package com.twitter.timelineranker.recap\_hydration

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FutureArrow

import com.twitter.storehaus.Store

import com.twitter.timelineranker.common.\_

import com.twitter.timelineranker.core.HydratedCandidatesAndFeaturesEnvelope

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

import com.twitter.timelineranker.model.\_

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

import com.twitter.timelineranker.parameters.recap\_hydration.RecapHydrationParams

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

import com.twitter.timelineranker.util.CopyContentFeaturesIntoHydratedTweetsTransform

import com.twitter.timelineranker.util.CopyContentFeaturesIntoThriftTweetFeaturesTransform

import com.twitter.timelineranker.visibility.FollowGraphDataProvider

import com.twitter.timelines.clients.gizmoduck.GizmoduckClient

import com.twitter.timelines.clients.manhattan.UserMetadataClient

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

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

import com.twitter.timelines.model.TweetId

import com.twitter.timelines.util.FailOpenHandler

import com.twitter.timelines.util.stats.RequestStatsReceiver

import com.twitter.util.Future

class RecapHydrationSource(

gizmoduckClient: GizmoduckClient,

searchClient: SearchClient,

tweetyPieClient: TweetyPieClient,

userMetadataClient: UserMetadataClient,

followGraphDataProvider: FollowGraphDataProvider,

contentFeaturesCache: Store[TweetId, ContentFeatures],

statsReceiver: StatsReceiver) {

private[this] val baseScope = statsReceiver.scope("recapHydration")

private[this] val requestStats = RequestStatsReceiver(baseScope)

private[this] val numInputTweetsStat = baseScope.stat("numInputTweets")

private[this] val failOpenScope = baseScope.scope("failOpen")

private[this] val userProfileHandler = new FailOpenHandler(failOpenScope, "userProfileInfo")

private[this] val userLanguagesHandler = new FailOpenHandler(failOpenScope, "userLanguages")

private[this] val maxFollowedUsersProvider =

DependencyProvider.value(RecapParams.MaxFollowedUsers.default)

private[this] val followGraphDataTransform =

new FollowGraphDataTransform(followGraphDataProvider, maxFollowedUsersProvider)

private[this] val searchResultsTransform =

new RecapHydrationSearchResultsTransform(searchClient, baseScope)

private[this] val userProfileInfoTransform =

new UserProfileInfoTransform(userProfileHandler, gizmoduckClient)

private[this] val languagesTransform =

new UserLanguagesTransform(userLanguagesHandler, userMetadataClient)

private[this] val candidateGenerationTransform = new CandidateGenerationTransform(baseScope)

private[this] val hydrationAndFilteringPipeline =

CreateCandidateEnvelopeTransform

.andThen(followGraphDataTransform)

.andThen(searchResultsTransform)

.andThen(CandidateTweetHydrationTransform)

// runs the main pipeline in parallel with fetching user profile info and user languages

private[this] val featureHydrationDataTransform = new FeatureHydrationDataTransform(

hydrationAndFilteringPipeline,

languagesTransform,

userProfileInfoTransform

)

private[this] val contentFeaturesHydrationTransform =

new ContentFeaturesHydrationTransformBuilder(

tweetyPieClient = tweetyPieClient,

contentFeaturesCache = contentFeaturesCache,

enableContentFeaturesGate =

RecapQuery.paramGate(RecapHydrationParams.EnableContentFeaturesHydrationParam),

enableTokensInContentFeaturesGate =

RecapQuery.paramGate(RecapHydrationParams.EnableTokensInContentFeaturesHydrationParam),

enableTweetTextInContentFeaturesGate =

RecapQuery.paramGate(RecapHydrationParams.EnableTweetTextInContentFeaturesHydrationParam),

enableConversationControlContentFeaturesGate = RecapQuery.paramGate(

RecapHydrationParams.EnableConversationControlInContentFeaturesHydrationParam),

enableTweetMediaHydrationGate = RecapQuery.paramGate(

RecapHydrationParams.EnableTweetMediaHydrationParam

),

hydrateInReplyToTweets = true,

statsReceiver = baseScope

).build()

private[this] def hydratesContentFeatures(

hydratedEnvelope: HydratedCandidatesAndFeaturesEnvelope

): Boolean =

hydratedEnvelope.candidateEnvelope.query.hydratesContentFeatures.getOrElse(true)

private[this] val contentFeaturesTransformer = FutureArrow.choose(

predicate = hydratesContentFeatures,

ifTrue = contentFeaturesHydrationTransform

.andThen(CopyContentFeaturesIntoThriftTweetFeaturesTransform)

.andThen(CopyContentFeaturesIntoHydratedTweetsTransform),

ifFalse = FutureArrow[

HydratedCandidatesAndFeaturesEnvelope,

HydratedCandidatesAndFeaturesEnvelope

](Future.value) // empty transformer

)

private[this] val featureHydrationPipeline =

featureHydrationDataTransform

.andThen(InNetworkTweetsSearchFeaturesHydrationTransform)

.andThen(contentFeaturesTransformer)

.andThen(candidateGenerationTransform)

def hydrate(queries: Seq[RecapQuery]): Future[Seq[CandidateTweetsResult]] = {

Future.collect(queries.map(hydrate))

}

def hydrate(query: RecapQuery): Future[CandidateTweetsResult] = {

require(query.tweetIds.isDefined && query.tweetIds.get.nonEmpty, "tweetIds must be present")

query.tweetIds.foreach(ids => numInputTweetsStat.add(ids.size))

requestStats.addEventStats {

featureHydrationPipeline(query)

}

}

}