package com.twitter.product\_mixer.component\_library.premarshaller.cursor

import com.twitter.product\_mixer.component\_library.model.cursor.OrderedCursor

import com.twitter.product\_mixer.component\_library.model.cursor.PassThroughCursor

import com.twitter.product\_mixer.component\_library.model.cursor.UnorderedBloomFilterCursor

import com.twitter.product\_mixer.component\_library.model.cursor.UnorderedExcludeIdsCursor

import com.twitter.product\_mixer.component\_library.{thriftscala => t}

import com.twitter.product\_mixer.core.pipeline.PipelineCursor

import com.twitter.product\_mixer.core.pipeline.PipelineCursorSerializer

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.IllegalStateFailure

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.MalformedCursor

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.scrooge.BinaryThriftStructSerializer

import com.twitter.scrooge.ThriftStructCodec

import com.twitter.search.common.util.bloomfilter.AdaptiveLongIntBloomFilterSerializer

import com.twitter.util.Base64UrlSafeStringEncoder

import com.twitter.util.StringEncoder

import com.twitter.product\_mixer.core.functional\_component.marshaller.response.slice.CursorTypeMarshaller

/\*\*

\* Handles serialization and deserialization for all supported generic cursors. Note that generic

\* cursors may be used for Slices or any bespoke marshalling format.

\*/

object CursorSerializer extends PipelineCursorSerializer[PipelineCursor] {

private[cursor] val CursorThriftSerializer: BinaryThriftStructSerializer[

t.ProductMixerRequestCursor

] =

new BinaryThriftStructSerializer[t.ProductMixerRequestCursor] {

override def codec: ThriftStructCodec[t.ProductMixerRequestCursor] =

t.ProductMixerRequestCursor

override def encoder: StringEncoder = Base64UrlSafeStringEncoder

}

override def serializeCursor(cursor: PipelineCursor): String =

cursor match {

case OrderedCursor(id, cursorType, gapBoundaryId) =>

val cursorTypeMarshaller = new CursorTypeMarshaller()

val thriftCursor = t.ProductMixerRequestCursor.OrderedCursor(

t.OrderedCursor(

id = id,

cursorType = cursorType.map(cursorTypeMarshaller.apply),

gapBoundaryId))

CursorThriftSerializer.toString(thriftCursor)

case UnorderedExcludeIdsCursor(excludedIds) =>

val thriftCursor = t.ProductMixerRequestCursor.UnorderedExcludeIdsCursor(

t.UnorderedExcludeIdsCursor(excludedIds = Some(excludedIds)))

CursorThriftSerializer.toString(thriftCursor)

case UnorderedBloomFilterCursor(longIntBloomFilter) =>

val thriftCursor = t.ProductMixerRequestCursor.UnorderedBloomFilterCursor(

t.UnorderedBloomFilterCursor(

serializedLongIntBloomFilter =

AdaptiveLongIntBloomFilterSerializer.serialize(longIntBloomFilter)

))

CursorThriftSerializer.toString(thriftCursor)

case PassThroughCursor(cursorValue, cursorType) =>

val cursorTypeMarshaller = new CursorTypeMarshaller()

val thriftCursor = t.ProductMixerRequestCursor.PassThroughCursor(

t.PassThroughCursor(

cursorValue = cursorValue,

cursorType = cursorType.map(cursorTypeMarshaller.apply)

))

CursorThriftSerializer.toString(thriftCursor)

case \_ =>

throw PipelineFailure(IllegalStateFailure, "Unknown cursor type")

}

def deserializeOrderedCursor(cursorString: String): Option[OrderedCursor] =

deserializeCursor(

cursorString,

{

case Some(

t.ProductMixerRequestCursor

.OrderedCursor(t.OrderedCursor(id, cursorType, gapBoundaryId))) =>

val cursorTypeMarshaller = new CursorTypeMarshaller()

Some(

OrderedCursor(

id = id,

cursorType = cursorType.map(cursorTypeMarshaller.unmarshall),

gapBoundaryId))

}

)

def deserializeUnorderedExcludeIdsCursor(

cursorString: String

): Option[UnorderedExcludeIdsCursor] = {

deserializeCursor(

cursorString,

{

case Some(

t.ProductMixerRequestCursor

.UnorderedExcludeIdsCursor(t.UnorderedExcludeIdsCursor(excludedIdsOpt))) =>

Some(UnorderedExcludeIdsCursor(excludedIds = excludedIdsOpt.getOrElse(Seq.empty)))

}

)

}

def deserializeUnorderedBloomFilterCursor(

cursorString: String

): Option[UnorderedBloomFilterCursor] =

deserializeCursor(

cursorString,

{

case Some(

t.ProductMixerRequestCursor.UnorderedBloomFilterCursor(

t.UnorderedBloomFilterCursor(serializedLongIntBloomFilter))) =>

val bloomFilter = AdaptiveLongIntBloomFilterSerializer

.deserialize(serializedLongIntBloomFilter).getOrElse(

throw PipelineFailure(

MalformedCursor,

s"Failed to deserialize UnorderedBloomFilterCursor from cursor string: $cursorString")

)

Some(UnorderedBloomFilterCursor(longIntBloomFilter = bloomFilter))

}

)

def deserializePassThroughCursor(cursorString: String): Option[PassThroughCursor] =

deserializeCursor(

cursorString,

{

case Some(

t.ProductMixerRequestCursor

.PassThroughCursor(t.PassThroughCursor(cursorValue, cursorType))) =>

val cursorTypeMarshaller = new CursorTypeMarshaller()

Some(

PassThroughCursor(

cursorValue = cursorValue,

cursorType = cursorType.map(cursorTypeMarshaller.unmarshall)))

}

)

// Note that the "A" type of the PartialFunction cannot be inferred due to the thrift type not

// being present on the PipelineCursorSerializer trait. By using this private def with the

// deserializePf type declared, it can be inferred.

private def deserializeCursor[Cursor <: PipelineCursor](

cursorString: String,

deserializePf: PartialFunction[Option[t.ProductMixerRequestCursor], Option[Cursor]]

): Option[Cursor] =

PipelineCursorSerializer.deserializeCursor(

cursorString,

CursorThriftSerializer,

deserializePf

)

}