/\*\* Copyright 2012 Twitter, Inc. \*/

package com.twitter.tweetypie

package service

import com.twitter.finagle.thrift.ClientId

import com.twitter.tweetypie.thriftscala.{TweetServiceProxy => BaseTweetServiceProxy, \_}

/\*\*

\* A trait for TweetService implementations that wrap an underlying TweetService and need to modify

\* only some of the methods.

\*

\* This proxy is the same as [[com.twitter.tweetypie.thriftscala.TweetServiceProxy]], except it also

\* extends [[com.twitter.tweetypie.thriftscala.TweetServiceInternal]] which gives us access to all

\* of the async\* methods.

\*/

trait TweetServiceProxy extends BaseTweetServiceProxy with ThriftTweetService {

protected override def underlying: ThriftTweetService

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

wrap(underlying.replicatedGetTweetCounts(request))

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

wrap(underlying.replicatedGetTweetFields(request))

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

wrap(underlying.replicatedGetTweets(request))

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

wrap(underlying.asyncSetAdditionalFields(request))

override def asyncDeleteAdditionalFields(

request: AsyncDeleteAdditionalFieldsRequest

): Future[Unit] =

wrap(underlying.asyncDeleteAdditionalFields(request))

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

wrap(underlying.cascadedDeleteTweet(request))

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

wrap(underlying.asyncInsert(request))

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

wrap(underlying.replicatedUpdatePossiblySensitiveTweet(tweet))

override def asyncUpdatePossiblySensitiveTweet(

request: AsyncUpdatePossiblySensitiveTweetRequest

): Future[Unit] =

wrap(underlying.asyncUpdatePossiblySensitiveTweet(request))

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

wrap(underlying.asyncUndeleteTweet(request))

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

wrap(underlying.eraseUserTweets(request))

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

wrap(underlying.asyncEraseUserTweets(request))

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

wrap(underlying.asyncDelete(request))

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

wrap(underlying.asyncIncrFavCount(request))

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

wrap(underlying.asyncIncrBookmarkCount(request))

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

wrap(underlying.scrubGeoUpdateUserTimestamp(request))

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

wrap(underlying.asyncSetRetweetVisibility(request))

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

wrap(underlying.setRetweetVisibility(request))

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

wrap(underlying.asyncTakedown(request))

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

wrap(underlying.setTweetUserTakedown(request))

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

wrap(underlying.replicatedUndeleteTweet2(request))

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

wrap(underlying.replicatedInsertTweet2(request))

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

wrap(underlying.replicatedDeleteTweet2(request))

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

wrap(underlying.replicatedIncrFavCount(tweetId, delta))

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

wrap(underlying.replicatedIncrBookmarkCount(tweetId, delta))

override def replicatedSetRetweetVisibility(

request: ReplicatedSetRetweetVisibilityRequest

): Future[Unit] =

wrap(underlying.replicatedSetRetweetVisibility(request))

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

wrap(underlying.replicatedScrubGeo(tweetIds))

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

wrap(underlying.replicatedSetAdditionalFields(request))

override def replicatedDeleteAdditionalFields(

request: ReplicatedDeleteAdditionalFieldsRequest

): Future[Unit] =

wrap(underlying.replicatedDeleteAdditionalFields(request))

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

wrap(underlying.replicatedTakedown(tweet))

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

wrap(underlying.quotedTweetDelete(request))

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

wrap(underlying.quotedTweetTakedown(request))

override def getStoredTweets(

request: GetStoredTweetsRequest

): Future[Seq[GetStoredTweetsResult]] =

wrap(underlying.getStoredTweets(request))

override def getStoredTweetsByUser(

request: GetStoredTweetsByUserRequest

): Future[GetStoredTweetsByUserResult] =

wrap(underlying.getStoredTweetsByUser(request))

}

/\*\*

\* A TweetServiceProxy with a mutable underlying field.

\*/

class MutableTweetServiceProxy(var underlying: ThriftTweetService) extends TweetServiceProxy

/\*\*

\* A TweetServiceProxy that sets the ClientId context before executing the method.

\*/

class ClientIdSettingTweetServiceProxy(clientId: ClientId, val underlying: ThriftTweetService)

extends TweetServiceProxy {

override def wrap[A](f: => Future[A]): Future[A] =

clientId.asCurrent(f)

}