package com.twitter.frigate.pushservice.take.sender

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

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

import com.twitter.frigate.pushservice.take.NotificationServiceRequest

import com.twitter.frigate.thriftscala.FrigateNotification

import com.twitter.hermit.store.common.ReadableWritableStore

import com.twitter.notificationservice.api.thriftscala.DeleteCurrentTimelineForUserRequest

import com.twitter.notificationservice.thriftscala.CreateGenericNotificationResponse

import com.twitter.notificationservice.thriftscala.DeleteGenericNotificationRequest

import com.twitter.notificationservice.thriftscala.GenericNotificationKey

import com.twitter.notificationservice.thriftscala.GenericNotificationOverrideKey

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

object OverrideCandidate extends Enumeration {

val One: String = "overrideEntry1"

}

class NtabSender(

notificationServiceSender: ReadableStore[

NotificationServiceRequest,

CreateGenericNotificationResponse

],

nTabHistoryStore: ReadableWritableStore[(Long, String), GenericNotificationOverrideKey],

nTabDelete: DeleteGenericNotificationRequest => Future[Unit],

nTabDeleteTimeline: DeleteCurrentTimelineForUserRequest => Future[Unit]

)(

implicit statsReceiver: StatsReceiver) {

private[this] val nTabDeleteRequests = statsReceiver.counter("ntab\_delete\_request")

private[this] val nTabDeleteTimelineRequests =

statsReceiver.counter("ntab\_delete\_timeline\_request")

private[this] val ntabOverrideImpressionNotFound =

statsReceiver.counter("ntab\_impression\_not\_found")

private[this] val nTabOverrideOverriddenStat =

statsReceiver.counter("ntab\_override\_overridden")

private[this] val storeGenericNotifOverrideKey =

statsReceiver.counter("ntab\_store\_generic\_notif\_key")

private[this] val prevGenericNotifKeyNotFound =

statsReceiver.counter("ntab\_prev\_generic\_notif\_key\_not\_found")

private[this] val ntabOverride =

statsReceiver.scope("ntab\_override")

private[this] val ntabRequestWithOverrideId =

ntabOverride.counter("request")

private[this] val storeGenericNotifOverrideKeyWithOverrideId =

ntabOverride.counter("store\_override\_key")

def send(

candidate: PushCandidate,

isNtabOnlyNotification: Boolean

): Future[Option[CreateGenericNotificationResponse]] = {

if (candidate.target.params(FSParams.EnableOverrideIdNTabRequest)) {

ntabRequestWithOverrideId.incr()

overridePreviousEntry(candidate).flatMap { \_ =>

if (shouldDisableNtabOverride(candidate)) {

sendNewEntry(candidate, isNtabOnlyNotification, None)

} else {

sendNewEntry(candidate, isNtabOnlyNotification, Some(OverrideCandidate.One))

}

}

} else {

for {

notificationOverwritten <- overrideNSlot(candidate)

\_ <- deleteCachedApiTimeline(candidate, notificationOverwritten)

gnResponse <- sendNewEntry(candidate, isNtabOnlyNotification)

} yield gnResponse

}

}

private def sendNewEntry(

candidate: PushCandidate,

isNtabOnlyNotif: Boolean,

overrideId: Option[String] = None

): Future[Option[CreateGenericNotificationResponse]] = {

notificationServiceSender

.get(

NotificationServiceRequest(

candidate,

impressionId = candidate.impressionId,

isBadgeUpdate = isNtabOnlyNotif,

overrideId = overrideId

)).flatMap {

case Some(response) =>

storeGenericNotifKey(candidate, response, overrideId).map { \_ => Some(response) }

case \_ => Future.None

}

}

private def storeGenericNotifKey(

candidate: PushCandidate,

createGenericNotificationResponse: CreateGenericNotificationResponse,

overrideId: Option[String]

): Future[Unit] = {

if (candidate.target.params(FSParams.EnableStoringNtabGenericNotifKey)) {

createGenericNotificationResponse.successKey match {

case Some(genericNotificationKey) =>

val userId = genericNotificationKey.userId

if (overrideId.nonEmpty) {

storeGenericNotifOverrideKeyWithOverrideId.incr()

}

val gnOverrideKey = GenericNotificationOverrideKey(

userId = userId,

hashKey = genericNotificationKey.hashKey,

timestampMillis = genericNotificationKey.timestampMillis,

overrideId = overrideId

)

val mhKeyVal =

((userId, candidate.impressionId), gnOverrideKey)

storeGenericNotifOverrideKey.incr()

nTabHistoryStore.put(mhKeyVal)

case \_ => Future.Unit

}

} else Future.Unit

}

private def candidateEligibleForOverride(

targetHistory: History,

targetEntries: Seq[FrigateNotification],

): FrigateNotification = {

val timestampToEntriesMap =

targetEntries.map { entry =>

PushOverrideInfo

.getTimestampInMillisForFrigateNotification(entry, targetHistory, statsReceiver)

.getOrElse(PushConstants.DefaultLookBackForHistory.ago.inMilliseconds) -> entry

}.toMap

PushOverrideInfo.getOldestFrigateNotification(timestampToEntriesMap)

}

private def overrideNSlot(candidate: PushCandidate): Future[Boolean] = {

if (candidate.target.params(FSParams.EnableNslotsForOverrideOnNtab)) {

val targetHistoryFut = candidate.target.history

targetHistoryFut.flatMap { targetHistory =>

val nonEligibleOverrideTypes =

Seq(RecTypes.RecommendedSpaceFanoutTypes ++ RecTypes.ScheduledSpaceReminderTypes)

val overrideNotifs = PushOverrideInfo

.getOverrideEligiblePushNotifications(

targetHistory,

candidate.target.params(FSParams.OverrideNotificationsLookbackDurationForNTab),

statsReceiver

).filterNot {

case notification =>

nonEligibleOverrideTypes.contains(notification.commonRecommendationType)

}

val maxNumUnreadEntries =

candidate.target.params(FSParams.OverrideNotificationsMaxCountForNTab)

if (overrideNotifs.nonEmpty && overrideNotifs.size >= maxNumUnreadEntries) {

val eligibleOverrideCandidateOpt = candidateEligibleForOverride(

targetHistory,

overrideNotifs

)

eligibleOverrideCandidateOpt match {

case overrideCandidate if overrideCandidate.impressionId.nonEmpty =>

deleteNTabEntryFromGenericNotificationStore(

candidate.target.targetId,

eligibleOverrideCandidateOpt.impressionId.head)

case \_ =>

ntabOverrideImpressionNotFound.incr()

Future.False

}

} else Future.False

}

} else {

Future.False

}

}

private def shouldDisableNtabOverride(candidate: PushCandidate): Boolean =

RecTypes.isSendHandlerType(candidate.commonRecType)

private def overridePreviousEntry(candidate: PushCandidate): Future[Boolean] = {

if (shouldDisableNtabOverride(candidate)) {

nTabOverrideOverriddenStat.incr()

Future.False

} else {

val targetHistoryFut = candidate.target.history

targetHistoryFut.flatMap { targetHistory =>

val impressionIds = PushOverrideInfo.getImpressionIdsOfPrevEligiblePushNotif(

targetHistory,

candidate.target.params(FSParams.OverrideNotificationsLookbackDurationForImpressionId),

statsReceiver)

if (impressionIds.nonEmpty) {

deleteNTabEntryFromGenericNotificationStore(candidate.target.targetId, impressionIds.head)

} else {

ntabOverrideImpressionNotFound.incr()

Future.False // no deletes issued

}

}

}

}

private def deleteCachedApiTimeline(

candidate: PushCandidate,

isNotificationOverridden: Boolean

): Future[Unit] = {

if (isNotificationOverridden && candidate.target.params(FSParams.EnableDeletingNtabTimeline)) {

val deleteTimelineRequest = DeleteCurrentTimelineForUserRequest(candidate.target.targetId)

nTabDeleteTimelineRequests.incr()

nTabDeleteTimeline(deleteTimelineRequest)

} else {

Future.Unit

}

}

private def deleteNTabEntryFromGenericNotificationStore(

targetUserId: Long,

targetImpressionId: String

): Future[Boolean] = {

val mhKey = (targetUserId, targetImpressionId)

val genericNotificationKeyFut = nTabHistoryStore.get(mhKey)

genericNotificationKeyFut.flatMap {

case Some(genericNotifOverrideKey) =>

val gnKey = GenericNotificationKey(

userId = genericNotifOverrideKey.userId,

hashKey = genericNotifOverrideKey.hashKey,

timestampMillis = genericNotifOverrideKey.timestampMillis

)

val deleteEntryRequest = DeleteGenericNotificationRequest(gnKey)

nTabDeleteRequests.incr()

nTabDelete(deleteEntryRequest).map(\_ => true)

case \_ =>

prevGenericNotifKeyNotFound.incr()

Future.False

}

}

}