package com.twitter.timelineranker.common

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.servo.util.Gate

import com.twitter.storehaus.Store

import com.twitter.timelineranker.contentfeatures.ContentFeaturesProvider

import com.twitter.timelineranker.core.FutureDependencyTransformer

import com.twitter.timelineranker.core.HydratedCandidatesAndFeaturesEnvelope

import com.twitter.timelineranker.model.RecapQuery

import com.twitter.timelineranker.recap.model.ContentFeatures

import com.twitter.timelineranker.util.SearchResultUtil.\_

import com.twitter.timelineranker.util.CachingContentFeaturesProvider

import com.twitter.timelineranker.util.TweetHydrator

import com.twitter.timelineranker.util.TweetypieContentFeaturesProvider

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

import com.twitter.timelines.model.TweetId

import com.twitter.util.Future

import com.twitter.timelines.configapi

import com.twitter.timelines.util.FutureUtils

class ContentFeaturesHydrationTransformBuilder(

tweetyPieClient: TweetyPieClient,

contentFeaturesCache: Store[TweetId, ContentFeatures],

enableContentFeaturesGate: Gate[RecapQuery],

enableTokensInContentFeaturesGate: Gate[RecapQuery],

enableTweetTextInContentFeaturesGate: Gate[RecapQuery],

enableConversationControlContentFeaturesGate: Gate[RecapQuery],

enableTweetMediaHydrationGate: Gate[RecapQuery],

hydrateInReplyToTweets: Boolean,

statsReceiver: StatsReceiver) {

val scopedStatsReceiver: StatsReceiver = statsReceiver.scope("ContentFeaturesHydrationTransform")

val tweetHydrator: TweetHydrator = new TweetHydrator(tweetyPieClient, scopedStatsReceiver)

val tweetypieContentFeaturesProvider: ContentFeaturesProvider =

new TweetypieContentFeaturesProvider(

tweetHydrator,

enableContentFeaturesGate,

enableTokensInContentFeaturesGate,

enableTweetTextInContentFeaturesGate,

enableConversationControlContentFeaturesGate,

enableTweetMediaHydrationGate,

scopedStatsReceiver

)

val cachingContentFeaturesProvider: ContentFeaturesProvider = new CachingContentFeaturesProvider(

underlying = tweetypieContentFeaturesProvider,

contentFeaturesCache = contentFeaturesCache,

statsReceiver = scopedStatsReceiver

)

val contentFeaturesProvider: configapi.FutureDependencyTransformer[RecapQuery, Seq[TweetId], Map[

TweetId,

ContentFeatures

]] = FutureDependencyTransformer.partition(

gate = enableContentFeaturesGate,

ifTrue = cachingContentFeaturesProvider,

ifFalse = tweetypieContentFeaturesProvider

)

lazy val contentFeaturesHydrationTransform: ContentFeaturesHydrationTransform =

new ContentFeaturesHydrationTransform(

contentFeaturesProvider,

enableContentFeaturesGate,

hydrateInReplyToTweets

)

def build(): ContentFeaturesHydrationTransform = contentFeaturesHydrationTransform

}

class ContentFeaturesHydrationTransform(

contentFeaturesProvider: ContentFeaturesProvider,

enableContentFeaturesGate: Gate[RecapQuery],

hydrateInReplyToTweets: Boolean)

extends FutureArrow[

HydratedCandidatesAndFeaturesEnvelope,

HydratedCandidatesAndFeaturesEnvelope

] {

override def apply(

request: HydratedCandidatesAndFeaturesEnvelope

): Future[HydratedCandidatesAndFeaturesEnvelope] = {

if (enableContentFeaturesGate(request.candidateEnvelope.query)) {

val searchResults = request.candidateEnvelope.searchResults

val sourceTweetIdMap = searchResults.map { searchResult =>

(searchResult.id, getRetweetSourceTweetId(searchResult).getOrElse(searchResult.id))

}.toMap

val inReplyToTweetIds = if (hydrateInReplyToTweets) {

searchResults.flatMap(getInReplyToTweetId)

} else {

Seq.empty

}

val tweetIdsToHydrate = (sourceTweetIdMap.values ++ inReplyToTweetIds).toSeq.distinct

val contentFeaturesMapFuture = if (tweetIdsToHydrate.nonEmpty) {

contentFeaturesProvider(request.candidateEnvelope.query, tweetIdsToHydrate)

} else {

FutureUtils.EmptyMap[TweetId, ContentFeatures]

}

Future.value(

request.copy(

contentFeaturesFuture = contentFeaturesMapFuture,

tweetSourceTweetMap = sourceTweetIdMap,

inReplyToTweetIds = inReplyToTweetIds.toSet

)

)

} else {

Future.value(request)

}

}

}