package com.twitter.tweetypie

package store

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

import com.twitter.tweetypie.thriftscala.\_

object UndeleteTweet extends TweetStore.SyncModule {

/\*\*

\* A TweetStoreEvent for Undeletion.

\*/

case class Event(

tweet: Tweet,

user: User,

timestamp: Time,

hydrateOptions: WritePathHydrationOptions,

\_internalTweet: Option[CachedTweet] = None,

deletedAt: Option[Time],

sourceTweet: Option[Tweet] = None,

sourceUser: Option[User] = None,

quotedTweet: Option[Tweet] = None,

quotedUser: Option[User] = None,

parentUserId: Option[UserId] = None,

quoterHasAlreadyQuotedTweet: Boolean = false)

extends SyncTweetStoreEvent("undelete\_tweet")

with QuotedTweetOps {

def internalTweet: CachedTweet =

\_internalTweet.getOrElse(

throw new IllegalStateException(

s"internalTweet should have been set in WritePathHydration, ${this}"

)

)

def toAsyncUndeleteTweetRequest: AsyncUndeleteTweetRequest =

AsyncUndeleteTweetRequest(

tweet = tweet,

cachedTweet = internalTweet,

user = user,

timestamp = timestamp.inMillis,

deletedAt = deletedAt.map(\_.inMillis),

sourceTweet = sourceTweet,

sourceUser = sourceUser,

quotedTweet = quotedTweet,

quotedUser = quotedUser,

parentUserId = parentUserId,

quoterHasAlreadyQuotedTweet = Some(quoterHasAlreadyQuotedTweet)

)

}

trait Store {

val undeleteTweet: FutureEffect[Event]

}

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

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

}

object Store {

def apply(

logLensStore: LogLensStore,

cachingTweetStore: CachingTweetStore,

tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore,

asyncEnqueueStore: AsyncEnqueueStore

): Store =

new Store {

override val undeleteTweet: FutureEffect[Event] =

FutureEffect.inParallel(

logLensStore.undeleteTweet,

// ignore failures writing to cache, will be retried in async-path

cachingTweetStore.ignoreFailures.undeleteTweet,

tweetCountsUpdatingStore.undeleteTweet,

asyncEnqueueStore.undeleteTweet

)

}

}

}

object AsyncUndeleteTweet extends TweetStore.AsyncModule {

object Event {

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

TweetStoreEventOrRetry(

AsyncUndeleteTweet.Event(

tweet = request.tweet,

cachedTweet = request.cachedTweet,

user = request.user,

optUser = Some(request.user),

timestamp = Time.fromMilliseconds(request.timestamp),

deletedAt = request.deletedAt.map(Time.fromMilliseconds),

sourceTweet = request.sourceTweet,

sourceUser = request.sourceUser,

quotedTweet = request.quotedTweet,

quotedUser = request.quotedUser,

parentUserId = request.parentUserId,

quoterHasAlreadyQuotedTweet = request.quoterHasAlreadyQuotedTweet.getOrElse(false)

),

request.retryAction,

RetryEvent

)

}

case class Event(

tweet: Tweet,

cachedTweet: CachedTweet,

user: User,

optUser: Option[User],

timestamp: Time,

deletedAt: Option[Time],

sourceTweet: Option[Tweet],

sourceUser: Option[User],

quotedTweet: Option[Tweet],

quotedUser: Option[User],

parentUserId: Option[UserId] = None,

quoterHasAlreadyQuotedTweet: Boolean = false)

extends AsyncTweetStoreEvent("async\_undelete\_tweet")

with QuotedTweetOps

with TweetStoreTweetEvent {

/\*\*

\* Convert this event into an AsyncUndeleteTweetRequest thrift request object

\*/

def toAsyncRequest(retryAction: Option[AsyncWriteAction] = None): AsyncUndeleteTweetRequest =

AsyncUndeleteTweetRequest(

tweet = tweet,

cachedTweet = cachedTweet,

user = user,

timestamp = timestamp.inMillis,

retryAction = retryAction,

deletedAt = deletedAt.map(\_.inMillis),

sourceTweet = sourceTweet,

sourceUser = sourceUser,

quotedTweet = quotedTweet,

quotedUser = quotedUser,

parentUserId = parentUserId,

quoterHasAlreadyQuotedTweet = Some(quoterHasAlreadyQuotedTweet)

)

override def toTweetEventData: Seq[TweetEventData] =

Seq(

TweetEventData.TweetUndeleteEvent(

TweetUndeleteEvent(

tweet = scrub(tweet),

user = Some(user),

sourceTweet = sourceTweet.map(scrub),

sourceUser = sourceUser,

retweetParentUserId = parentUserId,

quotedTweet = publicQuotedTweet.map(scrub),

quotedUser = publicQuotedUser,

deletedAtMsec = deletedAt.map(\_.inMilliseconds)

)

)

)

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

service.asyncUndeleteTweet(toAsyncRequest(Some(action)))

}

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

extends TweetStoreRetryEvent[Event] {

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

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

}

trait Store {

val asyncUndeleteTweet: FutureEffect[Event]

val retryAsyncUndeleteTweet: FutureEffect[TweetStoreRetryEvent[Event]]

}

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

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

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

underlying.retryAsyncUndeleteTweet)

}

object Store {

def apply(

cachingTweetStore: CachingTweetStore,

eventBusEnqueueStore: TweetEventBusStore,

indexingStore: TweetIndexingStore,

replicatingStore: ReplicatingTweetStore,

mediaServiceStore: MediaServiceStore,

timelineUpdatingStore: TlsTimelineUpdatingStore

): Store = {

val stores: Seq[Store] =

Seq(

cachingTweetStore,

eventBusEnqueueStore,

indexingStore,

replicatingStore,

mediaServiceStore,

timelineUpdatingStore

)

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

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

new Store {

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

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

\_.retryAsyncUndeleteTweet)

}

}

}

}

object ReplicatedUndeleteTweet extends TweetStore.ReplicatedModule {

case class Event(

tweet: Tweet,

cachedTweet: CachedTweet,

quoterHasAlreadyQuotedTweet: Boolean = false)

extends ReplicatedTweetStoreEvent("replicated\_undelete\_tweet")

trait Store {

val replicatedUndeleteTweet: FutureEffect[Event]

}

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

override val replicatedUndeleteTweet: FutureEffect[Event] = wrap(

underlying.replicatedUndeleteTweet)

}

object Store {

def apply(

cachingTweetStore: CachingTweetStore,

tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore

): Store =

new Store {

override val replicatedUndeleteTweet: FutureEffect[Event] =

FutureEffect.inParallel(

cachingTweetStore.replicatedUndeleteTweet.ignoreFailures,

tweetCountsUpdatingStore.replicatedUndeleteTweet.ignoreFailures

)

}

}

}