package com.twitter.tweetypie

package handler

import com.twitter.container.thriftscala.MaterializeAsTweetRequest

import com.twitter.context.TestingSignalsContext

import com.twitter.servo.exception.thriftscala.ClientError

import com.twitter.servo.exception.thriftscala.ClientErrorCause

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.additionalfields.AdditionalFields

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* Handler for the `getTweets` endpoint.

\*/

object GetTweetsHandler {

type Type = FutureArrow[GetTweetsRequest, Seq[GetTweetResult]]

/\*\*

\* A `TweetQuery.Include` instance with options set as the default base options

\* for the `getTweets` endpoint.

\*/

val BaseInclude: TweetQuery.Include =

TweetQuery.Include(

tweetFields = Set(

Tweet.CoreDataField.id,

Tweet.UrlsField.id,

Tweet.MentionsField.id,

Tweet.MediaField.id,

Tweet.HashtagsField.id,

Tweet.CashtagsField.id,

Tweet.TakedownCountryCodesField.id,

Tweet.TakedownReasonsField.id,

Tweet.DeviceSourceField.id,

Tweet.LanguageField.id,

Tweet.ContributorField.id,

Tweet.QuotedTweetField.id,

Tweet.UnderlyingCreativesContainerIdField.id,

),

pastedMedia = true

)

def apply(

tweetRepo: TweetResultRepository.Type,

creativesContainerRepo: CreativesContainerMaterializationRepository.GetTweetType,

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Type = {

FutureArrow[GetTweetsRequest, Seq[GetTweetResult]] { request =>

val requestOptions = request.options.getOrElse(GetTweetOptions())

val invalidAdditionalFields =

requestOptions.additionalFieldIds.filter(!AdditionalFields.isAdditionalFieldId(\_))

if (invalidAdditionalFields.nonEmpty) {

Future.exception(

ClientError(

ClientErrorCause.BadRequest,

"Requested additional fields contain invalid field id " +

s"${invalidAdditionalFields.mkString(", ")}. Additional fields ids must be greater than 100."

)

)

} else {

val opts = toTweetQueryOptions(requestOptions)

val measureRacyReads: TweetId => Unit = trackLossyReadsAfterWrite(

stats.stat("racy\_reads", "get\_tweets"),

Duration.fromSeconds(3)

)

Stitch.run(

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

tweetRepo(id, opts).liftToTry

.flatMap {

case Throw(NotFound) =>

measureRacyReads(id)

Stitch.value(GetTweetResult(id, StatusState.NotFound))

case Throw(ex) =>

failureResult(deletedTweetVisibilityRepo, id, requestOptions, ex)

case Return(r) =>

toGetTweetResult(

deletedTweetVisibilityRepo,

creativesContainerRepo,

requestOptions,

tweetResult = r,

includeSourceTweet = requestOptions.includeSourceTweet,

includeQuotedTweet = requestOptions.includeQuotedTweet,

stats,

shouldMaterializeContainers

)

}.flatMap { getTweetResult =>

// check if tweet data is backed by creatives container and needs to be hydrated from creatives

// container service.

hydrateCreativeContainerBackedTweet(

getTweetResult,

requestOptions,

creativesContainerRepo,

stats,

shouldMaterializeContainers

)

}

}

)

}

}

}

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

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

val cacheControl =

if (shouldSkipCache) CacheControl.NoCache

else if (options.doNotCache) CacheControl.ReadOnlyCache

else CacheControl.ReadWriteCache

val countsFields = toCountsFields(options)

val mediaFields = toMediaFields(options)

TweetQuery.Options(

include = BaseInclude.also(

tweetFields = toTweetFields(options, countsFields),

countsFields = countsFields,

mediaFields = mediaFields,

quotedTweet = Some(options.includeQuotedTweet)

),

cacheControl = cacheControl,

cardsPlatformKey = options.cardsPlatformKey,

excludeReported = options.excludeReported,

enforceVisibilityFiltering = !options.bypassVisibilityFiltering,

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

forUserId = options.forUserId,

languageTag = options.languageTag,

extensionsArgs = options.extensionsArgs,

forExternalConsumption = true,

simpleQuotedTweet = options.simpleQuotedTweet

)

}

private def toTweetFields(opts: GetTweetOptions, countsFields: Set[FieldId]): Set[FieldId] = {

val bldr = Set.newBuilder[FieldId]

bldr ++= opts.additionalFieldIds

if (opts.includePlaces) bldr += Tweet.PlaceField.id

if (opts.forUserId.nonEmpty) {

if (opts.includePerspectivals) bldr += Tweet.PerspectiveField.id

if (opts.includeConversationMuted) bldr += Tweet.ConversationMutedField.id

}

if (opts.includeCards && opts.cardsPlatformKey.isEmpty) bldr += Tweet.CardsField.id

if (opts.includeCards && opts.cardsPlatformKey.nonEmpty) bldr += Tweet.Card2Field.id

if (opts.includeProfileGeoEnrichment) bldr += Tweet.ProfileGeoEnrichmentField.id

if (countsFields.nonEmpty) bldr += Tweet.CountsField.id

if (opts.includeCardUri) bldr += Tweet.CardReferenceField.id

bldr.result()

}

private def toCountsFields(opts: GetTweetOptions): Set[FieldId] = {

val bldr = Set.newBuilder[FieldId]

if (opts.includeRetweetCount) bldr += StatusCounts.RetweetCountField.id

if (opts.includeReplyCount) bldr += StatusCounts.ReplyCountField.id

if (opts.includeFavoriteCount) bldr += StatusCounts.FavoriteCountField.id

if (opts.includeQuoteCount) bldr += StatusCounts.QuoteCountField.id

bldr.result()

}

private def toMediaFields(opts: GetTweetOptions): Set[FieldId] = {

if (opts.includeMediaAdditionalMetadata)

Set(MediaEntity.AdditionalMetadataField.id)

else

Set.empty

}

/\*\*

\* Converts a `TweetResult` into a `GetTweetResult`.

\*/

def toGetTweetResult(

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

creativesContainerRepo: CreativesContainerMaterializationRepository.GetTweetType,

options: GetTweetOptions,

tweetResult: TweetResult,

includeSourceTweet: Boolean,

includeQuotedTweet: Boolean,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Stitch[GetTweetResult] = {

val tweetData = tweetResult.value

// only include missing fields if non empty

def asMissingFields(set: Set[FieldByPath]): Option[Set[FieldByPath]] =

if (set.isEmpty) None else Some(set)

val missingFields = asMissingFields(tweetResult.state.failedFields)

val sourceTweetResult =

tweetData.sourceTweetResult

.filter(\_ => includeSourceTweet)

val sourceTweetData = tweetData.sourceTweetResult

.getOrElse(tweetResult)

.value

val quotedTweetResult: Option[QuotedTweetResult] = sourceTweetData.quotedTweetResult

.filter(\_ => includeQuotedTweet)

val qtFilteredReasonStitch =

((sourceTweetData.tweet.quotedTweet, quotedTweetResult) match {

case (Some(quotedTweet), Some(QuotedTweetResult.Filtered(filteredState))) =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

filteredState,

quotedTweet.tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = true

)

)

case \_ => Stitch.None

})

//Use quotedTweetResult filtered reason when VF filtered reason is not present

.map(fsOpt => fsOpt.orElse(quotedTweetResult.flatMap(\_.filteredReason)))

val suppress = tweetData.suppress.orElse(tweetData.sourceTweetResult.flatMap(\_.value.suppress))

val quotedTweetStitch: Stitch[Option[Tweet]] =

quotedTweetResult match {

// check if quote tweet is backed by creatives container and needs to be hydrated from creatives

// container service. detail see go/creatives-containers-tdd

case Some(QuotedTweetResult.Found(tweetResult)) =>

hydrateCreativeContainerBackedTweet(

originalGetTweetResult = GetTweetResult(

tweetId = tweetResult.value.tweet.id,

tweetState = StatusState.Found,

tweet = Some(tweetResult.value.tweet)

),

getTweetRequestOptions = options,

creativesContainerRepo = creativesContainerRepo,

stats = stats,

shouldMaterializeContainers

).map(\_.tweet)

case \_ =>

Stitch.value(

quotedTweetResult

.flatMap(\_.toOption)

.map(\_.value.tweet)

)

}

Stitch.join(qtFilteredReasonStitch, quotedTweetStitch).map {

case (qtFilteredReason, quotedTweet) =>

GetTweetResult(

tweetId = tweetData.tweet.id,

tweetState =

if (suppress.nonEmpty) StatusState.Suppress

else if (missingFields.nonEmpty) StatusState.Partial

else StatusState.Found,

tweet = Some(tweetData.tweet),

missingFields = missingFields,

filteredReason = suppress.map(\_.filteredReason),

sourceTweet = sourceTweetResult.map(\_.value.tweet),

sourceTweetMissingFields = sourceTweetResult

.map(\_.state.failedFields)

.flatMap(asMissingFields),

quotedTweet = quotedTweet,

quotedTweetMissingFields = quotedTweetResult

.flatMap(\_.toOption)

.map(\_.state.failedFields)

.flatMap(asMissingFields),

quotedTweetFilteredReason = qtFilteredReason

)

}

}

private[this] val AuthorAccountIsInactive = FilteredReason.AuthorAccountIsInactive(true)

def failureResult(

deletedTweetVisibilityRepo: DeletedTweetVisibilityRepository.Type,

tweetId: TweetId,

options: GetTweetOptions,

ex: Throwable

): Stitch[GetTweetResult] = {

def deletedState(deleted: Boolean, statusState: StatusState) =

if (deleted && options.enableDeletedState) {

statusState

} else {

StatusState.NotFound

}

ex match {

case FilteredState.Unavailable.Author.Deactivated =>

Stitch.value(GetTweetResult(tweetId, StatusState.DeactivatedUser))

case FilteredState.Unavailable.Author.NotFound =>

Stitch.value(GetTweetResult(tweetId, StatusState.NotFound))

case FilteredState.Unavailable.Author.Offboarded =>

Stitch.value(

GetTweetResult(tweetId, StatusState.Drop, filteredReason = Some(AuthorAccountIsInactive)))

case FilteredState.Unavailable.Author.Suspended =>

Stitch.value(GetTweetResult(tweetId, StatusState.SuspendedUser))

case FilteredState.Unavailable.Author.Protected =>

Stitch.value(GetTweetResult(tweetId, StatusState.ProtectedUser))

case FilteredState.Unavailable.Author.Unsafe =>

Stitch.value(GetTweetResult(tweetId, StatusState.Drop))

//Handle delete state with optional FilteredReason

case FilteredState.Unavailable.TweetDeleted =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

ex,

tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = false

)

).map(filteredReasonOpt => {

val deleteState = deletedState(deleted = true, StatusState.Deleted)

GetTweetResult(tweetId, deleteState, filteredReason = filteredReasonOpt)

})

case FilteredState.Unavailable.BounceDeleted =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

ex,

tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = false

)

).map(filteredReasonOpt => {

val deleteState = deletedState(deleted = true, StatusState.BounceDeleted)

GetTweetResult(tweetId, deleteState, filteredReason = filteredReasonOpt)

})

case FilteredState.Unavailable.SourceTweetNotFound(d) =>

deletedTweetVisibilityRepo(

DeletedTweetVisibilityRepository.VisibilityRequest(

ex,

tweetId,

options.safetyLevel,

options.forUserId,

isInnerQuotedTweet = false

)

).map(filteredReasonOpt => {

val deleteState = deletedState(d, StatusState.Deleted)

GetTweetResult(tweetId, deleteState, filteredReason = filteredReasonOpt)

})

case FilteredState.Unavailable.Reported =>

Stitch.value(GetTweetResult(tweetId, StatusState.ReportedTweet))

case fs: FilteredState.HasFilteredReason =>

Stitch.value(

GetTweetResult(tweetId, StatusState.Drop, filteredReason = Some(fs.filteredReason)))

case OverCapacity(\_) => Stitch.value(GetTweetResult(tweetId, StatusState.OverCapacity))

case \_ => Stitch.value(GetTweetResult(tweetId, StatusState.Failed))

}

}

private def hydrateCreativeContainerBackedTweet(

originalGetTweetResult: GetTweetResult,

getTweetRequestOptions: GetTweetOptions,

creativesContainerRepo: CreativesContainerMaterializationRepository.GetTweetType,

stats: StatsReceiver,

shouldMaterializeContainers: Gate[Unit]

): Stitch[GetTweetResult] = {

// creatives container backed tweet stats

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

val ccTweetMaterializeFiltered = ccTweetMaterialized.scope("filtered")

val ccTweetMaterializeSuccess = ccTweetMaterialized.counter("success")

val ccTweetMaterializeFailed = ccTweetMaterialized.counter("failed")

val ccTweetMaterializeRequests = ccTweetMaterialized.counter("requests")

val tweetId = originalGetTweetResult.tweetId

val tweetState = originalGetTweetResult.tweetState

val underlyingCreativesContainerId =

originalGetTweetResult.tweet.flatMap(\_.underlyingCreativesContainerId)

(

tweetState,

underlyingCreativesContainerId,

getTweetRequestOptions.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(originalGetTweetResult)

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

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

Stitch.value(GetTweetResult(tweetId, StatusState.NotFound))

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

ccTweetMaterializeRequests.incr()

val materializationRequest =

MaterializeAsTweetRequest(containerId, tweetId, Some(originalGetTweetResult))

creativesContainerRepo(

materializationRequest,

Some(getTweetRequestOptions)

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

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

.handle {

case \_ => GetTweetResult(tweetId, StatusState.Failed)

}

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

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

Stitch.value(GetTweetResult(tweetId, StatusState.NotFound))

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

ccTweetMaterializeFiltered.counter(state.name).incr()

Stitch.value(originalGetTweetResult)

}

}

}