package com.twitter.tweetypie

package store

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

import com.twitter.tweetypie.store.TweetStoreEvent.RetryStrategy

import com.twitter.tweetypie.thriftscala.AsyncIncrBookmarkCountRequest

import com.twitter.tweetypie.thriftscala.AsyncWriteAction

object IncrBookmarkCount extends TweetStore.SyncModule {

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

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

val toAsyncRequest: AsyncIncrBookmarkCountRequest =

AsyncIncrBookmarkCountRequest(tweetId = tweetId, delta = delta)

}

trait Store {

val incrBookmarkCount: FutureEffect[Event]

}

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

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

}

object Store {

def apply(

asyncEnqueueStore: AsyncEnqueueStore,

replicatingStore: ReplicatingTweetStore

): Store = {

new Store {

override val incrBookmarkCount: FutureEffect[Event] =

FutureEffect.inParallel(

asyncEnqueueStore.incrBookmarkCount,

replicatingStore.incrBookmarkCount

)

}

}

}

}

object AsyncIncrBookmarkCount extends TweetStore.AsyncModule {

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

extends AsyncTweetStoreEvent("async\_incr\_bookmark\_event") {

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

Future.Unit

override def retryStrategy: RetryStrategy = NoRetry

}

trait Store {

def asyncIncrBookmarkCount: FutureEffect[Event]

}

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

override val asyncIncrBookmarkCount: FutureEffect[Event] = wrap(

underlying.asyncIncrBookmarkCount)

}

object Store {

def apply(tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore): Store = {

new Store {

override def asyncIncrBookmarkCount: FutureEffect[AsyncIncrBookmarkCount.Event] =

tweetCountsUpdatingStore.asyncIncrBookmarkCount

}

}

}

}

object ReplicatedIncrBookmarkCount extends TweetStore.ReplicatedModule {

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

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

override def retryStrategy: RetryStrategy = NoRetry

}

trait Store {

val replicatedIncrBookmarkCount: FutureEffect[Event]

}

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

override val replicatedIncrBookmarkCount: FutureEffect[Event] = wrap(

underlying.replicatedIncrBookmarkCount)

}

object Store {

def apply(tweetCountsUpdatingStore: TweetCountsCacheUpdatingStore): Store = {

new Store {

override val replicatedIncrBookmarkCount: FutureEffect[Event] = {

tweetCountsUpdatingStore.replicatedIncrBookmarkCount

}

}

}

}

}