/\*\* Copyright 2010 Twitter, Inc. \*/

package com.twitter.tweetypie

package service

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

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

import com.twitter.tweetypie.additionalfields.AdditionalFields

import com.twitter.tweetypie.client\_id.ClientIdHelper

import com.twitter.tweetypie.handler.\_

import com.twitter.tweetypie.store.\_

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.util.Future

/\*\*

\* Implementation of the TweetService which dispatches requests to underlying

\* handlers and stores.

\*/

class DispatchingTweetService(

asyncDeleteAdditionalFieldsBuilder: AsyncDeleteAdditionalFieldsBuilder.Type,

asyncSetAdditionalFieldsBuilder: AsyncSetAdditionalFieldsBuilder.Type,

deleteAdditionalFieldsBuilder: DeleteAdditionalFieldsBuilder.Type,

deleteLocationDataHandler: DeleteLocationDataHandler.Type,

deletePathHandler: TweetDeletePathHandler,

eraseUserTweetsHandler: EraseUserTweetsHandler,

getDeletedTweetsHandler: GetDeletedTweetsHandler.Type,

getStoredTweetsHandler: GetStoredTweetsHandler.Type,

getStoredTweetsByUserHandler: GetStoredTweetsByUserHandler.Type,

getTweetCountsHandler: GetTweetCountsHandler.Type,

getTweetsHandler: GetTweetsHandler.Type,

getTweetFieldsHandler: GetTweetFieldsHandler.Type,

postTweetHandler: PostTweet.Type[PostTweetRequest],

postRetweetHandler: PostTweet.Type[RetweetRequest],

quotedTweetDeleteBuilder: QuotedTweetDeleteEventBuilder.Type,

quotedTweetTakedownBuilder: QuotedTweetTakedownEventBuilder.Type,

scrubGeoScrubTweetsBuilder: ScrubGeoEventBuilder.ScrubTweets.Type,

scrubGeoUpdateUserTimestampBuilder: ScrubGeoEventBuilder.UpdateUserTimestamp.Type,

setAdditionalFieldsBuilder: SetAdditionalFieldsBuilder.Type,

setRetweetVisibilityHandler: SetRetweetVisibilityHandler.Type,

statsReceiver: StatsReceiver,

takedownHandler: TakedownHandler.Type,

tweetStore: TotalTweetStore,

undeleteTweetHandler: UndeleteTweetHandler.Type,

unretweetHandler: UnretweetHandler.Type,

updatePossiblySensitiveTweetHandler: UpdatePossiblySensitiveTweetHandler.Type,

userTakedownHandler: UserTakedownHandler.Type,

clientIdHelper: ClientIdHelper)

extends ThriftTweetService {

import AdditionalFields.\_

// Incoming reads

override def getTweets(request: GetTweetsRequest): Future[Seq[GetTweetResult]] =

getTweetsHandler(request)

override def getTweetFields(request: GetTweetFieldsRequest): Future[Seq[GetTweetFieldsResult]] =

getTweetFieldsHandler(request)

override def getTweetCounts(request: GetTweetCountsRequest): Future[Seq[GetTweetCountsResult]] =

getTweetCountsHandler(request)

// Incoming deletes

override def cascadedDeleteTweet(request: CascadedDeleteTweetRequest): Future[Unit] =

deletePathHandler.cascadedDeleteTweet(request)

override def deleteTweets(request: DeleteTweetsRequest): Future[Seq[DeleteTweetResult]] =

deletePathHandler.deleteTweets(request)

// Incoming writes

override def postTweet(request: PostTweetRequest): Future[PostTweetResult] =

postTweetHandler(request)

override def postRetweet(request: RetweetRequest): Future[PostTweetResult] =

postRetweetHandler(request)

override def setAdditionalFields(request: SetAdditionalFieldsRequest): Future[Unit] = {

val setFields = AdditionalFields.nonEmptyAdditionalFieldIds(request.additionalFields)

if (setFields.isEmpty) {

Future.exception(

ClientError(

ClientErrorCause.BadRequest,

s"${SetAdditionalFieldsRequest.AdditionalFieldsField.name} is empty, there must be at least one field to set"

)

)

} else {

unsettableAdditionalFieldIds(request.additionalFields) match {

case Nil =>

setAdditionalFieldsBuilder(request).flatMap(tweetStore.setAdditionalFields)

case unsettableFieldIds =>

Future.exception(

ClientError(

ClientErrorCause.BadRequest,

unsettableAdditionalFieldIdsErrorMessage(unsettableFieldIds)

)

)

}

}

}

override def deleteAdditionalFields(request: DeleteAdditionalFieldsRequest): Future[Unit] =

if (request.tweetIds.isEmpty || request.fieldIds.isEmpty) {

Future.exception(

ClientError(ClientErrorCause.BadRequest, "request contains empty tweet ids or field ids")

)

} else if (request.fieldIds.exists(!isAdditionalFieldId(\_))) {

Future.exception(

ClientError(ClientErrorCause.BadRequest, "cannot delete non-additional fields")

)

} else {

deleteAdditionalFieldsBuilder(request).flatMap { events =>

Future.join(events.map(tweetStore.deleteAdditionalFields))

}

}

override def asyncInsert(request: AsyncInsertRequest): Future[Unit] =

AsyncInsertTweet.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncInsertTweet(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncInsertTweet(e)

}

override def asyncSetAdditionalFields(request: AsyncSetAdditionalFieldsRequest): Future[Unit] =

asyncSetAdditionalFieldsBuilder(request).map {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncSetAdditionalFields(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncSetAdditionalFields(e)

}

/\*\*

\* Set if a retweet should be included in its source tweet's retweet count.

\*

\* This is called by our RetweetVisibility daemon when a user enter/exit

\* suspended or read-only state and all their retweets visibility need to

\* be modified.

\*

\* @see [[SetRetweetVisibilityHandler]] for more implementation details

\*/

override def setRetweetVisibility(request: SetRetweetVisibilityRequest): Future[Unit] =

setRetweetVisibilityHandler(request)

override def asyncSetRetweetVisibility(request: AsyncSetRetweetVisibilityRequest): Future[Unit] =

AsyncSetRetweetVisibility.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncSetRetweetVisibility(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncSetRetweetVisibility(e)

}

/\*\*

\* When a tweet has been successfully undeleted from storage in Manhattan this endpoint will

\* enqueue requests to three related endpoints via deferredRPC:

\*

\* 1. asyncUndeleteTweet: Asynchronously handle aspects of the undelete not required for the response.

\* 2. replicatedUndeleteTweet2: Send the undeleted tweet to other clusters for cache caching.

\*

\* @see [[UndeleteTweetHandler]] for the core undelete implementation

\*/

override def undeleteTweet(request: UndeleteTweetRequest): Future[UndeleteTweetResponse] =

undeleteTweetHandler(request)

/\*\*

\* The async method that undeleteTweet calls to handle notifiying other services of the undelete

\* See [[TweetStores.asyncUndeleteTweetStore]] for all the stores that handle this event.

\*/

override def asyncUndeleteTweet(request: AsyncUndeleteTweetRequest): Future[Unit] =

AsyncUndeleteTweet.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncUndeleteTweet(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncUndeleteTweet(e)

}

override def getDeletedTweets(

request: GetDeletedTweetsRequest

): Future[Seq[GetDeletedTweetResult]] =

getDeletedTweetsHandler(request)

/\*\*

\* Triggers the deletion of all of a users tweets. Used by Gizmoduck when erasing a user

\* after they have been deactived for some number of days.

\*/

override def eraseUserTweets(request: EraseUserTweetsRequest): Future[Unit] =

eraseUserTweetsHandler.eraseUserTweetsRequest(request)

override def asyncEraseUserTweets(request: AsyncEraseUserTweetsRequest): Future[Unit] =

eraseUserTweetsHandler.asyncEraseUserTweetsRequest(request)

override def asyncDelete(request: AsyncDeleteRequest): Future[Unit] =

AsyncDeleteTweet.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncDeleteTweet(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncDeleteTweet(e)

}

/\*

\* unretweet a tweet.

\*

\* There are two ways to unretweet:

\* - call deleteTweets() with the retweetId

\* - call unretweet() with the retweeter userId and sourceTweetId

\*

\* This is useful if you want to be able to undo a retweet without having to

\* keep track of a retweetId

\*

\* Returns DeleteTweetResult for any deleted retweets.

\*/

override def unretweet(request: UnretweetRequest): Future[UnretweetResult] =

unretweetHandler(request)

override def asyncDeleteAdditionalFields(

request: AsyncDeleteAdditionalFieldsRequest

): Future[Unit] =

asyncDeleteAdditionalFieldsBuilder(request).map {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncDeleteAdditionalFields(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncDeleteAdditionalFields(e)

}

override def incrTweetFavCount(request: IncrTweetFavCountRequest): Future[Unit] =

tweetStore.incrFavCount(IncrFavCount.Event(request.tweetId, request.delta, Time.now))

override def asyncIncrFavCount(request: AsyncIncrFavCountRequest): Future[Unit] =

tweetStore.asyncIncrFavCount(AsyncIncrFavCount.Event(request.tweetId, request.delta, Time.now))

override def incrTweetBookmarkCount(request: IncrTweetBookmarkCountRequest): Future[Unit] =

tweetStore.incrBookmarkCount(IncrBookmarkCount.Event(request.tweetId, request.delta, Time.now))

override def asyncIncrBookmarkCount(request: AsyncIncrBookmarkCountRequest): Future[Unit] =

tweetStore.asyncIncrBookmarkCount(

AsyncIncrBookmarkCount.Event(request.tweetId, request.delta, Time.now))

override def scrubGeoUpdateUserTimestamp(request: DeleteLocationData): Future[Unit] =

scrubGeoUpdateUserTimestampBuilder(request).flatMap(tweetStore.scrubGeoUpdateUserTimestamp)

override def deleteLocationData(request: DeleteLocationDataRequest): Future[Unit] =

deleteLocationDataHandler(request)

override def scrubGeo(request: GeoScrub): Future[Unit] =

scrubGeoScrubTweetsBuilder(request).flatMap(tweetStore.scrubGeo)

override def takedown(request: TakedownRequest): Future[Unit] =

takedownHandler(request)

override def quotedTweetDelete(request: QuotedTweetDeleteRequest): Future[Unit] =

quotedTweetDeleteBuilder(request).flatMap {

case Some(event) => tweetStore.quotedTweetDelete(event)

case None => Future.Unit

}

override def quotedTweetTakedown(request: QuotedTweetTakedownRequest): Future[Unit] =

quotedTweetTakedownBuilder(request).flatMap {

case Some(event) => tweetStore.quotedTweetTakedown(event)

case None => Future.Unit

}

override def asyncTakedown(request: AsyncTakedownRequest): Future[Unit] =

AsyncTakedown.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(e) => tweetStore.asyncTakedown(e)

case TweetStoreEventOrRetry.Retry(e) => tweetStore.retryAsyncTakedown(e)

}

override def setTweetUserTakedown(request: SetTweetUserTakedownRequest): Future[Unit] =

userTakedownHandler(request)

override def asyncUpdatePossiblySensitiveTweet(

request: AsyncUpdatePossiblySensitiveTweetRequest

): Future[Unit] = {

AsyncUpdatePossiblySensitiveTweet.Event.fromAsyncRequest(request) match {

case TweetStoreEventOrRetry.First(event) =>

tweetStore.asyncUpdatePossiblySensitiveTweet(event)

case TweetStoreEventOrRetry.Retry(event) =>

tweetStore.retryAsyncUpdatePossiblySensitiveTweet(event)

}

}

override def flush(request: FlushRequest): Future[Unit] = {

// The logged "previous Tweet" value is intended to be used when interactively debugging an

// issue and an engineer flushes the tweet manually, e.g. from tweetypie.cmdline console.

// Don't log automated flushes originating from tweetypie-daemons to cut down noise.

val logExisting = !clientIdHelper.effectiveClientIdRoot.exists(\_ == "tweetypie-daemons")

tweetStore.flush(

Flush.Event(request.tweetIds, request.flushTweets, request.flushCounts, logExisting)

)

}

// Incoming replication events

override def replicatedGetTweetCounts(request: GetTweetCountsRequest): Future[Unit] =

getTweetCounts(request).unit

override def replicatedGetTweetFields(request: GetTweetFieldsRequest): Future[Unit] =

getTweetFields(request).unit

override def replicatedGetTweets(request: GetTweetsRequest): Future[Unit] =

getTweets(request).unit

override def replicatedInsertTweet2(request: ReplicatedInsertTweet2Request): Future[Unit] =

tweetStore.replicatedInsertTweet(

ReplicatedInsertTweet

.Event(

request.cachedTweet.tweet,

request.cachedTweet,

request.quoterHasAlreadyQuotedTweet.getOrElse(false),

request.initialTweetUpdateRequest

)

)

override def replicatedDeleteTweet2(request: ReplicatedDeleteTweet2Request): Future[Unit] =

tweetStore.replicatedDeleteTweet(

ReplicatedDeleteTweet.Event(

tweet = request.tweet,

isErasure = request.isErasure,

isBounceDelete = request.isBounceDelete,

isLastQuoteOfQuoter = request.isLastQuoteOfQuoter.getOrElse(false)

)

)

override def replicatedIncrFavCount(tweetId: TweetId, delta: Int): Future[Unit] =

tweetStore.replicatedIncrFavCount(ReplicatedIncrFavCount.Event(tweetId, delta))

override def replicatedIncrBookmarkCount(tweetId: TweetId, delta: Int): Future[Unit] =

tweetStore.replicatedIncrBookmarkCount(ReplicatedIncrBookmarkCount.Event(tweetId, delta))

override def replicatedScrubGeo(tweetIds: Seq[TweetId]): Future[Unit] =

tweetStore.replicatedScrubGeo(ReplicatedScrubGeo.Event(tweetIds))

override def replicatedSetAdditionalFields(request: SetAdditionalFieldsRequest): Future[Unit] =

tweetStore.replicatedSetAdditionalFields(

ReplicatedSetAdditionalFields.Event(request.additionalFields)

)

override def replicatedSetRetweetVisibility(

request: ReplicatedSetRetweetVisibilityRequest

): Future[Unit] =

tweetStore.replicatedSetRetweetVisibility(

ReplicatedSetRetweetVisibility.Event(request.srcId, request.visible)

)

override def replicatedDeleteAdditionalFields(

request: ReplicatedDeleteAdditionalFieldsRequest

): Future[Unit] =

Future.join(

request.fieldsMap.map {

case (tweetId, fieldIds) =>

tweetStore.replicatedDeleteAdditionalFields(

ReplicatedDeleteAdditionalFields.Event(tweetId, fieldIds)

)

}.toSeq

)

override def replicatedUndeleteTweet2(request: ReplicatedUndeleteTweet2Request): Future[Unit] =

tweetStore.replicatedUndeleteTweet(

ReplicatedUndeleteTweet

.Event(

request.cachedTweet.tweet,

request.cachedTweet,

request.quoterHasAlreadyQuotedTweet.getOrElse(false)

))

override def replicatedTakedown(tweet: Tweet): Future[Unit] =

tweetStore.replicatedTakedown(ReplicatedTakedown.Event(tweet))

override def updatePossiblySensitiveTweet(

request: UpdatePossiblySensitiveTweetRequest

): Future[Unit] =

updatePossiblySensitiveTweetHandler(request)

override def replicatedUpdatePossiblySensitiveTweet(tweet: Tweet): Future[Unit] =

tweetStore.replicatedUpdatePossiblySensitiveTweet(

ReplicatedUpdatePossiblySensitiveTweet.Event(tweet)

)

override def getStoredTweets(

request: GetStoredTweetsRequest

): Future[Seq[GetStoredTweetsResult]] =

getStoredTweetsHandler(request)

override def getStoredTweetsByUser(

request: GetStoredTweetsByUserRequest

): Future[GetStoredTweetsByUserResult] =

getStoredTweetsByUserHandler(request)

}