package com.twitter.tweetypie

package service

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

import com.twitter.servo.util.SynchronizedHashMap

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

import com.twitter.tweetypie.service.observer.\_

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.finagle.tracing.Trace

/\*\*

\* Wraps an underlying TweetService, observing requests and results.

\*/

class ObservedTweetService(

protected val underlying: ThriftTweetService,

stats: StatsReceiver,

clientIdHelper: ClientIdHelper)

extends TweetServiceProxy {

private[this] val asyncEventOrRetryScope = stats.scope("async\_event\_or\_retry")

private[this] val deleteFieldsScope = stats.scope("delete\_additional\_fields")

private[this] val deleteTweetsScope = stats.scope("delete\_tweets")

private[this] val getDeletedTweetsScope = stats.scope("get\_deleted\_tweets")

private[this] val getTweetCountsScope = stats.scope("get\_tweet\_counts")

private[this] val getTweetsScope = stats.scope("get\_tweets")

private[this] val getTweetFieldsScope = stats.scope("get\_tweet\_fields")

private[this] val postTweetScope = stats.scope("post\_tweet")

private[this] val replicatedInsertTweet2Scope = stats.scope("replicated\_insert\_tweet2")

private[this] val retweetScope = stats.scope("post\_retweet")

private[this] val scrubGeoScope = stats.scope("scrub\_geo")

private[this] val setFieldsScope = stats.scope("set\_additional\_fields")

private[this] val setRetweetVisibilityScope = stats.scope("set\_retweet\_visibility")

private[this] val getStoredTweetsScope = stats.scope("get\_stored\_tweets")

private[this] val getStoredTweetsByUserScope = stats.scope("get\_stored\_tweets\_by\_user")

private[this] val defaultGetTweetsRequestOptions = GetTweetOptions()

/\*\* Increments the appropriate write success/failure counter \*/

private[this] val observeWriteResult: Effect[Try[\_]] = {

withAndWithoutClientId(stats) { (stats, \_) =>

val successCounter = stats.counter("write\_successes")

val failureCounter = stats.counter("write\_failures")

val clientErrorCounter = stats.counter("write\_client\_errors")

Effect[Try[\_]] {

case Return(\_) => successCounter.incr()

case Throw(ClientError(\_, \_)) | Throw(AccessDenied(\_, \_)) => clientErrorCounter.incr()

case Throw(\_) => failureCounter.incr()

}

}

}

/\*\* Increments the tweet\_creates counter on future success. \*/

private[this] val observeTweetWriteSuccess: Effect[Any] = {

withAndWithoutClientId(stats) { (stats, \_) =>

val counter = stats.counter("tweet\_writes")

Effect[Any] { \_ => counter.incr() }

}

}

private[this] val observeGetTweetsRequest =

withAndWithoutClientId(getTweetsScope) {

GetTweetsObserver.observeRequest

}

private[this] val observeGetTweetFieldsRequest =

withAndWithoutClientId(getTweetFieldsScope) {

GetTweetFieldsObserver.observeRequest

}

private[this] val observeGetTweetCountsRequest =

withAndWithoutClientId(getTweetCountsScope) { (s, \_) =>

GetTweetCountsObserver.observeRequest(s)

}

private[this] val observeRetweetRequest: Effect[RetweetRequest] =

withAndWithoutClientId(retweetScope) { (s, \_) => Observer.observeRetweetRequest(s) }

private[this] val observeDeleteTweetsRequest =

withAndWithoutClientId(deleteTweetsScope) { (s, \_) => Observer.observeDeleteTweetsRequest(s) }

private[this] val observeSetFieldsRequest: Effect[SetAdditionalFieldsRequest] =

withAndWithoutClientId(setFieldsScope) { (s, \_) => Observer.observeSetFieldsRequest(s) }

private[this] val observeSetRetweetVisibilityRequest: Effect[SetRetweetVisibilityRequest] =

withAndWithoutClientId(setRetweetVisibilityScope) { (s, \_) =>

Observer.observeSetRetweetVisibilityRequest(s)

}

private[this] val observeDeleteFieldsRequest: Effect[DeleteAdditionalFieldsRequest] =

withAndWithoutClientId(deleteFieldsScope) { (s, \_) => Observer.observeDeleteFieldsRequest(s) }

private[this] val observePostTweetAdditionals: Effect[Tweet] =

withAndWithoutClientId(postTweetScope) { (s, \_) => Observer.observeAdditionalFields(s) }

private[this] val observePostTweetRequest: Effect[PostTweetRequest] =

withAndWithoutClientId(postTweetScope) { (s, \_) => PostTweetObserver.observerRequest(s) }

private[this] val observeGetTweetResults =

withAndWithoutClientId(getTweetsScope) {

GetTweetsObserver.observeResults

}

private[this] val observeGetTweetFieldsResults: Effect[Seq[GetTweetFieldsResult]] =

GetTweetFieldsObserver.observeResults(getTweetFieldsScope)

private[this] val observeTweetCountsResults =

GetTweetCountsObserver.observeResults(getTweetCountsScope)

private[this] val observeScrubGeoRequest =

Observer.observeScrubGeo(scrubGeoScope)

private[this] val observeRetweetResponse =

PostTweetObserver.observeResults(retweetScope, byClient = false)

private[this] val observePostTweetResponse =

PostTweetObserver.observeResults(postTweetScope, byClient = false)

private[this] val observeAsyncInsertRequest =

Observer.observeAsyncInsertRequest(asyncEventOrRetryScope)

private[this] val observeAsyncSetAdditionalFieldsRequest =

Observer.observeAsyncSetAdditionalFieldsRequest(asyncEventOrRetryScope)

private[this] val observeAsyncSetRetweetVisibilityRequest =

Observer.observeAsyncSetRetweetVisibilityRequest(asyncEventOrRetryScope)

private[this] val observeAsyncUndeleteTweetRequest =

Observer.observeAsyncUndeleteTweetRequest(asyncEventOrRetryScope)

private[this] val observeAsyncDeleteTweetRequest =

Observer.observeAsyncDeleteTweetRequest(asyncEventOrRetryScope)

private[this] val observeAsyncDeleteAdditionalFieldsRequest =

Observer.observeAsyncDeleteAdditionalFieldsRequest(asyncEventOrRetryScope)

private[this] val observeAsyncTakedownRequest =

Observer.observeAsyncTakedownRequest(asyncEventOrRetryScope)

private[this] val observeAsyncUpdatePossiblySensitiveTweetRequest =

Observer.observeAsyncUpdatePossiblySensitiveTweetRequest(asyncEventOrRetryScope)

private[this] val observedReplicatedInsertTweet2Request =

Observer.observeReplicatedInsertTweetRequest(replicatedInsertTweet2Scope)

private[this] val observeGetTweetFieldsResultState: Effect[GetTweetFieldsObserver.Type] =

withAndWithoutClientId(getTweetFieldsScope) { (statsReceiver, \_) =>

GetTweetFieldsObserver.observeExchange(statsReceiver)

}

private[this] val observeGetTweetsResultState: Effect[GetTweetsObserver.Type] =

withAndWithoutClientId(getTweetsScope) { (statsReceiver, \_) =>

GetTweetsObserver.observeExchange(statsReceiver)

}

private[this] val observeGetTweetCountsResultState: Effect[GetTweetCountsObserver.Type] =

withAndWithoutClientId(getTweetCountsScope) { (statsReceiver, \_) =>

GetTweetCountsObserver.observeExchange(statsReceiver)

}

private[this] val observeGetDeletedTweetsResultState: Effect[GetDeletedTweetsObserver.Type] =

withAndWithoutClientId(getDeletedTweetsScope) { (statsReceiver, \_) =>

GetDeletedTweetsObserver.observeExchange(statsReceiver)

}

private[this] val observeGetStoredTweetsRequest: Effect[GetStoredTweetsRequest] =

GetStoredTweetsObserver.observeRequest(getStoredTweetsScope)

private[this] val observeGetStoredTweetsResult: Effect[Seq[GetStoredTweetsResult]] =

GetStoredTweetsObserver.observeResult(getStoredTweetsScope)

private[this] val observeGetStoredTweetsResultState: Effect[GetStoredTweetsObserver.Type] =

GetStoredTweetsObserver.observeExchange(getStoredTweetsScope)

private[this] val observeGetStoredTweetsByUserRequest: Effect[GetStoredTweetsByUserRequest] =

GetStoredTweetsByUserObserver.observeRequest(getStoredTweetsByUserScope)

private[this] val observeGetStoredTweetsByUserResult: Effect[GetStoredTweetsByUserResult] =

GetStoredTweetsByUserObserver.observeResult(getStoredTweetsByUserScope)

private[this] val observeGetStoredTweetsByUserResultState: Effect[

GetStoredTweetsByUserObserver.Type

] =

GetStoredTweetsByUserObserver.observeExchange(getStoredTweetsByUserScope)

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

val actualRequest =

if (request.options.nonEmpty) request

else request.copy(options = Some(defaultGetTweetsRequestOptions))

observeGetTweetsRequest(actualRequest)

Trace.recordBinary("query\_width", request.tweetIds.length)

super

.getTweets(request)

.onSuccess(observeGetTweetResults)

.respond(response => observeGetTweetsResultState((request, response)))

}

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

observeGetTweetFieldsRequest(request)

Trace.recordBinary("query\_width", request.tweetIds.length)

super

.getTweetFields(request)

.onSuccess(observeGetTweetFieldsResults)

.respond(response => observeGetTweetFieldsResultState((request, response)))

}

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

observeGetTweetCountsRequest(request)

Trace.recordBinary("query\_width", request.tweetIds.length)

super

.getTweetCounts(request)

.onSuccess(observeTweetCountsResults)

.respond(response => observeGetTweetCountsResultState((request, response)))

}

override def getDeletedTweets(

request: GetDeletedTweetsRequest

): Future[Seq[GetDeletedTweetResult]] = {

Trace.recordBinary("query\_width", request.tweetIds.length)

super

.getDeletedTweets(request)

.respond(response => observeGetDeletedTweetsResultState((request, response)))

}

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

observePostTweetRequest(request)

request.additionalFields.foreach(observePostTweetAdditionals)

super

.postTweet(request)

.onSuccess(observePostTweetResponse)

.onSuccess(observeTweetWriteSuccess)

.respond(observeWriteResult)

}

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

observeRetweetRequest(request)

super

.postRetweet(request)

.onSuccess(observeRetweetResponse)

.onSuccess(observeTweetWriteSuccess)

.respond(observeWriteResult)

}

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

observeSetFieldsRequest(request)

super

.setAdditionalFields(request)

.respond(observeWriteResult)

}

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

observeSetRetweetVisibilityRequest(request)

super

.setRetweetVisibility(request)

.respond(observeWriteResult)

}

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

observeDeleteFieldsRequest(request)

super

.deleteAdditionalFields(request)

.respond(observeWriteResult)

}

override def updatePossiblySensitiveTweet(

request: UpdatePossiblySensitiveTweetRequest

): Future[Unit] =

super

.updatePossiblySensitiveTweet(request)

.respond(observeWriteResult)

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

super

.deleteLocationData(request)

.respond(observeWriteResult)

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

observeScrubGeoRequest(geoScrub)

super

.scrubGeo(geoScrub)

.respond(observeWriteResult)

}

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

super.scrubGeoUpdateUserTimestamp(request).respond(observeWriteResult)

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

super

.takedown(request)

.respond(observeWriteResult)

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

super

.setTweetUserTakedown(request)

.respond(observeWriteResult)

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

super

.incrTweetFavCount(request)

.respond(observeWriteResult)

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

super

.incrTweetBookmarkCount(request)

.respond(observeWriteResult)

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

observeDeleteTweetsRequest(request)

super

.deleteTweets(request)

.respond(observeWriteResult)

}

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

super

.cascadedDeleteTweet(request)

.respond(observeWriteResult)

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

observeAsyncInsertRequest(request)

super

.asyncInsert(request)

.respond(observeWriteResult)

}

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

observeAsyncSetAdditionalFieldsRequest(request)

super

.asyncSetAdditionalFields(request)

.respond(observeWriteResult)

}

override def asyncSetRetweetVisibility(

request: AsyncSetRetweetVisibilityRequest

): Future[Unit] = {

observeAsyncSetRetweetVisibilityRequest(request)

super

.asyncSetRetweetVisibility(request)

.respond(observeWriteResult)

}

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

observeAsyncUndeleteTweetRequest(request)

super

.asyncUndeleteTweet(request)

.respond(observeWriteResult)

}

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

observeAsyncDeleteTweetRequest(request)

super

.asyncDelete(request)

.respond(observeWriteResult)

}

override def asyncDeleteAdditionalFields(

request: AsyncDeleteAdditionalFieldsRequest

): Future[Unit] = {

observeAsyncDeleteAdditionalFieldsRequest(request)

super

.asyncDeleteAdditionalFields(request)

.respond(observeWriteResult)

}

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

observeAsyncTakedownRequest(request)

super

.asyncTakedown(request)

.respond(observeWriteResult)

}

override def asyncUpdatePossiblySensitiveTweet(

request: AsyncUpdatePossiblySensitiveTweetRequest

): Future[Unit] = {

observeAsyncUpdatePossiblySensitiveTweetRequest(request)

super

.asyncUpdatePossiblySensitiveTweet(request)

.respond(observeWriteResult)

}

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

observedReplicatedInsertTweet2Request(request.cachedTweet.tweet)

super.replicatedInsertTweet2(request)

}

override def getStoredTweets(

request: GetStoredTweetsRequest

): Future[Seq[GetStoredTweetsResult]] = {

observeGetStoredTweetsRequest(request)

super

.getStoredTweets(request)

.onSuccess(observeGetStoredTweetsResult)

.respond(response => observeGetStoredTweetsResultState((request, response)))

}

override def getStoredTweetsByUser(

request: GetStoredTweetsByUserRequest

): Future[GetStoredTweetsByUserResult] = {

observeGetStoredTweetsByUserRequest(request)

super

.getStoredTweetsByUser(request)

.onSuccess(observeGetStoredTweetsByUserResult)

.respond(response => observeGetStoredTweetsByUserResultState((request, response)))

}

private def withAndWithoutClientId[A](

stats: StatsReceiver

)(

f: (StatsReceiver, Boolean) => Effect[A]

) =

f(stats, false).also(withClientId(stats)(f))

private def withClientId[A](stats: StatsReceiver)(f: (StatsReceiver, Boolean) => Effect[A]) = {

val map = new SynchronizedHashMap[String, Effect[A]]

Effect[A] { value =>

clientIdHelper.effectiveClientIdRoot.foreach { clientId =>

val clientObserver = map.getOrElseUpdate(clientId, f(stats.scope(clientId), true))

clientObserver(value)

}

}

}

}