package com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator

import com.google.inject.name.Named

import com.twitter.conversions.DurationOps.RichDuration

import com.twitter.home\_mixer.model.HomeFeatures.AuthorIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.DirectedAtUserIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.ExclusiveConversationAuthorIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.HasImageFeature

import com.twitter.home\_mixer.model.HomeFeatures.HasVideoFeature

import com.twitter.home\_mixer.model.HomeFeatures.InReplyToTweetIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.InReplyToUserIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.IsRetweetFeature

import com.twitter.home\_mixer.model.HomeFeatures.MentionScreenNameFeature

import com.twitter.home\_mixer.model.HomeFeatures.MentionUserIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.QuotedTweetIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.QuotedUserIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.SourceTweetIdFeature

import com.twitter.home\_mixer.model.HomeFeatures.SourceUserIdFeature

import com.twitter.home\_mixer.param.HomeMixerInjectionNames.TweetypieStaticEntitiesCache

import com.twitter.home\_mixer.util.tweetypie.RequestFields

import com.twitter.home\_mixer.util.tweetypie.content.TweetMediaFeaturesExtractor

import com.twitter.product\_mixer.component\_library.model.candidate.TweetCandidate

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMapBuilder

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BulkCandidateFeatureHydrator

import com.twitter.product\_mixer.core.model.common.CandidateWithFeatures

import com.twitter.product\_mixer.core.model.common.identifier.FeatureHydratorIdentifier

import com.twitter.product\_mixer.core.pipeline.PipelineQuery

import com.twitter.servo.cache.TtlCache

import com.twitter.spam.rtf.{thriftscala => sp}

import com.twitter.stitch.Stitch

import com.twitter.stitch.tweetypie.{TweetyPie => TweetypieStitchClient}

import com.twitter.tweetypie.{thriftscala => tp}

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class TweetypieStaticEntitiesFeatureHydrator @Inject() (

tweetypieStitchClient: TweetypieStitchClient,

@Named(TweetypieStaticEntitiesCache) cacheClient: TtlCache[Long, tp.Tweet])

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate] {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("TweetypieStaticEntities")

override val features: Set[Feature[\_, \_]] = Set(

AuthorIdFeature,

DirectedAtUserIdFeature,

ExclusiveConversationAuthorIdFeature,

HasImageFeature,

HasVideoFeature,

InReplyToTweetIdFeature,

InReplyToUserIdFeature,

IsRetweetFeature,

MentionScreenNameFeature,

MentionUserIdFeature,

QuotedTweetIdFeature,

QuotedUserIdFeature,

SourceTweetIdFeature,

SourceUserIdFeature

)

private val CacheTTL = 24.hours

private val DefaultFeatureMap = FeatureMapBuilder()

.add(AuthorIdFeature, None)

.add(DirectedAtUserIdFeature, None)

.add(ExclusiveConversationAuthorIdFeature, None)

.add(HasImageFeature, false)

.add(HasVideoFeature, false)

.add(InReplyToTweetIdFeature, None)

.add(InReplyToUserIdFeature, None)

.add(IsRetweetFeature, false)

.add(MentionScreenNameFeature, Seq.empty)

.add(MentionUserIdFeature, Seq.empty)

.add(QuotedTweetIdFeature, None)

.add(QuotedUserIdFeature, None)

.add(SourceTweetIdFeature, None)

.add(SourceUserIdFeature, None)

.build()

/\*\*

\* Steps:

\* 1. query cache with all candidates

\* 2. create a cached feature map

\* 3. iterate candidates to hydrate features

\* 3.a transform cached candidates

\* 3.b hydrate non-cached candidates from Tweetypie and write to cache

\*/

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = {

val tweetIds = candidates.map(\_.candidate.id)

val cachedTweetsMapFu = cacheClient

.get(tweetIds)

.map(\_.found)

Stitch.callFuture(cachedTweetsMapFu).flatMap { cachedTweets =>

Stitch.collect {

candidates.map { candidate =>

if (cachedTweets.contains(candidate.candidate.id))

Stitch.value(createFeatureMap(cachedTweets(candidate.candidate.id)))

else readFromTweetypie(query, candidate)

}

}

}

}

private def createFeatureMap(tweet: tp.Tweet): FeatureMap = {

val coreData = tweet.coreData

val quotedTweet = tweet.quotedTweet

val mentions = tweet.mentions.getOrElse(Seq.empty)

val share = coreData.flatMap(\_.share)

val reply = coreData.flatMap(\_.reply)

FeatureMapBuilder()

.add(AuthorIdFeature, coreData.map(\_.userId))

.add(DirectedAtUserIdFeature, coreData.flatMap(\_.directedAtUser.map(\_.userId)))

.add(

ExclusiveConversationAuthorIdFeature,

tweet.exclusiveTweetControl.map(\_.conversationAuthorId))

.add(HasImageFeature, TweetMediaFeaturesExtractor.hasImage(tweet))

.add(HasVideoFeature, TweetMediaFeaturesExtractor.hasVideo(tweet))

.add(InReplyToTweetIdFeature, reply.flatMap(\_.inReplyToStatusId))

.add(InReplyToUserIdFeature, reply.map(\_.inReplyToUserId))

.add(IsRetweetFeature, share.isDefined)

.add(MentionScreenNameFeature, mentions.map(\_.screenName))

.add(MentionUserIdFeature, mentions.flatMap(\_.userId))

.add(QuotedTweetIdFeature, quotedTweet.map(\_.tweetId))

.add(QuotedUserIdFeature, quotedTweet.map(\_.userId))

.add(SourceTweetIdFeature, share.map(\_.sourceStatusId))

.add(SourceUserIdFeature, share.map(\_.sourceUserId))

.build()

}

private def readFromTweetypie(

query: PipelineQuery,

candidate: CandidateWithFeatures[TweetCandidate]

): Stitch[FeatureMap] = {

tweetypieStitchClient

.getTweetFields(

tweetId = candidate.candidate.id,

options = tp.GetTweetFieldsOptions(

tweetIncludes = RequestFields.TweetStaticEntitiesFields,

includeRetweetedTweet = false,

includeQuotedTweet = false,

forUserId = query.getOptionalUserId, // Needed to get protected Tweets for certain users

visibilityPolicy = tp.TweetVisibilityPolicy.UserVisible,

safetyLevel = Some(sp.SafetyLevel.FilterNone) // VF is handled in the For You product

)

).map {

case tp.GetTweetFieldsResult(\_, tp.TweetFieldsResultState.Found(found), \_, \_) =>

cacheClient.set(candidate.candidate.id, found.tweet, CacheTTL)

createFeatureMap(found.tweet)

case \_ =>

DefaultFeatureMap + (AuthorIdFeature, candidate.features.getOrElse(AuthorIdFeature, None))

}

}

}