package com.twitter.product\_mixer.core.pipeline.step.async\_feature\_map

import com.twitter.product\_mixer.core.feature.featuremap.FeatureMap

import com.twitter.product\_mixer.core.feature.featuremap.asyncfeaturemap.AsyncFeatureMap

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.async\_feature\_map\_executor.AsyncFeatureMapExecutor

import com.twitter.product\_mixer.core.service.async\_feature\_map\_executor.AsyncFeatureMapExecutorResults

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* Async Feature Hydrator Step, it takes an existing asyn feature map and executes any hydration

\* needed before the next step. The state object is responsible for keeping the updated query

\* with the updated feature map.

\*

\* @param asyncFeatureMapExecutor Async feature map executor

\*

\* @tparam Query Type of PipelineQuery domain model

\* @tparam State The pipeline state domain model.

\*/

case class AsyncFeatureMapStep[

Query <: PipelineQuery,

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

asyncFeatureMapExecutor: AsyncFeatureMapExecutor)

extends Step[

State,

AsyncFeatureMapStepConfig,

AsyncFeatureMap,

AsyncFeatureMapExecutorResults

] {

override def isEmpty(config: AsyncFeatureMapStepConfig): Boolean = false

override def adaptInput(

state: State,

config: AsyncFeatureMapStepConfig

): AsyncFeatureMap = state.asyncFeatureMap

override def arrow(

config: AsyncFeatureMapStepConfig,

context: Executor.Context

): Arrow[AsyncFeatureMap, AsyncFeatureMapExecutorResults] =

asyncFeatureMapExecutor.arrow(config.stepToHydrateFor, config.currentStep, context)

override def updateState(

state: State,

executorResult: AsyncFeatureMapExecutorResults,

config: AsyncFeatureMapStepConfig

): State = {

val hydratedFeatureMap =

executorResult.featureMapsByStep.getOrElse(config.stepToHydrateFor, FeatureMap.empty)

if (hydratedFeatureMap.isEmpty) {

state

} else {

val updatedFeatureMap = state.query.features

.getOrElse(FeatureMap.empty) ++ hydratedFeatureMap

state.updateQuery(

state.query

.withFeatureMap(updatedFeatureMap).asInstanceOf[Query])

}

}

}

case class AsyncFeatureMapStepConfig(

stepToHydrateFor: PipelineStepIdentifier,

currentStep: PipelineStepIdentifier)