package com.twitter.home\_mixer.util

import com.twitter.home\_mixer.model.HomeFeatures.\_

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

object ReplyRetweetUtil {

def isEligibleReply(candidate: CandidateWithFeatures[TweetCandidate]): Boolean = {

candidate.features.getOrElse(InReplyToTweetIdFeature, None).nonEmpty &&

!candidate.features.getOrElse(IsRetweetFeature, false)

}

/\*\*

\* Builds a map from reply tweet to all ancestors that are also hydrated candidates. If a reply

\* does not have any ancestors which are also candidates, it will not add to the returned Map.

\* Make sure ancestors are bottom-up ordered such that:

\* (1) if parent tweet is a candidate, it should be the first item at the returned ancestors;

\* (2) if root tweet is a candidate, it should be the last item at the returned ancestors.

\* Retweets of replies or replies to retweets are not included.

\*/

def replyToAncestorTweetCandidatesMap(

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Map[Long, Seq[CandidateWithFeatures[TweetCandidate]]] = {

val replyToAncestorTweetIdsMap: Map[Long, Seq[Long]] =

candidates.flatMap { candidate =>

if (isEligibleReply(candidate)) {

val ancestorIds =

if (candidate.features.getOrElse(AncestorsFeature, Seq.empty).nonEmpty) {

candidate.features.getOrElse(AncestorsFeature, Seq.empty).map(\_.tweetId)

} else {

Seq(

candidate.features.getOrElse(InReplyToTweetIdFeature, None),

candidate.features.getOrElse(ConversationModuleIdFeature, None)

).flatten.distinct

}

Some(candidate.candidate.id -> ancestorIds)

} else {

None

}

}.toMap

val ancestorTweetIds = replyToAncestorTweetIdsMap.values.flatten.toSet

val ancestorTweetsMapById: Map[Long, CandidateWithFeatures[TweetCandidate]] = candidates

.filter { maybeAncestor =>

ancestorTweetIds.contains(maybeAncestor.candidate.id)

}.map { ancestor =>

ancestor.candidate.id -> ancestor

}.toMap

replyToAncestorTweetIdsMap

.mapValues { ancestorTweetIds =>

ancestorTweetIds.flatMap { ancestorTweetId =>

ancestorTweetsMapById.get(ancestorTweetId)

}

}.filter {

case (reply, ancestors) =>

ancestors.nonEmpty

}

}

/\*\*

\* This map is the opposite of [[replyToAncestorTweetCandidatesMap]].

\* Builds a map from ancestor tweet to all descendant replies that are also hydrated candidates.

\* Currently, we only return two ancestors at most: one is inReplyToTweetId and the other

\* is conversationId.

\* Retweets of replies are not included.

\*/

def ancestorTweetIdToDescendantRepliesMap(

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Map[Long, Seq[CandidateWithFeatures[TweetCandidate]]] = {

val tweetToCandidateMap = candidates.map(c => c.candidate.id -> c).toMap

replyToAncestorTweetCandidatesMap(candidates).toSeq

.flatMap {

case (reply, ancestorTweets) =>

ancestorTweets.map { ancestor =>

(ancestor.candidate.id, reply)

}

}.groupBy { case (ancestor, reply) => ancestor }

.mapValues { ancestorReplyPairs =>

ancestorReplyPairs.map(\_.\_2).distinct

}.mapValues(tweetIds => tweetIds.map(tid => tweetToCandidateMap(tid)))

}

/\*\*

\* Builds a map from reply tweet to inReplyToTweet which is also a candidate.

\* Retweets of replies or replies to retweets are not included

\*/

def replyTweetIdToInReplyToTweetMap(

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Map[Long, CandidateWithFeatures[TweetCandidate]] = {

val eligibleReplyCandidates = candidates.filter { candidate =>

isEligibleReply(candidate) && candidate.features

.getOrElse(InReplyToTweetIdFeature, None)

.nonEmpty

}

val inReplyToTweetIds = eligibleReplyCandidates

.flatMap(\_.features.getOrElse(InReplyToTweetIdFeature, None))

.toSet

val inReplyToTweetIdToTweetMap: Map[Long, CandidateWithFeatures[TweetCandidate]] = candidates

.filter { maybeInReplyToTweet =>

inReplyToTweetIds.contains(maybeInReplyToTweet.candidate.id)

}.map { inReplyToTweet =>

inReplyToTweet.candidate.id -> inReplyToTweet

}.toMap

eligibleReplyCandidates.flatMap { reply =>

val inReplyToTweetId = reply.features.getOrElse(InReplyToTweetIdFeature, None)

if (inReplyToTweetId.nonEmpty) {

inReplyToTweetIdToTweetMap.get(inReplyToTweetId.get).map { inReplyToTweet =>

reply.candidate.id -> inReplyToTweet

}

} else {

None

}

}.toMap

}

}