package com.twitter.frigate.pushservice.take

import com.twitter.finagle.stats.BroadcastStatsReceiver

import com.twitter.finagle.stats.Counter

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.frigate.common.base.CandidateResult

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.base.Stats.track

import com.twitter.frigate.common.config.CommonConstants

import com.twitter.frigate.common.logger.MRLogger

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

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

import com.twitter.frigate.pushservice.refresh\_handler.RFPHStatsRecorder

import com.twitter.frigate.pushservice.thriftscala.LoggedOutResponse

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

import com.twitter.util.Future

import com.twitter.util.JavaTimer

import com.twitter.util.Timer

class LoggedOutRefreshForPushNotifier(

rfphStatsRecorder: RFPHStatsRecorder,

loCandidateNotifier: CandidateNotifier

)(

globalStats: StatsReceiver) {

private implicit val statsReceiver: StatsReceiver =

globalStats.scope("LoggedOutRefreshForPushHandler")

private val loPushStats: StatsReceiver = statsReceiver.scope("logged\_out\_push")

private val loSendLatency: StatsReceiver = statsReceiver.scope("logged\_out\_send")

private val processedCandidatesCounter: Counter =

statsReceiver.counter("processed\_candidates\_count")

private val validCandidatesCounter: Counter = statsReceiver.counter("valid\_candidates\_count")

private val okayCandidateCounter: Counter = statsReceiver.counter("ok\_candidate\_count")

private val nonOkayCandidateCounter: Counter = statsReceiver.counter("non\_ok\_candidate\_count")

private val successNotifyCounter: Counter = statsReceiver.counter("success\_notify\_count")

private val notifyCandidate: Counter = statsReceiver.counter("notify\_candidate")

private val noneCandidateResultCounter: Counter = statsReceiver.counter("none\_candidate\_count")

private val nonOkayPredsResult: Counter = statsReceiver.counter("non\_okay\_preds\_result")

private val invalidResultCounter: Counter = statsReceiver.counter("invalid\_result\_count")

private val filteredLoggedOutResponse: Counter = statsReceiver.counter("filtered\_response\_count")

implicit private val timer: Timer = new JavaTimer(true)

val log = MRLogger("LoggedOutRefreshForNotifier")

private def notify(

candidatesResult: CandidateResult[PushCandidate, Result]

): Future[LoggedOutResponse] = {

val candidate = candidatesResult.candidate

if (candidate != null)

notifyCandidate.incr()

val predsResult = candidatesResult.result

if (predsResult != OK) {

nonOkayPredsResult.incr()

val invalidResult = predsResult

invalidResult match {

case Invalid(Some(reason)) =>

invalidResultCounter.incr()

Future.value(LoggedOutResponse(PushStatus.Filtered, Some(reason)))

case \_ =>

filteredLoggedOutResponse.incr()

Future.value(LoggedOutResponse(PushStatus.Filtered, None))

}

} else {

track(loSendLatency)(loCandidateNotifier.loggedOutNotify(candidate).map { res =>

LoggedOutResponse(res.status)

})

}

}

def checkResponseAndNotify(

response: Response[PushCandidate, Result]

): Future[LoggedOutResponse] = {

val receivers = Seq(statsReceiver)

val loggedOutResponse = response match {

case Response(OK, processedCandidates) =>

processedCandidatesCounter.incr(processedCandidates.size)

val validCandidates = processedCandidates.filter(\_.result == OK)

validCandidatesCounter.incr(validCandidates.size)

validCandidates.headOption match {

case Some(candidatesResult) =>

candidatesResult.result match {

case OK =>

okayCandidateCounter.incr()

notify(candidatesResult)

.onSuccess { nr =>

successNotifyCounter.incr()

loPushStats.scope("lo\_result").counter(nr.status.name).incr()

}

case \_ =>

nonOkayCandidateCounter.incr()

FilteredLoggedOutResponseFut

}

case \_ =>

noneCandidateResultCounter.incr()

FilteredLoggedOutResponseFut

}

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

FilteredLoggedOutResponseFut.map(\_.copy(filteredBy = reason))

case \_ =>

FilteredLoggedOutResponseFut

}

val bstats = BroadcastStatsReceiver(receivers)

Stat

.timeFuture(bstats.stat("logged\_out\_latency"))(

loggedOutResponse.raiseWithin(CommonConstants.maxPushRequestDuration)

)

.onFailure { exception =>

rfphStatsRecorder.loggedOutRequestExceptionStats(exception, bstats)

}

loggedOutResponse

}

}