package com.twitter.product\_mixer.component\_library.premarshaller.urt.builder

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

import com.twitter.product\_mixer.component\_library.premarshaller.cursor.UrtCursorSerializer

import com.twitter.product\_mixer.core.model.marshalling.response.urt.TimelineEntry

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.CursorType

import com.twitter.product\_mixer.core.model.marshalling.response.urt.operation.GapCursor

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

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

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

/\*\*

\* Builds [[UrtOrderedCursor]] in the Bottom position as a Gap cursor.

\*

\* @param idSelector Specifies the entry from which to derive the `id` field

\* @param includeOperation Logic to determine whether or not to build the gap cursor, which should

\* always be the inverse of the logic used to decide whether or not to build

\* the bottom cursor via [[OrderedBottomCursorBuilder]], since either the

\* gap or the bottom cursor must always be returned.

\* @param serializer Converts the cursor to an encoded string

\*/

case class OrderedGapCursorBuilder[

-Query <: PipelineQuery with HasPipelineCursor[UrtOrderedCursor]

](

idSelector: PartialFunction[TimelineEntry, Long],

override val includeOperation: IncludeInstruction[Query],

serializer: PipelineCursorSerializer[UrtOrderedCursor] = UrtCursorSerializer)

extends UrtCursorBuilder[Query] {

override val cursorType: CursorType = GapCursor

override def cursorValue(

query: Query,

timelineEntries: Seq[TimelineEntry]

): String = {

// To determine the gap boundary, use any existing cursor gap boundary id (i.e. if submitted

// from a previous gap cursor, else use the existing cursor id (i.e. from a previous top cursor)

val gapBoundaryId = query.pipelineCursor.flatMap(\_.gapBoundaryId).orElse {

query.pipelineCursor.flatMap(\_.id)

}

val bottomId = timelineEntries.reverseIterator.collectFirst(idSelector)

val id = bottomId.orElse(gapBoundaryId)

val cursor = UrtOrderedCursor(

initialSortIndex = nextBottomInitialSortIndex(query, timelineEntries),

id = id,

cursorType = Some(cursorType),

gapBoundaryId = gapBoundaryId

)

serializer.serializeCursor(cursor)

}

}