIdParser;

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

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

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

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

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

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

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

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

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

import com.twitter.search.ingester.pipeline.util.CardFieldUtil;

import com.twitter.service.spiderduck.gen.MediaTypes;

import com.twitter.tweetypie.thriftjava.DeviceSource;

import com.twitter.tweetypie.thriftjava.DirectedAtUser;

import com.twitter.tweetypie.thriftjava.EscherbirdEntityAnnotations;

import com.twitter.tweetypie.thriftjava.ExclusiveTweetControl;

import com.twitter.tweetypie.thriftjava.GeoCoordinates;

import com.twitter.tweetypie.thriftjava.HashtagEntity;

import com.twitter.tweetypie.thriftjava.MediaEntity;

import com.twitter.tweetypie.thriftjava.MentionEntity;

import com.twitter.tweetypie.thriftjava.Place;

import com.twitter.tweetypie.thriftjava.QuotedTweet;

import com.twitter.tweetypie.thriftjava.Reply;

import com.twitter.tweetypie.thriftjava.Tweet;

import com.twitter.tweetypie.thriftjava.TweetCoreData;

import com.twitter.tweetypie.thriftjava.TweetCreateEvent;

import com.twitter.tweetypie.thriftjava.TweetDeleteEvent;

import com.twitter.tweetypie.thriftjava.UrlEntity;

import com.twitter.tweetypie.tweettext.PartialHtmlEncoding;

/\*\*

\* This is an utility class for converting Thrift TweetEvent messages sent by TweetyPie

\* into ingester internal representation, IngesterTwitterMessage.

\*/

public final class TweetEventParseHelper {

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

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_NULL\_TEXT =

SearchCounter.export("tweets\_with\_null\_text\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter TWEET\_SIZE = SearchCounter.export("tweet\_size\_from\_thrift");

@VisibleForTesting

static final Percentile<Long> TWEET\_SIZE\_PERCENTILES =

PercentileUtil.createPercentile("tweet\_size\_from\_thrift");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_CONVERSATION\_ID =

SearchCounter.export("tweets\_with\_conversation\_id\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_QUOTE =

SearchCounter.export("tweets\_with\_quote\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_ANNOTATIONS =

SearchCounter.export("tweets\_with\_annotation\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_ANNOTATIONS\_ADDED =

SearchCounter.export("num\_annotations\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_COORDINATE\_FIELD =

SearchCounter.export("tweets\_with\_coordinate\_field\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_PLACE\_ADDED =

SearchCounter.export("num\_places\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_PLACE\_FIELD =

SearchCounter.export("tweets\_with\_place\_field\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_PLACE\_COUNTRY\_CODE =

SearchCounter.export("tweets\_with\_place\_country\_code\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_USE\_PLACE\_FIELD =

SearchCounter.export("tweets\_use\_place\_field\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_CANNOT\_PARSE\_PLACE\_FIELD =

SearchCounter.export("tweets\_cannot\_parse\_place\_field\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_PROFILE\_GEO\_ENRICHMENT =

SearchCounter.export("tweets\_with\_profile\_geo\_enrichment\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_MENTIONS =

SearchCounter.export("tweets\_with\_mentions\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_MENTIONS\_ADDED =

SearchCounter.export("num\_mentions\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_HASHTAGS =

SearchCounter.export("tweets\_with\_hashtags\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_HASHTAGS\_ADDED =

SearchCounter.export("num\_hashtags\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_MEDIA\_URL =

SearchCounter.export("tweets\_with\_media\_url\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_MEDIA\_URLS\_ADDED =

SearchCounter.export("num\_media\_urls\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_PHOTO\_MEDIA\_URL =

SearchCounter.export("tweets\_with\_photo\_media\_url\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_VIDEO\_MEDIA\_URL =

SearchCounter.export("tweets\_with\_video\_media\_url\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_WITH\_NON\_MEDIA\_URL =

SearchCounter.export("tweets\_with\_non\_media\_url\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_NON\_MEDIA\_URLS\_ADDED =

SearchCounter.export("num\_non\_media\_urls\_from\_thrift\_cnt");

@VisibleForTesting

static final SearchCounter NUM\_TWEETS\_MISSING\_QUOTE\_URLS =

SearchCounter.export("num\_tweets\_missing\_quote\_urls\_cnt");

// Utility class, disallow instantiation.

private TweetEventParseHelper() {

}

/\*\* Builds an IngesterTwitterMessage instance from a TweetCreateEvent. \*/

@Nonnull

public static IngesterTwitterMessage getTwitterMessageFromCreationEvent(

@Nonnull TweetCreateEvent createEvent,

@Nonnull List<PenguinVersion> supportedPenguinVersions,

@Nullable DebugEvents debugEvents) throws ThriftTweetParsingException {

Tweet tweet = createEvent.getTweet();

if (tweet == null) {

throw new ThriftTweetParsingException("No tweet field in TweetCreateEvent");

}

TweetCoreData coreData = tweet.getCore\_data();

if (coreData == null) {

throw new ThriftTweetParsingException("No core\_data field in Tweet in TweetCreateEvent");

}

User user = createEvent.getUser();

if (user == null) {

throw new ThriftTweetParsingException("No user field in TweetCreateEvent");

}

if (!user.isSetProfile()) {

throw new ThriftTweetParsingException("No profile field in User in TweetCreateEvent");

}

if (!user.isSetSafety()) {

throw new ThriftTweetParsingException("No safety field in User in TweetCreateEvent");

}

long twitterId = tweet.getId();

IngesterTwitterMessage message = new IngesterTwitterMessage(

twitterId,

supportedPenguinVersions,

debugEvents);

// Set the creation time based on the tweet ID, because it has millisecond granularity,

// and coreData.created\_at\_secs has only second granularity.

message.setDate(new Date(SnowflakeIdParser.getTimestampFromTweetId(twitterId)));

boolean isNsfw = coreData.isNsfw\_admin() || coreData.isNsfw\_user();

boolean hasMediaOrUrlsOrCards =

tweet.getMediaSize() > 0

|| tweet.getUrlsSize() > 0

|| tweet.getCardsSize() > 0

|| tweet.isSetCard2();

message.setIsSensitiveContent(isNsfw && hasMediaOrUrlsOrCards);

message.setFromUser(getFromUser(user));

if (user.isSetCounts()) {

message.setFollowersCount((int) user.getCounts().getFollowers());

}

message.setUserProtected(user.getSafety().isIs\_protected());

message.setUserVerified(user.getSafety().isVerified());

message.setUserBlueVerified(user.getSafety().isIs\_blue\_verified());

if (tweet.isSetLanguage()) {

message.setLanguage(tweet.getLanguage().getLanguage()); // language ID like "en"

}

if (tweet.isSetSelf\_thread\_metadata()) {

message.setSelfThread(true);

}

ExclusiveTweetControl exclusiveTweetControl = tweet.getExclusive\_tweet\_control();

if (exclusiveTweetControl != null) {

if (exclusiveTweetControl.isSetConversation\_author\_id()) {

message.setExclusiveConversationAuthorId(

exclusiveTweetControl.getConversation\_author\_id());

}

}

setDirectedAtUser(message, coreData);

addMentionsToMessage(message, tweet);

addHashtagsToMessage(message, tweet);

addMediaEntitiesToMessage(message, tweet.getId(), tweet.getMedia());

addUrlsToMessage(message, tweet.getUrls());

addEscherbirdAnnotationsToMessage(message, tweet);

message.setNullcast(coreData.isNullcast());

if (coreData.isSetConversation\_id()) {

message.setConversationId(coreData.getConversation\_id());

NUM\_TWEETS\_WITH\_CONVERSATION\_ID.increment();

}

// quotes

if (tweet.isSetQuoted\_tweet()) {

QuotedTweet quotedTweet = tweet.getQuoted\_tweet();

if (quotedTweet.getTweet\_id() > 0 && quotedTweet.getUser\_id() > 0) {

if (quotedTweet.isSetPermalink()) {

String quotedURL = quotedTweet.getPermalink().getLong\_url();

UrlEntity quotedURLEntity = new UrlEntity();

quotedURLEntity.setExpanded(quotedURL);

quotedURLEntity.setUrl(quotedTweet.getPermalink().getShort\_url());

quotedURLEntity.setDisplay(quotedTweet.getPermalink().getDisplay\_text());

addUrlsToMessage(message, Lists.newArrayList(quotedURLEntity));

} else {

LOG.warn("Tweet {} has quoted tweet, but is missing quoted tweet URL: {}",

tweet.getId(), quotedTweet);

NUM\_TWEETS\_MISSING\_QUOTE\_URLS.increment();

}

TwitterQuotedMessage quotedMessage =

new TwitterQuotedMessage(

quotedTweet.getTweet\_id(),

quotedTweet.getUser\_id());

message.setQuotedMessage(quotedMessage);

NUM\_TWEETS\_WITH\_QUOTE.increment();

}

}

// card fields

if (createEvent.getTweet().isSetCard2()) {

Card2 card = createEvent.getTweet().getCard2();

message.setCardName(card.getName());

message.setCardTitle(

CardFieldUtil.extractBindingValue(CardFieldUtil.TITLE\_BINDING\_KEY, card));

message.setCardDescription(

CardFieldUtil.extractBindingValue(CardFieldUtil.DESCRIPTION\_BINDING\_KEY, card));

CardFieldUtil.deriveCardLang(message);

message.setCardUrl(card.getUrl());

}

// Some fields should be set based on the "original" tweet. So if this tweet is a retweet,

// we want to extract those fields from the retweeted tweet.

Tweet retweetOrTweet = tweet;

TweetCoreData retweetOrTweetCoreData = coreData;

User retweetOrTweetUser = user;

// retweets

boolean isRetweet = coreData.isSetShare();

if (isRetweet) {

retweetOrTweet = createEvent.getSource\_tweet();

retweetOrTweetCoreData = retweetOrTweet.getCore\_data();

retweetOrTweetUser = createEvent.getSource\_user();

TwitterRetweetMessage retweetMessage = new TwitterRetweetMessage();

retweetMessage.setRetweetId(twitterId);

if (retweetOrTweetUser != null) {

if (retweetOrTweetUser.isSetProfile()) {

retweetMessage.setSharedUserDisplayName(retweetOrTweetUser.getProfile().getName());

}

retweetMessage.setSharedUserTwitterId(retweetOrTweetUser.getId());

}

retweetMessage.setSharedDate(new Date(retweetOrTweetCoreData.getCreated\_at\_secs() \* 1000));

retweetMessage.setSharedId(retweetOrTweet.getId());

addMediaEntitiesToMessage(message, retweetOrTweet.getId(), retweetOrTweet.getMedia());

addUrlsToMessage(message, retweetOrTweet.getUrls());

// If a tweet's text is longer than 140 characters, the text for any retweet of that tweet

// will be truncated. And if the original tweet has hashtags or mentions after character 140,

// the Tweetypie event for the retweet will not include those hashtags/mentions, which will

// make the retweet unsearchable by those hashtags/mentions. So in order to avoid this

// problem, we add to the retweet all hashtags/mentions set on the original tweet.

addMentionsToMessage(message, retweetOrTweet);

addHashtagsToMessage(message, retweetOrTweet);

message.setRetweetMessage(retweetMessage);

}

// Some fields should be set based on the "original" tweet.

// Only set geo fields if this is not a retweet

if (!isRetweet) {

setGeoFields(message, retweetOrTweetCoreData, retweetOrTweetUser);

setPlacesFields(message, retweetOrTweet);

}

setText(message, retweetOrTweetCoreData);

setInReplyTo(message, retweetOrTweetCoreData, isRetweet);

setDeviceSourceField(message, retweetOrTweet);

// Profile geo enrichment fields should be set based on this tweet, even if it's a retweet.

setProfileGeoEnrichmentFields(message, tweet);

// The composer used to create this tweet: standard tweet creator or the camera flow.

setComposerSource(message, tweet);

return message;

}

private static void setGeoFields(

TwitterMessage message, TweetCoreData coreData, User user) {

if (coreData.isSetCoordinates()) {

NUM\_TWEETS\_WITH\_COORDINATE\_FIELD.increment();

GeoCoordinates coords = coreData.getCoordinates();

message.setGeoTaggedLocation(

GeoObject.createForIngester(coords.getLatitude(), coords.getLongitude()));

String location =

String.format("GeoAPI:%.4f,%.4f", coords.getLatitude(), coords.getLongitude());

TwitterMessageUtil.setAndTruncateLocationOnMessage(message, location);

}

// If the location was not set from the coordinates.

if ((message.getOrigLocation() == null) && (user != null) && user.isSetProfile()) {

TwitterMessageUtil.setAndTruncateLocationOnMessage(message, user.getProfile().getLocation());

}

}

private static void setPlacesFields(TwitterMessage message, Tweet tweet) {

if (!tweet.isSetPlace()) {

return;

}

Place place = tweet.getPlace();

if (place.isSetContainers() && place.getContainersSize() > 0) {

NUM\_TWEETS\_WITH\_PLACE\_FIELD.increment();

NUM\_PLACE\_ADDED.add(place.getContainersSize());

for (String placeId : place.getContainers()) {

message.addPlace(placeId);

}

}

Preconditions.checkArgument(place.isSetId(), "Tweet.Place without id.");

message.setPlaceId(place.getId());

Preconditions.checkArgument(place.isSetFull\_name(), "Tweet.Place without full\_name.");

message.setPlaceFullName(place.getFull\_name());

if (place.isSetCountry\_code()) {

message.setPlaceCountryCode(place.getCountry\_code());

NUM\_TWEETS\_WITH\_PLACE\_COUNTRY\_CODE.increment();

}

if (message.getGeoTaggedLocation() == null) {

Optional<GeoObject> location = GeoObject.fromPlace(place);

if (location.isPresent()) {

NUM\_TWEETS\_USE\_PLACE\_FIELD.increment();

message.setGeoTaggedLocation(location.get());

} else {

NUM\_TWEETS\_CANNOT\_PARSE\_PLACE\_FIELD.increment();

}

}

}

private static void setText(TwitterMessage message, TweetCoreData coreData) {

/\*\*

\* TweetyPie doesn't do a full HTML escaping of the text, only a partial escaping

\* so we use their code to unescape it first, then we do

\* a second unescaping because when the tweet text itself has HTML escape

\* sequences, we want to index the unescaped version, not the escape sequence itself.

\* --

\* Yes, we \*double\* unescape html. About 1-2 tweets per second are double escaped,

\* and we probably want to index the real text and not things like '&#9733;'.

\* Unescaping already unescaped text seems safe in practice.

\* --

\*

\* This may seem wrong, because one thinks we should index whatever the user posts,

\* but given punctuation stripping this creates odd behavior:

\*

\* If someone tweets &amp; they won't be able to find it by searching for '&amp;' because

\* the tweet will be indexed as 'amp'

\*

\* It would also prevent some tweets from surfacing for certain searches, for example:

\*

\* User Tweets: John Mayer &amp; Dave Chappelle

\* We Unescape To: John Mayer & Dave Chappelle

\* We Strip/Normalize To: john mayer dave chappelle

\*

\* A user searching for 'John Mayer Dave Chappelle' would get the above tweet.

\*

\* If we didn't double unescape

\*

\* User Tweets: John Mayer &amp; Dave Chappelle

\* We Strip/Normalize To: john mayer amp dave chappelle

\*

\* A user searching for 'John Mayer Dave Chappelle' would miss the above tweet.

\*

\* Second example

\*

\* User Tweets: L'Humanit&eacute;

\* We Unescape To: L'Humanité

\* We Strip/Normalize To: l humanite

\*

\* If we didn't double escape

\*

\* User Tweets: L'Humanit&eacute;

\* We Strip/Normalize To: l humanit eacute

\*

\*/

String text = coreData.isSetText()

? StringEscapeUtils.unescapeHtml(PartialHtmlEncoding.decode(coreData.getText()))

: coreData.getText();

message.setText(text);

if (text != null) {

long tweetLength = text.length();

TWEET\_SIZE.add(tweetLength);

TWEET\_SIZE\_PERCENTILES.record(tweetLength);

} else {

NUM\_TWEETS\_WITH\_NULL\_TEXT.increment();

}

}

private static void setInReplyTo(

TwitterMessage message, TweetCoreData coreData, boolean isRetweet) {

Reply reply = coreData.getReply();

if (!isRetweet && reply != null) {

String inReplyToScreenName = reply.getIn\_reply\_to\_screen\_name();

long inReplyToUserId = reply.getIn\_reply\_to\_user\_id();

message.replaceToUserWithInReplyToUserIfNeeded(inReplyToScreenName, inReplyToUserId);

}

if ((reply != null) && reply.isSetIn\_reply\_to\_status\_id()) {

message.setInReplyToStatusId(reply.getIn\_reply\_to\_status\_id());

}

}

private static void setProfileGeoEnrichmentFields(TwitterMessage message, Tweet tweet) {

if (!tweet.isSetProfile\_geo\_enrichment()) {

return;

}

ProfileGeoEnrichment profileGeoEnrichment = tweet.getProfile\_geo\_enrichment();

List<PotentialLocation> thriftPotentialLocations =

profileGeoEnrichment.getPotential\_locations();

if (!thriftPotentialLocations.isEmpty()) {

NUM\_TWEETS\_WITH\_PROFILE\_GEO\_ENRICHMENT.increment();

List<PotentialLocationObject> potentialLocations = Lists.newArrayList();

for (PotentialLocation potentialLocation : thriftPotentialLocations) {

GeoEntity geoEntity = potentialLocation.getGeo\_entity();

potentialLocations.add(new PotentialLocationObject(geoEntity.getCountry\_code(),

geoEntity.getRegion(),

geoEntity.getLocality()));

}

message.setPotentialLocations(potentialLocations);

}

}

private static void setDeviceSourceField(TwitterMessage message, Tweet tweet) {

DeviceSource deviceSource = tweet.getDevice\_source();

TwitterMessageUtil.setSourceOnMessage(message, modifyDeviceSourceWithNofollow(deviceSource));

}

/\*\* Builds an IngesterTwitterMessage instance from a TweetDeleteEvent. \*/

@Nonnull

public static IngesterTwitterMessage getTwitterMessageFromDeletionEvent(

@Nonnull TweetDeleteEvent deleteEvent,

@Nonnull List<PenguinVersion> supportedPenguinVersions,

@Nullable DebugEvents debugEvents) throws ThriftTweetParsingException {

Tweet tweet = deleteEvent.getTweet();

if (tweet == null) {

throw new ThriftTweetParsingException("No tweet field in TweetDeleteEvent");

}

long tweetId = tweet.getId();

TweetCoreData coreData = tweet.getCore\_data();

if (coreData == null) {

throw new ThriftTweetParsingException("No TweetCoreData in TweetDeleteEvent");

}

long userId = coreData.getUser\_id();

IngesterTwitterMessage message = new IngesterTwitterMessage(

tweetId,

supportedPenguinVersions,

debugEvents);

message.setDeleted(true);

message.setText("delete");

message.setFromUser(TwitterMessageUser.createWithNamesAndId("delete", "delete", userId));

return message;

}

private static TwitterMessageUser getFromUser(User user) {

String screenName = user.getProfile().getScreen\_name();

long id = user.getId();

String displayName = user.getProfile().getName();

return TwitterMessageUser.createWithNamesAndId(screenName, displayName, id);

}

private static void addMentionsToMessage(IngesterTwitterMessage message, Tweet tweet) {

List<MentionEntity> mentions = tweet.getMentions();

if (mentions != null) {

NUM\_TWEETS\_WITH\_MENTIONS.increment();

NUM\_MENTIONS\_ADDED.add(mentions.size());

for (MentionEntity mention : mentions) {

addMention(message, mention);

}

}

}

private static void addMention(IngesterTwitterMessage message, MentionEntity mention) {

// Default values. They are weird, but are consistent with JSON parsing behavior.

Optional<Long> id = Optional.of(-1L);

Optional<String> screenName = Optional.of("");

Optional<String> displayName = Optional.of("");

if (mention.isSetUser\_id()) {

id = Optional.of(mention.getUser\_id());

}

if (mention.isSetScreen\_name()) {

screenName = Optional.of(mention.getScreen\_name());

}

if (mention.isSetName()) {

displayName = Optional.of(mention.getName());

}

TwitterMessageUser mentionedUser = TwitterMessageUser

.createWithOptionalNamesAndId(screenName, displayName, id);

if (isToUser(mention, message.getToUserObject())) {

message.setToUserObject(mentionedUser);

}

message.addUserToMentions(mentionedUser);

}

private static boolean isToUser(

MentionEntity mention, Optional<TwitterMessageUser> optionalToUser) {

if (mention.getFrom\_index() == 0) {

return true;

}

if (optionalToUser.isPresent()) {

TwitterMessageUser toUser = optionalToUser.get();

if (toUser.getId().isPresent()) {

long toUserId = toUser.getId().get();

return mention.getUser\_id() == toUserId;

}

}

return false;

}

private static void addHashtagsToMessage(IngesterTwitterMessage message, Tweet tweet) {

List<HashtagEntity> hashtags = tweet.getHashtags();

if (hashtags != null) {

NUM\_TWEETS\_WITH\_HASHTAGS.increment();

NUM\_HASHTAGS\_ADDED.add(hashtags.size());

for (HashtagEntity hashtag : hashtags) {

addHashtag(message, hashtag);

}

}

}

private static void addHashtag(IngesterTwitterMessage message, HashtagEntity hashtag) {

String hashtagString = hashtag.getText();

message.addHashtag(hashtagString);

}

/\*\* Add the given media entities to the given message. \*/

public static void addMediaEntitiesToMessage(

IngesterTwitterMessage message,

long photoStatusId,

@Nullable List<MediaEntity> medias) {

if (medias != null) {

NUM\_TWEETS\_WITH\_MEDIA\_URL.increment();

NUM\_MEDIA\_URLS\_ADDED.add(medias.size());

boolean hasPhotoMediaUrl = false;

boolean hasVideoMediaUrl = false;

for (MediaEntity media : medias) {

MediaTypes mediaType = null;

if (media.isSetMedia\_info()) {

MediaInfo mediaInfo = media.getMedia\_info();

if (mediaInfo != null) {

if (mediaInfo.isSet(MediaInfo.\_Fields.IMAGE\_INFO)) {

mediaType = MediaTypes.NATIVE\_IMAGE;

String mediaUrl = media.getMedia\_url\_https();

if (mediaUrl != null) {

hasPhotoMediaUrl = true;

message.addPhotoUrl(photoStatusId, mediaUrl);

// Add this link to the expanded URLs too, so that the HAS\_NATIVE\_IMAGE\_FLAG is set

// correctly too. See EncodedFeatureBuilder.updateLinkEncodedFeatures().

}

} else if (mediaInfo.isSet(MediaInfo.\_Fields.VIDEO\_INFO)) {

mediaType = MediaTypes.VIDEO;

hasVideoMediaUrl = true;

}

}

}

String originalUrl = media.getUrl();

String expandedUrl = media.getExpanded\_url();

message.addExpandedMediaUrl(originalUrl, expandedUrl, mediaType);

}

if (hasPhotoMediaUrl) {

NUM\_TWEETS\_WITH\_PHOTO\_MEDIA\_URL.increment();

}

if (hasVideoMediaUrl) {

NUM\_TWEETS\_WITH\_VIDEO\_MEDIA\_URL.increment();

}

}

}

/\*\* Adds the given urls to the given message. \*/

public static void addUrlsToMessage(

IngesterTwitterMessage message,

@Nullable List<UrlEntity> urls) {

if (urls != null) {

NUM\_TWEETS\_WITH\_NON\_MEDIA\_URL.increment();

NUM\_NON\_MEDIA\_URLS\_ADDED.add(urls.size());

for (UrlEntity url : urls) {

String originalUrl = url.getUrl();

String expandedUrl = url.getExpanded();

message.addExpandedNonMediaUrl(originalUrl, expandedUrl);

}

}

}

private static void addEscherbirdAnnotationsToMessage(

IngesterTwitterMessage message, Tweet tweet) {

if (tweet.isSetEscherbird\_entity\_annotations()) {

EscherbirdEntityAnnotations entityAnnotations = tweet.getEscherbird\_entity\_annotations();

if (entityAnnotations.isSetEntity\_annotations()) {

NUM\_TWEETS\_WITH\_ANNOTATIONS.increment();

NUM\_ANNOTATIONS\_ADDED.add(entityAnnotations.getEntity\_annotationsSize());

for (TweetEntityAnnotation entityAnnotation : entityAnnotations.getEntity\_annotations()) {

EscherbirdAnnotation escherbirdAnnotation =

new EscherbirdAnnotation(entityAnnotation.getGroupId(),

entityAnnotation.getDomainId(),

entityAnnotation.getEntityId());

message.addEscherbirdAnnotation(escherbirdAnnotation);

}

}

}

}

private static void setComposerSource(IngesterTwitterMessage message, Tweet tweet) {

if (tweet.isSetComposer\_source()) {

message.setComposerSource(tweet.getComposer\_source());

}

}

private static String modifyDeviceSourceWithNofollow(@Nullable DeviceSource deviceSource) {

if (deviceSource != null) {

String source = deviceSource.getDisplay();

int i = source.indexOf("\">");

if (i == -1) {

return source;

} else {

return source.substring(0, i) + "\" rel=\"nofollow\">" + source.substring(i + 2);

}

} else {

return "<a href=\"http://twitter.com\" rel=\"nofollow\">Twitter</a>";

}

}

private static void setDirectedAtUser(

IngesterTwitterMessage message,

TweetCoreData tweetCoreData) {

if (!tweetCoreData.isSetDirected\_at\_user()) {

return;

}

DirectedAtUser directedAtUser = tweetCoreData.getDirected\_at\_user();

if (!directedAtUser.isSetUser\_id()) {

return;

}

message.setDirectedAtUserId(Optional.of(directedAtUser.getUser\_id()));

}

}