package com.twitter.tweetypie.storage

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Time

import com.twitter.util.Try

import java.util.Arrays

import scala.util.control.NoStackTrace

import scala.util.control.NonFatal

sealed abstract class TimestampType(val keyName: String)

object TimestampType {

object Default extends TimestampType("timestamp")

object SoftDelete extends TimestampType("softdelete\_timestamp")

}

/\*\*

\* TimestampDecoder gets the timestamps associated with state records. The Manhattan timestamp is

\* used for legacy records (with value "1"), otherwise the timestamp is extracted from the

\* JSON value.

\*

\* See "Metadata" in README.md for further information about state records.

\*/

object TimestampDecoder {

case class UnparsableJson(msg: String, t: Throwable) extends Exception(msg, t) with NoStackTrace

case class MissingJsonTimestamp(msg: String) extends Exception(msg) with NoStackTrace

case class UnexpectedJsonValue(msg: String) extends Exception(msg) with NoStackTrace

case class MissingManhattanTimestamp(msg: String) extends Exception(msg) with NoStackTrace

private[storage] val LegacyValue: Array[Byte] = Array('1')

/\*\*

\* The first backfill of tweet data to Manhattan supplied timestamps in milliseconds where

\* nanoseconds were expected. The result is that some values have an incorrect Manhattan

\* timestamp. For these bad timestamps, time.inNanoseconds is actually milliseconds.

\*

\* For example, the deletion record for tweet 22225781 has Manhattan timestamp 1970-01-01 00:23:24 +0000.

\* Contrast with the deletion record for tweet 435404491999813632 with Manhattan timestamp 2014-11-09 14:24:04 +0000

\*

\* This threshold value comes from the last time in milliseconds that was interpreted

\* as nanoseconds, e.g. Time.fromNanoseconds(1438387200000L) == 1970-01-01 00:23:58 +0000

\*/

private[storage] val BadTimestampThreshold = Time.at("1970-01-01 00:23:58 +0000")

def decode(record: TweetManhattanRecord, tsType: TimestampType): Try[Long] =

decode(record.value, tsType)

def decode(mhValue: TweetManhattanValue, tsType: TimestampType): Try[Long] = {

val value = ByteArrayCodec.fromByteBuffer(mhValue.contents)

if (isLegacyRecord(value)) {

nativeManhattanTimestamp(mhValue)

} else {

jsonTimestamp(value, tsType)

}

}

private def isLegacyRecord(value: Array[Byte]) = Arrays.equals(value, LegacyValue)

private def nativeManhattanTimestamp(mhValue: TweetManhattanValue): Try[Long] =

mhValue.timestamp match {

case Some(ts) => Return(correctedTimestamp(ts))

case None =>

Throw(MissingManhattanTimestamp(s"Manhattan timestamp missing in value $mhValue"))

}

private def jsonTimestamp(value: Array[Byte], tsType: TimestampType): Try[Long] =

Try { Json.decode(value) }

.rescue { case NonFatal(e) => Throw(UnparsableJson(e.getMessage, e)) }

.flatMap { m =>

m.get(tsType.keyName) match {

case Some(v) =>

v match {

case l: Long => Return(l)

case i: Integer => Return(i.toLong)

case \_ =>

Throw(

UnexpectedJsonValue(s"Unexpected value for ${tsType.keyName} in record data $m")

)

}

case None =>

Throw(MissingJsonTimestamp(s"Missing key ${tsType.keyName} in record data $m"))

}

}

def correctedTime(t: Time): Time =

if (t < BadTimestampThreshold) Time.fromMilliseconds(t.inNanoseconds) else t

def correctedTime(t: Long): Time = correctedTime(Time.fromNanoseconds(t))

def correctedTimestamp(t: Time): Long =

if (t < BadTimestampThreshold) t.inNanoseconds else t.inMilliseconds

}