package com.twitter.tweetypie

package handler

import com.twitter.container.thriftscala.MaterializeAsTweetFieldsRequest

import com.twitter.context.TestingSignalsContext

import com.twitter.servo.util.FutureArrow

import com.twitter.spam.rtf.thriftscala.FilteredReason

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.DeletedTweetVisibilityRepository

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.TweetFieldsResultState

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* Handler for the `getTweetFields` endpoint.

\*/

object GetTweetFieldsHandler {

type Type = GetTweetFieldsRequest => Future[Seq[GetTweetFieldsResult]]

def apply(

tweetRepo: TweetResultRepository.Type,

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

containerAsGetTweetFieldsResultRepo: CreativesContainerMaterializationRepository.GetTweetFieldsType,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Type = {

FutureArrow[GetTweetFieldsRequest, Seq[GetTweetFieldsResult]] { request =>

val queryOptions = toTweetQueryOptions(request.options)

Stitch.run(

Stitch.traverse(request.tweetIds) { id =>

tweetRepo(id, queryOptions).liftToTry.flatMap { tweetResult =>

toGetTweetFieldsResult(

id,

tweetResult,

request.options,

deletedTweetVisibilityRepo,

containerAsGetTweetFieldsResultRepo,

stats,

shouldMaterializeContainers

)

}

}

)

}

}

/\*\*

\* Converts a `GetTweetFieldsOptions` into an internal `TweetQuery.Options`.

\*/

def toTweetQueryOptions(options: GetTweetFieldsOptions): TweetQuery.Options = {

val includes = options.tweetIncludes

val shouldSkipCache = TestingSignalsContext().flatMap(\_.simulateBackPressure).nonEmpty

val cacheControl =

if (shouldSkipCache) CacheControl.NoCache

else if (options.doNotCache) CacheControl.ReadOnlyCache

else CacheControl.ReadWriteCache

TweetQuery.Options(

include = TweetQuery

.Include(

tweetFields = includes.collect {

case TweetInclude.TweetFieldId(id) => id

case TweetInclude.CountsFieldId(\_) => Tweet.CountsField.id

case TweetInclude.MediaEntityFieldId(\_) => Tweet.MediaField.id

}.toSet,

countsFields = includes.collect { case TweetInclude.CountsFieldId(id) => id }.toSet,

mediaFields = includes.collect { case TweetInclude.MediaEntityFieldId(id) => id }.toSet,

quotedTweet = options.includeQuotedTweet,

pastedMedia = true

).also(

/\*\*

\* Always fetching underlying creatives container id. see

\* [[hydrateCreativeContainerBackedTweet]] for more detail.

\*/

tweetFields = Seq(Tweet.UnderlyingCreativesContainerIdField.id)

),

cacheControl = cacheControl,

enforceVisibilityFiltering = options.visibilityPolicy == TweetVisibilityPolicy.UserVisible,

safetyLevel = options.safetyLevel.getOrElse(SafetyLevel.FilterNone),

forUserId = options.forUserId,

languageTag = options.languageTag.getOrElse("en"),

cardsPlatformKey = options.cardsPlatformKey,

extensionsArgs = options.extensionsArgs,

forExternalConsumption = true,

simpleQuotedTweet = options.simpleQuotedTweet

)

}

def toGetTweetFieldsResult(

tweetId: TweetId,

res: Try[TweetResult],

options: GetTweetFieldsOptions,

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

containerAsGetTweetFieldsResultRepo: CreativesContainerMaterializationRepository.GetTweetFieldsType,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Stitch[GetTweetFieldsResult] = {

val measureRacyReads: TweetId => Unit = trackLossyReadsAfterWrite(

stats.stat("racy\_reads", "get\_tweet\_fields"),

Duration.fromSeconds(3)

)

res match {

case Throw(NotFound) =>

measureRacyReads(tweetId)

Stitch.value(GetTweetFieldsResult(tweetId, NotFoundResultState))

case Throw(ex) =>

val resultStateStitch = failureResultState(ex) match {

case notFoundResultState @ TweetFieldsResultState.NotFound(\_) =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

ex,

tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = false

)

).map(withVisibilityFilteredReason(notFoundResultState, \_))

case res => Stitch.value(res)

}

resultStateStitch.map(res => GetTweetFieldsResult(tweetId, res))

case Return(r) =>

toTweetFieldsResult(

r,

options,

deletedTweetVisibilityRepo,

containerAsGetTweetFieldsResultRepo,

stats,

shouldMaterializeContainers

).flatMap { getTweetFieldsResult =>

hydrateCreativeContainerBackedTweet(

r.value.tweet.underlyingCreativesContainerId,

getTweetFieldsResult,

options,

containerAsGetTweetFieldsResultRepo,

tweetId,

stats,

shouldMaterializeContainers

)

}

}

}

private def failureResultState(ex: Throwable): TweetFieldsResultState =

ex match {

case FilteredState.Unavailable.TweetDeleted => DeletedResultState

case FilteredState.Unavailable.BounceDeleted => BounceDeletedResultState

case FilteredState.Unavailable.SourceTweetNotFound(d) => notFoundResultState(deleted = d)

case FilteredState.Unavailable.Author.NotFound => NotFoundResultState

case fs: FilteredState.HasFilteredReason => toFilteredState(fs.filteredReason)

case OverCapacity(\_) => toFailedState(overcapacity = true, None)

case \_ => toFailedState(overcapacity = false, Some(ex.toString))

}

private val NotFoundResultState = TweetFieldsResultState.NotFound(TweetFieldsResultNotFound())

private val DeletedResultState = TweetFieldsResultState.NotFound(

TweetFieldsResultNotFound(deleted = true)

)

private val BounceDeletedResultState = TweetFieldsResultState.NotFound(

TweetFieldsResultNotFound(deleted = true, bounceDeleted = true)

)

def notFoundResultState(deleted: Boolean): TweetFieldsResultState.NotFound =

if (deleted) DeletedResultState else NotFoundResultState

private def toFailedState(

overcapacity: Boolean,

message: Option[String]

): TweetFieldsResultState =

TweetFieldsResultState.Failed(TweetFieldsResultFailed(overcapacity, message))

private def toFilteredState(reason: FilteredReason): TweetFieldsResultState =

TweetFieldsResultState.Filtered(

TweetFieldsResultFiltered(reason = reason)

)

/\*\*

\* Converts a `TweetResult` into a `GetTweetFieldsResult`. For retweets, missing or filtered source

\* tweets cause the retweet to be treated as missing or filtered.

\*/

private def toTweetFieldsResult(

tweetResult: TweetResult,

options: GetTweetFieldsOptions,

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

creativesContainerRepo: CreativesContainerMaterializationRepository.GetTweetFieldsType,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Stitch[GetTweetFieldsResult] = {

val primaryResultState = toTweetFieldsResultState(tweetResult, options)

val quotedResultStateStitch = primaryResultState match {

case TweetFieldsResultState.Found(\_) if options.includeQuotedTweet =>

val tweetData = tweetResult.value.sourceTweetResult

.getOrElse(tweetResult)

.value

tweetData.quotedTweetResult

.map {

case QuotedTweetResult.NotFound => Stitch.value(NotFoundResultState)

case QuotedTweetResult.Filtered(state) =>

val resultState = failureResultState(state)

(tweetData.tweet.quotedTweet, resultState) match {

//When QT exists => contribute VF filtered reason to result state

case (Some(qt), notFoundResultState @ TweetFieldsResultState.NotFound(\_)) =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

state,

qt.tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = true

)

).map(withVisibilityFilteredReason(notFoundResultState, \_))

//When QT is absent => result state without filtered reason

case \_ => Stitch.value(resultState)

}

case QuotedTweetResult.Found(res) =>

Stitch

.value(toTweetFieldsResultState(res, options))

.flatMap { resultState =>

hydrateCreativeContainerBackedTweet(

creativesContainerId = res.value.tweet.underlyingCreativesContainerId,

originalGetTweetFieldsResult = GetTweetFieldsResult(

tweetId = res.value.tweet.id,

tweetResult = resultState,

),

getTweetFieldsRequestOptions = options,

creativesContainerRepo = creativesContainerRepo,

res.value.tweet.id,

stats,

shouldMaterializeContainers

)

}

.map(\_.tweetResult)

}

//Quoted tweet result not requested

case \_ => None

}

quotedResultStateStitch

.map(qtStitch => qtStitch.map(Some(\_)))

.getOrElse(Stitch.None)

.map(qtResult =>

GetTweetFieldsResult(

tweetId = tweetResult.value.tweet.id,

tweetResult = primaryResultState,

quotedTweetResult = qtResult

))

}

/\*\*

\* @return a copy of resultState with filtered reason when @param filteredReasonOpt is present

\*/

private def withVisibilityFilteredReason(

resultState: TweetFieldsResultState.NotFound,

filteredReasonOpt: Option[FilteredReason]

): TweetFieldsResultState.NotFound = {

filteredReasonOpt match {

case Some(fs) =>

resultState.copy(

notFound = resultState.notFound.copy(

filteredReason = Some(fs)

))

case \_ => resultState

}

}

private def toTweetFieldsResultState(

tweetResult: TweetResult,

options: GetTweetFieldsOptions

): TweetFieldsResultState = {

val tweetData = tweetResult.value

val suppressReason = tweetData.suppress.map(\_.filteredReason)

val tweetFailedFields = tweetResult.state.failedFields

val sourceTweetFailedFields =

tweetData.sourceTweetResult.map(\_.state.failedFields).getOrElse(Set())

val sourceTweetOpt = tweetData.sourceTweetResult.map(\_.value.tweet)

val sourceTweetSuppressReason =

tweetData.sourceTweetResult.flatMap(\_.value.suppress.map(\_.filteredReason))

val isTweetPartial = tweetFailedFields.nonEmpty || sourceTweetFailedFields.nonEmpty

val tweetFoundResult = tweetData.sourceTweetResult match {

case None =>

// if `sourceTweetResult` is empty, this isn't a retweet

TweetFieldsResultFound(

tweet = tweetData.tweet,

suppressReason = suppressReason

)

case Some(r) =>

// if the source tweet result state is Found, merge that into the primary result

TweetFieldsResultFound(

tweet = tweetData.tweet,

retweetedTweet = sourceTweetOpt.filter(\_ => options.includeRetweetedTweet),

suppressReason = suppressReason.orElse(sourceTweetSuppressReason)

)

}

if (isTweetPartial) {

TweetFieldsResultState.Failed(

TweetFieldsResultFailed(

overCapacity = false,

message = Some(

"Failed to load: " + (tweetFailedFields ++ sourceTweetFailedFields).mkString(", ")),

partial = Some(

TweetFieldsPartial(

found = tweetFoundResult,

missingFields = tweetFailedFields,

sourceTweetMissingFields = sourceTweetFailedFields

)

)

)

)

} else {

TweetFieldsResultState.Found(

tweetFoundResult

)

}

}

/\*\*

\* if tweet data is backed by creatives container, it'll be hydrated from creatives

\* container service.

\*/

private def hydrateCreativeContainerBackedTweet(

creativesContainerId: Option[Long],

originalGetTweetFieldsResult: GetTweetFieldsResult,

getTweetFieldsRequestOptions: GetTweetFieldsOptions,

creativesContainerRepo: CreativesContainerMaterializationRepository.GetTweetFieldsType,

tweetId: Long,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Stitch[GetTweetFieldsResult] = {

// creatives container backed tweet stats

val ccTweetMaterialized = stats.scope("creatives\_container", "get\_tweet\_fields")

val ccTweetMaterializeRequests = ccTweetMaterialized.counter("requests")

val ccTweetMaterializeSuccess = ccTweetMaterialized.counter("success")

val ccTweetMaterializeFailed = ccTweetMaterialized.counter("failed")

val ccTweetMaterializeFiltered = ccTweetMaterialized.scope("filtered")

(

creativesContainerId,

originalGetTweetFieldsResult.tweetResult,

getTweetFieldsRequestOptions.disableTweetMaterialization,

shouldMaterializeContainers()

) match {

// 1. creatives container backed tweet is determined by `underlyingCreativesContainerId` field presence.

// 2. if the frontend tweet is suppressed by any reason, respect that and not do this hydration.

// (this logic can be revisited and improved further)

case (None, \_, \_, \_) =>

Stitch.value(originalGetTweetFieldsResult)

case (Some(\_), \_, \_, false) =>

ccTweetMaterializeFiltered.counter("decider\_suppressed").incr()

Stitch.value {

GetTweetFieldsResult(

tweetId = tweetId,

tweetResult = TweetFieldsResultState.NotFound(TweetFieldsResultNotFound())

)

}

case (Some(containerId), TweetFieldsResultState.Found(\_), false, \_) =>

ccTweetMaterializeRequests.incr()

val materializationRequest =

MaterializeAsTweetFieldsRequest(containerId, tweetId, Some(originalGetTweetFieldsResult))

creativesContainerRepo(

materializationRequest,

getTweetFieldsRequestOptions

).onSuccess(\_ => ccTweetMaterializeSuccess.incr())

.onFailure(\_ => ccTweetMaterializeFailed.incr())

.handle {

case ex =>

GetTweetFieldsResult(

tweetId = tweetId,

tweetResult = failureResultState(ex)

)

}

case (Some(\_), \_, true, \_) =>

ccTweetMaterializeFiltered.counter("suppressed").incr()

Stitch.value(

GetTweetFieldsResult(

tweetId = tweetId,

tweetResult = TweetFieldsResultState.NotFound(TweetFieldsResultNotFound())

)

)

case (Some(\_), state, \_, \_) =>

ccTweetMaterializeFiltered.counter(state.getClass.getName).incr()

Stitch.value(originalGetTweetFieldsResult)

}

}

}