package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.NotFound

import com.twitter.stitch.Stitch

import com.twitter.tweetutil.TweetPermalink

import com.twitter.tweetypie.core.FilteredState

import com.twitter.tweetypie.core.ValueState

import com.twitter.tweetypie.repository.\_

import com.twitter.tweetypie.thriftscala.\_

/\*\*

\* Adds QuotedTweet structs to tweets that contain a tweet permalink url at the end of the

\* tweet text. After introduction of QT + Media, we stopped storing inner tweet permalinks

\* in the outer tweet text. So this hydrator would run only for below cases:

\*

\* - historical quote tweets which have inner tweet url in the tweet text and url entities.

\* - new quote tweets created with pasted tweet permalinks, going forward we want to persist

\* quoted\_tweet struct in MH for these tweets

\*/

object QuotedTweetRefHydrator {

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

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

val hydratedField: FieldByPath = fieldByPath(Tweet.QuotedTweetField)

private val partial = ValueState.partial(None, hydratedField)

val queryOptions: TweetQuery.Options =

TweetQuery.Options(

include = TweetQuery.Include(Set(Tweet.CoreDataField.id)),

// Don't enforce visibility filtering when loading the QuotedTweet struct because it is

// cacheable. The filtering happens in QuoteTweetVisibilityHydrator.

enforceVisibilityFiltering = false,

forUserId = None

)

def once(h: Type): Type =

TweetHydration.completeOnlyOnce(

queryFilter = queryFilter,

hydrationType = HydrationType.QuotedTweetRef,

dependsOn = Set(HydrationType.Urls),

hydrator = h

)

case class UrlHydrationFailed(url: String) extends Exception

/\*\*

\* Iterate through UrlEntity objects in reverse to identify a quoted-tweet ID

\* to hydrate. Quoted tweets are indicated by a TweetPermalink in the tweet text

\* that references an older tweet ID. If a quoted tweet permalink is found, also

\* return the corresponding UrlEntity.

\*

\* @throws UrlHydrationFailed if we encounter a partial URL entity before

\* finding a tweet permalink URL.

\*/

def quotedTweetId(ctx: Ctx): Option[(UrlEntity, TweetId)] =

ctx.urlEntities.reverseIterator // we want the rightmost tweet permalink

.map { e: UrlEntity =>

if (UrlEntityHydrator.hydrationFailed(e)) throw UrlHydrationFailed(e.url)

else (e, e.expanded)

}

.collectFirst {

case (e, Some(TweetPermalink(\_, quotedTweetId))) => (e, quotedTweetId)

}

// Prevent tweet-quoting cycles

.filter { case (\_, quotedTweetId) => ctx.tweetId > quotedTweetId }

def buildShortenedUrl(e: UrlEntity): ShortenedUrl =

ShortenedUrl(

shortUrl = e.url,

// Reading from MH will also default the following to "".

// QuotedTweetRefUrlsHydrator will hydrate these cases

longUrl = e.expanded.getOrElse(""),

displayText = e.display.getOrElse("")

)

/\*\*

\* We run this hydrator only if:

\*

\* - quoted\_tweet struct is empty

\* - quoted\_tweet is present but permalink is not

\* - url entities is present. QT hydration depends on urls - long term goal

\* is to entirely rely on persisted quoted\_tweet struct in MH

\* - requested tweet is not a retweet

\*

\* Hydration steps:

\* - We determine the last tweet permalink from url entities

\* - Extract the inner tweet Id from the permalink

\* - Query tweet repo with inner tweet Id

\* - Construct quoted\_tweet struct from hydrated tweet object and last permalink

\*/

def apply(repo: TweetRepository.Type): Type =

ValueHydrator[Option[QuotedTweet], Ctx] { (\_, ctx) =>

// propagate errors from quotedTweetId in Stitch

Stitch(quotedTweetId(ctx)).liftToTry.flatMap {

case Return(Some((lastPermalinkEntity, quotedTweetId))) =>

repo(quotedTweetId, queryOptions).liftToTry.map {

case Return(tweet) =>

ValueState.modified(

Some(asQuotedTweet(tweet, lastPermalinkEntity))

)

case Throw(NotFound | \_: FilteredState) => ValueState.UnmodifiedNone

case Throw(\_) => partial

}

case Return(None) => Stitch(ValueState.UnmodifiedNone)

case Throw(\_) => Stitch(partial)

}

}.onlyIf { (curr, ctx) =>

(curr.isEmpty || curr.exists(\_.permalink.isEmpty)) &&

!ctx.isRetweet && ctx.urlEntities.nonEmpty

}

def queryFilter(opts: TweetQuery.Options): Boolean =

opts.include.tweetFields(Tweet.QuotedTweetField.id)

/\*\*

\* We construct Tweet.quoted\_tweet from hydrated inner tweet.

\* Note: if the inner tweet is a Retweet, we populate the quoted\_tweet struct from source tweet.

\*/

def asQuotedTweet(tweet: Tweet, entity: UrlEntity): QuotedTweet = {

val shortenedUrl = Some(buildShortenedUrl(entity))

getShare(tweet) match {

case None => QuotedTweet(tweet.id, getUserId(tweet), shortenedUrl)

case Some(share) => QuotedTweet(share.sourceStatusId, share.sourceUserId, shortenedUrl)

}

}

}