package com.twitter.tweetypie

package hydrator

import com.twitter.featureswitches.v2.FeatureSwitchResults

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

import com.twitter.stitch.Stitch

import com.twitter.stitch.timelineservice.TimelineService.GetPerspectives.Query

import com.twitter.tweetypie.core.ValueState

import com.twitter.tweetypie.repository.PerspectiveRepository

import com.twitter.tweetypie.thriftscala.EditControl

import com.twitter.tweetypie.thriftscala.FieldByPath

import com.twitter.tweetypie.thriftscala.StatusPerspective

import com.twitter.tweetypie.thriftscala.TweetPerspective

object EditPerspectiveHydrator {

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

val HydratedField: FieldByPath = fieldByPath(Tweet.EditPerspectiveField)

case class Ctx(

currentTweetPerspective: Option[StatusPerspective],

editControl: Option[EditControl],

featureSwitchResults: Option[FeatureSwitchResults],

underlyingTweetCtx: TweetCtx)

extends TweetCtx.Proxy

// Timeline safety levels determine some part of high level traffic

// that we might want to turn off with a decider if edits traffic

// is too big for perspectives to handle. The decider allows us

// to turn down the traffic without the impact on tweet detail.

val TimelinesSafetyLevels: Set[SafetyLevel] = Set(

SafetyLevel.TimelineFollowingActivity,

SafetyLevel.TimelineHome,

SafetyLevel.TimelineConversations,

SafetyLevel.DeprecatedTimelineConnect,

SafetyLevel.TimelineMentions,

SafetyLevel.DeprecatedTimelineActivity,

SafetyLevel.TimelineFavorites,

SafetyLevel.TimelineLists,

SafetyLevel.TimelineInjection,

SafetyLevel.StickersTimeline,

SafetyLevel.LiveVideoTimeline,

SafetyLevel.QuoteTweetTimeline,

SafetyLevel.TimelineHomeLatest,

SafetyLevel.TimelineLikedBy,

SafetyLevel.TimelineRetweetedBy,

SafetyLevel.TimelineBookmark,

SafetyLevel.TimelineMedia,

SafetyLevel.TimelineReactiveBlending,

SafetyLevel.TimelineProfile,

SafetyLevel.TimelineFocalTweet,

SafetyLevel.TimelineHomeRecommendations,

SafetyLevel.NotificationsTimelineDeviceFollow,

SafetyLevel.TimelineConversationsDownranking,

SafetyLevel.TimelineHomeTopicFollowRecommendations,

SafetyLevel.TimelineHomeHydration,

SafetyLevel.FollowedTopicsTimeline,

SafetyLevel.ModeratedTweetsTimeline,

SafetyLevel.TimelineModeratedTweetsHydration,

SafetyLevel.ElevatedQuoteTweetTimeline,

SafetyLevel.TimelineConversationsDownrankingMinimal,

SafetyLevel.BirdwatchNoteTweetsTimeline,

SafetyLevel.TimelineSuperLikedBy,

SafetyLevel.UserScopedTimeline,

SafetyLevel.TweetScopedTimeline,

SafetyLevel.TimelineHomePromotedHydration,

SafetyLevel.NearbyTimeline,

SafetyLevel.TimelineProfileAll,

SafetyLevel.TimelineProfileSuperFollows,

SafetyLevel.SpaceTweetAvatarHomeTimeline,

SafetyLevel.SpaceHomeTimelineUpranking,

SafetyLevel.BlockMuteUsersTimeline,

SafetyLevel.RitoActionedTweetTimeline,

SafetyLevel.TimelineScorer,

SafetyLevel.ArticleTweetTimeline,

SafetyLevel.DesQuoteTweetTimeline,

SafetyLevel.EditHistoryTimeline,

SafetyLevel.DirectMessagesConversationTimeline,

SafetyLevel.DesHomeTimeline,

SafetyLevel.TimelineContentControls,

SafetyLevel.TimelineFavoritesSelfView,

SafetyLevel.TimelineProfileSpaces,

)

val TweetDetailSafetyLevels: Set[SafetyLevel] = Set(

SafetyLevel.TweetDetail,

SafetyLevel.TweetDetailNonToo,

SafetyLevel.TweetDetailWithInjectionsHydration,

SafetyLevel.DesTweetDetail,

)

def apply(

repo: PerspectiveRepository.Type,

timelinesGate: Gate[Unit],

tweetDetailsGate: Gate[Unit],

otherSafetyLevelsGate: Gate[Unit],

bookmarksGate: Gate[Long],

stats: StatsReceiver

): Type = {

val statsByLevel =

SafetyLevel.list.map { level =>

(level, stats.counter("perspective\_by\_safety\_label", level.name, "calls"))

}.toMap

val editsAggregated = stats.counter("edit\_perspective", "edits\_aggregated")

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

val safetyLevel = ctx.opts.safetyLevel

val lookupsDecider =

if (TimelinesSafetyLevels.contains(safetyLevel)) timelinesGate

else if (TweetDetailSafetyLevels.contains(safetyLevel)) tweetDetailsGate

else otherSafetyLevelsGate

val tweetIds: Seq[TweetId] = if (lookupsDecider()) tweetIdsToAggregate(ctx).toSeq else Seq()

statsByLevel

.getOrElse(

safetyLevel,

stats.counter("perspective\_by\_safety\_label", safetyLevel.name, "calls"))

.incr(tweetIds.size)

editsAggregated.incr(tweetIds.size)

Stitch

.traverse(tweetIds) { id =>

repo(

Query(

ctx.opts.forUserId.get,

id,

PerspectiveHydrator.evaluatePerspectiveTypes(

ctx.opts.forUserId.get,

bookmarksGate,

ctx.featureSwitchResults))).liftToTry

}.map { seq =>

if (seq.isEmpty) {

val editPerspective = ctx.currentTweetPerspective.map { c =>

TweetPerspective(

c.favorited,

c.retweeted,

c.bookmarked

)

}

ValueState.delta(curr, editPerspective)

} else {

val returns = seq.collect { case Return(r) => r }

val aggregate = Some(

TweetPerspective(

favorited =

returns.exists(\_.favorited) || ctx.currentTweetPerspective.exists(\_.favorited),

retweeted =

returns.exists(\_.retweeted) || ctx.currentTweetPerspective.exists(\_.retweeted),

bookmarked = Some(

returns.exists(\_.bookmarked.contains(true)) || ctx.currentTweetPerspective.exists(

\_.bookmarked.contains(true)))

)

)

if (seq.exists(\_.isThrow)) {

ValueState.partial(aggregate, HydratedField)

} else {

ValueState.modified(aggregate)

}

}

}

}.onlyIf { (curr, ctx) =>

curr.isEmpty &&

ctx.opts.forUserId.isDefined &&

ctx.tweetFieldRequested(Tweet.EditPerspectiveField)

}

}

private def tweetIdsToAggregate(ctx: Ctx): Set[TweetId] = {

ctx.editControl

.flatMap {

case EditControl.Initial(initial) => Some(initial)

case EditControl.Edit(edit) => edit.editControlInitial

case \_ => None

}

.map(\_.editTweetIds.toSet)

.getOrElse(Set()) - ctx.tweetId

}

}