package com.twitter.tweetypie

package service

package observer

import com.twitter.servo.exception.thriftscala.ClientError

import com.twitter.tweetypie.thriftscala.\_

private[service] object GetTweetFieldsObserver {

type Type = ObserveExchange[GetTweetFieldsRequest, Seq[GetTweetFieldsResult]]

def observeExchange(statsReceiver: StatsReceiver): Effect[Type] = {

val resultStateStats = ResultStateStats(statsReceiver)

val stats = statsReceiver.scope("results")

val tweetResultFailed = stats.counter("tweet\_result\_failed")

val quoteResultFailed = stats.counter("quote\_result\_failed")

val overCapacity = stats.counter("over\_capacity")

def observeFailedResult(r: GetTweetFieldsResult): Unit = {

r.tweetResult match {

case TweetFieldsResultState.Failed(failed) =>

tweetResultFailed.incr()

if (failed.overCapacity) overCapacity.incr()

case \_ =>

}

if (r.quotedTweetResult.exists(\_.isInstanceOf[TweetFieldsResultState.Failed]))

quoteResultFailed.incr()

}

Effect {

case (request, response) =>

response match {

case Return(xs) =>

xs foreach {

case x if isFailedResult(x) =>

observeFailedResult(x)

resultStateStats.failed()

case \_ =>

resultStateStats.success()

}

case Throw(ClientError(\_)) =>

resultStateStats.success(request.tweetIds.size)

case Throw(\_) =>

resultStateStats.failed(request.tweetIds.size)

}

}

}

def observeRequest(stats: StatsReceiver, byClient: Boolean): Effect[GetTweetFieldsRequest] = {

val requestSizeStat = stats.stat("request\_size")

val optionsScope = stats.scope("options")

val tweetFieldsScope = optionsScope.scope("tweet\_field")

val countsFieldsScope = optionsScope.scope("counts\_field")

val mediaFieldsScope = optionsScope.scope("media\_field")

val includeRetweetedTweetCounter = optionsScope.counter("include\_retweeted\_tweet")

val includeQuotedTweetCounter = optionsScope.counter("include\_quoted\_tweet")

val forUserIdCounter = optionsScope.counter("for\_user\_id")

val cardsPlatformKeyCounter = optionsScope.counter("cards\_platform\_key")

val cardsPlatformKeyScope = optionsScope.scope("cards\_platform\_key")

val extensionsArgsCounter = optionsScope.counter("extensions\_args")

val doNotCacheCounter = optionsScope.counter("do\_not\_cache")

val simpleQuotedTweetCounter = optionsScope.counter("simple\_quoted\_tweet")

val visibilityPolicyScope = optionsScope.scope("visibility\_policy")

val userVisibleCounter = visibilityPolicyScope.counter("user\_visible")

val noFilteringCounter = visibilityPolicyScope.counter("no\_filtering")

val noSafetyLevelCounter = optionsScope.counter("no\_safety\_level")

val safetyLevelCounter = optionsScope.counter("safety\_level")

val safetyLevelScope = optionsScope.scope("safety\_level")

Effect {

case GetTweetFieldsRequest(tweetIds, options) =>

requestSizeStat.add(tweetIds.size)

options.tweetIncludes.foreach {

case TweetInclude.TweetFieldId(id) => tweetFieldsScope.counter(id.toString).incr()

case TweetInclude.CountsFieldId(id) => countsFieldsScope.counter(id.toString).incr()

case TweetInclude.MediaEntityFieldId(id) => mediaFieldsScope.counter(id.toString).incr()

case \_ =>

}

if (options.includeRetweetedTweet) includeRetweetedTweetCounter.incr()

if (options.includeQuotedTweet) includeQuotedTweetCounter.incr()

if (options.forUserId.nonEmpty) forUserIdCounter.incr()

if (options.cardsPlatformKey.nonEmpty) cardsPlatformKeyCounter.incr()

if (!byClient) {

options.cardsPlatformKey.foreach { cardsPlatformKey =>

cardsPlatformKeyScope.counter(cardsPlatformKey).incr()

}

}

if (options.extensionsArgs.nonEmpty) extensionsArgsCounter.incr()

if (options.safetyLevel.nonEmpty) {

safetyLevelCounter.incr()

} else {

noSafetyLevelCounter.incr()

}

options.visibilityPolicy match {

case TweetVisibilityPolicy.UserVisible => userVisibleCounter.incr()

case TweetVisibilityPolicy.NoFiltering => noFilteringCounter.incr()

case \_ =>

}

options.safetyLevel.foreach { level => safetyLevelScope.counter(level.toString).incr() }

if (options.doNotCache) doNotCacheCounter.incr()

if (options.simpleQuotedTweet) simpleQuotedTweetCounter.incr()

}

}

def observeResults(stats: StatsReceiver): Effect[Seq[GetTweetFieldsResult]] = {

val resultsCounter = stats.counter("results")

val resultsScope = stats.scope("results")

val observeState = GetTweetFieldsObserver.observeResultState(resultsScope)

Effect { results =>

resultsCounter.incr(results.size)

results.foreach { r =>

observeState(r.tweetResult)

r.quotedTweetResult.foreach { qtResult =>

resultsCounter.incr()

observeState(qtResult)

}

}

}

}

/\*\*

\* Given a GetTweetFieldsResult result, do we observe the result as a failure or not.

\*/

private def isFailedResult(result: GetTweetFieldsResult): Boolean = {

result.tweetResult.isInstanceOf[TweetFieldsResultState.Failed] ||

result.quotedTweetResult.exists(\_.isInstanceOf[TweetFieldsResultState.Failed])

}

private def observeResultState(stats: StatsReceiver): Effect[TweetFieldsResultState] = {

val foundCounter = stats.counter("found")

val notFoundCounter = stats.counter("not\_found")

val failedCounter = stats.counter("failed")

val filteredCounter = stats.counter("filtered")

val filteredReasonScope = stats.scope("filtered\_reason")

val otherCounter = stats.counter("other")

val observeTweet = Observer

.countTweetAttributes(stats.scope("found"), byClient = false)

Effect {

case TweetFieldsResultState.Found(found) =>

foundCounter.incr()

observeTweet(found.tweet)

found.retweetedTweet.foreach(observeTweet)

case TweetFieldsResultState.NotFound(\_) => notFoundCounter.incr()

case TweetFieldsResultState.Failed(\_) => failedCounter.incr()

case TweetFieldsResultState.Filtered(f) =>

filteredCounter.incr()

// Since reasons have parameters, eg. AuthorBlockViewer(true) and we don't

// need the "(true)" part, we do .getClass.getSimpleName to get rid of that

filteredReasonScope.counter(f.reason.getClass.getSimpleName).incr()

case \_ => otherCounter.incr()

}

}

}