package com.twitter.product\_mixer.core.pipeline.step.query\_feature\_hydrator

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.BaseQueryFeatureHydrator

import com.twitter.product\_mixer.core.model.common.identifier.PipelineStepIdentifier

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

import com.twitter.product\_mixer.core.pipeline.state.HasAsyncFeatureMap

import com.twitter.product\_mixer.core.pipeline.state.HasQuery

import com.twitter.product\_mixer.core.pipeline.step.Step

import com.twitter.product\_mixer.core.service.Executor

import com.twitter.product\_mixer.core.service.query\_feature\_hydrator\_executor.QueryFeatureHydratorExecutor

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A query level feature hydration step, it takes the input list of candidates and the given

\* hydrators and executes them. The [[State]] object is responsible for merging the resulting

\* feature maps with the hydrated ones in its updateCandidatesWithFeatures.

\*

\* @param queryFeatureHydratorExecutor Hydrator Executor

\* @tparam Query Type of PipelineQuery domain model

\* @tparam State The pipeline state domain model.

\*/

case class QueryFeatureHydratorStep[

Query <: PipelineQuery,

State <: HasQuery[Query, State] with HasAsyncFeatureMap[State]] @Inject() (

queryFeatureHydratorExecutor: QueryFeatureHydratorExecutor)

extends Step[State, QueryFeatureHydratorStepConfig[

Query

], Query, QueryFeatureHydratorExecutor.Result] {

override def isEmpty(config: QueryFeatureHydratorStepConfig[Query]): Boolean =

config.hydrators.isEmpty

override def adaptInput(state: State, config: QueryFeatureHydratorStepConfig[Query]): Query =

state.query

override def arrow(

config: QueryFeatureHydratorStepConfig[Query],

context: Executor.Context

): Arrow[Query, QueryFeatureHydratorExecutor.Result] =

queryFeatureHydratorExecutor.arrow(

config.hydrators,

config.validPipelineStepIdentifiers,

context)

override def updateState(

state: State,

executorResult: QueryFeatureHydratorExecutor.Result,

config: QueryFeatureHydratorStepConfig[Query]

): State = {

val updatedQuery = state.query

.withFeatureMap(executorResult.featureMap).asInstanceOf[Query]

state

.updateQuery(updatedQuery).addAsyncFeatureMap(executorResult.asyncFeatureMap)

}

}

case class QueryFeatureHydratorStepConfig[Query <: PipelineQuery](

hydrators: Seq[BaseQueryFeatureHydrator[Query, \_]],

validPipelineStepIdentifiers: Set[PipelineStepIdentifier])