package com.twitter.tweetypie

package store

import com.twitter.tweetypie.store.TweetEventDataScrubber.scrub

import com.twitter.tweetypie.thriftscala.\_

object DeleteTweet extends TweetStore.SyncModule {

case class Event(

tweet: Tweet,

timestamp: Time,

user: Option[User] = None,

byUserId: Option[UserId] = None,

auditPassthrough: Option[AuditDeleteTweet] = None,

cascadedFromTweetId: Option[TweetId] = None,

isUserErasure: Boolean = false,

isBounceDelete: Boolean = false,

isLastQuoteOfQuoter: Boolean = false,

isAdminDelete: Boolean)

extends SyncTweetStoreEvent("delete\_tweet") {

def toAsyncRequest: AsyncDeleteRequest =

AsyncDeleteRequest(

tweet = tweet,

user = user,

byUserId = byUserId,

timestamp = timestamp.inMillis,

auditPassthrough = auditPassthrough,

cascadedFromTweetId = cascadedFromTweetId,

isUserErasure = isUserErasure,

isBounceDelete = isBounceDelete,

isLastQuoteOfQuoter = Some(isLastQuoteOfQuoter),

isAdminDelete = Some(isAdminDelete)

)

}

trait Store {

val deleteTweet: FutureEffect[Event]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val deleteTweet: FutureEffect[Event] = wrap(underlying.deleteTweet)

}

object Store {

def apply(

cachingTweetStore: CachingTweetStore,

asyncEnqueueStore: AsyncEnqueueStore,

userCountsUpdatingStore: GizmoduckUserCountsUpdatingStore,

tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore,

logLensStore: LogLensStore

): Store =

new Store {

override val deleteTweet: FutureEffect[Event] =

FutureEffect.inParallel(

cachingTweetStore.ignoreFailures.deleteTweet,

asyncEnqueueStore.deleteTweet,

userCountsUpdatingStore.deleteTweet,

tweetCountsUpdatingStore.deleteTweet,

logLensStore.deleteTweet

)

}

}

}

object AsyncDeleteTweet extends TweetStore.AsyncModule {

object Event {

def fromAsyncRequest(request: AsyncDeleteRequest): TweetStoreEventOrRetry[Event] =

TweetStoreEventOrRetry(

AsyncDeleteTweet.Event(

tweet = request.tweet,

timestamp = Time.fromMilliseconds(request.timestamp),

optUser = request.user,

byUserId = request.byUserId,

auditPassthrough = request.auditPassthrough,

cascadedFromTweetId = request.cascadedFromTweetId,

isUserErasure = request.isUserErasure,

isBounceDelete = request.isBounceDelete,

isLastQuoteOfQuoter = request.isLastQuoteOfQuoter.getOrElse(false),

isAdminDelete = request.isAdminDelete.getOrElse(false)

),

request.retryAction,

RetryEvent

)

}

case class Event(

tweet: Tweet,

timestamp: Time,

optUser: Option[User] = None,

byUserId: Option[UserId] = None,

auditPassthrough: Option[AuditDeleteTweet] = None,

cascadedFromTweetId: Option[TweetId] = None,

isUserErasure: Boolean = false,

isBounceDelete: Boolean,

isLastQuoteOfQuoter: Boolean = false,

isAdminDelete: Boolean)

extends AsyncTweetStoreEvent("async\_delete\_tweet")

with TweetStoreTweetEvent {

val tweetEventTweetId: TweetId = tweet.id

def toAsyncRequest(action: Option[AsyncWriteAction] = None): AsyncDeleteRequest =

AsyncDeleteRequest(

tweet = tweet,

user = optUser,

byUserId = byUserId,

timestamp = timestamp.inMillis,

auditPassthrough = auditPassthrough,

cascadedFromTweetId = cascadedFromTweetId,

retryAction = action,

isUserErasure = isUserErasure,

isBounceDelete = isBounceDelete,

isLastQuoteOfQuoter = Some(isLastQuoteOfQuoter),

isAdminDelete = Some(isAdminDelete)

)

override def toTweetEventData: Seq[TweetEventData] =

Seq(

TweetEventData.TweetDeleteEvent(

TweetDeleteEvent(

tweet = scrub(tweet),

user = optUser,

isUserErasure = Some(isUserErasure),

audit = auditPassthrough,

byUserId = byUserId,

isAdminDelete = Some(isAdminDelete)

)

)

)

override def enqueueRetry(service: ThriftTweetService, action: AsyncWriteAction): Future[Unit] =

service.asyncDelete(toAsyncRequest(Some(action)))

}

case class RetryEvent(action: AsyncWriteAction, event: Event)

extends TweetStoreRetryEvent[Event] {

override val eventType: AsyncWriteEventType.Delete.type = AsyncWriteEventType.Delete

override val scribedTweetOnFailure: Option[Tweet] = Some(event.tweet)

}

trait Store {

val asyncDeleteTweet: FutureEffect[Event]

val retryAsyncDeleteTweet: FutureEffect[TweetStoreRetryEvent[Event]]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val asyncDeleteTweet: FutureEffect[Event] = wrap(underlying.asyncDeleteTweet)

override val retryAsyncDeleteTweet: FutureEffect[TweetStoreRetryEvent[Event]] = wrap(

underlying.retryAsyncDeleteTweet)

}

object Store {

def apply(

manhattanStore: ManhattanTweetStore,

cachingTweetStore: CachingTweetStore,

replicatingStore: ReplicatingTweetStore,

indexingStore: TweetIndexingStore,

eventBusEnqueueStore: TweetEventBusStore,

timelineUpdatingStore: TlsTimelineUpdatingStore,

tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore,

guanoServiceStore: GuanoServiceStore,

mediaServiceStore: MediaServiceStore

): Store = {

val stores: Seq[Store] =

Seq(

manhattanStore,

cachingTweetStore,

replicatingStore,

indexingStore,

eventBusEnqueueStore,

timelineUpdatingStore,

tweetCountsUpdatingStore,

guanoServiceStore,

mediaServiceStore

)

def build[E <: TweetStoreEvent](extract: Store => FutureEffect[E]): FutureEffect[E] =

FutureEffect.inParallel[E](stores.map(extract): \_\*)

new Store {

override val asyncDeleteTweet: FutureEffect[Event] = build(\_.asyncDeleteTweet)

override val retryAsyncDeleteTweet: FutureEffect[TweetStoreRetryEvent[Event]] = build(

\_.retryAsyncDeleteTweet)

}

}

}

}

object ReplicatedDeleteTweet extends TweetStore.ReplicatedModule {

case class Event(

tweet: Tweet,

isErasure: Boolean,

isBounceDelete: Boolean,

isLastQuoteOfQuoter: Boolean = false)

extends ReplicatedTweetStoreEvent("replicated\_delete\_tweet")

trait Store {

val replicatedDeleteTweet: FutureEffect[Event]

}

trait StoreWrapper extends Store { self: TweetStoreWrapper[Store] =>

override val replicatedDeleteTweet: FutureEffect[Event] = wrap(underlying.replicatedDeleteTweet)

}

object Store {

def apply(

cachingTweetStore: CachingTweetStore,

tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore

): Store = {

new Store {

override val replicatedDeleteTweet: FutureEffect[Event] =

FutureEffect.inParallel(

cachingTweetStore.replicatedDeleteTweet,

tweetCountsUpdatingStore.replicatedDeleteTweet.ignoreFailures

)

}

}

}

}