package com.twitter.timelineranker.util

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.timelineranker.core.HydratedTweets

import com.twitter.timelines.clients.tweetypie.TweetyPieClient

import com.twitter.timelines.model.\_

import com.twitter.timelines.model.tweet.HydratedTweet

import com.twitter.timelines.model.tweet.HydratedTweetUtils

import com.twitter.timelines.util.stats.RequestStats

import com.twitter.tweetypie.thriftscala.TweetInclude

import com.twitter.util.Future

object TweetHydrator {

val FieldsToHydrate: Set[TweetInclude] = TweetyPieClient.CoreTweetFields

val EmptyHydratedTweets: HydratedTweets =

HydratedTweets(Seq.empty[HydratedTweet], Seq.empty[HydratedTweet])

val EmptyHydratedTweetsFuture: Future[HydratedTweets] = Future.value(EmptyHydratedTweets)

}

class TweetHydrator(tweetyPieClient: TweetyPieClient, statsReceiver: StatsReceiver)

extends RequestStats {

private[this] val hydrateScope = statsReceiver.scope("tweetHydrator")

private[this] val outerTweetsScope = hydrateScope.scope("outerTweets")

private[this] val innerTweetsScope = hydrateScope.scope("innerTweets")

private[this] val totalCounter = outerTweetsScope.counter(Total)

private[this] val totalInnerCounter = innerTweetsScope.counter(Total)

/\*\*

\* Hydrates zero or more tweets from the given seq of tweet IDs. Returns requested tweets ordered

\* by tweetIds and out of order inner tweet ids.

\*

\* Inner tweets that were also requested as outer tweets are returned as outer tweets.

\*

\* Note that some tweet may not be hydrated due to hydration errors or because they are deleted.

\* Consequently, the size of output is <= size of input. That is the intended usage pattern.

\*/

def hydrate(

viewerId: Option[UserId],

tweetIds: Seq[TweetId],

fieldsToHydrate: Set[TweetInclude] = TweetyPieClient.CoreTweetFields,

includeQuotedTweets: Boolean = false

): Future[HydratedTweets] = {

if (tweetIds.isEmpty) {

TweetHydrator.EmptyHydratedTweetsFuture

} else {

val tweetStateMapFuture = tweetyPieClient.getHydratedTweetFields(

tweetIds,

viewerId,

fieldsToHydrate,

safetyLevel = Some(SafetyLevel.FilterNone),

bypassVisibilityFiltering = true,

includeSourceTweets = false,

includeQuotedTweets = includeQuotedTweets,

ignoreTweetSuppression = true

)

tweetStateMapFuture.map { tweetStateMap =>

val innerTweetIdSet = tweetStateMap.keySet -- tweetIds.toSet

val hydratedTweets =

HydratedTweetUtils.extractAndOrder(tweetIds ++ innerTweetIdSet.toSeq, tweetStateMap)

val (outer, inner) = hydratedTweets.partition { tweet =>

!innerTweetIdSet.contains(tweet.tweetId)

}

totalCounter.incr(outer.size)

totalInnerCounter.incr(inner.size)

HydratedTweets(outer, inner)

}

}

}

}