package com.twitter.tweetypie

package config

import com.twitter.featureswitches.v2.FeatureSwitches

import com.twitter.servo.cache.Cached

import com.twitter.servo.cache.LockingCache

import com.twitter.servo.util.ExceptionCategorizer

import com.twitter.servo.util.ExceptionCounter

import com.twitter.servo.util.FutureEffect

import com.twitter.servo.util.Scribe

import com.twitter.stitch.NotFound

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.core.TweetData

import com.twitter.tweetypie.core.ValueState

import com.twitter.tweetypie.hydrator.\_

import com.twitter.tweetypie.repository.TweetQuery

import com.twitter.tweetypie.serverutil.{ExceptionCounter => TpExceptionCounter}

import com.twitter.tweetypie.thriftscala.\_

import com.twitter.tweetypie.client\_id.ClientIdHelper

trait TweetHydrators {

/\*\*

\* Hydrator that has all the Tweet hydrators (entire "pipeline") configured

\* and wired up.

\* This hydrator is used both on the read and write path and is

\* customized by different TweetQuery.Options.

\* Modifications are not automatically written back to cache.

\* `cacheChanges` must be used for that.

\*/

def hydrator: TweetDataValueHydrator

/\*\*

\* The `Effect` to use to write modified tweets back to cache.

\*/

def cacheChangesEffect: Effect[ValueState[TweetData]]

}

object TweetHydrators {

/\*\*

\* Creates all the hydrators and calls TweetHydration to wire them up.

\*/

def apply(

stats: StatsReceiver,

deciderGates: TweetypieDeciderGates,

repos: LogicalRepositories,

tweetDataCache: LockingCache[TweetId, Cached[TweetData]],

hasMedia: Tweet => Boolean,

featureSwitchesWithoutExperiments: FeatureSwitches,

clientIdHelper: ClientIdHelper

): TweetHydrators = {

import repos.\_

val repairStats = stats.scope("repairs")

val hydratorStats = stats.scope("hydrators")

def scoped[A](stats: StatsReceiver, name: String)(f: StatsReceiver => A): A = {

val scopedStats = stats.scope(name)

f(scopedStats)

}

val isFailureException: Throwable => Boolean = {

case \_: FilteredState => false

case NotFound => false

case \_ => true

}

def hydratorExceptionCategorizer(failureScope: String) =

ExceptionCategorizer.const("filtered").onlyIf(\_.isInstanceOf[FilteredState]) ++

ExceptionCategorizer.const("not\_found").onlyIf(\_ == NotFound) ++

TpExceptionCounter.defaultCategorizer(failureScope).onlyIf(isFailureException)

val hydratorExceptionCounter: (StatsReceiver, String) => ExceptionCounter =

(stats, scope) => TpExceptionCounter(stats, hydratorExceptionCategorizer(scope))

val tweetHydrator =

TweetHydration(

hydratorStats = hydratorStats,

hydrateFeatureSwitchResults =

FeatureSwitchResultsHydrator(featureSwitchesWithoutExperiments, clientIdHelper),

hydrateMentions = MentionEntitiesHydrator

.once(MentionEntityHydrator(userIdentityRepo))

.observe(hydratorStats.scope("mentions"), hydratorExceptionCounter),

hydrateLanguage = LanguageHydrator(languageRepo)

.observe(hydratorStats.scope("language"), hydratorExceptionCounter),

hydrateUrls = scoped(hydratorStats, "url") { stats =>

UrlEntitiesHydrator

.once(UrlEntityHydrator(urlRepo, stats))

.observe(stats, hydratorExceptionCounter)

},

hydrateQuotedTweetRef = QuotedTweetRefHydrator

.once(

QuotedTweetRefHydrator(tweetRepo)

)

.observe(hydratorStats.scope("quoted\_tweet\_ref"), hydratorExceptionCounter),

hydrateQuotedTweetRefUrls = QuotedTweetRefUrlsHydrator(userIdentityRepo)

.observe(hydratorStats.scope("quoted\_tweet\_ref\_urls"), hydratorExceptionCounter),

hydrateMediaCacheable = MediaEntitiesHydrator.Cacheable

.once(

MediaEntityHydrator.Cacheable(

hydrateMediaUrls = MediaUrlFieldsHydrator()

.observe(hydratorStats.scope("media\_urls"), hydratorExceptionCounter),

hydrateMediaIsProtected = MediaIsProtectedHydrator(userProtectionRepo)

.observe(hydratorStats.scope("media\_is\_protected"), hydratorExceptionCounter)

)

)

.observe(hydratorStats.scope("media\_cacheable"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateMedia),

hydrateReplyScreenName = ReplyScreenNameHydrator

.once(ReplyScreenNameHydrator(userIdentityRepo))

.observe(hydratorStats.scope("in\_reply\_to\_screen\_name"), hydratorExceptionCounter),

hydrateConvoId = ConversationIdHydrator(conversationIdRepo)

.observe(hydratorStats.scope("conversation\_id"), hydratorExceptionCounter),

hydratePerspective = // Don't cache with the tweet because it depends on the request

PerspectiveHydrator(

repo = perspectiveRepo,

shouldHydrateBookmarksPerspective = deciderGates.hydrateBookmarksPerspective,

stats = hydratorStats.scope("perspective\_by\_safety\_label")

).observe(hydratorStats.scope("perspective"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydratePerspectives),

hydrateEditPerspective = EditPerspectiveHydrator(

repo = perspectiveRepo,

timelinesGate = deciderGates.hydratePerspectivesEditsForTimelines,

tweetDetailsGate = deciderGates.hydratePerspectivesEditsForTweetDetail,

otherSafetyLevelsGate = deciderGates.hydratePerspectivesEditsForOtherSafetyLevels,

bookmarksGate = deciderGates.hydrateBookmarksPerspective,

stats = hydratorStats

).observe(hydratorStats.scope("edit\_perspective"), hydratorExceptionCounter),

hydrateConversationMuted = // Don't cache because it depends on the request. If

// possible, this hydrator should be in the same stage as

// PerspectiveHydrator, so that the calls can be batched

// together.

ConversationMutedHydrator(conversationMutedRepo)

.observe(hydratorStats.scope("conversation\_muted"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateConversationMuted),

hydrateContributor = ContributorHydrator

.once(ContributorHydrator(userIdentityRepo))

.observe(hydratorStats.scope("contributors"), hydratorExceptionCounter),

hydrateTakedowns = TakedownHydrator(takedownRepo)

.observe(hydratorStats.scope("takedowns"), hydratorExceptionCounter),

hydrateDirectedAt = scoped(hydratorStats, "directed\_at") { stats =>

DirectedAtHydrator

.once(DirectedAtHydrator(userIdentityRepo, stats))

.observe(stats, hydratorExceptionCounter)

},

hydrateGeoScrub = GeoScrubHydrator(

geoScrubTimestampRepo,

Scribe("test\_tweetypie\_read\_time\_geo\_scrubs")

.contramap[TweetId](\_.toString)

).observe(hydratorStats.scope("geo\_scrub"), hydratorExceptionCounter),

hydrateCacheableRepairs = ValueHydrator

.fromMutation[Tweet, TweetQuery.Options](

RepairMutation(

repairStats.scope("on\_read"),

"created\_at" ->

new CreatedAtRepairer(Scribe("test\_tweetypie\_bad\_created\_at")),

"retweet\_media" -> RetweetMediaRepairer,

"parent\_status\_id" -> RetweetParentStatusIdRepairer.tweetMutation,

"visible\_text\_range" -> NegativeVisibleTextRangeRepairer.tweetMutation

)

)

.lensed(TweetData.Lenses.tweet)

.onlyIf((td, opts) => opts.cause.reading(td.tweet.id)),

hydrateMediaUncacheable = MediaEntityHydrator

.Uncacheable(

hydrateMediaKey = MediaKeyHydrator()

.observe(hydratorStats.scope("media\_key"), hydratorExceptionCounter),

hydrateMediaInfo = scoped(hydratorStats, "media\_info") { stats =>

MediaInfoHydrator(mediaMetadataRepo, stats)

.observe(stats, hydratorExceptionCounter)

}

)

.observe(hydratorStats.scope("media\_uncacheable"), hydratorExceptionCounter)

.liftSeq

.ifEnabled(deciderGates.hydrateMedia),

hydratePostCacheRepairs =

// clean-up partially hydrated entities before any of the hydrators that look at

// url and media entities run, so that they never see bad entities.

ValueHydrator.fromMutation[TweetData, TweetQuery.Options](

RepairMutation(

repairStats.scope("on\_read"),

"partial\_entity\_cleanup" -> PartialEntityCleaner(repairStats),

"strip\_not\_display\_coords" -> StripHiddenGeoCoordinates

).lensed(TweetData.Lenses.tweet)

),

hydrateTweetLegacyFormat = scoped(hydratorStats, "tweet\_legacy\_formatter") { stats =>

TweetLegacyFormatter(stats)

.observe(stats, hydratorExceptionCounter)

.onlyIf((td, opts) => opts.cause.reading(td.tweet.id))

},

hydrateQuoteTweetVisibility = QuoteTweetVisibilityHydrator(quotedTweetVisibilityRepo)

.observe(hydratorStats.scope("quote\_tweet\_visibility"), hydratorExceptionCounter),

hydrateQuotedTweet = QuotedTweetHydrator(tweetResultRepo)

.observe(hydratorStats.scope("quoted\_tweet"), hydratorExceptionCounter),

hydratePastedMedia =

// Don't cache with the tweet because we want to automatically drop this media if

// the referenced tweet is deleted or becomes non-public.

PastedMediaHydrator(pastedMediaRepo)

.observe(hydratorStats.scope("pasted\_media"))

.ifEnabled(deciderGates.hydratePastedMedia),

hydrateMediaRefs = MediaRefsHydrator(

optionalTweetRepo,

deciderGates.mediaRefsHydratorIncludePastedMedia

).observe(hydratorStats.scope("media\_refs"))

.ifEnabled(deciderGates.hydrateMediaRefs),

hydrateMediaTags = // depends on AdditionalFieldsHydrator

MediaTagsHydrator(userViewRepo)

.observe(hydratorStats.scope("media\_tags"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateMediaTags),

hydrateClassicCards = CardHydrator(cardRepo)

.observe(hydratorStats.scope("cards"), hydratorExceptionCounter),

hydrateCard2 = Card2Hydrator(card2Repo)

.observe(hydratorStats.scope("card2")),

hydrateContributorVisibility =

// Filter out contributors field for all but the user who owns the tweet

ContributorVisibilityFilter()

.observe(hydratorStats.scope("contributor\_visibility"), hydratorExceptionCounter),

hydrateHasMedia =

// Sets hasMedia. Comes after PastedMediaHydrator in order to include pasted

// pics as well as other media & urls.

HasMediaHydrator(hasMedia)

.observe(hydratorStats.scope("has\_media"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateHasMedia),

hydrateTweetCounts = // Don't cache counts with the tweet because it has its own cache with

// a different TTL

TweetCountsHydrator(tweetCountsRepo, deciderGates.hydrateBookmarksCount)

.observe(hydratorStats.scope("tweet\_counts"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateCounts),

hydratePreviousTweetCounts = // previous counts are not cached

scoped(hydratorStats, "previous\_counts") { stats =>

PreviousTweetCountsHydrator(tweetCountsRepo, deciderGates.hydrateBookmarksCount)

.observe(stats, hydratorExceptionCounter)

.ifEnabled(deciderGates.hydratePreviousCounts)

},

hydratePlace =

// Don't cache with the tweet because Place has its own tweetypie cache keyspace

// with a different TTL, and it's more efficient to store separately.

// See com.twitter.tweetypie.repository.PlaceKey

PlaceHydrator(placeRepo)

.observe(hydratorStats.scope("place"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydratePlaces),

hydrateDeviceSource = // Don't cache with the tweet because it has its own cache,

// and it's more efficient to cache it separately

DeviceSourceHydrator(deviceSourceRepo)

.observe(hydratorStats.scope("device\_source"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateDeviceSources),

hydrateProfileGeo =

// Don't cache gnip profile geo as read request volume is expected to be low

ProfileGeoHydrator(profileGeoRepo)

.observe(hydratorStats.scope("profile\_geo"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateGnipProfileGeoEnrichment),

hydrateSourceTweet = scoped(hydratorStats, "source\_tweet") { stats =>

SourceTweetHydrator(

tweetResultRepo,

stats,

FutureEffect

.inParallel(

Scribe(DetachedRetweet, "tweetypie\_detached\_retweets"),

Scribe(DetachedRetweet, "test\_tweetypie\_detached\_retweets"),

)

).observe(stats, hydratorExceptionCounter)

},

hydrateIM1837State = IM1837FilterHydrator()

.observe(hydratorStats.scope("im1837\_filter"), hydratorExceptionCounter)

.onlyIf { (\_, ctx) =>

ctx.opts.forExternalConsumption && ctx.opts.cause.reading(ctx.tweetId)

},

hydrateIM2884State = scoped(hydratorStats, "im2884\_filter") { stats =>

IM2884FilterHydrator(stats)

.observe(stats, hydratorExceptionCounter)

.onlyIf { (\_, ctx) =>

ctx.opts.forExternalConsumption && ctx.opts.cause.reading(ctx.tweetId)

}

},

hydrateIM3433State = scoped(hydratorStats, "im3433\_filter") { stats =>

IM3433FilterHydrator(stats)

.observe(stats, hydratorExceptionCounter)

.onlyIf { (\_, ctx) =>

ctx.opts.forExternalConsumption && ctx.opts.cause.reading(ctx.tweetId)

}

},

hydrateTweetAuthorVisibility = TweetAuthorVisibilityHydrator(userVisibilityRepo)

.observe(hydratorStats.scope("tweet\_author\_visibility"), hydratorExceptionCounter)

.onlyIf((\_, ctx) => ctx.opts.cause.reading(ctx.tweetId)),

hydrateReportedTweetVisibility = ReportedTweetFilter()

.observe(hydratorStats.scope("reported\_tweet\_filter"), hydratorExceptionCounter),

scrubSuperfluousUrlEntities = ValueHydrator

.fromMutation[Tweet, TweetQuery.Options](SuperfluousUrlEntityScrubber.mutation)

.lensed(TweetData.Lenses.tweet),

copyFromSourceTweet = CopyFromSourceTweet.hydrator

.observe(hydratorStats.scope("copy\_from\_source\_tweet"), hydratorExceptionCounter),

hydrateTweetVisibility = scoped(hydratorStats, "tweet\_visibility") { stats =>

TweetVisibilityHydrator(

tweetVisibilityRepo,

deciderGates.failClosedInVF,

stats

).observe(stats, hydratorExceptionCounter)

},

hydrateEscherbirdAnnotations = EscherbirdAnnotationHydrator(escherbirdAnnotationRepo)

.observe(hydratorStats.scope("escherbird\_annotations"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateEscherbirdAnnotations),

hydrateScrubEngagements = ScrubEngagementHydrator()

.observe(hydratorStats.scope("scrub\_engagements"), hydratorExceptionCounter)

.ifEnabled(deciderGates.hydrateScrubEngagements),

hydrateConversationControl = scoped(hydratorStats, "tweet\_conversation\_control") { stats =>

ConversationControlHydrator(

conversationControlRepo,

deciderGates.disableInviteViaMention,

stats

).observe(stats, hydratorExceptionCounter)

},

hydrateEditControl = scoped(hydratorStats, "tweet\_edit\_control") { stats =>

EditControlHydrator(

tweetRepo,

deciderGates.setEditTimeWindowToSixtyMinutes,

stats

).observe(stats, hydratorExceptionCounter)

},

hydrateUnmentionData = UnmentionDataHydrator(),

hydrateNoteTweetSuffix = NoteTweetSuffixHydrator().observe(stats, hydratorExceptionCounter)

)

new TweetHydrators {

val hydrator: TweetDataValueHydrator =

tweetHydrator.onlyIf { (tweetData, opts) =>

// When the caller requests fetchStoredTweets and Tweets are fetched from Manhattan

// irrespective of state, the stored data for some Tweets may be incomplete.

// We skip the hydration of those Tweets.

!opts.fetchStoredTweets ||

tweetData.storedTweetResult.exists(\_.canHydrate)

}

val cacheChangesEffect: Effect[ValueState[TweetData]] =

TweetHydration.cacheChanges(

tweetDataCache,

hydratorStats.scope("tweet\_caching")

)

}

}

}