package com.twitter.frigate.pushservice.adaptor

import com.twitter.events.recos.thriftscala.DisplayLocation

import com.twitter.events.recos.thriftscala.TrendsContext

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateSource

import com.twitter.frigate.common.base.CandidateSourceEligible

import com.twitter.frigate.common.base.TrendTweetCandidate

import com.twitter.frigate.common.base.TrendsCandidate

import com.twitter.frigate.common.candidate.RecommendedTrendsCandidateSource

import com.twitter.frigate.common.candidate.RecommendedTrendsCandidateSource.Query

import com.twitter.frigate.common.predicate.CommonOutNetworkTweetCandidatesSourcePredicates.filterOutReplyTweet

import com.twitter.frigate.pushservice.model.PushTypes.RawCandidate

import com.twitter.frigate.pushservice.model.PushTypes.Target

import com.twitter.frigate.pushservice.adaptor.TrendsCandidatesAdaptor.\_

import com.twitter.frigate.pushservice.params.PushFeatureSwitchParams

import com.twitter.frigate.pushservice.params.PushParams

import com.twitter.frigate.pushservice.predicate.TargetPredicates

import com.twitter.frigate.pushservice.util.PushDeviceUtil

import com.twitter.frigate.thriftscala.CommonRecommendationType

import com.twitter.geoduck.common.thriftscala.Location

import com.twitter.gizmoduck.thriftscala.UserType

import com.twitter.hermit.store.tweetypie.UserTweet

import com.twitter.stitch.tweetypie.TweetyPie.TweetyPieResult

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import scala.collection.Map

object TrendsCandidatesAdaptor {

type TweetId = Long

type EventId = Long

}

case class TrendsCandidatesAdaptor(

softUserGeoLocationStore: ReadableStore[Long, Location],

recommendedTrendsCandidateSource: RecommendedTrendsCandidateSource,

tweetyPieStore: ReadableStore[Long, TweetyPieResult],

tweetyPieStoreNoVF: ReadableStore[Long, TweetyPieResult],

safeUserTweetTweetyPieStore: ReadableStore[UserTweet, TweetyPieResult],

statsReceiver: StatsReceiver)

extends CandidateSource[Target, RawCandidate]

with CandidateSourceEligible[Target, RawCandidate] {

override val name = this.getClass.getSimpleName

private val trendAdaptorStats = statsReceiver.scope("TrendsCandidatesAdaptor")

private val trendTweetCandidateNumber = trendAdaptorStats.counter("trend\_tweet\_candidate")

private val nonReplyTweetsCounter = trendAdaptorStats.counter("non\_reply\_tweets")

private def getQuery(target: Target): Future[Query] = {

def getUserCountryCode(target: Target): Future[Option[String]] = {

target.targetUser.flatMap {

case Some(user) if user.userType == UserType.Soft =>

softUserGeoLocationStore

.get(user.id)

.map(\_.flatMap(\_.simpleRgcResult.flatMap(\_.countryCodeAlpha2)))

case \_ => target.accountCountryCode

}

}

for {

countryCode <- getUserCountryCode(target)

inferredLanguage <- target.inferredUserDeviceLanguage

} yield {

Query(

userId = target.targetId,

displayLocation = DisplayLocation.MagicRecs,

languageCode = inferredLanguage,

countryCode = countryCode,

maxResults = target.params(PushFeatureSwitchParams.MaxRecommendedTrendsToQuery)

)

}

}

/\*\*

\* Query candidates only if sent at most [[PushFeatureSwitchParams.MaxTrendTweetNotificationsInDuration]]

\* trend tweet notifications in [[PushFeatureSwitchParams.TrendTweetNotificationsFatigueDuration]]

\*/

val trendTweetFatiguePredicate = TargetPredicates.pushRecTypeFatiguePredicate(

CommonRecommendationType.TrendTweet,

PushFeatureSwitchParams.TrendTweetNotificationsFatigueDuration,

PushFeatureSwitchParams.MaxTrendTweetNotificationsInDuration,

trendAdaptorStats

)

private val recommendedTrendsWithTweetsCandidateSource: CandidateSource[

Target,

RawCandidate with TrendsCandidate

] = recommendedTrendsCandidateSource

.convert[Target, TrendsCandidate](

getQuery,

recommendedTrendsCandidateSource.identityCandidateMapper

)

.batchMapValues[Target, RawCandidate with TrendsCandidate](

trendsCandidatesToTweetCandidates(\_, \_, getTweetyPieResults))

private def getTweetyPieResults(

tweetIds: Seq[TweetId],

target: Target

): Future[Map[TweetId, TweetyPieResult]] = {

if (target.params(PushFeatureSwitchParams.EnableSafeUserTweetTweetypieStore)) {

Future

.collect(

safeUserTweetTweetyPieStore.multiGet(

tweetIds.toSet.map(UserTweet(\_, Some(target.targetId))))).map {

\_.collect {

case (userTweet, Some(tweetyPieResult)) => userTweet.tweetId -> tweetyPieResult

}

}

} else {

Future

.collect((target.params(PushFeatureSwitchParams.EnableVFInTweetypie) match {

case true => tweetyPieStore

case false => tweetyPieStoreNoVF

}).multiGet(tweetIds.toSet)).map { tweetyPieResultMap =>

filterOutReplyTweet(tweetyPieResultMap, nonReplyTweetsCounter).collect {

case (tweetId, Some(tweetyPieResult)) => tweetId -> tweetyPieResult

}

}

}

}

/\*\*

\*

\* @param \_target: [[Target]] object representing notificaion recipient user

\* @param trendsCandidates: Sequence of [[TrendsCandidate]] returned from ERS

\* @return: Seq of trends candidates expanded to associated tweets.

\*/

private def trendsCandidatesToTweetCandidates(

\_target: Target,

trendsCandidates: Seq[TrendsCandidate],

getTweetyPieResults: (Seq[TweetId], Target) => Future[Map[TweetId, TweetyPieResult]]

): Future[Seq[RawCandidate with TrendsCandidate]] = {

def generateTrendTweetCandidates(

trendCandidate: TrendsCandidate,

tweetyPieResults: Map[TweetId, TweetyPieResult]

) = {

val tweetIds = trendCandidate.context.curatedRepresentativeTweets.getOrElse(Seq.empty) ++

trendCandidate.context.algoRepresentativeTweets.getOrElse(Seq.empty)

tweetIds.flatMap { tweetId =>

tweetyPieResults.get(tweetId).map { \_tweetyPieResult =>

new RawCandidate with TrendTweetCandidate {

override val trendId: String = trendCandidate.trendId

override val trendName: String = trendCandidate.trendName

override val landingUrl: String = trendCandidate.landingUrl

override val timeBoundedLandingUrl: Option[String] =

trendCandidate.timeBoundedLandingUrl

override val context: TrendsContext = trendCandidate.context

override val tweetyPieResult: Option[TweetyPieResult] = Some(\_tweetyPieResult)

override val tweetId: TweetId = \_tweetyPieResult.tweet.id

override val target: Target = \_target

}

}

}

}

// collect all tweet ids associated with all trends

val allTweetIds = trendsCandidates.flatMap { trendsCandidate =>

val context = trendsCandidate.context

context.curatedRepresentativeTweets.getOrElse(Seq.empty) ++

context.algoRepresentativeTweets.getOrElse(Seq.empty)

}

getTweetyPieResults(allTweetIds, \_target)

.map { tweetIdToTweetyPieResult =>

val trendTweetCandidates = trendsCandidates.flatMap { trendCandidate =>

val allTrendTweetCandidates = generateTrendTweetCandidates(

trendCandidate,

tweetIdToTweetyPieResult

)

val (tweetCandidatesFromCuratedTrends, tweetCandidatesFromNonCuratedTrends) =

allTrendTweetCandidates.partition(\_.isCuratedTrend)

tweetCandidatesFromCuratedTrends.filter(

\_.target.params(PushFeatureSwitchParams.EnableCuratedTrendTweets)) ++

tweetCandidatesFromNonCuratedTrends.filter(

\_.target.params(PushFeatureSwitchParams.EnableNonCuratedTrendTweets))

}

trendTweetCandidateNumber.incr(trendTweetCandidates.size)

trendTweetCandidates

}

}

/\*\*

\*

\* @param target: [[Target]] user

\* @return: true if customer is eligible to receive trend tweet notifications

\*

\*/

override def isCandidateSourceAvailable(target: Target): Future[Boolean] = {

PushDeviceUtil

.isRecommendationsEligible(target)

.map(target.params(PushParams.TrendsCandidateDecider) && \_)

}

override def get(target: Target): Future[Option[Seq[RawCandidate with TrendsCandidate]]] = {

recommendedTrendsWithTweetsCandidateSource

.get(target)

.flatMap {

case Some(candidates) if candidates.nonEmpty =>

trendTweetFatiguePredicate(Seq(target))

.map(\_.head)

.map { isTargetFatigueEligible =>

if (isTargetFatigueEligible) Some(candidates)

else None

}

case \_ => Future.None

}

}

}