package com.twitter.tweetypie.storage

import com.twitter.servo.util.FutureEffect

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.\_

import com.twitter.scrooge.BinaryThriftStructSerializer

import com.twitter.servo.util.{Scribe => ServoScribe}

import com.twitter.tweetypie.storage\_internal.thriftscala.\_

import com.twitter.tbird.thriftscala.Added

import com.twitter.tbird.thriftscala.Removed

import com.twitter.tbird.thriftscala.Scrubbed

import com.twitter.util.Time

/\*\*

\* Scribe is used to log tweet writes which are used to generate /tables/statuses in HDFS.

\*

\* Write Scribe Category Message

\* ----- --------------- -------

\* add tbird\_add\_status [[com.twitter.tbird.thriftscala.Added]]

\* remove tbird\_remove\_status [[com.twitter.tbird.thriftscala.Removed]]

\* scrub tbird\_scrub\_status [[com.twitter.tbird.thriftscala.Scrubbed]]

\*

\* The thrift representation is encoded using binary thrift protocol format, followed by base64

\* encoding and converted to string using default character set (utf8). The logger uses BareFormatter.

\*

\* The thrift ops are scribed only after the write API call has succeeded.

\*

\* The class is thread safe except initial configuration and registration routines,

\* and no exception is expected unless java heap is out of memory.

\*

\* If exception does get thrown, add/remove/scrub operations will fail and

\* client will have to retry

\*/

class Scribe(factory: Scribe.ScribeHandlerFactory, statsReceiver: StatsReceiver) {

import Scribe.\_

private val AddedSerializer = BinaryThriftStructSerializer(Added)

private val RemovedSerializer = BinaryThriftStructSerializer(Removed)

private val ScrubbedSerializer = BinaryThriftStructSerializer(Scrubbed)

private val addCounter = statsReceiver.counter("scribe/add/count")

private val removeCounter = statsReceiver.counter("scribe/remove/count")

private val scrubCounter = statsReceiver.counter("scribe/scrub/count")

val addHandler: FutureEffect[String] = ServoScribe(factory(scribeAddedCategory)())

val removeHandler: FutureEffect[String] = ServoScribe(factory(scribeRemovedCategory)())

val scrubHandler: FutureEffect[String] = ServoScribe(factory(scribeScrubbedCategory)())

private def addedToString(tweet: StoredTweet): String =

AddedSerializer.toString(

Added(StatusConversions.toTBirdStatus(tweet), Time.now.inMilliseconds, Some(false))

)

private def removedToString(id: Long, at: Time, isSoftDeleted: Boolean): String =

RemovedSerializer.toString(Removed(id, at.inMilliseconds, Some(isSoftDeleted)))

private def scrubbedToString(id: Long, cols: Seq[Int], at: Time): String =

ScrubbedSerializer.toString(Scrubbed(id, cols, at.inMilliseconds))

def logAdded(tweet: StoredTweet): Unit = {

addHandler(addedToString(tweet))

addCounter.incr()

}

def logRemoved(id: Long, at: Time, isSoftDeleted: Boolean): Unit = {

removeHandler(removedToString(id, at, isSoftDeleted))

removeCounter.incr()

}

def logScrubbed(id: Long, cols: Seq[Int], at: Time): Unit = {

scrubHandler(scrubbedToString(id, cols, at))

scrubCounter.incr()

}

}

object Scribe {

type ScribeHandlerFactory = (String) => HandlerFactory

/\*\* WARNING: These categories are white-listed. If you are changing them, the new categories should be white-listed.

\* You should followup with CoreWorkflows team (CW) for that.

\*/

private val scribeAddedCategory = "tbird\_add\_status"

private val scribeRemovedCategory = "tbird\_remove\_status"

private val scribeScrubbedCategory = "tbird\_scrub\_status"

}