package com.twitter.tweetypie

package store

import com.twitter.conversions.DurationOps.\_

import com.twitter.servo.cache.Cached

import com.twitter.servo.cache.CachedValueStatus

import com.twitter.servo.cache.LockingCache

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.tweetypie.backends.GeoScrubEventStore

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* Scrub geo information from Tweets.

\*/

object ScrubGeo extends TweetStore.SyncModule {

case class Event(

tweetIdSet: Set[TweetId],

userId: UserId,

optUser: Option[User],

timestamp: Time,

enqueueMax: Boolean)

extends SyncTweetStoreEvent("scrub\_geo")

with TweetStoreTweetEvent {

val tweetIds: Seq[TweetId] = tweetIdSet.toSeq

override def toTweetEventData: Seq[TweetEventData] =

tweetIds.map { tweetId =>

TweetEventData.TweetScrubGeoEvent(

TweetScrubGeoEvent(

tweetId = tweetId,

userId = userId

)

)

}

}

trait Store {

val scrubGeo: FutureEffect[Event]

}

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

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

}

object Store {

def apply(

logLensStore: LogLensStore,

manhattanStore: ManhattanTweetStore,

cachingTweetStore: CachingTweetStore,

eventBusEnqueueStore: TweetEventBusStore,

replicatingStore: ReplicatingTweetStore

): Store =

new Store {

override val scrubGeo: FutureEffect[Event] =

FutureEffect.inParallel(

logLensStore.scrubGeo,

manhattanStore.scrubGeo,

cachingTweetStore.scrubGeo,

eventBusEnqueueStore.scrubGeo,

replicatingStore.scrubGeo

)

}

}

}

object ReplicatedScrubGeo extends TweetStore.ReplicatedModule {

case class Event(tweetIds: Seq[TweetId]) extends ReplicatedTweetStoreEvent("replicated\_scrub\_geo")

trait Store {

val replicatedScrubGeo: FutureEffect[Event]

}

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

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

}

object Store {

def apply(cachingTweetStore: CachingTweetStore): Store = {

new Store {

override val replicatedScrubGeo: FutureEffect[Event] =

cachingTweetStore.replicatedScrubGeo

}

}

}

}

/\*\*

\* Update the timestamp of the user's most recent request to delete all

\* location data attached to her tweets. We use the timestamp to ensure

\* that even if we fail to scrub a particular tweet in storage, we will

\* not return geo information with that tweet.

\*

\* See http://go/geoscrub for more details.

\*/

object ScrubGeoUpdateUserTimestamp extends TweetStore.SyncModule {

case class Event(userId: UserId, timestamp: Time, optUser: Option[User])

extends SyncTweetStoreEvent("scrub\_geo\_update\_user\_timestamp")

with TweetStoreTweetEvent {

def mightHaveGeotaggedStatuses: Boolean =

optUser.forall(\_.account.forall(\_.hasGeotaggedStatuses == true))

def maxTweetId: TweetId = SnowflakeId.firstIdFor(timestamp + 1.millisecond) - 1

override def toTweetEventData: Seq[TweetEventData] =

Seq(

TweetEventData.UserScrubGeoEvent(

UserScrubGeoEvent(

userId = userId,

maxTweetId = maxTweetId

)

)

)

/\*\*

\* How to update a geo scrub timestamp cache entry. Always prefers

\* the highest timestamp value that is available, regardless of when

\* it was added to cache.

\*/

def cacheHandler: LockingCache.Handler[Cached[Time]] = {

case Some(c) if c.value.exists(\_ >= timestamp) => None

case \_ => Some(Cached(Some(timestamp), CachedValueStatus.Found, Time.now))

}

}

trait Store {

val scrubGeoUpdateUserTimestamp: FutureEffect[Event]

}

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

override val scrubGeoUpdateUserTimestamp: FutureEffect[Event] = wrap(

underlying.scrubGeoUpdateUserTimestamp)

}

object Store {

def apply(

geotagUpdateStore: GizmoduckUserGeotagUpdateStore,

tweetEventBusStore: TweetEventBusStore,

setInManhattan: GeoScrubEventStore.SetGeoScrubTimestamp,

cache: LockingCache[UserId, Cached[Time]]

): Store = {

val manhattanEffect =

setInManhattan.asFutureEffect

.contramap[Event](e => (e.userId, e.timestamp))

val cacheEffect =

FutureEffect[Event](e => cache.lockAndSet(e.userId, e.cacheHandler).unit)

new Store {

override val scrubGeoUpdateUserTimestamp: FutureEffect[Event] =

FutureEffect.inParallel(

manhattanEffect,

cacheEffect,

geotagUpdateStore.scrubGeoUpdateUserTimestamp,

tweetEventBusStore.scrubGeoUpdateUserTimestamp

)

}

}

}

}