package com.twitter.tweetypie.util

import com.twitter.conversions.DurationOps.\_

import com.twitter.logging.Logger

import com.twitter.mediaservices.commons.mediainformation.thriftscala.UserDefinedProductMetadata

import com.twitter.scrooge.BinaryThriftStructSerializer

import com.twitter.servo.cache.ScopedCacheKey

import com.twitter.servo.util.Transformer

import com.twitter.tweetypie.thriftscala.PostTweetRequest

import com.twitter.util.Base64Long

import com.twitter.util.Time

import java.nio.ByteBuffer

import java.security.MessageDigest

import org.apache.commons.codec.binary.Base64

import scala.collection.immutable.SortedMap

object TweetCreationLock {

case class Key private (userId: UserId, typeCode: String, idOrMd5: String)

extends ScopedCacheKey("t", "locker", 2, Base64Long.toBase64(userId), typeCode, idOrMd5) {

def uniquenessId: Option[String] =

if (typeCode == Key.TypeCode.UniquenessId) Some(idOrMd5) else None

}

object Key {

private[this] val log = Logger(getClass)

object TypeCode {

val SourceTweetId = "r"

val UniquenessId = "u"

val PostTweetRequest = "p"

}

private[this] val serializer = BinaryThriftStructSerializer(PostTweetRequest)

// normalize the representation of no media ids.

private[util] def sanitizeMediaUploadIds(mediaUploadIds: Option[Seq[Long]]) =

mediaUploadIds.filter(\_.nonEmpty)

/\*\*

\* Request deduplication depends on the hash of a serialized Thrift value.

\*

\* In order to guarantee that a Map has a reproducible serialized form,

\* it's necessary to fix the ordering of its keys.

\*/

private[util] def sanitizeMediaMetadata(

mediaMetadata: Option[scala.collection.Map[MediaId, UserDefinedProductMetadata]]

): Option[scala.collection.Map[MediaId, UserDefinedProductMetadata]] =

mediaMetadata.map(m => SortedMap(m.toSeq: \_\*))

/\*\*

\* Make sure to sanitize request fields with map/set since serialized

\* bytes ordering is not guaranteed for same thrift values.

\*/

private[util] def sanitizeRequest(request: PostTweetRequest): PostTweetRequest =

PostTweetRequest(

userId = request.userId,

text = request.text,

createdVia = "",

inReplyToTweetId = request.inReplyToTweetId,

geo = request.geo,

mediaUploadIds = sanitizeMediaUploadIds(request.mediaUploadIds),

narrowcast = request.narrowcast,

nullcast = request.nullcast,

additionalFields = request.additionalFields,

attachmentUrl = request.attachmentUrl,

mediaMetadata = sanitizeMediaMetadata(request.mediaMetadata),

conversationControl = request.conversationControl,

underlyingCreativesContainerId = request.underlyingCreativesContainerId,

editOptions = request.editOptions,

noteTweetOptions = request.noteTweetOptions

)

def bySourceTweetId(userId: UserId, sourceTweetId: TweetId): Key =

Key(userId, TypeCode.SourceTweetId, Base64Long.toBase64(sourceTweetId))

def byRequest(request: PostTweetRequest): Key =

request.uniquenessId match {

case Some(uqid) =>

byUniquenessId(request.userId, uqid)

case None =>

val sanitized = sanitizeRequest(request)

val sanitizedBytes = serializer.toBytes(sanitized)

val digested = MessageDigest.getInstance("SHA-256").digest(sanitizedBytes)

val base64Digest = Base64.encodeBase64String(digested)

val key = Key(request.userId, TypeCode.PostTweetRequest, base64Digest)

log.ifDebug(s"Generated key $key from request:\n${sanitized}")

key

}

/\*\*

\* Key for tweets that have a uniqueness id set. There is only one

\* namespace of uniqueness ids, across all clients. They are

\* expected to be Snowflake ids, in order to avoid cache

\* collisions.

\*/

def byUniquenessId(userId: UserId, uniquenessId: Long): Key =

Key(userId, TypeCode.UniquenessId, Base64Long.toBase64(uniquenessId))

}

/\*\*

\* The state of tweet creation for a given Key (request).

\*/

sealed trait State

object State {

/\*\*

\* There is no tweet creation currently in progress. (This can

\* either be represented by no entry in the cache, or this special

\* marker. This lets us use checkAndSet for deletion to avoid

\* accidentally overwriting other process' values.)

\*/

case object Unlocked extends State

/\*\*

\* Some process is attempting to create the tweet.

\*/

case class InProgress(token: Long, timestamp: Time) extends State

/\*\*

\* The tweet has already been successfully created, and has the

\* specified id.

\*/

case class AlreadyCreated(tweetId: TweetId, timestamp: Time) extends State

/\*\*

\* When stored in cache, each state is prefixed by a byte

\* indicating the type of the entry.

\*/

object TypeCode {

val Unlocked: Byte = 0.toByte

val InProgress: Byte = 1.toByte // + random long + timestamp

val AlreadyCreated: Byte = 2.toByte // + tweet id + timestamp

}

private[this] val BufferSize = 17 // type byte + 64-bit value + 64-bit timestamp

// Constant buffer to use for storing the serialized form on

// Unlocked.

private[this] val UnlockedBuf = Array[Byte](TypeCode.Unlocked)

// Store the serialization function in a ThreadLocal so that we can

// reuse the buffer between invocations.

private[this] val threadLocalSerialize = new ThreadLocal[State => Array[Byte]] {

override def initialValue(): State => Array[Byte] = {

// Allocate the thread-local state

val ary = new Array[Byte](BufferSize)

val buf = ByteBuffer.wrap(ary)

{

case Unlocked => UnlockedBuf

case InProgress(token, timestamp) =>

buf.clear()

buf

.put(TypeCode.InProgress)

.putLong(token)

.putLong(timestamp.sinceEpoch.inNanoseconds)

ary

case AlreadyCreated(tweetId, timestamp) =>

buf.clear()

buf

.put(TypeCode.AlreadyCreated)

.putLong(tweetId)

.putLong(timestamp.sinceEpoch.inNanoseconds)

ary

}

}

}

/\*\*

\* Convert this State to the cache representation.

\*/

private[this] def toBytes(state: State): Array[Byte] =

threadLocalSerialize.get()(state)

/\*\*

\* Convert this byte array into a LockState.

\*

\* @throws RuntimeException if the buffer is not of the right size

\* and format

\*/

private[this] def fromBytes(bytes: Array[Byte]): State = {

val buf = ByteBuffer.wrap(bytes)

val result = buf.get() match {

case TypeCode.Unlocked => Unlocked

case TypeCode.InProgress => InProgress(buf.getLong(), buf.getLong().nanoseconds.afterEpoch)

case TypeCode.AlreadyCreated =>

AlreadyCreated(buf.getLong(), buf.getLong().nanoseconds.afterEpoch)

case other => throw new RuntimeException("Invalid type code: " + other)

}

if (buf.remaining != 0) {

throw new RuntimeException("Extra data in buffer: " + bytes)

}

result

}

/\*\*

\* How to serialize the State for storage in cache.

\*/

val Serializer: Transformer[State, Array[Byte]] =

Transformer[State, Array[Byte]](tTo = toBytes \_, tFrom = fromBytes \_)

}

}