package com.twitter.timelineranker.server

import com.twitter.abdecider.LoggingABDecider

import com.twitter.finagle.TimeoutException

import com.twitter.finagle.stats.Stat

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.servo.util.FunctionArrow

import com.twitter.timelineranker.entity\_tweets.EntityTweetsRepository

import com.twitter.timelineranker.in\_network\_tweets.InNetworkTweetRepository

import com.twitter.timelineranker.model.\_

import com.twitter.timelineranker.observe.ObservedRequests

import com.twitter.timelineranker.recap\_author.RecapAuthorRepository

import com.twitter.timelineranker.recap\_hydration.RecapHydrationRepository

import com.twitter.timelineranker.repository.\_

import com.twitter.timelineranker.uteg\_liked\_by\_tweets.UtegLikedByTweetsRepository

import com.twitter.timelineranker.{thriftscala => thrift}

import com.twitter.timelines.authorization.TimelinesClientRequestAuthorizer

import com.twitter.timelines.observe.DebugObserver

import com.twitter.timelines.observe.ObservedAndValidatedRequests

import com.twitter.timelines.observe.QueryWidth

import com.twitter.timelines.observe.ServiceObserver

import com.twitter.util.Future

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

object TimelineRanker {

def toTimelineErrorThriftResponse(

ex: Throwable,

reason: Option[thrift.ErrorReason] = None

): thrift.GetTimelineResponse = {

thrift.GetTimelineResponse(

error = Some(thrift.TimelineError(message = ex.toString, reason))

)

}

def getTimelinesExceptionHandler: PartialFunction[

Throwable,

Future[thrift.GetTimelineResponse]

] = {

case e: TimeoutException =>

Future.value(toTimelineErrorThriftResponse(e, Some(thrift.ErrorReason.UpstreamTimeout)))

case e: Throwable if ObservedAndValidatedRequests.isOverCapacityException(e) =>

Future.value(toTimelineErrorThriftResponse(e, Some(thrift.ErrorReason.OverCapacity)))

case e => Future.value(toTimelineErrorThriftResponse(e))

}

def toErrorThriftResponse(

ex: Throwable,

reason: Option[thrift.ErrorReason] = None

): thrift.GetCandidateTweetsResponse = {

thrift.GetCandidateTweetsResponse(

error = Some(thrift.TimelineError(message = ex.toString, reason))

)

}

def exceptionHandler: PartialFunction[Throwable, Future[thrift.GetCandidateTweetsResponse]] = {

case e: TimeoutException =>

Future.value(toErrorThriftResponse(e, Some(thrift.ErrorReason.UpstreamTimeout)))

case e: Throwable if ObservedAndValidatedRequests.isOverCapacityException(e) =>

Future.value(toErrorThriftResponse(e, Some(thrift.ErrorReason.OverCapacity)))

case e => Future.value(toErrorThriftResponse(e))

}

}

class TimelineRanker(

routingRepository: RoutingTimelineRepository,

inNetworkTweetRepository: InNetworkTweetRepository,

recapHydrationRepository: RecapHydrationRepository,

recapAuthorRepository: RecapAuthorRepository,

entityTweetsRepository: EntityTweetsRepository,

utegLikedByTweetsRepository: UtegLikedByTweetsRepository,

serviceObserver: ServiceObserver,

val abdecider: Option[LoggingABDecider],

override val clientRequestAuthorizer: TimelinesClientRequestAuthorizer,

override val debugObserver: DebugObserver,

queryParamInitializer: FunctionArrow[RecapQuery, Future[RecapQuery]],

statsReceiver: StatsReceiver)

extends thrift.TimelineRanker.MethodPerEndpoint

with ObservedRequests {

override val requestWidthStats: Stat = statsReceiver.stat("TimelineRanker/requestWidth")

private[this] val getTimelinesStats = serviceObserver.readMethodStats(

"getTimelines",

QueryWidth.one[TimelineQuery]

)

private[this] val getInNetworkTweetCandidatesStats = serviceObserver.readMethodStats(

"getInNetworkTweetCandidates",

QueryWidth.one[RecapQuery]

)

private[this] val hydrateTweetCandidatesStats = serviceObserver.readMethodStats(

"hydrateTweetCandidates",

QueryWidth.one[RecapQuery]

)

private[this] val getRecapCandidatesFromAuthorsStats = serviceObserver.readMethodStats(

"getRecapCandidatesFromAuthors",

QueryWidth.one[RecapQuery]

)

private[this] val getEntityTweetCandidatesStats = serviceObserver.readMethodStats(

"getEntityTweetCandidates",

QueryWidth.one[RecapQuery]

)

private[this] val getUtegLikedByTweetCandidatesStats = serviceObserver.readMethodStats(

"getUtegLikedByTweetCandidates",

QueryWidth.one[RecapQuery]

)

def getTimelines(

thriftQueries: Seq[thrift.TimelineQuery]

): Future[Seq[thrift.GetTimelineResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(TimelineQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

getTimelinesStats,

TimelineRanker.getTimelinesExceptionHandler) { validatedQuery =>

routingRepository.get(validatedQuery).map { timeline =>

thrift.GetTimelineResponse(Some(timeline.toThrift))

}

}

case Throw(e) => Future.value(TimelineRanker.toTimelineErrorThriftResponse(e))

}

}

)

}

def getRecycledTweetCandidates(

thriftQueries: Seq[thrift.RecapQuery]

): Future[Seq[thrift.GetCandidateTweetsResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(RecapQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

getInNetworkTweetCandidatesStats,

TimelineRanker.exceptionHandler

) { validatedQuery =>

Future(queryParamInitializer(validatedQuery)).flatten.liftToTry.flatMap {

case Return(q) => inNetworkTweetRepository.get(q).map(\_.toThrift)

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

)

}

def hydrateTweetCandidates(

thriftQueries: Seq[thrift.RecapHydrationQuery]

): Future[Seq[thrift.GetCandidateTweetsResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(RecapQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

hydrateTweetCandidatesStats,

TimelineRanker.exceptionHandler

) { validatedQuery =>

Future(queryParamInitializer(validatedQuery)).flatten.liftToTry.flatMap {

case Return(q) => recapHydrationRepository.hydrate(q).map(\_.toThrift)

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

)

}

def getRecapCandidatesFromAuthors(

thriftQueries: Seq[thrift.RecapQuery]

): Future[Seq[thrift.GetCandidateTweetsResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(RecapQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

getRecapCandidatesFromAuthorsStats,

TimelineRanker.exceptionHandler

) { validatedQuery =>

Future(queryParamInitializer(validatedQuery)).flatten.liftToTry.flatMap {

case Return(q) => recapAuthorRepository.get(q).map(\_.toThrift)

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

)

}

def getEntityTweetCandidates(

thriftQueries: Seq[thrift.EntityTweetsQuery]

): Future[Seq[thrift.GetCandidateTweetsResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(RecapQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

getEntityTweetCandidatesStats,

TimelineRanker.exceptionHandler

) { validatedQuery =>

Future(queryParamInitializer(validatedQuery)).flatten.liftToTry.flatMap {

case Return(q) => entityTweetsRepository.get(q).map(\_.toThrift)

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

)

}

def getUtegLikedByTweetCandidates(

thriftQueries: Seq[thrift.UtegLikedByTweetsQuery]

): Future[Seq[thrift.GetCandidateTweetsResponse]] = {

Future.collect(

thriftQueries.map { thriftQuery =>

Try(RecapQuery.fromThrift(thriftQuery)) match {

case Return(query) =>

observeAndValidate(

query,

Seq(query.userId),

getUtegLikedByTweetCandidatesStats,

TimelineRanker.exceptionHandler

) { validatedQuery =>

Future(queryParamInitializer(validatedQuery)).flatten.liftToTry.flatMap {

case Return(q) => utegLikedByTweetsRepository.get(q).map(\_.toThrift)

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

case Throw(e) => Future.value(TimelineRanker.toErrorThriftResponse(e))

}

}

)

}

}