package com.twitter.unified\_user\_actions.enricher.hydrator

import com.google.common.util.concurrent.RateLimiter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.unified\_user\_actions.enricher.FatalException

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentEnvelop

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentInstruction

import com.twitter.unified\_user\_actions.enricher.internal.thriftscala.EnrichmentKey

import com.twitter.util.Future

import com.twitter.util.logging.Logging

abstract class AbstractHydrator(scopedStatsReceiver: StatsReceiver) extends Hydrator with Logging {

object StatsNames {

val Exceptions = "exceptions"

val EmptyKeys = "empty\_keys"

val Hydrations = "hydrations"

}

private val exceptionsCounter = scopedStatsReceiver.counter(StatsNames.Exceptions)

private val emptyKeysCounter = scopedStatsReceiver.counter(StatsNames.EmptyKeys)

private val hydrationsCounter = scopedStatsReceiver.counter(StatsNames.Hydrations)

// at most 1 log message per second

private val rateLimiter = RateLimiter.create(1.0)

private def rateLimitedLogError(e: Throwable): Unit =

if (rateLimiter.tryAcquire()) {

error(e.getMessage, e)

}

protected def safelyHydrate(

instruction: EnrichmentInstruction,

keyOpt: EnrichmentKey,

envelop: EnrichmentEnvelop

): Future[EnrichmentEnvelop]

override def hydrate(

instruction: EnrichmentInstruction,

keyOpt: Option[EnrichmentKey],

envelop: EnrichmentEnvelop

): Future[EnrichmentEnvelop] = {

keyOpt

.map(key => {

safelyHydrate(instruction, key, envelop)

.onSuccess(\_ => hydrationsCounter.incr())

.rescue {

case e: FatalException => Future.exception(e)

case e =>

rateLimitedLogError(e)

exceptionsCounter.incr()

Future.value(envelop)

}

}).getOrElse({

emptyKeysCounter.incr()

Future.value(envelop)

})

}

}