package com.twitter.frigate.pushservice.model.ibis

import com.twitter.frigate.common.rec\_types.RecTypes

import com.twitter.frigate.common.util.MRPushCopy

import com.twitter.frigate.common.util.MrPushCopyObjects

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

import com.twitter.frigate.pushservice.params.{PushFeatureSwitchParams => FS}

import com.twitter.ibis2.service.thriftscala.Flags

import com.twitter.ibis2.service.thriftscala.Ibis2Request

import com.twitter.ibis2.service.thriftscala.RecipientSelector

import com.twitter.ibis2.service.thriftscala.ResponseFlags

import com.twitter.util.Future

import scala.util.control.NoStackTrace

import com.twitter.ni.lib.logged\_out\_transform.Ibis2RequestTransform

class PushCopyIdNotFoundException(private val message: String)

extends Exception(message)

with NoStackTrace

class InvalidPushCopyIdException(private val message: String)

extends Exception(message)

with NoStackTrace

trait Ibis2HydratorForCandidate

extends CandidatePushCopy

with OverrideForIbis2Request

with CustomConfigurationMapForIbis {

self: PushCandidate =>

lazy val silentPushModelValue: Map[String, String] =

if (RecTypes.silentPushDefaultEnabledCrts.contains(commonRecType)) {

Map.empty

} else {

Map("is\_silent\_push" -> "true")

}

private def transformRelevanceScore(

mlScore: Double,

scoreRange: Seq[Double]

): Double = {

val (lowerBound, upperBound) = (scoreRange.head, scoreRange.last)

(mlScore \* (upperBound - lowerBound)) + lowerBound

}

private def getBoundedMlScore(mlScore: Double): Double = {

if (RecTypes.isMagicFanoutEventType(commonRecType)) {

val mfScoreRange = target.params(FS.MagicFanoutRelevanceScoreRange)

transformRelevanceScore(mlScore, mfScoreRange)

} else {

val mrScoreRange = target.params(FS.MagicRecsRelevanceScoreRange)

transformRelevanceScore(mlScore, mrScoreRange)

}

}

lazy val relevanceScoreMapFut: Future[Map[String, String]] = {

mrWeightedOpenOrNtabClickRankingProbability.map {

case Some(mlScore) if target.params(FS.IncludeRelevanceScoreInIbis2Payload) =>

val boundedMlScore = getBoundedMlScore(mlScore)

Map("relevance\_score" -> boundedMlScore.toString)

case \_ => Map.empty[String, String]

}

}

def customFieldsMapFut: Future[Map[String, String]] = relevanceScoreMapFut

//override is only enabled for RFPH CRT

def modelValues: Future[Map[String, String]] = {

Future.join(overrideModelValueFut, customConfigMapsFut).map {

case (overrideModelValue, customConfig) =>

overrideModelValue ++ silentPushModelValue ++ customConfig

}

}

def modelName: String = pushCopy.ibisPushModelName

def senderId: Option[Long] = None

def ibis2Request: Future[Option[Ibis2Request]] = {

Future.join(self.target.loggedOutMetadata, modelValues).map {

case (Some(metadata), modelVals) =>

Some(

Ibis2RequestTransform

.apply(metadata, modelName, modelVals).copy(

senderId = senderId,

flags = Some(Flags(

darkWrite = Some(target.isDarkWrite),

skipDupcheck = target.pushContext.flatMap(\_.useDebugHandler),

responseFlags = Some(ResponseFlags(stringTelemetry = Some(true)))

))

))

case (None, modelVals) =>

Some(

Ibis2Request(

recipientSelector = RecipientSelector(Some(target.targetId)),

modelName = modelName,

modelValues = Some(modelVals),

senderId = senderId,

flags = Some(

Flags(

darkWrite = Some(target.isDarkWrite),

skipDupcheck = target.pushContext.flatMap(\_.useDebugHandler),

responseFlags = Some(ResponseFlags(stringTelemetry = Some(true)))

)

)

))

}

}

}

trait CandidatePushCopy {

self: PushCandidate =>

final lazy val pushCopy: MRPushCopy =

pushCopyId match {

case Some(pushCopyId) =>

MrPushCopyObjects

.getCopyFromId(pushCopyId)

.getOrElse(

throw new InvalidPushCopyIdException(

s"Invalid push copy id: $pushCopyId for ${self.commonRecType}"))

case None =>

throw new PushCopyIdNotFoundException(

s"PushCopy not found in frigateNotification for ${self.commonRecType}"

)

}

}