package com.twitter.tweetypie

package store

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* A TweetStore that sends write events to the replication endpoints

\* of a ThriftTweetService.

\*

\* The events that are sent are sufficient to keep the other

\* instance's caches up to date. The calls contain sufficient data so

\* that the remote caches can be updated without requiring the remote

\* Tweetypie to access any other services.

\*

\* The replication services two purposes:

\*

\* 1. Maintain consistency between caches in different data centers.

\*

\* 2. Keep the caches in all data centers warm, protecting backend

\* services.

\*

\* Correctness bugs are worse than bugs that make data less available.

\* All of these events affect data consistency.

\*

\* IncrFavCount.Event and InsertEvents are the least important

\* from a data consistency standpoint, because the only data

\* consistency issues are counts, which are cached for a shorter time,

\* and are not as noticable to end users if they fail to occur.

\* (Failure to apply them is both less severe and self-correcting.)

\*

\* Delete and GeoScrub events are critical, because the cached data

\* has a long expiration and failure to apply them can result in

\* violations of user privacy.

\*

\* Update events are also important from a legal perspective, since

\* the update may be updating the per-country take-down status.

\*

\* @param svc: The ThriftTweetService implementation that will receive the

\* replication events. In practice, this will usually be a

\* deferredrpc service.

\*/

trait ReplicatingTweetStore

extends TweetStoreBase[ReplicatingTweetStore]

with AsyncInsertTweet.Store

with AsyncDeleteTweet.Store

with AsyncUndeleteTweet.Store

with AsyncSetRetweetVisibility.Store

with AsyncSetAdditionalFields.Store

with AsyncDeleteAdditionalFields.Store

with ScrubGeo.Store

with IncrFavCount.Store

with IncrBookmarkCount.Store

with AsyncTakedown.Store

with AsyncUpdatePossiblySensitiveTweet.Store {

def wrap(w: TweetStore.Wrap): ReplicatingTweetStore =

new TweetStoreWrapper(w, this)

with ReplicatingTweetStore

with AsyncInsertTweet.StoreWrapper

with AsyncDeleteTweet.StoreWrapper

with AsyncUndeleteTweet.StoreWrapper

with AsyncSetRetweetVisibility.StoreWrapper

with AsyncSetAdditionalFields.StoreWrapper

with AsyncDeleteAdditionalFields.StoreWrapper

with ScrubGeo.StoreWrapper

with IncrFavCount.StoreWrapper

with IncrBookmarkCount.StoreWrapper

with AsyncTakedown.StoreWrapper

with AsyncUpdatePossiblySensitiveTweet.StoreWrapper

}

object ReplicatingTweetStore {

val Action: AsyncWriteAction.Replication.type = AsyncWriteAction.Replication

def apply(

svc: ThriftTweetService

): ReplicatingTweetStore =

new ReplicatingTweetStore {

override val asyncInsertTweet: FutureEffect[AsyncInsertTweet.Event] =

FutureEffect[AsyncInsertTweet.Event] { e =>

svc.replicatedInsertTweet2(

ReplicatedInsertTweet2Request(

e.cachedTweet,

initialTweetUpdateRequest = e.initialTweetUpdateRequest

))

}

override val retryAsyncInsertTweet: FutureEffect[

TweetStoreRetryEvent[AsyncInsertTweet.Event]

] =

TweetStore.retry(Action, asyncInsertTweet)

override val asyncDeleteTweet: FutureEffect[AsyncDeleteTweet.Event] =

FutureEffect[AsyncDeleteTweet.Event] { e =>

svc.replicatedDeleteTweet2(

ReplicatedDeleteTweet2Request(

tweet = e.tweet,

isErasure = e.isUserErasure,

isBounceDelete = e.isBounceDelete

)

)

}

override val retryAsyncDeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteTweet.Event]

] =

TweetStore.retry(Action, asyncDeleteTweet)

override val asyncUndeleteTweet: FutureEffect[AsyncUndeleteTweet.Event] =

FutureEffect[AsyncUndeleteTweet.Event] { e =>

svc.replicatedUndeleteTweet2(ReplicatedUndeleteTweet2Request(e.cachedTweet))

}

override val retryAsyncUndeleteTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUndeleteTweet.Event]

] =

TweetStore.retry(Action, asyncUndeleteTweet)

override val asyncSetAdditionalFields: FutureEffect[AsyncSetAdditionalFields.Event] =

FutureEffect[AsyncSetAdditionalFields.Event] { e =>

svc.replicatedSetAdditionalFields(SetAdditionalFieldsRequest(e.additionalFields))

}

override val retryAsyncSetAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncSetAdditionalFields.Event]

] =

TweetStore.retry(Action, asyncSetAdditionalFields)

override val asyncSetRetweetVisibility: FutureEffect[AsyncSetRetweetVisibility.Event] =

FutureEffect[AsyncSetRetweetVisibility.Event] { e =>

svc.replicatedSetRetweetVisibility(

ReplicatedSetRetweetVisibilityRequest(e.srcId, e.visible)

)

}

override val retryAsyncSetRetweetVisibility: FutureEffect[

TweetStoreRetryEvent[AsyncSetRetweetVisibility.Event]

] =

TweetStore.retry(Action, asyncSetRetweetVisibility)

override val asyncDeleteAdditionalFields: FutureEffect[AsyncDeleteAdditionalFields.Event] =

FutureEffect[AsyncDeleteAdditionalFields.Event] { e =>

svc.replicatedDeleteAdditionalFields(

ReplicatedDeleteAdditionalFieldsRequest(Map(e.tweetId -> e.fieldIds))

)

}

override val retryAsyncDeleteAdditionalFields: FutureEffect[

TweetStoreRetryEvent[AsyncDeleteAdditionalFields.Event]

] =

TweetStore.retry(Action, asyncDeleteAdditionalFields)

override val scrubGeo: FutureEffect[ScrubGeo.Event] =

FutureEffect[ScrubGeo.Event](e => svc.replicatedScrubGeo(e.tweetIds))

override val incrFavCount: FutureEffect[IncrFavCount.Event] =

FutureEffect[IncrFavCount.Event](e => svc.replicatedIncrFavCount(e.tweetId, e.delta))

override val incrBookmarkCount: FutureEffect[IncrBookmarkCount.Event] =

FutureEffect[IncrBookmarkCount.Event](e =>

svc.replicatedIncrBookmarkCount(e.tweetId, e.delta))

override val asyncTakedown: FutureEffect[AsyncTakedown.Event] =

FutureEffect[AsyncTakedown.Event](e => svc.replicatedTakedown(e.tweet))

override val retryAsyncTakedown: FutureEffect[TweetStoreRetryEvent[AsyncTakedown.Event]] =

TweetStore.retry(Action, asyncTakedown)

override val asyncUpdatePossiblySensitiveTweet: FutureEffect[

AsyncUpdatePossiblySensitiveTweet.Event

] =

FutureEffect[AsyncUpdatePossiblySensitiveTweet.Event](e =>

svc.replicatedUpdatePossiblySensitiveTweet(e.tweet))

override val retryAsyncUpdatePossiblySensitiveTweet: FutureEffect[

TweetStoreRetryEvent[AsyncUpdatePossiblySensitiveTweet.Event]

] =

TweetStore.retry(Action, asyncUpdatePossiblySensitiveTweet)

}

}