package com.twitter.tweetypie

package hydrator

import com.twitter.spam.rtf.thriftscala.SafetyLevel

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.\_

import com.twitter.tweetypie.media.MediaUrl

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

object PastedMediaHydrator {

type Type = ValueHydrator[PastedMedia, Ctx]

/\*\*

\* Ensure that the final tweet has at most 4 media entities.

\*/

val MaxMediaEntitiesPerTweet = 4

/\*\*

\* Enforce visibility rules when hydrating media for a write.

\*/

val writeSafetyLevel = SafetyLevel.TweetWritesApi

case class Ctx(urlEntities: Seq[UrlEntity], underlyingTweetCtx: TweetCtx) extends TweetCtx.Proxy {

def includePastedMedia: Boolean = opts.include.pastedMedia

def includeMediaEntities: Boolean = tweetFieldRequested(Tweet.MediaField)

def includeAdditionalMetadata: Boolean =

mediaFieldRequested(MediaEntity.AdditionalMetadataField.id)

def includeMediaTags: Boolean = tweetFieldRequested(Tweet.MediaTagsField)

}

def getPastedMedia(t: Tweet): PastedMedia = PastedMedia(getMedia(t), Map.empty)

def apply(repo: PastedMediaRepository.Type): Type = {

def hydrateOneReference(

tweetId: TweetId,

urlEntity: UrlEntity,

repoCtx: PastedMediaRepository.Ctx

): Stitch[PastedMedia] =

repo(tweetId, repoCtx).liftToTry.map {

case Return(pastedMedia) => pastedMedia.updateEntities(urlEntity)

case \_ => PastedMedia.empty

}

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

val repoCtx = asRepoCtx(ctx)

val idsAndEntities = pastedIdsAndEntities(ctx.tweetId, ctx.urlEntities)

val res = Stitch.traverse(idsAndEntities) {

case (tweetId, urlEntity) =>

hydrateOneReference(tweetId, urlEntity, repoCtx)

}

res.liftToTry.map {

case Return(pastedMedias) =>

val merged = pastedMedias.foldLeft(curr)(\_.merge(\_))

val limited = merged.take(MaxMediaEntitiesPerTweet)

ValueState.delta(curr, limited)

case Throw(\_) => ValueState.unmodified(curr)

}

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

// we only attempt to hydrate pasted media if media is requested

ctx.includePastedMedia &&

!ctx.isRetweet &&

ctx.includeMediaEntities

}

}

/\*\*

\* Finds url entities for foreign permalinks, and returns a sequence of tuples containing

\* the foreign tweet IDs and the associated UrlEntity containing the permalink. If the same

\* permalink appears multiple times, only one of the duplicate entities is returned.

\*/

def pastedIdsAndEntities(

tweetId: TweetId,

urlEntities: Seq[UrlEntity]

): Seq[(TweetId, UrlEntity)] =

urlEntities

.foldLeft(Map.empty[TweetId, UrlEntity]) {

case (z, e) =>

MediaUrl.Permalink.getTweetId(e).filter(\_ != tweetId) match {

case Some(id) if !z.contains(id) => z + (id -> e)

case \_ => z

}

}

.toSeq

def asRepoCtx(ctx: Ctx) =

PastedMediaRepository.Ctx(

ctx.includeMediaEntities,

ctx.includeAdditionalMetadata,

ctx.includeMediaTags,

ctx.opts.extensionsArgs,

if (ctx.opts.cause == TweetQuery.Cause.Insert(ctx.tweetId) ||

ctx.opts.cause == TweetQuery.Cause.Undelete(ctx.tweetId)) {

writeSafetyLevel

} else {

ctx.opts.safetyLevel

}

)

}