package com.twitter.tweetypie

package hydrator

import com.twitter.stitch.Stitch

import com.twitter.tweetypie.core.ValueState

import com.twitter.tweetypie.repository.ConversationControlRepository

import com.twitter.tweetypie.serverutil.ExceptionCounter

import com.twitter.tweetypie.thriftscala.ConversationControl

private object ReplyTweetConversationControlHydrator {

type Type = ConversationControlHydrator.Type

type Ctx = ConversationControlHydrator.Ctx

// The conversation control thrift field was added Feb 17th, 2020.

// No conversation before this will have a conversation control field to hydrate.

// We explicitly short circuit to save resources from querying for tweets we

// know do not have conversation control fields set.

val FirstValidDate: Time = Time.fromMilliseconds(1554076800000L) // 2020-02-17

def apply(

repo: ConversationControlRepository.Type,

stats: StatsReceiver

): Type = {

val exceptionCounter = ExceptionCounter(stats)

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

repo(ctx.conversationId.get, ctx.opts.cacheControl).liftToTry.map {

case Return(conversationControl) =>

ValueState.delta(curr, conversationControl)

case Throw(exception) => {

// In the case where we get an exception, we want to count the

// exception but fail open.

exceptionCounter(exception)

// Reply Tweet Tweet.ConversationControlField hydration should fail open.

// Ideally we would return ValueState.partial here to notify Tweetypie the caller

// that requested the Tweet.ConversationControlField field was not hydrated.

// We cannot do so because GetTweetFields will return TweetFieldsResultFailed

// for partial results which would fail closed.

ValueState.unmodified(curr)

}

}

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

// This hydrator is specifically for replies so only run when Tweet is a reply

ctx.inReplyToTweetId.isDefined &&

// See comment for FirstValidDate

ctx.createdAt > FirstValidDate &&

// We need conversation id to get ConversationControl

ctx.conversationId.isDefined &&

// Only run if the ConversationControl was requested

ctx.tweetFieldRequested(Tweet.ConversationControlField)

}

}

}

/\*\*

\* ConversationControlHydrator is used to hydrate the conversationControl field.

\* For root Tweets, this hydrator just passes through the existing conversationControl.

\* For reply Tweets, it loads the conversationControl from the root Tweet of the conversation.

\* Only root Tweets in a conversation (i.e. the Tweet pointed to by conversationId) have

\* a persisted conversationControl, so we have to hydrate that field for all replies in order

\* to know if a Tweet in a conversation can be replied to.

\*/

object ConversationControlHydrator {

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

case class Ctx(conversationId: Option[ConversationId], underlyingTweetCtx: TweetCtx)

extends TweetCtx.Proxy

private def scrubInviteViaMention(

ccOpt: Option[ConversationControl]

): Option[ConversationControl] = {

ccOpt collect {

case ConversationControl.ByInvitation(byInvitation) =>

ConversationControl.ByInvitation(byInvitation.copy(inviteViaMention = None))

case ConversationControl.Community(community) =>

ConversationControl.Community(community.copy(inviteViaMention = None))

case ConversationControl.Followers(followers) =>

ConversationControl.Followers(followers.copy(inviteViaMention = None))

}

}

def apply(

repo: ConversationControlRepository.Type,

disableInviteViaMention: Gate[Unit],

stats: StatsReceiver

): Type = {

val replyTweetConversationControlHydrator = ReplyTweetConversationControlHydrator(

repo,

stats

)

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

val ccUpdated = if (disableInviteViaMention()) {

scrubInviteViaMention(curr)

} else {

curr

}

if (ctx.inReplyToTweetId.isEmpty) {

// For non-reply tweets, pass through the existing conversation control

Stitch.value(ValueState.delta(curr, ccUpdated))

} else {

replyTweetConversationControlHydrator(ccUpdated, ctx)

}

}

}

}