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

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

import com.twitter.frigate.common.store.deviceinfo.DeviceInfo

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

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

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

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

import com.twitter.frigate.pushservice.predicate.ntab\_caret\_fatigue.ContinuousFunction

import com.twitter.frigate.pushservice.predicate.ntab\_caret\_fatigue.ContinuousFunctionParam

import com.twitter.frigate.pushservice.util.OverrideNotificationUtil

import com.twitter.frigate.pushservice.util.PushCapUtil

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

import com.twitter.frigate.thriftscala.CommonRecommendationType.MagicFanoutSportsEvent

import com.twitter.ibis2.lib.util.JsonMarshal

import com.twitter.util.Future

trait OverrideForIbis2Request {

self: PushCandidate =>

private lazy val overrideStats = self.statsReceiver.scope("override\_for\_ibis2")

private lazy val addedOverrideAndroidCounter =

overrideStats.scope("android").counter("added\_override\_for\_ibis2\_request")

private lazy val addedSmartPushConfigAndroidCounter =

overrideStats.scope("android").counter("added\_smart\_push\_config\_for\_ibis2\_request")

private lazy val addedOverrideIosCounter =

overrideStats.scope("ios").counter("added\_override\_for\_ibis2\_request")

private lazy val noOverrideCounter = overrideStats.counter("no\_override\_for\_ibis2\_request")

private lazy val noOverrideDueToDeviceInfoCounter =

overrideStats.counter("no\_override\_due\_to\_device\_info")

private lazy val addedMlScoreToPayloadAndroid =

overrideStats.scope("android").counter("added\_ml\_score")

private lazy val noMlScoreAddedToPayload =

overrideStats.counter("no\_ml\_score")

private lazy val addedNSlotsToPayload =

overrideStats.counter("added\_n\_slots")

private lazy val noNSlotsAddedToPayload =

overrideStats.counter("no\_n\_slots")

private lazy val addedCustomThreadIdToPayload =

overrideStats.counter("added\_custom\_thread\_id")

private lazy val noCustomThreadIdAddedToPayload =

overrideStats.counter("no\_custom\_thread\_id")

private lazy val enableTargetIdOverrideForMagicFanoutSportsEventCounter =

overrideStats.counter("enable\_target\_id\_override\_for\_mf\_sports\_event")

lazy val candidateModelScoreFut: Future[Option[Double]] = {

if (RecTypes.notEligibleForModelScoreTracking(commonRecType)) Future.None

else mrWeightedOpenOrNtabClickRankingProbability

}

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

if (self.target.isLoggedOutUser) {

Future.value(Map.empty[String, String])

} else {

Future

.join(

target.deviceInfo,

target.accountCountryCode,

OverrideNotificationUtil.getCollapseAndImpressionIdForOverride(self),

candidateModelScoreFut,

target.dynamicPushcap,

target.optoutAdjustedPushcap,

PushCapUtil.getDefaultPushCap(target)

).map {

case (

deviceInfoOpt,

countryCodeOpt,

Some((collapseId, impressionIds)),

mlScore,

dynamicPushcapOpt,

optoutAdjustedPushcapOpt,

defaultPushCap) =>

val pushCap: Int = (dynamicPushcapOpt, optoutAdjustedPushcapOpt) match {

case (\_, Some(optoutAdjustedPushcap)) => optoutAdjustedPushcap

case (Some(pushcapInfo), \_) => pushcapInfo.pushcap

case \_ => defaultPushCap

}

getClientSpecificOverrideModelValues(

target,

deviceInfoOpt,

countryCodeOpt,

collapseId,

impressionIds,

mlScore,

pushCap)

case \_ =>

noOverrideCounter.incr()

Map.empty[String, String]

}

}

}

/\*\*

\* Determines the appropriate Override Notification model values based on the client

\* @param target Target that will be receiving the push recommendation

\* @param deviceInfoOpt Target's Device Info

\* @param collapseId Collapse ID determined by OverrideNotificationUtil

\* @param impressionIds Impression IDs of previously sent Override Notifications

\* @param mlScore Open/NTab click ranking score of the current push candidate

\* @param pushCap Push cap for the target

\* @return Map consisting of the model values that need to be added to the Ibis2 Request

\*/

def getClientSpecificOverrideModelValues(

target: Target,

deviceInfoOpt: Option[DeviceInfo],

countryCodeOpt: Option[String],

collapseId: String,

impressionIds: Seq[String],

mlScoreOpt: Option[Double],

pushCap: Int

): Map[String, String] = {

val primaryDeviceIos = PushDeviceUtil.isPrimaryDeviceIOS(deviceInfoOpt)

val primaryDeviceAndroid = PushDeviceUtil.isPrimaryDeviceAndroid(deviceInfoOpt)

if (primaryDeviceIos ||

(primaryDeviceAndroid &&

target.params(FSParams.EnableOverrideNotificationsSmartPushConfigForAndroid))) {

if (primaryDeviceIos) addedOverrideIosCounter.incr()

else addedSmartPushConfigAndroidCounter.incr()

val impressionIdsSeq = {

if (target.params(FSParams.EnableTargetIdsInSmartPushPayload)) {

if (target.params(FSParams.EnableOverrideNotificationsMultipleTargetIds))

impressionIds

else Seq(impressionIds.head)

}

// Explicitly enable targetId-based override for MagicFanoutSportsEvent candidates (live sport update notifications)

else if (self.commonRecType == MagicFanoutSportsEvent && target.params(

FSParams.EnableTargetIdInSmartPushPayloadForMagicFanoutSportsEvent)) {

enableTargetIdOverrideForMagicFanoutSportsEventCounter.incr()

Seq(impressionIds.head)

} else Seq.empty[String]

}

val mlScoreMap = mlScoreOpt match {

case Some(mlScore)

if target.params(FSParams.EnableOverrideNotificationsScoreBasedOverride) =>

addedMlScoreToPayloadAndroid.incr()

Map("score" -> mlScore)

case \_ =>

noMlScoreAddedToPayload.incr()

Map.empty

}

val nSlotsMap = {

if (target.params(FSParams.EnableOverrideNotificationsNSlots)) {

if (target.params(FSParams.EnableOverrideMaxSlotFn)) {

val nslotFnParam = ContinuousFunctionParam(

target

.params(PushFeatureSwitchParams.OverrideMaxSlotFnPushCapKnobs),

target

.params(PushFeatureSwitchParams.OverrideMaxSlotFnNSlotKnobs),

target

.params(PushFeatureSwitchParams.OverrideMaxSlotFnPowerKnobs),

target

.params(PushFeatureSwitchParams.OverrideMaxSlotFnWeight),

target.params(FSParams.OverrideNotificationsMaxNumOfSlots)

)

val numOfSlots = ContinuousFunction.safeEvaluateFn(

pushCap,

nslotFnParam,

overrideStats.scope("max\_nslot\_fn"))

overrideStats.counter("max\_notification\_slots\_num\_" + numOfSlots.toString).incr()

addedNSlotsToPayload.incr()

Map("max\_notification\_slots" -> numOfSlots)

} else {

addedNSlotsToPayload.incr()

val numOfSlots = target.params(FSParams.OverrideNotificationsMaxNumOfSlots)

Map("max\_notification\_slots" -> numOfSlots)

}

} else {

noNSlotsAddedToPayload.incr()

Map.empty

}

}

val baseActionDetailsMap = Map("target\_ids" -> impressionIdsSeq)

val actionDetailsMap =

Map("action\_details" -> (baseActionDetailsMap ++ nSlotsMap))

val baseSmartPushConfigMap = Map("notification\_action" -> "REPLACE")

val customThreadId = {

if (target.params(FSParams.EnableCustomThreadIdForOverride)) {

addedCustomThreadIdToPayload.incr()

Map("custom\_thread\_id" -> impressionId)

} else {

noCustomThreadIdAddedToPayload.incr()

Map.empty

}

}

val smartPushConfigMap =

JsonMarshal.toJson(

baseSmartPushConfigMap ++ actionDetailsMap ++ mlScoreMap ++ customThreadId)

Map("smart\_notification\_configuration" -> smartPushConfigMap)

} else if (primaryDeviceAndroid) {

addedOverrideAndroidCounter.incr()

Map("notification\_id" -> collapseId, "overriding\_impression\_id" -> impressionIds.head)

} else {

noOverrideDueToDeviceInfoCounter.incr()

Map.empty[String, String]

}

}

}