package com.twitter.tweetypie

package store

import com.twitter.tweetypie.thriftscala.\_

object DeleteAdditionalFields extends TweetStore.SyncModule {

case class Event(tweetId: TweetId, fieldIds: Seq[FieldId], userId: UserId, timestamp: Time)

extends SyncTweetStoreEvent("delete\_additional\_fields") {

def toAsyncRequest: AsyncDeleteAdditionalFieldsRequest =

AsyncDeleteAdditionalFieldsRequest(

tweetId = tweetId,

fieldIds = fieldIds,

userId = userId,

timestamp = timestamp.inMillis

)

}

trait Store {

val deleteAdditionalFields: FutureEffect[Event]

}

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

override val deleteAdditionalFields: FutureEffect[Event] = wrap(

underlying.deleteAdditionalFields)

}

object Store {

def apply(

cachingTweetStore: CachingTweetStore,

asyncEnqueueStore: AsyncEnqueueStore,

logLensStore: LogLensStore

): Store =

new Store {

override val deleteAdditionalFields: FutureEffect[Event] =

FutureEffect.inParallel(

// ignore failures deleting from cache, will be retried in async-path

cachingTweetStore.ignoreFailures.deleteAdditionalFields,

asyncEnqueueStore.deleteAdditionalFields,

logLensStore.deleteAdditionalFields

)

}

}

}

object AsyncDeleteAdditionalFields extends TweetStore.AsyncModule {

object Event {

def fromAsyncRequest(

request: AsyncDeleteAdditionalFieldsRequest,

user: User

): TweetStoreEventOrRetry[Event] =

TweetStoreEventOrRetry(

Event(

tweetId = request.tweetId,

fieldIds = request.fieldIds,

userId = request.userId,

optUser = Some(user),

timestamp = Time.fromMilliseconds(request.timestamp)

),

request.retryAction,

RetryEvent

)

}

case class Event(

tweetId: TweetId,

fieldIds: Seq[FieldId],

userId: UserId,

optUser: Option[User],

timestamp: Time)

extends AsyncTweetStoreEvent("async\_delete\_additional\_fields")

with TweetStoreTweetEvent {

def toAsyncRequest(

action: Option[AsyncWriteAction] = None

): AsyncDeleteAdditionalFieldsRequest =

AsyncDeleteAdditionalFieldsRequest(

tweetId = tweetId,

fieldIds = fieldIds,

userId = userId,

timestamp = timestamp.inMillis,

retryAction = action

)

override def toTweetEventData: Seq[TweetEventData] =

Seq(

TweetEventData.AdditionalFieldDeleteEvent(

AdditionalFieldDeleteEvent(

deletedFields = Map(tweetId -> fieldIds),

userId = optUser.map(\_.id)

)

)

)

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

service.asyncDeleteAdditionalFields(toAsyncRequest(Some(action)))

}

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

extends TweetStoreRetryEvent[Event] {

override val eventType: AsyncWriteEventType.DeleteAdditionalFields.type =

AsyncWriteEventType.DeleteAdditionalFields

override val scribedTweetOnFailure: None.type = None

}

trait Store {

val asyncDeleteAdditionalFields: FutureEffect[Event]

val retryAsyncDeleteAdditionalFields: FutureEffect[TweetStoreRetryEvent[Event]]

}

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

override val asyncDeleteAdditionalFields: FutureEffect[Event] = wrap(

underlying.asyncDeleteAdditionalFields)

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

underlying.retryAsyncDeleteAdditionalFields

)

}

object Store {

def apply(

manhattanStore: ManhattanTweetStore,

cachingTweetStore: CachingTweetStore,

replicatingStore: ReplicatingTweetStore,

eventBusEnqueueStore: TweetEventBusStore

): Store = {

val stores: Seq[Store] =

Seq(

manhattanStore,

cachingTweetStore,

replicatingStore,

eventBusEnqueueStore

)

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

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

new Store {

override val asyncDeleteAdditionalFields: FutureEffect[Event] = build(

\_.asyncDeleteAdditionalFields)

override val retryAsyncDeleteAdditionalFields: FutureEffect[TweetStoreRetryEvent[Event]] =

build(\_.retryAsyncDeleteAdditionalFields)

}

}

}

}

object ReplicatedDeleteAdditionalFields extends TweetStore.ReplicatedModule {

case class Event(tweetId: TweetId, fieldIds: Seq[FieldId])

extends ReplicatedTweetStoreEvent("replicated\_delete\_additional\_fields")

trait Store {

val replicatedDeleteAdditionalFields: FutureEffect[Event]

}

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

override val replicatedDeleteAdditionalFields: FutureEffect[Event] =

wrap(underlying.replicatedDeleteAdditionalFields)

}

object Store {

def apply(cachingTweetStore: CachingTweetStore): Store = {

new Store {

override val replicatedDeleteAdditionalFields: FutureEffect[Event] =

cachingTweetStore.replicatedDeleteAdditionalFields

}

}

}

}