package com.twitter.frigate.pushservice.take

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.Invalid

import com.twitter.frigate.common.base.OK

import com.twitter.frigate.common.base.Response

import com.twitter.frigate.common.base.Result

import com.twitter.frigate.common.util.NotificationScribeUtil

import com.twitter.frigate.common.util.PushServiceUtil

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

import com.twitter.frigate.pushservice.thriftscala.PushResponse

import com.twitter.frigate.pushservice.thriftscala.PushStatus

import com.twitter.util.Future

class SendHandlerNotifier(

candidateNotifier: CandidateNotifier,

private val statsReceiver: StatsReceiver) {

val missingResponseCounter = statsReceiver.counter("missing\_response")

val filteredResponseCounter = statsReceiver.counter("filtered")

/\*\*

\*

\* @param isScribeInfoRequired: [[Boolean]] to indicate if scribe info is required

\* @param candidate: [[PushCandidate]] to build the scribe data from

\* @return: scribe response string

\*/

private def scribeInfoForResponse(

isScribeInfoRequired: Boolean,

candidate: PushCandidate

): Future[Option[String]] = {

if (isScribeInfoRequired) {

candidate.scribeData().map { scribedInfo =>

Some(NotificationScribeUtil.convertToJsonString(scribedInfo))

}

} else Future.None

}

/\*\*

\*

\* @param response: Candidate validation response

\* @param responseWithScribedInfo: boolean indicating if scribe data is expected in push response

\* @return: [[PushResponse]] containing final result of send request for [[com.twitter.frigate.pushservice.thriftscala.PushRequest]]

\*/

final def checkResponseAndNotify(

response: Response[PushCandidate, Result],

responseWithScribedInfo: Boolean

): Future[PushResponse] = {

response match {

case Response(OK, processedCandidates) =>

val (validCandidates, invalidCandidates) = processedCandidates.partition(\_.result == OK)

validCandidates.headOption match {

case Some(candidateResult) =>

val scribeInfo =

scribeInfoForResponse(responseWithScribedInfo, candidateResult.candidate)

scribeInfo.flatMap { scribedData =>

val response: Future[PushResponse] =

candidateNotifier.notify(candidateResult.candidate)

response.map(\_.copy(notifScribe = scribedData))

}

case None =>

invalidCandidates.headOption match {

case Some(candidateResult) =>

filteredResponseCounter.incr()

val response = candidateResult.result match {

case Invalid(reason) => PushResponse(PushStatus.Filtered, filteredBy = reason)

case \_ => PushResponse(PushStatus.Filtered, filteredBy = Some("unknown"))

}

val scribeInfo =

scribeInfoForResponse(responseWithScribedInfo, candidateResult.candidate)

scribeInfo.map(scribeData => response.copy(notifScribe = scribeData))

case None =>

missingResponseCounter.incr()

PushServiceUtil.FilteredPushResponseFut

}

}

case Response(Invalid(reason), \_) =>

throw new IllegalStateException(s"Unexpected target filtering in SendHandler: $reason")

}

}

}