package com.twitter.frigate.pushservice.util

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.MagicFanoutEventCandidate

import com.twitter.frigate.common.history.History

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.RawCandidate

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

import com.twitter.frigate.pushservice.model.ibis.PushOverrideInfo

import com.twitter.frigate.pushservice.params.PushConstants

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

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

import com.twitter.frigate.thriftscala.CollapseInfo

import com.twitter.frigate.thriftscala.CommonRecommendationType

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

import com.twitter.frigate.thriftscala.OverrideInfo

import com.twitter.util.Future

import java.util.UUID

object OverrideNotificationUtil {

/\*\*

\* Gets Override Info for the current notification.

\* @param candidate [[PushCandidate]] object representing the recommendation candidate

\* @param stats StatsReceiver to track stats for this function as well as the subsequent funcs. called

\* @return Returns OverrideInfo if CollapseInfo exists, else None

\*/

def getOverrideInfo(

candidate: PushCandidate,

stats: StatsReceiver

): Future[Option[OverrideInfo]] = {

if (candidate.target.isLoggedOutUser) {

Future.None

} else if (isOverrideEnabledForCandidate(candidate))

getCollapseInfo(candidate, stats).map(\_.map(OverrideInfo(\_)))

else Future.None

}

private def getCollapseInfo(

candidate: PushCandidate,

stats: StatsReceiver

): Future[Option[CollapseInfo]] = {

val target = candidate.target

for {

targetHistory <- target.history

deviceInfo <- target.deviceInfo

} yield getCollapseInfo(target, targetHistory, deviceInfo, stats)

}

/\*\*

\* Get Collapse Info for the current notification.

\* @param target Push Target - recipient of the notification

\* @param targetHistory Target's History

\* @param deviceInfoOpt `Option` of the Target's Device Info

\* @param stats StatsReceiver to track stats for this function as well as the subsequent funcs. called

\* @return Returns CollapseInfo if the Target is eligible for Override Notifs, else None

\*/

def getCollapseInfo(

target: PushTypes.Target,

targetHistory: History,

deviceInfoOpt: Option[DeviceInfo],

stats: StatsReceiver

): Option[CollapseInfo] = {

val overrideInfoOfLastNotif =

PushOverrideInfo.getOverrideInfoOfLastEligiblePushNotif(

targetHistory,

target.params(FSParams.OverrideNotificationsLookbackDurationForOverrideInfo),

stats)

overrideInfoOfLastNotif match {

case Some(prevOverrideInfo) if isOverrideEnabled(target, deviceInfoOpt, stats) =>

val notifsInLastOverrideChain =

PushOverrideInfo.getMrPushNotificationsInOverrideChain(

targetHistory,

prevOverrideInfo.collapseInfo.overrideChainId,

stats)

val numNotifsInLastOverrideChain = notifsInLastOverrideChain.size

val timestampOfFirstNotifInOverrideChain =

PushOverrideInfo

.getTimestampInMillisForFrigateNotification(

notifsInLastOverrideChain.last,

targetHistory,

stats).getOrElse(PushConstants.DefaultLookBackForHistory.ago.inMilliseconds)

if (numNotifsInLastOverrideChain < target.params(FSParams.MaxMrPushSends24HoursParam) &&

timestampOfFirstNotifInOverrideChain > PushConstants.DefaultLookBackForHistory.ago.inMilliseconds) {

Some(prevOverrideInfo.collapseInfo)

} else {

val prevCollapseId = prevOverrideInfo.collapseInfo.collapseId

val newOverrideChainId = UUID.randomUUID.toString.replaceAll("-", "")

Some(CollapseInfo(prevCollapseId, newOverrideChainId))

}

case None if isOverrideEnabled(target, deviceInfoOpt, stats) =>

val newOverrideChainId = UUID.randomUUID.toString.replaceAll("-", "")

Some(CollapseInfo("", newOverrideChainId))

case \_ => None // Override is disabled for everything else

}

}

/\*\*

\* Gets the collapse and impression identifier for the current override notification

\* @param target Push Target - recipient of the notification

\* @param stats StatsReceiver to track stats for this function as well as the subsequent funcs. called

\* @return A Future of Collapse ID as well as the Impression ID.

\*/

def getCollapseAndImpressionIdForOverride(

candidate: PushCandidate

): Future[Option[(String, Seq[String])]] = {

if (isOverrideEnabledForCandidate(candidate)) {

val target = candidate.target

val stats = candidate.statsReceiver

Future.join(target.history, target.deviceInfo).map {

case (targetHistory, deviceInfoOpt) =>

val collapseInfoOpt = getCollapseInfo(target, targetHistory, deviceInfoOpt, stats)

val impressionIds = candidate.commonRecType match {

case MagicFanoutSportsEvent

if target.params(FSParams.EnableEventIdBasedOverrideForSportsCandidates) =>

PushOverrideInfo.getImpressionIdsForPrevEligibleMagicFanoutEventCandidates(

targetHistory,

target.params(FSParams.OverrideNotificationsLookbackDurationForImpressionId),

stats,

MagicFanoutSportsEvent,

candidate

.asInstanceOf[RawCandidate with MagicFanoutEventCandidate].eventId

)

case \_ =>

PushOverrideInfo.getImpressionIdsOfPrevEligiblePushNotif(

targetHistory,

target.params(FSParams.OverrideNotificationsLookbackDurationForImpressionId),

stats)

}

collapseInfoOpt match {

case Some(collapseInfo) if impressionIds.nonEmpty =>

val notifsInLastOverrideChain =

PushOverrideInfo.getMrPushNotificationsInOverrideChain(

targetHistory,

collapseInfo.overrideChainId,

stats)

stats

.scope("OverrideNotificationUtil").stat("number\_of\_notifications\_sent").add(

notifsInLastOverrideChain.size + 1)

Some((collapseInfo.collapseId, impressionIds))

case \_ => None

}

case \_ => None

}

} else Future.None

}

/\*\*

\* Checks to see if override notifications are enabled based on the Target's Device Info and Params

\* @param target Push Target - recipient of the notification

\* @param deviceInfoOpt `Option` of the Target's Device Info

\* @param stats StatsReceiver to track stats for this function

\* @return Returns True if Override Notifications are enabled for the provided

\* Target, else False.

\*/

private def isOverrideEnabled(

target: PushTypes.Target,

deviceInfoOpt: Option[DeviceInfo],

stats: StatsReceiver

): Boolean = {

val scopedStats = stats.scope("OverrideNotificationUtil").scope("isOverrideEnabled")

val enabledForAndroidCounter = scopedStats.counter("android\_enabled")

val disabledForAndroidCounter = scopedStats.counter("android\_disabled")

val enabledForIosCounter = scopedStats.counter("ios\_enabled")

val disabledForIosCounter = scopedStats.counter("ios\_disabled")

val disabledForOtherDevicesCounter = scopedStats.counter("other\_disabled")

val isPrimaryDeviceAndroid = PushDeviceUtil.isPrimaryDeviceAndroid(deviceInfoOpt)

val isPrimaryDeviceIos = PushDeviceUtil.isPrimaryDeviceIOS(deviceInfoOpt)

lazy val validAndroidDevice =

isPrimaryDeviceAndroid && target.params(FSParams.EnableOverrideNotificationsForAndroid)

lazy val validIosDevice =

isPrimaryDeviceIos && target.params(FSParams.EnableOverrideNotificationsForIos)

if (isPrimaryDeviceAndroid) {

if (validAndroidDevice) enabledForAndroidCounter.incr() else disabledForAndroidCounter.incr()

} else if (isPrimaryDeviceIos) {

if (validIosDevice) enabledForIosCounter.incr() else disabledForIosCounter.incr()

} else {

disabledForOtherDevicesCounter.incr()

}

validAndroidDevice || validIosDevice

}

/\*\*

\* Checks if override is enabled for the currently supported types for SendHandler or not.

\* This method is package private for unit testing.

\* @param candidate [[PushCandidate]]

\* @param stats StatsReceiver to track statistics for this function

\* @return Returns True if override notifications are enabled for the current type, otherwise False.

\*/

private def isOverrideEnabledForSendHandlerCandidate(

candidate: PushCandidate

): Boolean = {

val scopedStats = candidate.statsReceiver

.scope("OverrideNotificationUtil").scope("isOverrideEnabledForSendHandlerType")

val overrideSupportedTypesForSpaces: Set[CommonRecommendationType] = Set(

CommonRecommendationType.SpaceSpeaker,

CommonRecommendationType.SpaceHost

)

val isOverrideSupportedForSpaces = {

overrideSupportedTypesForSpaces.contains(candidate.commonRecType) &&

candidate.target.params(FSParams.EnableOverrideForSpaces)

}

val isOverrideSupportedForSports = {

candidate.commonRecType == CommonRecommendationType.MagicFanoutSportsEvent &&

candidate.target

.params(PushFeatureSwitchParams.EnableOverrideForSportsCandidates)

}

val isOverrideSupported = isOverrideSupportedForSpaces || isOverrideSupportedForSports

scopedStats.counter(s"$isOverrideSupported").incr()

isOverrideSupported

}

private[util] def isOverrideEnabledForCandidate(candidate: PushCandidate) =

!RecTypes.isSendHandlerType(

candidate.commonRecType) || isOverrideEnabledForSendHandlerCandidate(candidate)

}