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

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.home\_mixer.product.scored\_tweets.feature\_hydrator.adapters.author\_features.AuthorFeaturesAdapter

import com.twitter.home\_mixer.util.CandidatesUtil

import com.twitter.home\_mixer.util.ObservedKeyValueResultHandler

import com.twitter.ml.api.DataRecord

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.FeatureWithDefaultOnFailure

import com.twitter.product\_mixer.core.feature.datarecord.DataRecordInAFeature

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.product\_mixer.core.util.OffloadFuturePools

import com.twitter.servo.repository.KeyValueRepository

import com.twitter.servo.repository.KeyValueResult

import com.twitter.stitch.Stitch

import com.twitter.timelines.author\_features.v1.{thriftjava => af}

import com.twitter.util.Future

import com.twitter.util.Try

import javax.inject.Inject

import javax.inject.Named

import javax.inject.Singleton

import scala.collection.JavaConverters.\_

object AuthorFeature

extends DataRecordInAFeature[TweetCandidate]

with FeatureWithDefaultOnFailure[TweetCandidate, DataRecord] {

override def defaultValue: DataRecord = new DataRecord()

}

@Singleton

class AuthorFeatureHydrator @Inject() (

@Named(AuthorFeatureRepository) client: KeyValueRepository[Seq[Long], Long, af.AuthorFeatures],

override val statsReceiver: StatsReceiver)

extends BulkCandidateFeatureHydrator[PipelineQuery, TweetCandidate]

with ObservedKeyValueResultHandler {

override val identifier: FeatureHydratorIdentifier =

FeatureHydratorIdentifier("AuthorFeature")

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

override val statScope: String = identifier.toString

override def apply(

query: PipelineQuery,

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Stitch[Seq[FeatureMap]] = OffloadFuturePools.offloadFuture {

val possiblyAuthorIds = extractKeys(candidates)

val authorIds = possiblyAuthorIds.flatten

val response: Future[KeyValueResult[Long, af.AuthorFeatures]] =

if (authorIds.nonEmpty) client(authorIds)

else Future.value(KeyValueResult.empty)

response.map { result =>

possiblyAuthorIds.map { possiblyAuthorId =>

val value = observedGet(key = possiblyAuthorId, keyValueResult = result)

val transformedValue = postTransformer(value)

FeatureMapBuilder().add(AuthorFeature, transformedValue).build()

}

}

}

private def postTransformer(authorFeatures: Try[Option[af.AuthorFeatures]]): Try[DataRecord] = {

authorFeatures.map {

\_.map { features => AuthorFeaturesAdapter.adaptToDataRecords(features).asScala.head }

.getOrElse(new DataRecord())

}

}

private def extractKeys(

candidates: Seq[CandidateWithFeatures[TweetCandidate]]

): Seq[Option[Long]] = {

candidates.map { candidate =>

CandidatesUtil.getOriginalAuthorId(candidate.features)

}

}

}