package com.twitter.timelineranker.common

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.timelineranker.core.CandidateEnvelope

import com.twitter.timelineranker.model.RecapQuery.DependencyProvider

import com.twitter.timelineranker.model.TweetIdRange

import com.twitter.timelineranker.parameters.recap.RecapParams

import com.twitter.timelines.clients.relevance\_search.SearchClient

import com.twitter.timelines.clients.relevance\_search.SearchClient.TweetTypes

import com.twitter.util.Future

/\*\*

\* Fetch recap/recycled search results using the search client

\* and populate them into the CandidateEnvelope

\*/

class RecapSearchResultsTransform(

searchClient: SearchClient,

maxCountProvider: DependencyProvider[Int],

returnAllResultsProvider: DependencyProvider[Boolean],

relevanceOptionsMaxHitsToProcessProvider: DependencyProvider[Int],

enableExcludeSourceTweetIdsProvider: DependencyProvider[Boolean],

enableSettingTweetTypesWithTweetKindOptionProvider: DependencyProvider[Boolean],

perRequestSearchClientIdProvider: DependencyProvider[Option[String]],

relevanceSearchProvider: DependencyProvider[Boolean] =

DependencyProvider.from(RecapParams.EnableRelevanceSearchParam),

statsReceiver: StatsReceiver,

logSearchDebugInfo: Boolean = true)

extends FutureArrow[CandidateEnvelope, CandidateEnvelope] {

private[this] val maxCountStat = statsReceiver.stat("maxCount")

private[this] val numResultsFromSearchStat = statsReceiver.stat("numResultsFromSearch")

private[this] val excludedTweetIdsStat = statsReceiver.stat("excludedTweetIds")

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

val maxCount = maxCountProvider(envelope.query)

maxCountStat.add(maxCount)

val excludedTweetIdsOpt = envelope.query.excludedTweetIds

excludedTweetIdsOpt.foreach { excludedTweetIds =>

excludedTweetIdsStat.add(excludedTweetIds.size)

}

val tweetIdRange = envelope.query.range

.map(TweetIdRange.fromTimelineRange)

.getOrElse(TweetIdRange.default)

val beforeTweetIdExclusive = tweetIdRange.toId

val afterTweetIdExclusive = tweetIdRange.fromId

val returnAllResults = returnAllResultsProvider(envelope.query)

val relevanceOptionsMaxHitsToProcess = relevanceOptionsMaxHitsToProcessProvider(envelope.query)

Future

.join(

envelope.followGraphData.followedUserIdsFuture,

envelope.followGraphData.retweetsMutedUserIdsFuture

).flatMap {

case (followedIds, retweetsMutedIds) =>

val followedIdsIncludingSelf = followedIds.toSet + envelope.query.userId

searchClient

.getUsersTweetsForRecap(

userId = envelope.query.userId,

followedUserIds = followedIdsIncludingSelf,

retweetsMutedUserIds = retweetsMutedIds,

maxCount = maxCount,

tweetTypes = TweetTypes.fromTweetKindOption(envelope.query.options),

searchOperator = envelope.query.searchOperator,

beforeTweetIdExclusive = beforeTweetIdExclusive,

afterTweetIdExclusive = afterTweetIdExclusive,

enableSettingTweetTypesWithTweetKindOption =

enableSettingTweetTypesWithTweetKindOptionProvider(envelope.query),

excludedTweetIds = excludedTweetIdsOpt,

earlybirdOptions = envelope.query.earlybirdOptions,

getOnlyProtectedTweets = false,

logSearchDebugInfo = logSearchDebugInfo,

returnAllResults = returnAllResults,

enableExcludeSourceTweetIdsQuery =

enableExcludeSourceTweetIdsProvider(envelope.query),

relevanceSearch = relevanceSearchProvider(envelope.query),

searchClientId = perRequestSearchClientIdProvider(envelope.query),

relevanceOptionsMaxHitsToProcess = relevanceOptionsMaxHitsToProcess

).map { results =>

numResultsFromSearchStat.add(results.size)

envelope.copy(searchResults = results)

}

}

}

}