package com.twitter.tweetypie

package store

import com.twitter.tweetypie.store.TweetStoreEvent.NoRetry

import com.twitter.tweetypie.thriftscala.\_

object IncrFavCount extends TweetStore.SyncModule {

case class Event(tweetId: TweetId, delta: Int, timestamp: Time)

extends SyncTweetStoreEvent("incr\_fav\_count") {

val toAsyncRequest: AsyncIncrFavCountRequest = AsyncIncrFavCountRequest(tweetId, delta)

}

trait Store {

val incrFavCount: FutureEffect[Event]

}

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

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

}

object Store {

def apply(

asyncEnqueueStore: AsyncEnqueueStore,

replicatingStore: ReplicatingTweetStore

): Store =

new Store {

override val incrFavCount: FutureEffect[Event] =

FutureEffect.inParallel(

asyncEnqueueStore.incrFavCount,

replicatingStore.incrFavCount

)

}

}

}

object AsyncIncrFavCount extends TweetStore.AsyncModule {

case class Event(tweetId: TweetId, delta: Int, timestamp: Time)

extends AsyncTweetStoreEvent("async\_incr\_fav\_count") {

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

Future.Unit // We need to define this method for TweetStoreEvent.Async but we don't use it

override def retryStrategy: TweetStoreEvent.RetryStrategy = NoRetry

}

trait Store {

val asyncIncrFavCount: FutureEffect[Event]

}

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

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

}

object Store {

def apply(tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore): Store = {

new Store {

override val asyncIncrFavCount: FutureEffect[Event] =

tweetCountsUpdatingStore.asyncIncrFavCount

}

}

}

}

object ReplicatedIncrFavCount extends TweetStore.ReplicatedModule {

case class Event(tweetId: TweetId, delta: Int)

extends ReplicatedTweetStoreEvent("replicated\_incr\_fav\_count") {

override def retryStrategy: TweetStoreEvent.NoRetry.type = NoRetry

}

trait Store {

val replicatedIncrFavCount: FutureEffect[Event]

}

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

override val replicatedIncrFavCount: FutureEffect[Event] = wrap(

underlying.replicatedIncrFavCount)

}

object Store {

def apply(tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore): Store = {

new Store {

override val replicatedIncrFavCount: FutureEffect[Event] =

tweetCountsUpdatingStore.replicatedIncrFavCount.ignoreFailures

}

}

}

}