package com.twitter.timelineranker.common

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

import com.twitter.servo.util.FutureArrow

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.core.HydratedTweets

import com.twitter.timelineranker.util.TweetFilters

import com.twitter.timelineranker.util.TweetsPostFilter

import com.twitter.timelines.model.UserId

import com.twitter.util.Future

object HydratedTweetsFilterTransform {

val EmptyFollowGraphDataTuple: (Seq[UserId], Seq[UserId], Set[UserId]) =

(Seq.empty[UserId], Seq.empty[UserId], Set.empty[UserId])

val DefaultNumRetweetsAllowed = 1

// Number of duplicate retweets (including the first one) allowed.

// For example,

// If there are 7 retweets of a given tweet, the following value will cause 5 of them

// to be returned after filtering and the additional 2 will be filtered out.

val NumDuplicateRetweetsAllowed = 5

}

/\*\*

\* Transform which takes TweetFilters ValueSets for inner and outer tweets and uses

\* TweetsPostFilter to filter down the HydratedTweets using the supplied filters

\*

\* @param useFollowGraphData - use follow graph for filtering; otherwise only does filtering

\* independent of follow graph data

\* @param useSourceTweets - only needed when filtering extended replies

\* @param statsReceiver - scoped stats receiver

\*/

class HydratedTweetsFilterTransform(

outerFilters: TweetFilters.ValueSet,

innerFilters: TweetFilters.ValueSet,

useFollowGraphData: Boolean,

useSourceTweets: Boolean,

statsReceiver: StatsReceiver,

numRetweetsAllowed: Int = HydratedTweetsFilterTransform.DefaultNumRetweetsAllowed)

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

import HydratedTweetsFilterTransform.\_

val logger: Logger = Logger.get(getClass.getSimpleName)

override def apply(envelope: CandidateEnvelope): Future[CandidateEnvelope] = {

if (outerFilters == TweetFilters.None) {

Future.value(envelope)

} else {

val tweetsPostOuterFilter = new TweetsPostFilter(outerFilters, logger, statsReceiver)

val tweetsPostInnerFilter = new TweetsPostFilter(innerFilters, logger, statsReceiver)

val graphData = if (useFollowGraphData) {

Future.join(

envelope.followGraphData.followedUserIdsFuture,

envelope.followGraphData.inNetworkUserIdsFuture,

envelope.followGraphData.mutedUserIdsFuture

)

} else {

Future.value(EmptyFollowGraphDataTuple)

}

val sourceTweets = if (useSourceTweets) {

envelope.sourceHydratedTweets.outerTweets

} else {

Nil

}

graphData.map {

case (followedUserIds, inNetworkUserIds, mutedUserIds) =>

val outerTweets = tweetsPostOuterFilter(

userId = envelope.query.userId,

followedUserIds = followedUserIds,

inNetworkUserIds = inNetworkUserIds,

mutedUserIds = mutedUserIds,

tweets = envelope.hydratedTweets.outerTweets,

numRetweetsAllowed = numRetweetsAllowed,

sourceTweets = sourceTweets

)

val innerTweets = tweetsPostInnerFilter(

userId = envelope.query.userId,

followedUserIds = followedUserIds,

inNetworkUserIds = inNetworkUserIds,

mutedUserIds = mutedUserIds,

// inner tweets refers to quoted tweets not source tweets, and special rulesets

// in birdherd handle visibility of viewer to inner tweet author for these tweets.

tweets = envelope.hydratedTweets.innerTweets,

numRetweetsAllowed = numRetweetsAllowed,

sourceTweets = sourceTweets

)

envelope.copy(hydratedTweets = HydratedTweets(outerTweets, innerTweets))

}

}

}

}