package com.twitter.tweetypie.storage

import com.twitter.bijection.Conversion.asMethod

import com.twitter.bijection.Injection

import com.twitter.scrooge.TFieldBlob

import com.twitter.storage.client.manhattan.kv.\_

import com.twitter.tweetypie.storage.Response.FieldResponse

import com.twitter.tweetypie.storage.Response.FieldResponseCode

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

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

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

import java.io.ByteArrayOutputStream

import java.nio.ByteBuffer

import org.apache.thrift.protocol.TBinaryProtocol

import org.apache.thrift.transport.TIOStreamTransport

import org.apache.thrift.transport.TMemoryInputTransport

import scala.collection.immutable

import scala.util.control.NoStackTrace

// NOTE: All field ids and Tweet structure in this file correspond to the StoredTweet struct ONLY

object ByteArrayCodec {

def toByteBuffer(byteArray: Array[Byte]): ByteBuffer = byteArray.as[ByteBuffer]

def fromByteBuffer(buffer: ByteBuffer): Array[Byte] = buffer.as[Array[Byte]]

}

object StringCodec {

private val string2ByteBuffer = Injection.connect[String, Array[Byte], ByteBuffer]

def toByteBuffer(strValue: String): ByteBuffer = string2ByteBuffer(strValue)

def fromByteBuffer(buffer: ByteBuffer): String = string2ByteBuffer.invert(buffer).get

}

/\*\*

\* Terminology

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

\* Tweet id field : The field number of 'tweetId' in the 'Tweet' thrift structure (i.e "1")

\*

\* First AdditionalField id : The ID if the first additional field in 'Tweet' thrift structure. All field Ids less than this are

\* considered internal and all the ids greater than or equal to this field id are considered 'Additional fields'.

\* This is set to 100.

\*

\* Internal Fields : Fields with ids [1 to firstAdditionalFieldid) (excluding firstAdditionalFieldId)

\*

\* Core fields : (Subset of Internal fields)- Fields with ids [1 to 8, 19]. These fields are "packed" together and stored

\* under a single key. This key is referred to as "CoreFieldsKey" (see @TweetKeyType.CoreFieldsKey).

\* Note: Actually field 1 is skipped when packing as this field is the tweet id and it need not be

\* explicitly stored since the pkey already contains the tweet Id)

\*

\* Root Core field id : The field id under which the packed core fields are stored in Manhattan. (This is field Id "1")

\*

\* Required fields : (Subset of Core fields) - Fields with ids [1 to 5] that MUST be present on every tweet.

\*

\* Additional Fields : All fields with field ids >= 'firstAdditionalFieldId'

\*

\* Compiled Additional fields : (Subset of Additional Fields) - All fields that the storage library knows about

\* (i.e present on the latest storage\_internal.thrift that is compiled-in).

\*

\* Passthrough fields : (Subset of Additional Fields) - The fields on storage\_internal.thrift that the storage library is NOT aware of

\* These field ids are is obtained looking at the "\_passThroughFields" member of the scrooge-generated

\* 'Tweet' object.

\*

\* coreFieldsIdInInternalTweet: This is the field id of the core fields (the only field) in the Internal Tweet struct

\*/

object TweetFields {

val firstAdditionalFieldId: Short = 100

val tweetIdField: Short = 1

val geoFieldId: Short = 9

// The field under which all the core field values are stored (in serialized form).

val rootCoreFieldId: Short = 1

val coreFieldIds: immutable.IndexedSeq[FieldId] = {

val quotedTweetFieldId: Short = 19

(1 to 8).map(\_.toShort) ++ Seq(quotedTweetFieldId)

}

val requiredFieldIds: immutable.IndexedSeq[FieldId] = (1 to 5).map(\_.toShort)

val coreFieldsIdInInternalTweet: Short = 1

val compiledAdditionalFieldIds: Seq[FieldId] =

StoredTweet.metaData.fields.filter(\_.id >= firstAdditionalFieldId).map(\_.id)

val internalFieldIds: Seq[FieldId] =

StoredTweet.metaData.fields.filter(\_.id < firstAdditionalFieldId).map(\_.id)

val nonCoreInternalFields: Seq[FieldId] =

(internalFieldIds.toSet -- coreFieldIds.toSet).toSeq

def getAdditionalFieldIds(tweet: StoredTweet): Seq[FieldId] =

compiledAdditionalFieldIds ++ tweet.\_passthroughFields.keys.toSeq

}

/\*\*

\* Helper object to convert TFieldBlob to ByteBuffer that gets stored in Manhattan.

\*

\* The following is the format in which the TFieldBlob gets stored:

\* [Version][TField][TFieldBlob]

\*/

object TFieldBlobCodec {

val BinaryProtocolFactory: TBinaryProtocol.Factory = new TBinaryProtocol.Factory()

val FormatVersion = 1.0

def toByteBuffer(tFieldBlob: TFieldBlob): ByteBuffer = {

val baos = new ByteArrayOutputStream()

val prot = BinaryProtocolFactory.getProtocol(new TIOStreamTransport(baos))

prot.writeDouble(FormatVersion)

prot.writeFieldBegin(tFieldBlob.field)

prot.writeBinary(ByteArrayCodec.toByteBuffer(tFieldBlob.data))

ByteArrayCodec.toByteBuffer(baos.toByteArray)

}

def fromByteBuffer(buffer: ByteBuffer): TFieldBlob = {

val byteArray = ByteArrayCodec.fromByteBuffer(buffer)

val prot = BinaryProtocolFactory.getProtocol(new TMemoryInputTransport(byteArray))

val version = prot.readDouble()

if (version != FormatVersion) {

throw new VersionMismatchError(

"Version mismatch in decoding ByteBuffer to TFieldBlob. " +

"Actual version: " + version + ". Expected version: " + FormatVersion

)

}

val tField = prot.readFieldBegin()

val dataBuffer = prot.readBinary()

val data = ByteArrayCodec.fromByteBuffer(dataBuffer)

TFieldBlob(tField, data)

}

}

/\*\*

\* Helper object to help convert 'CoreFields' object to/from TFieldBlob (and also to construct

\* 'CoreFields' object from a 'StoredTweet' object)

\*

\* More details:

\* - A subset of fields on the 'StoredTweet' thrift structure (2-8,19) are 'packaged' and stored

\* together as a serialized TFieldBlob object under a single key in Manhattan (see TweetKeyCodec

\* helper object above for more details).

\*

\* - To make the packing/unpacking the fields to/from TFieldBlob object, we created the following

\* two helper thrift structures 'CoreFields' and 'InternalTweet'

\*

\* // The field Ids and types here MUST exactly match field Ids on 'StoredTweet' thrift structure.

\* struct CoreFields {

\* 2: optional i64 user\_id

\* ...

\* 8: optional i64 contributor\_id

\* ...

\* 19: optional StoredQuotedTweet stored\_quoted\_tweet

\*

\* }

\*

\* // The field id of core fields MUST be "1"

\* struct InternalTweet {

\* 1: CoreFields coreFields

\* }

\*

\* - Given the above two structures, packing/unpacking fields (2-8,19) on StoredTweet object into a TFieldBlob

\* becomes very trivial:

\* For packing:

\* (i) Copy fields (2-8,19) from StoredTweet object to a new CoreFields object

\* (ii) Create a new InternalTweet object with the 'CoreFields' object constructed in step (i) above

\* (iii) Extract field "1" as a TFieldBlob from InternalField (by calling the scrooge generated "getFieldBlob(1)"

\* function on the InternalField objecton

\*

\* For unpacking:

\* (i) Create an empty 'InternalField' object

\* (ii) Call scrooge-generated 'setField' by passing the tFieldBlob blob (created by packing steps above)

\* (iii) Doing step (ii) above will create a hydrated 'CoreField' object that can be accessed by 'coreFields'

\* member of 'InternalTweet' object.

\*/

object CoreFieldsCodec {

val coreFieldIds: Seq[FieldId] = CoreFields.metaData.fields.map(\_.id)

// "Pack" the core fields i.e converts 'CoreFields' object to "packed" tFieldBlob (See description

// above for more details)

def toTFieldBlob(coreFields: CoreFields): TFieldBlob = {

InternalTweet(Some(coreFields)).getFieldBlob(TweetFields.coreFieldsIdInInternalTweet).get

}

// "Unpack" the core fields from a packed TFieldBlob into a CoreFields object (see description above for

// more details)

def fromTFieldBlob(tFieldBlob: TFieldBlob): CoreFields = {

InternalTweet().setField(tFieldBlob).coreFields.get

}

// "Unpack" the core fields from a packed TFieldBlob into a Map of core-fieldId-> TFieldBlob

def unpackFields(tFieldBlob: TFieldBlob): Map[Short, TFieldBlob] =

fromTFieldBlob(tFieldBlob).getFieldBlobs(coreFieldIds)

// Create a 'CoreFields' thrift object from 'Tweet' thrift object.

def fromTweet(tweet: StoredTweet): CoreFields = {

// As mentioned above, the field ids and types on the 'CoreFields' struct exactly match the

// corresponding fields on StoredTweet structure. So it is safe to call .getField() on Tweet object and

// and pass the returned tFleldBlob a 'setField' on 'CoreFields' object.

coreFieldIds.foldLeft(CoreFields()) {

case (core, fieldId) =>

tweet.getFieldBlob(fieldId) match {

case None => core

case Some(tFieldBlob) => core.setField(tFieldBlob)

}

}

}

}

/\*\*

\* Helper object to convert ManhattanException to FieldResponseCode thrift object

\*/

object FieldResponseCodeCodec {

import FieldResponseCodec.ValueNotFoundException

def fromManhattanException(mhException: ManhattanException): FieldResponseCode = {

mhException match {

case \_: ValueNotFoundException => FieldResponseCode.ValueNotFound

case \_: InternalErrorManhattanException => FieldResponseCode.Error

case \_: InvalidRequestManhattanException => FieldResponseCode.InvalidRequest

case \_: DeniedManhattanException => FieldResponseCode.Error

case \_: UnsatisfiableManhattanException => FieldResponseCode.Error

case \_: TimeoutManhattanException => FieldResponseCode.Timeout

}

}

}

/\*\*

\* Helper object to construct FieldResponse thrift object from an Exception.

\* This is typically called to convert 'ManhattanException' object to 'FieldResponse' thrift object

\*/

object FieldResponseCodec {

class ValueNotFoundException extends ManhattanException("Value not found!") with NoStackTrace

private[storage] val NotFound = new ValueNotFoundException

def fromThrowable(e: Throwable, additionalMsg: Option[String] = None): FieldResponse = {

val (respCode, errMsg) = e match {

case mhException: ManhattanException =>

(FieldResponseCodeCodec.fromManhattanException(mhException), mhException.getMessage)

case \_ => (FieldResponseCode.Error, e.getMessage)

}

val respMsg = additionalMsg.map(\_ + ". " + errMsg).orElse(Some(errMsg.toString))

FieldResponse(respCode, respMsg)

}

}