package com.twitter.tweetypie

package service

package observer

import com.twitter.escherbird.thriftscala.TweetEntityAnnotation

import com.twitter.tweetypie.thriftscala.BatchComposeMode

import com.twitter.tweetypie.thriftscala.PostTweetRequest

import com.twitter.tweetypie.thriftscala.PostTweetResult

import com.twitter.tweetypie.thriftscala.TweetCreateState

import com.twitter.util.Memoize

private[service] object PostTweetObserver {

def observeResults(stats: StatsReceiver, byClient: Boolean): Effect[PostTweetResult] = {

val stateScope = stats.scope("state")

val tweetObserver = Observer.countTweetAttributes(stats, byClient)

val stateCounters =

Memoize { st: TweetCreateState => stateScope.counter(Observer.camelToUnderscore(st.name)) }

Effect { result =>

stateCounters(result.state).incr()

if (result.state == TweetCreateState.Ok) result.tweet.foreach(tweetObserver)

}

}

private def isCommunity(req: PostTweetRequest): Boolean = {

val CommunityGroupId = 8L

val CommunityDomainId = 31L

req.additionalFields

.flatMap(\_.escherbirdEntityAnnotations).exists { e =>

e.entityAnnotations.collect {

case TweetEntityAnnotation(CommunityGroupId, CommunityDomainId, \_) => true

}.nonEmpty

}

}

def observerRequest(stats: StatsReceiver): Effect[PostTweetRequest] = {

val optionsScope = stats.scope("options")

val narrowcastCounter = optionsScope.counter("narrowcast")

val nullcastCounter = optionsScope.counter("nullcast")

val inReplyToStatusIdCounter = optionsScope.counter("in\_reply\_to\_status\_id")

val placeIdCounter = optionsScope.counter("place\_id")

val geoCoordinatesCounter = optionsScope.counter("geo\_coordinates")

val placeMetadataCounter = optionsScope.counter("place\_metadata")

val mediaUploadIdCounter = optionsScope.counter("media\_upload\_id")

val darkCounter = optionsScope.counter("dark")

val tweetToNarrowcastingCounter = optionsScope.counter("tweet\_to\_narrowcasting")

val autoPopulateReplyMetadataCounter = optionsScope.counter("auto\_populate\_reply\_metadata")

val attachmentUrlCounter = optionsScope.counter("attachment\_url")

val excludeReplyUserIdsCounter = optionsScope.counter("exclude\_reply\_user\_ids")

val excludeReplyUserIdsStat = optionsScope.stat("exclude\_reply\_user\_ids")

val uniquenessIdCounter = optionsScope.counter("uniqueness\_id")

val batchModeScope = optionsScope.scope("batch\_mode")

val batchModeFirstCounter = batchModeScope.counter("first")

val batchModeSubsequentCounter = batchModeScope.counter("subsequent")

val communitiesCounter = optionsScope.counter("communities")

Effect { request =>

if (request.narrowcast.nonEmpty) narrowcastCounter.incr()

if (request.nullcast) nullcastCounter.incr()

if (request.inReplyToTweetId.nonEmpty) inReplyToStatusIdCounter.incr()

if (request.geo.flatMap(\_.placeId).nonEmpty) placeIdCounter.incr()

if (request.geo.flatMap(\_.coordinates).nonEmpty) geoCoordinatesCounter.incr()

if (request.geo.flatMap(\_.placeMetadata).nonEmpty) placeMetadataCounter.incr()

if (request.mediaUploadIds.nonEmpty) mediaUploadIdCounter.incr()

if (request.dark) darkCounter.incr()

if (request.enableTweetToNarrowcasting) tweetToNarrowcastingCounter.incr()

if (request.autoPopulateReplyMetadata) autoPopulateReplyMetadataCounter.incr()

if (request.attachmentUrl.nonEmpty) attachmentUrlCounter.incr()

if (request.excludeReplyUserIds.exists(\_.nonEmpty)) excludeReplyUserIdsCounter.incr()

if (isCommunity(request)) communitiesCounter.incr()

if (request.uniquenessId.nonEmpty) uniquenessIdCounter.incr()

request.transientContext.flatMap(\_.batchCompose).foreach {

case BatchComposeMode.BatchFirst => batchModeFirstCounter.incr()

case BatchComposeMode.BatchSubsequent => batchModeSubsequentCounter.incr()

case \_ =>

}

excludeReplyUserIdsStat.add(request.excludeReplyUserIds.size)

}

}

}