package com.twitter.tweetypie

package hydrator

import com.twitter.featureswitches.v2.FeatureSwitchResults

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

import scala.collection.mutable

object TweetCountsHydrator {

type Type = ValueHydrator[Option[StatusCounts], Ctx]

case class Ctx(featureSwitchResults: Option[FeatureSwitchResults], underlyingTweetCtx: TweetCtx)

extends TweetCtx.Proxy

val retweetCountField: FieldByPath =

fieldByPath(Tweet.CountsField, StatusCounts.RetweetCountField)

val replyCountField: FieldByPath = fieldByPath(Tweet.CountsField, StatusCounts.ReplyCountField)

val favoriteCountField: FieldByPath =

fieldByPath(Tweet.CountsField, StatusCounts.FavoriteCountField)

val quoteCountField: FieldByPath = fieldByPath(Tweet.CountsField, StatusCounts.QuoteCountField)

val bookmarkCountField: FieldByPath =

fieldByPath(Tweet.CountsField, StatusCounts.BookmarkCountField)

val emptyCounts = StatusCounts()

val retweetCountPartial = ValueState.partial(emptyCounts, retweetCountField)

val replyCountPartial = ValueState.partial(emptyCounts, replyCountField)

val favoriteCountPartial = ValueState.partial(emptyCounts, favoriteCountField)

val quoteCountPartial = ValueState.partial(emptyCounts, quoteCountField)

val bookmarkCountPartial = ValueState.partial(emptyCounts, bookmarkCountField)

val bookmarksCountHydrationEnabledKey = "bookmarks\_count\_hydration\_enabled"

/\*\*

\* Take a Seq of StatusCounts and reduce down to a single StatusCounts.

\* Note: `reduce` here is safe because we are guaranteed to always have at least

\* one value.

\*/

def reduceStatusCounts(counts: Seq[StatusCounts]): StatusCounts =

counts.reduce { (a, b) =>

StatusCounts(

retweetCount = b.retweetCount.orElse(a.retweetCount),

replyCount = b.replyCount.orElse(a.replyCount),

favoriteCount = b.favoriteCount.orElse(a.favoriteCount),

quoteCount = b.quoteCount.orElse(a.quoteCount),

bookmarkCount = b.bookmarkCount.orElse(a.bookmarkCount)

)

}

def toKeys(

tweetId: TweetId,

countsFields: Set[FieldId],

curr: Option[StatusCounts]

): Seq[TweetCountKey] = {

val keys = new mutable.ArrayBuffer[TweetCountKey](4)

countsFields.foreach {

case StatusCounts.RetweetCountField.id =>

if (curr.flatMap(\_.retweetCount).isEmpty)

keys += RetweetsKey(tweetId)

case StatusCounts.ReplyCountField.id =>

if (curr.flatMap(\_.replyCount).isEmpty)

keys += RepliesKey(tweetId)

case StatusCounts.FavoriteCountField.id =>

if (curr.flatMap(\_.favoriteCount).isEmpty)

keys += FavsKey(tweetId)

case StatusCounts.QuoteCountField.id =>

if (curr.flatMap(\_.quoteCount).isEmpty)

keys += QuotesKey(tweetId)

case StatusCounts.BookmarkCountField.id =>

if (curr.flatMap(\_.bookmarkCount).isEmpty)

keys += BookmarksKey(tweetId)

case \_ =>

}

keys

}

/\*

\* Get a StatusCounts object for a specific tweet and specific field (e.g. only fav, or reply etc).

\* StatusCounts returned from here can be combined with other StatusCounts using `sumStatusCount`

\*/

def statusCountsRepo(

key: TweetCountKey,

repo: TweetCountsRepository.Type

): Stitch[ValueState[StatusCounts]] =

repo(key).liftToTry.map {

case Return(count) =>

ValueState.modified(

key match {

case \_: RetweetsKey => StatusCounts(retweetCount = Some(count))

case \_: RepliesKey => StatusCounts(replyCount = Some(count))

case \_: FavsKey => StatusCounts(favoriteCount = Some(count))

case \_: QuotesKey => StatusCounts(quoteCount = Some(count))

case \_: BookmarksKey => StatusCounts(bookmarkCount = Some(count))

}

)

case Throw(\_) =>

key match {

case \_: RetweetsKey => retweetCountPartial

case \_: RepliesKey => replyCountPartial

case \_: FavsKey => favoriteCountPartial

case \_: QuotesKey => quoteCountPartial

case \_: BookmarksKey => bookmarkCountPartial

}

}

def filterRequestedCounts(

userId: UserId,

requestedCounts: Set[FieldId],

bookmarkCountsDecider: Gate[Long],

featureSwitchResults: Option[FeatureSwitchResults]

): Set[FieldId] = {

if (requestedCounts.contains(StatusCounts.BookmarkCountField.id))

if (bookmarkCountsDecider(userId) ||

featureSwitchResults

.flatMap(\_.getBoolean(bookmarksCountHydrationEnabledKey, false))

.getOrElse(false))

requestedCounts

else

requestedCounts.filter(\_ != StatusCounts.BookmarkCountField.id)

else

requestedCounts

}

def apply(repo: TweetCountsRepository.Type, shouldHydrateBookmarksCount: Gate[Long]): Type = {

val all: Set[FieldId] = StatusCounts.fieldInfos.map(\_.tfield.id).toSet

val modifiedZero: Map[Set[FieldId], ValueState[Some[StatusCounts]]] = {

for (set <- all.subsets) yield {

@inline

def zeroOrNone(fieldId: FieldId) =

if (set.contains(fieldId)) Some(0L) else None

val statusCounts =

StatusCounts(

retweetCount = zeroOrNone(StatusCounts.RetweetCountField.id),

replyCount = zeroOrNone(StatusCounts.ReplyCountField.id),

favoriteCount = zeroOrNone(StatusCounts.FavoriteCountField.id),

quoteCount = zeroOrNone(StatusCounts.QuoteCountField.id),

bookmarkCount = zeroOrNone(StatusCounts.BookmarkCountField.id)

)

set -> ValueState.modified(Some(statusCounts))

}

}.toMap

ValueHydrator[Option[StatusCounts], Ctx] { (curr, ctx) =>

val countsFields: Set[FieldId] = filterRequestedCounts(

ctx.opts.forUserId.getOrElse(ctx.userId),

ctx.opts.include.countsFields,

shouldHydrateBookmarksCount,

ctx.featureSwitchResults

)

if (ctx.isRetweet) {

// To avoid a reflection-induced key error where the countsFields can contain a fieldId

// that is not in the thrift schema loaded at start, we strip unknown field\_ids using

// `intersect`

Stitch.value(modifiedZero(countsFields.intersect(all)))

} else {

val keys = toKeys(ctx.tweetId, countsFields, curr)

Stitch.traverse(keys)(key => statusCountsRepo(key, repo)).map { results =>

// always flag modified if starting from None

val vs0 = ValueState.success(curr.getOrElse(emptyCounts), curr.isEmpty)

val vs = vs0 +: results

ValueState.sequence(vs).map(reduceStatusCounts).map(Some(\_))

}

}

}.onlyIf { (\_, ctx) =>

filterRequestedCounts(

ctx.opts.forUserId.getOrElse(ctx.userId),

ctx.opts.include.countsFields,

shouldHydrateBookmarksCount,

ctx.featureSwitchResults

).nonEmpty

}

}

}