package com.twitter.search.common.relevance.entities;

import java.text.DateFormat;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collection;

import java.util.Collections;

import java.util.Date;

import java.util.HashSet;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Optional;

import java.util.Set;

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.ComparisonChain;

import com.google.common.collect.Lists;

import com.google.common.collect.Maps;

import com.google.common.collect.Sets;

import org.apache.commons.lang.StringUtils;

import org.apache.commons.lang3.builder.EqualsBuilder;

import org.apache.commons.lang3.builder.HashCodeBuilder;

import org.apache.commons.lang3.builder.ToStringBuilder;

import org.apache.lucene.analysis.TokenStream;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.text.language.LocaleUtil;

import com.twitter.common.text.pipeline.TwitterLanguageIdentifier;

import com.twitter.common.text.token.TokenizedCharSequence;

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

import com.twitter.cuad.ner.plain.thriftjava.NamedEntity;

import com.twitter.search.common.indexing.thriftjava.ThriftExpandedUrl;

import com.twitter.search.common.relevance.features.TweetFeatures;

import com.twitter.search.common.relevance.features.TweetTextFeatures;

import com.twitter.search.common.relevance.features.TweetTextQuality;

import com.twitter.search.common.relevance.features.TweetUserFeatures;

import com.twitter.search.common.util.text.NormalizerHelper;

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

import com.twitter.tweetypie.thriftjava.ComposerSource;

import com.twitter.util.TwitterDateFormat;

/\*\*

\* A representation of tweets used as an intermediate object during ingestion. As we proceed

\* in ingestion, we fill this object with data. We then convert it to ThriftVersionedEvents (which

\* itself represents a single tweet too, in different penguin versions potentially).

\*/

public class TwitterMessage {

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

public static class EscherbirdAnnotation implements Comparable<EscherbirdAnnotation> {

public final long groupId;

public final long domainId;

public final long entityId;

public EscherbirdAnnotation(long groupId, long domainId, long entityId) {

this.groupId = groupId;

this.domainId = domainId;

this.entityId = entityId;

}

@Override

public boolean equals(Object o2) {

if (o2 instanceof EscherbirdAnnotation) {

EscherbirdAnnotation a2 = (EscherbirdAnnotation) o2;

return groupId == a2.groupId && domainId == a2.domainId && entityId == a2.entityId;

}

return false;

}

@Override

public int hashCode() {

return new HashCodeBuilder()

.append(groupId)

.append(domainId)

.append(entityId)

.toHashCode();

}

@Override

public int compareTo(EscherbirdAnnotation o) {

return ComparisonChain.start()

.compare(this.groupId, o.groupId)

.compare(this.domainId, o.domainId)

.compare(this.entityId, o.entityId)

.result();

}

}

private final List<EscherbirdAnnotation> escherbirdAnnotations = Lists.newArrayList();

// tweet features for multiple penguin versions

private static class VersionedTweetFeatures {

// TweetFeatures populated by relevance classifiers, structure defined

// in src/main/thrift/classifier.thrift.

private TweetFeatures tweetFeatures = new TweetFeatures();

private TokenizedCharSequence tokenizedCharSequence = null;

private Set<String> normalizedHashtags = Sets.newHashSet();

public TweetFeatures getTweetFeatures() {

return this.tweetFeatures;

}

public void setTweetFeatures(final TweetFeatures tweetFeatures) {

this.tweetFeatures = tweetFeatures;

}

public TweetTextQuality getTweetTextQuality() {

return this.tweetFeatures.getTweetTextQuality();

}

public TweetTextFeatures getTweetTextFeatures() {

return this.tweetFeatures.getTweetTextFeatures();

}

public TweetUserFeatures getTweetUserFeatures() {

return this.tweetFeatures.getTweetUserFeatures();

}

public TokenizedCharSequence getTokenizedCharSequence() {

return this.tokenizedCharSequence;

}

public void setTokenizedCharSequence(TokenizedCharSequence sequence) {

this.tokenizedCharSequence = sequence;

}

public Set<String> getNormalizedHashtags() {

return this.normalizedHashtags;

}

public void addNormalizedHashtags(String normalizedHashtag) {

this.normalizedHashtags.add(normalizedHashtag);

}

}

public static final int INT\_FIELD\_NOT\_PRESENT = -1;

public static final long LONG\_FIELD\_NOT\_PRESENT = -1;

public static final double DOUBLE\_FIELD\_NOT\_PRESENT = -1;

public static final int MAX\_USER\_REPUTATION = 100;

private final long tweetId;

private String text;

private Date date;

@Nonnull

private Optional<TwitterMessageUser> optionalFromUser = Optional.empty();

@Nonnull

private Optional<TwitterMessageUser> optionalToUser = Optional.empty();

private Locale locale = null;

private Locale linkLocale = null;

// Original source text.

private String origSource;

// Source with HTML tags removed and truncated.

private String strippedSource;

// Original location text.

private String origLocation;

// Location truncated for mysql field-width reasons (see TwitterMessageUtil.java).

private String truncatedNormalizedLocation;

// User's country

private String fromUserLocCountry;

private Integer followersCount = INT\_FIELD\_NOT\_PRESENT;

private boolean deleted = false;

// Fields extracted from entities (in the JSON object)

private List<TwitterMessageUser> mentions = new ArrayList<>();

private Set<String> hashtags = Sets.newHashSet();

// Lat/lon and region accuracy tuples extracted from tweet text, or null.

private GeoObject geoLocation = null;

private boolean uncodeableLocation = false;

// This is set if the tweet is geotagged. (i.e. "geo" or "coordinate" section is present

// in the json)

// This field has only a getter but no setter --- it is filled in when the json is parsed.

private GeoObject geoTaggedLocation = null;

private double userReputation = DOUBLE\_FIELD\_NOT\_PRESENT;

private boolean geocodeRequired = false;

private boolean sensitiveContent = false;

private boolean userProtected;

private boolean userVerified;

private boolean userBlueVerified;

private TwitterRetweetMessage retweetMessage;

private TwitterQuotedMessage quotedMessage;

private List<String> places;

// maps from original url (the t.co url) to ThriftExpandedUrl, which contains the

// expanded url and the spiderduck response (canoicalLastHopUrl and mediatype)

private final Map<String, ThriftExpandedUrl> expandedUrls;

// maps the photo status id to the media url

private Map<Long, String> photoUrls;

private Optional<Long> inReplyToStatusId = Optional.empty();

private Optional<Long> directedAtUserId = Optional.empty();

private long conversationId = -1;

// True if tweet is nullcasted.

private boolean nullcast = false;

// True if tweet is a self-threaded tweet

private boolean selfThread = false;

// If the tweet is a part of an exclusive conversation, the author who started

// that conversation.

private Optional<Long> exclusiveConversationAuthorId = Optional.empty();

// tweet features map for multiple versions of penguin

private Map<PenguinVersion, VersionedTweetFeatures> versionedTweetFeaturesMap;

// Engagments count: favorites, retweets and replies

private int numFavorites = 0;

private int numRetweets = 0;

private int numReplies = 0;

// Card information

private String cardName;

private String cardDomain;

private String cardTitle;

private String cardDescription;

private String cardLang;

private String cardUrl;

private String placeId;

private String placeFullName;

private String placeCountryCode;

private Set<NamedEntity> namedEntities = Sets.newHashSet();

// Spaces data

private Set<String> spaceIds = Sets.newHashSet();

private Set<TwitterMessageUser> spaceAdmins = Sets.newHashSet();

private String spaceTitle;

private Optional<ComposerSource> composerSource = Optional.empty();

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

// one or two penguin versions supported by this system

private final List<PenguinVersion> supportedPenguinVersions;

public TwitterMessage(Long tweetId, List<PenguinVersion> supportedPenguinVersions) {

this.tweetId = tweetId;

this.places = new ArrayList<>();

this.expandedUrls = new LinkedHashMap<>();

// make sure we support at least one, but no more than two versions of penguin

this.supportedPenguinVersions = supportedPenguinVersions;

this.versionedTweetFeaturesMap = getVersionedTweetFeaturesMap();

Preconditions.checkArgument(this.supportedPenguinVersions.size() <= 2

&& this.supportedPenguinVersions.size() > 0);

}

/\*\*

\* Replace to-user with in-reply-to user if needed.

\*/

public void replaceToUserWithInReplyToUserIfNeeded(

String inReplyToScreenName, long inReplyToUserId) {

if (shouldUseReplyUserAsToUser(optionalToUser, inReplyToUserId)) {

TwitterMessageUser replyUser =

TwitterMessageUser.createWithNamesAndId(inReplyToScreenName, "", inReplyToUserId);

optionalToUser = Optional.of(replyUser);

}

}

// To-user could have been inferred from the mention at the position 0.

// But if there is an explicit in-reply-to user, we might need to use it as to-user instead.

private static boolean shouldUseReplyUserAsToUser(

Optional<TwitterMessageUser> currentToUser,

long inReplyToUserId) {

if (!currentToUser.isPresent()) {

// There is no mention in the tweet that qualifies as to-user.

return true;

}

// We already have a mention in the tweet that qualifies as to-user.

TwitterMessageUser toUser = currentToUser.get();

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

// The to-user from the mention is a stub.

return true;

}

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

if (toUserId != inReplyToUserId) {

// The to-user from the mention is different that the in-reply-to user,

// use in-reply-to user instead.

return true;

}

return false;

}

public double getUserReputation() {

return userReputation;

}

/\*\*

\* Sets the user reputation.

\*/

public TwitterMessage setUserReputation(double newUserReputation) {

if (newUserReputation > MAX\_USER\_REPUTATION) {

LOG.warn("Out of bounds user reputation {} for status id {}", newUserReputation, tweetId);

this.userReputation = (float) MAX\_USER\_REPUTATION;

} else {

this.userReputation = newUserReputation;

}

return this;

}

public String getText() {

return text;

}

public Optional<TwitterMessageUser> getOptionalToUser() {

return optionalToUser;

}

public void setOptionalToUser(Optional<TwitterMessageUser> optionalToUser) {

this.optionalToUser = optionalToUser;

}

public void setText(String text) {

this.text = text;

}

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

public void setFromUser(@Nonnull TwitterMessageUser fromUser) {

Preconditions.checkNotNull(fromUser, "Don't set a null fromUser");

optionalFromUser = Optional.of(fromUser);

}

public Optional<String> getFromUserScreenName() {

return optionalFromUser.isPresent()

? optionalFromUser.get().getScreenName()

: Optional.empty();

}

/\*\*

\* Sets the fromUserScreenName.

\*/

public void setFromUserScreenName(@Nonnull String fromUserScreenName) {

TwitterMessageUser newFromUser = optionalFromUser.isPresent()

? optionalFromUser.get().copyWithScreenName(fromUserScreenName)

: TwitterMessageUser.createWithScreenName(fromUserScreenName);

optionalFromUser = Optional.of(newFromUser);

}

public Optional<TokenStream> getTokenizedFromUserScreenName() {

return optionalFromUser.flatMap(TwitterMessageUser::getTokenizedScreenName);

}

public Optional<String> getFromUserDisplayName() {

return optionalFromUser.flatMap(TwitterMessageUser::getDisplayName);

}

/\*\*

\* Sets the fromUserDisplayName.

\*/

public void setFromUserDisplayName(@Nonnull String fromUserDisplayName) {

TwitterMessageUser newFromUser = optionalFromUser.isPresent()

? optionalFromUser.get().copyWithDisplayName(fromUserDisplayName)

: TwitterMessageUser.createWithDisplayName(fromUserDisplayName);

optionalFromUser = Optional.of(newFromUser);

}

public Optional<Long> getFromUserTwitterId() {

return optionalFromUser.flatMap(TwitterMessageUser::getId);

}

/\*\*

\* Sets the fromUserId.

\*/

public void setFromUserId(long fromUserId) {

TwitterMessageUser newFromUser = optionalFromUser.isPresent()

? optionalFromUser.get().copyWithId(fromUserId)

: TwitterMessageUser.createWithId(fromUserId);

optionalFromUser = Optional.of(newFromUser);

}

public long getConversationId() {

return conversationId;

}

public void setConversationId(long conversationId) {

this.conversationId = conversationId;

}

public boolean isUserProtected() {

return this.userProtected;

}

public void setUserProtected(boolean userProtected) {

this.userProtected = userProtected;

}

public boolean isUserVerified() {

return this.userVerified;

}

public void setUserVerified(boolean userVerified) {

this.userVerified = userVerified;

}

public boolean isUserBlueVerified() {

return this.userBlueVerified;

}

public void setUserBlueVerified(boolean userBlueVerified) {

this.userBlueVerified = userBlueVerified;

}

public void setIsSensitiveContent(boolean isSensitiveContent) {

this.sensitiveContent = isSensitiveContent;

}

public boolean isSensitiveContent() {

return this.sensitiveContent;

}

public Optional<TwitterMessageUser> getToUserObject() {

return optionalToUser;

}

public void setToUserObject(@Nonnull TwitterMessageUser user) {

Preconditions.checkNotNull(user, "Don't set a null to-user");

optionalToUser = Optional.of(user);

}

public Optional<Long> getToUserTwitterId() {

return optionalToUser.flatMap(TwitterMessageUser::getId);

}

/\*\*

\* Sets toUserId.

\*/

public void setToUserTwitterId(long toUserId) {

TwitterMessageUser newToUser = optionalToUser.isPresent()

? optionalToUser.get().copyWithId(toUserId)

: TwitterMessageUser.createWithId(toUserId);

optionalToUser = Optional.of(newToUser);

}

public Optional<String> getToUserLowercasedScreenName() {

return optionalToUser.flatMap(TwitterMessageUser::getScreenName).map(String::toLowerCase);

}

public Optional<String> getToUserScreenName() {

return optionalToUser.flatMap(TwitterMessageUser::getScreenName);

}

/\*\*

\* Sets toUserScreenName.

\*/

public void setToUserScreenName(@Nonnull String screenName) {

Preconditions.checkNotNull(screenName, "Don't set a null to-user screenname");

TwitterMessageUser newToUser = optionalToUser.isPresent()

? optionalToUser.get().copyWithScreenName(screenName)

: TwitterMessageUser.createWithScreenName(screenName);

optionalToUser = Optional.of(newToUser);

}

// to use from TweetEventParseHelper

public void setDirectedAtUserId(Optional<Long> directedAtUserId) {

this.directedAtUserId = directedAtUserId;

}

@VisibleForTesting

public Optional<Long> getDirectedAtUserId() {

return directedAtUserId;

}

/\*\*

\* Returns the referenceAuthorId.

\*/

public Optional<Long> getReferenceAuthorId() {

// The semantics of reference-author-id:

// - if the tweet is a retweet, it should be the user id of the author of the original tweet

// - else, if the tweet is directed at a user, it should be the id of the user it's directed at.

// - else, if the tweet is a reply in a root self-thread, directed-at is not set, so it's

// the id of the user who started the self-thread.

//

// For definitive info on replies and directed-at, take a look at go/replies. To view these

// for a certain tweet, use http://go/t.

//

// Note that if directed-at is set, reply is always set.

// If reply is set, directed-at is not necessarily set.

if (isRetweet() && retweetMessage.hasSharedUserTwitterId()) {

long retweetedUserId = retweetMessage.getSharedUserTwitterId();

return Optional.of(retweetedUserId);

} else if (directedAtUserId.isPresent()) {

// Why not replace directedAtUserId with reply and make this function depend

// on the "reply" field of TweetCoreData?

// Well, verified by counters, it seems for ~1% of tweets, which contain both directed-at

// and reply, directed-at-user is different than the reply-to-user id. This happens in the

// following case:

//

// author / reply-to / directed-at

// T1 A - -

// T2 B A A

// T3 B B A

//

// T2 is a reply to T1, T3 is a reply to T2.

//

// It's up to us to decide who this tweet is "referencing", but with the current code,

// we choose that T3 is referencing user A.

return directedAtUserId;

} else {

// This is the case of a root self-thread reply. directed-at is not set.

Optional<Long> fromUserId = this.getFromUserTwitterId();

Optional<Long> toUserId = this.getToUserTwitterId();

if (fromUserId.isPresent() && fromUserId.equals(toUserId)) {

return fromUserId;

}

}

return Optional.empty();

}

public void setNumFavorites(int numFavorites) {

this.numFavorites = numFavorites;

}

public void setNumRetweets(int numRetweets) {

this.numRetweets = numRetweets;

}

public void setNumReplies(int numRepliess) {

this.numReplies = numRepliess;

}

public void addEscherbirdAnnotation(EscherbirdAnnotation annotation) {

escherbirdAnnotations.add(annotation);

}

public List<EscherbirdAnnotation> getEscherbirdAnnotations() {

return escherbirdAnnotations;

}

public List<PotentialLocationObject> getPotentialLocations() {

return potentialLocations;

}

public void setPotentialLocations(Collection<PotentialLocationObject> potentialLocations) {

this.potentialLocations.clear();

this.potentialLocations.addAll(potentialLocations);

}

@Override

public String toString() {

return ToStringBuilder.reflectionToString(this);

}

// Tweet language related getters and setters.

/\*\*

\* Returns the locale.

\* <p>

\* Note the getLocale() will never return null, this is for the convenience of text related

\* processing in the ingester. If you want the real locale, you need to check isSetLocale()

\* first to see if we really have any information about the locale of this tweet.

\*/

public Locale getLocale() {

if (locale == null) {

return TwitterLanguageIdentifier.UNKNOWN;

} else {

return locale;

}

}

public void setLocale(Locale locale) {

this.locale = locale;

}

/\*\*

\* Determines if the locate is set.

\*/

public boolean isSetLocale() {

return locale != null;

}

/\*\*

\* Returns the language of the locale. E.g. zh

\*/

public String getLanguage() {

if (isSetLocale()) {

return getLocale().getLanguage();

} else {

return null;

}

}

/\*\*

\* Returns the IETF BCP 47 Language Tag of the locale. E.g. zh-CN

\*/

public String getBCP47LanguageTag() {

if (isSetLocale()) {

return getLocale().toLanguageTag();

} else {

return null;

}

}

public void setLanguage(String language) {

if (language != null) {

locale = LocaleUtil.getLocaleOf(language);

}

}

// Tweet link language related getters and setters.

public Locale getLinkLocale() {

return linkLocale;

}

public void setLinkLocale(Locale linkLocale) {

this.linkLocale = linkLocale;

}

/\*\*

\* Returns the language of the link locale.

\*/

public String getLinkLanguage() {

if (this.linkLocale == null) {

return null;

} else {

return this.linkLocale.getLanguage();

}

}

public String getOrigSource() {

return origSource;

}

public void setOrigSource(String origSource) {

this.origSource = origSource;

}

public String getStrippedSource() {

return strippedSource;

}

public void setStrippedSource(String strippedSource) {

this.strippedSource = strippedSource;

}

public String getOrigLocation() {

return origLocation;

}

public String getLocation() {

return truncatedNormalizedLocation;

}

public void setOrigLocation(String origLocation) {

this.origLocation = origLocation;

}

public void setTruncatedNormalizedLocation(String truncatedNormalizedLocation) {

this.truncatedNormalizedLocation = truncatedNormalizedLocation;

}

public boolean hasFromUserLocCountry() {

return fromUserLocCountry != null;

}

public String getFromUserLocCountry() {

return fromUserLocCountry;

}

public void setFromUserLocCountry(String fromUserLocCountry) {

this.fromUserLocCountry = fromUserLocCountry;

}

public String getTruncatedNormalizedLocation() {

return truncatedNormalizedLocation;

}

public Integer getFollowersCount() {

return followersCount;

}

public void setFollowersCount(Integer followersCount) {

this.followersCount = followersCount;

}

public boolean hasFollowersCount() {

return followersCount != INT\_FIELD\_NOT\_PRESENT;

}

public boolean isDeleted() {

return deleted;

}

public void setDeleted(boolean deleted) {

this.deleted = deleted;

}

public boolean hasCard() {

return !StringUtils.isBlank(getCardName());

}

@Override

public int hashCode() {

return ((Long) getId()).hashCode();

}

/\*\*

\* Parses the given date using the TwitterDateFormat.

\*/

public static Date parseDate(String date) {

DateFormat parser = TwitterDateFormat.apply("EEE MMM d HH:mm:ss Z yyyy");

try {

return parser.parse(date);

} catch (Exception e) {

return null;

}

}

public boolean hasGeoLocation() {

return geoLocation != null;

}

public void setGeoLocation(GeoObject location) {

this.geoLocation = location;

}

public GeoObject getGeoLocation() {

return geoLocation;

}

public String getPlaceId() {

return placeId;

}

public void setPlaceId(String placeId) {

this.placeId = placeId;

}

public String getPlaceFullName() {

return placeFullName;

}

public void setPlaceFullName(String placeFullName) {

this.placeFullName = placeFullName;

}

public String getPlaceCountryCode() {

return placeCountryCode;

}

public void setPlaceCountryCode(String placeCountryCode) {

this.placeCountryCode = placeCountryCode;

}

public void setGeoTaggedLocation(GeoObject geoTaggedLocation) {

this.geoTaggedLocation = geoTaggedLocation;

}

public GeoObject getGeoTaggedLocation() {

return geoTaggedLocation;

}

public void setLatLon(double latitude, double longitude) {

geoLocation = new GeoObject(latitude, longitude, null);

}

public Double getLatitude() {

return hasGeoLocation() ? geoLocation.getLatitude() : null;

}

public Double getLongitude() {

return hasGeoLocation() ? geoLocation.getLongitude() : null;

}

public boolean isUncodeableLocation() {

return uncodeableLocation;

}

public void setUncodeableLocation() {

uncodeableLocation = true;

}

public void setGeocodeRequired() {

this.geocodeRequired = true;

}

public boolean isGeocodeRequired() {

return geocodeRequired;

}

public Map<Long, String> getPhotoUrls() {

return photoUrls;

}

/\*\*

\* Associates the given mediaUrl with the given photoStatusId.

\*/

public void addPhotoUrl(long photoStatusId, String mediaUrl) {

if (photoUrls == null) {

photoUrls = new LinkedHashMap<>();

}

photoUrls.putIfAbsent(photoStatusId, mediaUrl);

}

public Map<String, ThriftExpandedUrl> getExpandedUrlMap() {

return expandedUrls;

}

public int getExpandedUrlMapSize() {

return expandedUrls.size();

}

/\*\*

\* Associates the given originalUrl with the given expanderUrl.

\*/

public void addExpandedUrl(String originalUrl, ThriftExpandedUrl expandedUrl) {

this.expandedUrls.put(originalUrl, expandedUrl);

}

/\*\*

\* Replaces urls with resolved ones.

\*/

public String getTextReplacedWithResolvedURLs() {

String retText = text;

for (Map.Entry<String, ThriftExpandedUrl> entry : expandedUrls.entrySet()) {

ThriftExpandedUrl urlInfo = entry.getValue();

String resolvedUrl;

String canonicalLastHopUrl = urlInfo.getCanonicalLastHopUrl();

String expandedUrl = urlInfo.getExpandedUrl();

if (canonicalLastHopUrl != null) {

resolvedUrl = canonicalLastHopUrl;

LOG.debug("{} has canonical last hop url set", urlInfo);

} else if (expandedUrl != null) {

LOG.debug("{} has no canonical last hop url set, using expanded url instead", urlInfo);

resolvedUrl = expandedUrl;

} else {

LOG.debug("{} has no canonical last hop url or expanded url set, skipping", urlInfo);

continue;

}

retText = retText.replace(entry.getKey(), resolvedUrl);

}

return retText;

}

public long getId() {

return tweetId;

}

public boolean isRetweet() {

return retweetMessage != null;

}

public boolean hasQuote() {

return quotedMessage != null;

}

public boolean isReply() {

return getToUserScreenName().isPresent()

|| getToUserTwitterId().isPresent()

|| getInReplyToStatusId().isPresent();

}

public boolean isReplyToTweet() {

return getInReplyToStatusId().isPresent();

}

public TwitterRetweetMessage getRetweetMessage() {

return retweetMessage;

}

public void setRetweetMessage(TwitterRetweetMessage retweetMessage) {

this.retweetMessage = retweetMessage;

}

public TwitterQuotedMessage getQuotedMessage() {

return quotedMessage;

}

public void setQuotedMessage(TwitterQuotedMessage quotedMessage) {

this.quotedMessage = quotedMessage;

}

public List<String> getPlaces() {

return places;

}

public void addPlace(String place) {

// Places are used for earlybird serialization

places.add(place);

}

public Optional<Long> getInReplyToStatusId() {

return inReplyToStatusId;

}

public void setInReplyToStatusId(long inReplyToStatusId) {

Preconditions.checkArgument(inReplyToStatusId > 0, "In-reply-to status ID should be positive");

this.inReplyToStatusId = Optional.of(inReplyToStatusId);

}

public boolean getNullcast() {

return nullcast;

}

public void setNullcast(boolean nullcast) {

this.nullcast = nullcast;

}

public List<PenguinVersion> getSupportedPenguinVersions() {

return supportedPenguinVersions;

}

private VersionedTweetFeatures getVersionedTweetFeatures(PenguinVersion penguinVersion) {

VersionedTweetFeatures versionedTweetFeatures = versionedTweetFeaturesMap.get(penguinVersion);

return Preconditions.checkNotNull(versionedTweetFeatures);

}

public TweetFeatures getTweetFeatures(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTweetFeatures();

}

@VisibleForTesting

// only used in Tests

public void setTweetFeatures(PenguinVersion penguinVersion, TweetFeatures tweetFeatures) {

versionedTweetFeaturesMap.get(penguinVersion).setTweetFeatures(tweetFeatures);

}

public int getTweetSignature(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTweetTextFeatures().getSignature();

}

public TweetTextQuality getTweetTextQuality(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTweetTextQuality();

}

public TweetTextFeatures getTweetTextFeatures(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTweetTextFeatures();

}

public TweetUserFeatures getTweetUserFeatures(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTweetUserFeatures();

}

public TokenizedCharSequence getTokenizedCharSequence(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getTokenizedCharSequence();

}

public void setTokenizedCharSequence(PenguinVersion penguinVersion,

TokenizedCharSequence sequence) {

getVersionedTweetFeatures(penguinVersion).setTokenizedCharSequence(sequence);

}

// True if the features contain multiple hash tags or multiple trends.

// This is intended as an anti-trend-spam measure.

public static boolean hasMultipleHashtagsOrTrends(TweetTextFeatures textFeatures) {

// Allow at most 1 trend and 2 hashtags.

return textFeatures.getTrendingTermsSize() > 1 || textFeatures.getHashtagsSize() > 2;

}

/\*\*

\* Returns the expanded URLs.

\*/

public Collection<ThriftExpandedUrl> getExpandedUrls() {

return expandedUrls.values();

}

/\*\*

\* Returns the canonical last hop URLs.

\*/

public Set<String> getCanonicalLastHopUrls() {

Set<String> result = new HashSet<>(expandedUrls.size());

for (ThriftExpandedUrl url : expandedUrls.values()) {

result.add(url.getCanonicalLastHopUrl());

}

return result;

}

public String getCardName() {

return cardName;

}

public void setCardName(String cardName) {

this.cardName = cardName;

}

public String getCardDomain() {

return cardDomain;

}

public void setCardDomain(String cardDomain) {

this.cardDomain = cardDomain;

}

public String getCardTitle() {

return cardTitle;

}

public void setCardTitle(String cardTitle) {

this.cardTitle = cardTitle;

}

public String getCardDescription() {

return cardDescription;

}

public void setCardDescription(String cardDescription) {

this.cardDescription = cardDescription;

}

public String getCardLang() {

return cardLang;

}

public void setCardLang(String cardLang) {

this.cardLang = cardLang;

}

public String getCardUrl() {

return cardUrl;

}

public void setCardUrl(String cardUrl) {

this.cardUrl = cardUrl;

}

public List<TwitterMessageUser> getMentions() {

return this.mentions;

}

public void setMentions(List<TwitterMessageUser> mentions) {

this.mentions = mentions;

}

public List<String> getLowercasedMentions() {

return Lists.transform(getMentions(), user -> {

// This condition is also checked in addUserToMentions().

Preconditions.checkState(user.getScreenName().isPresent(), "Invalid mention");

return user.getScreenName().get().toLowerCase();

});

}

public Set<String> getHashtags() {

return this.hashtags;

}

public Set<String> getNormalizedHashtags(PenguinVersion penguinVersion) {

return getVersionedTweetFeatures(penguinVersion).getNormalizedHashtags();

}

public void addNormalizedHashtag(String normalizedHashtag, PenguinVersion penguinVersion) {

getVersionedTweetFeatures(penguinVersion).addNormalizedHashtags(normalizedHashtag);

}

public Optional<ComposerSource> getComposerSource() {

return composerSource;

}

public void setComposerSource(ComposerSource composerSource) {

Preconditions.checkNotNull(composerSource, "composerSource should not be null");

this.composerSource = Optional.of(composerSource);

}

public boolean isSelfThread() {

return selfThread;

}

public void setSelfThread(boolean selfThread) {

this.selfThread = selfThread;

}

public boolean isExclusive() {

return exclusiveConversationAuthorId.isPresent();

}

public long getExclusiveConversationAuthorId() {

return exclusiveConversationAuthorId.get();

}

public void setExclusiveConversationAuthorId(long exclusiveConversationAuthorId) {

this.exclusiveConversationAuthorId = Optional.of(exclusiveConversationAuthorId);

}

/\*\*

\* Adds an expanded media url based on the given parameters.

\*/

public void addExpandedMediaUrl(String originalUrl,

String expandedUrl,

@Nullable MediaTypes mediaType) {

if (!StringUtils.isBlank(originalUrl) && !StringUtils.isBlank(expandedUrl)) {

ThriftExpandedUrl thriftExpandedUrl = new ThriftExpandedUrl();

if (mediaType != null) {

thriftExpandedUrl.setMediaType(mediaType);

}

thriftExpandedUrl.setOriginalUrl(originalUrl);

thriftExpandedUrl.setExpandedUrl(expandedUrl); // This will be tokenized and indexed

// Note that the mediaURL is not indexed. We could also index it, but it is not indexed

// to reduce RAM usage.

thriftExpandedUrl.setCanonicalLastHopUrl(expandedUrl); // This will be tokenized and indexed

addExpandedUrl(originalUrl, thriftExpandedUrl);

thriftExpandedUrl.setConsumerMedia(true);

}

}

/\*\*

\* Adds an expanded non-media url based on the given parameters.

\*/

public void addExpandedNonMediaUrl(String originalUrl, String expandedUrl) {

if (!StringUtils.isBlank(originalUrl)) {

ThriftExpandedUrl thriftExpandedUrl = new ThriftExpandedUrl(originalUrl);

if (!StringUtils.isBlank(expandedUrl)) {

thriftExpandedUrl.setExpandedUrl(expandedUrl);

}

addExpandedUrl(originalUrl, thriftExpandedUrl);

thriftExpandedUrl.setConsumerMedia(false);

}

}

/\*\*

\* Only used in tests.

\*

\* Simulates resolving compressed URLs, which is usually done by ResolveCompressedUrlsStage.

\*/

@VisibleForTesting

public void replaceUrlsWithResolvedUrls(Map<String, String> resolvedUrls) {

for (Map.Entry<String, ThriftExpandedUrl> urlEntry : expandedUrls.entrySet()) {

String tcoUrl = urlEntry.getKey();

if (resolvedUrls.containsKey(tcoUrl)) {

ThriftExpandedUrl expandedUrl = urlEntry.getValue();

expandedUrl.setCanonicalLastHopUrl(resolvedUrls.get(tcoUrl));

}

}

}

/\*\*

\* Adds a mention for a user with the given screen name.

\*/

public void addMention(String screenName) {

TwitterMessageUser user = TwitterMessageUser.createWithScreenName(screenName);

addUserToMentions(user);

}

/\*\*

\* Adds the given user to mentions.

\*/

public void addUserToMentions(TwitterMessageUser user) {

Preconditions.checkArgument(user.getScreenName().isPresent(), "Don't add invalid mentions");

this.mentions.add(user);

}

/\*\*

\* Adds the given hashtag.

\*/

public void addHashtag(String hashtag) {

this.hashtags.add(hashtag);

for (PenguinVersion penguinVersion : supportedPenguinVersions) {

addNormalizedHashtag(NormalizerHelper.normalize(hashtag, getLocale(), penguinVersion),

penguinVersion);

}

}

private Map<PenguinVersion, VersionedTweetFeatures> getVersionedTweetFeaturesMap() {

Map<PenguinVersion, VersionedTweetFeatures> versionedMap =

Maps.newEnumMap(PenguinVersion.class);

for (PenguinVersion penguinVersion : getSupportedPenguinVersions()) {

versionedMap.put(penguinVersion, new VersionedTweetFeatures());

}

return versionedMap;

}

public int getNumFavorites() {

return numFavorites;

}

public int getNumRetweets() {

return numRetweets;

}

public int getNumReplies() {

return numReplies;

}

public Set<NamedEntity> getNamedEntities() {

return namedEntities;

}

public void addNamedEntity(NamedEntity namedEntity) {

namedEntities.add(namedEntity);

}

public Set<String> getSpaceIds() {

return spaceIds;

}

public void setSpaceIds(Set<String> spaceIds) {

this.spaceIds = Sets.newHashSet(spaceIds);

}

public Set<TwitterMessageUser> getSpaceAdmins() {

return spaceAdmins;

}

public void addSpaceAdmin(TwitterMessageUser admin) {

spaceAdmins.add(admin);

}

public String getSpaceTitle() {

return spaceTitle;

}

public void setSpaceTitle(String spaceTitle) {

this.spaceTitle = spaceTitle;

}

private static boolean equals(List<EscherbirdAnnotation> l1, List<EscherbirdAnnotation> l2) {

EscherbirdAnnotation[] arr1 = l1.toArray(new EscherbirdAnnotation[l1.size()]);

Arrays.sort(arr1);

EscherbirdAnnotation[] arr2 = l1.toArray(new EscherbirdAnnotation[l2.size()]);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

/\*\*

\* Compares the given messages using reflection and determines if they're approximately equal.

\*/

public static boolean reflectionApproxEquals(

TwitterMessage a,

TwitterMessage b,

List<String> additionalExcludeFields) {

List<String> excludeFields = Lists.newArrayList(

"versionedTweetFeaturesMap",

"geoLocation",

"geoTaggedLocation",

"escherbirdAnnotations"

);

excludeFields.addAll(additionalExcludeFields);

return EqualsBuilder.reflectionEquals(a, b, excludeFields)

&& GeoObject.approxEquals(a.getGeoLocation(), b.getGeoLocation())

&& GeoObject.approxEquals(a.getGeoTaggedLocation(), b.getGeoTaggedLocation())

&& equals(a.getEscherbirdAnnotations(), b.getEscherbirdAnnotations());

}

public static boolean reflectionApproxEquals(TwitterMessage a, TwitterMessage b) {

return reflectionApproxEquals(a, b, Collections.emptyList());

}

}