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

import com.twitter.product\_mixer.component\_library.premarshaller.urt.builder.UrtCursorBuilder.DefaultSortIndex

import com.twitter.product\_mixer.component\_library.premarshaller.urt.builder.UrtCursorBuilder.NextPageTopCursorEntryOffset

import com.twitter.product\_mixer.component\_library.premarshaller.urt.builder.UrtCursorBuilder.UrtEntryOffset

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

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

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

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

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.model.marshalling.response.urt.operation.TopCursor

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

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

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

import com.twitter.product\_mixer.core.util.SortIndexBuilder

object UrtCursorBuilder {

val NextPageTopCursorEntryOffset = 1L

val UrtEntryOffset = 1L

val DefaultSortIndex = (query: PipelineQuery) => SortIndexBuilder.timeToId(query.queryTime)

}

trait UrtCursorBuilder[-Query <: PipelineQuery] {

val includeOperation: IncludeInstruction[Query] = AlwaysInclude

def cursorType: CursorType

def cursorValue(query: Query, entries: Seq[TimelineEntry]): String

/\*\*

\* Identifier of an \*existing\* timeline cursor that this new cursor would replace, if this cursor

\* is returned in a `ReplaceEntry` timeline instruction.

\*

\* Note:

\* - This id is used to populate the `entryIdToReplace` field on the URT TimelineEntry

\* generated. More details at [[CursorOperation.entryIdToReplace]].

\* - As a convention, we use the sortIndex of the cursor for its id/entryId fields. So the

\* `idToReplace` should represent the sortIndex of the existing cursor to be replaced.

\*/

def idToReplace(query: Query): Option[Long] = None

def cursorSortIndex(query: Query, entries: Seq[TimelineEntry]): Long =

(query, cursorType) match {

case (query: PipelineQuery with HasPipelineCursor[\_], TopCursor) =>

topCursorSortIndex(query, entries)

case (query: PipelineQuery with HasPipelineCursor[\_], BottomCursor | GapCursor) =>

bottomCursorSortIndex(query, entries)

case \_ =>

throw new UnsupportedOperationException(

"Automatic sort index support limited to top and bottom cursors")

}

def build(query: Query, entries: Seq[TimelineEntry]): Option[CursorOperation] = {

if (includeOperation(query, entries)) {

val sortIndex = cursorSortIndex(query, entries)

val cursorOperation = CursorOperation(

id = sortIndex,

sortIndex = Some(sortIndex),

value = cursorValue(query, entries),

cursorType = cursorType,

displayTreatment = None,

idToReplace = idToReplace(query),

)

Some(cursorOperation)

} else None

}

/\*\*

\* Build the top cursor sort index which handles the following cases:

\* 1. When there is at least one non-cursor entry, use the first entry's sort index + UrtEntryOffset

\* 2. When there are no non-cursor entries, and initialSortIndex is not set which indicates that

\* it is the first page, use DefaultSortIndex + UrtEntryOffset

\* 3. When there are no non-cursor entries, and initialSortIndex is set which indicates that it is

\* not the first page, use the query.initialSortIndex from the passed-in cursor + UrtEntryOffset

\*/

protected def topCursorSortIndex(

query: PipelineQuery with HasPipelineCursor[\_],

entries: Seq[TimelineEntry]

): Long = {

val nonCursorEntries = entries.filter {

case \_: CursorOperation => false

case \_: CursorItem => false

case \_ => true

}

lazy val initialSortIndex =

UrtPipelineCursor.getCursorInitialSortIndex(query).getOrElse(DefaultSortIndex(query))

nonCursorEntries.headOption.flatMap(\_.sortIndex).getOrElse(initialSortIndex) + UrtEntryOffset

}

/\*\*

\* Specifies the point at which the next page's entries' sort indices will start counting.

\*

\* Note that in the case of URT, the next page's entries' does not include the top cursor. As

\* such, the value of initialSortIndex passed back in the cursor is typically the bottom cursor's

\* sort index - 2. Subtracting 2 leaves room for the next page's top cursor, which will have a

\* sort index of top entry + 1.

\*/

protected def nextBottomInitialSortIndex(

query: PipelineQuery with HasPipelineCursor[\_],

entries: Seq[TimelineEntry]

): Long = {

bottomCursorSortIndex(query, entries) - NextPageTopCursorEntryOffset - UrtEntryOffset

}

/\*\*

\* Build the bottom cursor sort index which handles the following cases:

\* 1. When there is at least one non-cursor entry, use the last entry's sort index - UrtEntryOffset

\* 2. When there are no non-cursor entries, and initialSortIndex is not set which indicates that

\* it is the first page, use DefaultSortIndex

\* 3. When there are no non-cursor entries, and initialSortIndex is set which indicates that it is

\* not the first page, use the query.initialSortIndex from the passed-in cursor

\*/

protected def bottomCursorSortIndex(

query: PipelineQuery with HasPipelineCursor[\_],

entries: Seq[TimelineEntry]

): Long = {

val nonCursorEntries = entries.filter {

case \_: CursorOperation => false

case \_: CursorItem => false

case \_ => true

}

lazy val initialSortIndex =

UrtPipelineCursor.getCursorInitialSortIndex(query).getOrElse(DefaultSortIndex(query))

nonCursorEntries.lastOption

.flatMap(\_.sortIndex).map(\_ - UrtEntryOffset).getOrElse(initialSortIndex)

}

}