package com.twitter.frigate.pushservice.refresh\_handler

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateDetails

import com.twitter.frigate.common.base.FeatureMap

import com.twitter.frigate.data\_pipeline.features\_common.MrRequestContextForFeatureStore

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

import com.twitter.frigate.pushservice.ml.HydrationContextBuilder

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

import com.twitter.frigate.pushservice.util.MrUserStateUtil

import com.twitter.nrel.heavyranker.FeatureHydrator

import com.twitter.util.Future

class RFPHFeatureHydrator(

featureHydrator: FeatureHydrator

)(

implicit globalStats: StatsReceiver) {

implicit val statsReceiver: StatsReceiver =

globalStats.scope("RefreshForPushHandler")

//stat for feature hydration

private val featureHydrationEnabledCounter = statsReceiver.counter("featureHydrationEnabled")

private val mrUserStateStat = statsReceiver.scope("mr\_user\_state")

private def hydrateFromRelevanceHydrator(

candidateDetails: Seq[CandidateDetails[PushCandidate]],

mrRequestContextForFeatureStore: MrRequestContextForFeatureStore

): Future[Unit] = {

val pushCandidates = candidateDetails.map(\_.candidate)

val candidatesAndContextsFut = Future.collect(pushCandidates.map { pc =>

val contextFut = HydrationContextBuilder.build(pc)

contextFut.map { ctx => (pc, ctx) }

})

candidatesAndContextsFut.flatMap { candidatesAndContexts =>

val contexts = candidatesAndContexts.map(\_.\_2)

val resultsFut = featureHydrator.hydrateCandidate(contexts, mrRequestContextForFeatureStore)

resultsFut.map { hydrationResult =>

candidatesAndContexts.foreach {

case (pushCandidate, context) =>

val resultFeatures = hydrationResult.getOrElse(context, FeatureMap())

pushCandidate.mergeFeatures(resultFeatures)

}

}

}

}

def candidateFeatureHydration(

candidateDetails: Seq[CandidateDetails[PushCandidate]],

mrRequestContextForFeatureStore: MrRequestContextForFeatureStore

): Future[Seq[CandidateDetails[PushCandidate]]] = {

candidateDetails.headOption match {

case Some(cand) =>

val target = cand.candidate.target

MrUserStateUtil.updateMrUserStateStats(target)(mrUserStateStat)

if (target.params(PushParams.DisableAllRelevanceParam)) {

Future.value(candidateDetails)

} else {

featureHydrationEnabledCounter.incr()

for {

\_ <- hydrateFromRelevanceHydrator(candidateDetails, mrRequestContextForFeatureStore)

} yield {

candidateDetails

}

}

case \_ => Future.Nil

}

}

}