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

import com.fasterxml.jackson.databind.annotation.JsonSerialize

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.feature.Feature

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.functional\_component.feature\_hydrator.AsyncHydrator

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

import com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.featurestorev1.FeatureStoreV1QueryFeatureHydrator

import com.twitter.product\_mixer.core.model.common.Conditionally

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

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

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

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

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

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

import com.twitter.product\_mixer.core.service.feature\_hydrator\_observer.FeatureHydratorObserver

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

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

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

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

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

import com.twitter.stitch.Arrow

import com.twitter.stitch.Stitch

import javax.inject.Inject

import javax.inject.Singleton

@Singleton

class QueryFeatureHydratorExecutor @Inject() (override val statsReceiver: StatsReceiver)

extends Executor {

def arrow[Query <: PipelineQuery](

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

validPipelineSteps: Set[PipelineStepIdentifier],

context: Executor.Context

): Arrow[Query, QueryFeatureHydratorExecutor.Result] = {

val observer = new FeatureHydratorObserver(statsReceiver, hydrators, context)

val hydratorsWithErrorHandling =

hydrators.map { hydrator =>

val queryFeatureHydratorArrow =

getQueryHydratorArrow(hydrator, context, observer)

val wrappedWithAsyncHandling =

handleAsyncHydrator(hydrator, validPipelineSteps, queryFeatureHydratorArrow)

handleConditionally(hydrator, wrappedWithAsyncHandling)

}

Arrow

.collect(hydratorsWithErrorHandling)

.map {

results: Seq[

(FeatureHydratorIdentifier, BaseIndividualFeatureHydratorResult)

] =>

val combinedFeatureMap = FeatureMap.merge(results.collect {

case (\_, IndividualFeatureHydratorResult(featureMap)) => featureMap

})

val asyncFeatureMaps = results.collect {

case (

hydratorIdentifier,

AsyncIndividualFeatureHydratorResult(hydrateBefore, featuresToHydrate, ref)) =>

(hydratorIdentifier, hydrateBefore, featuresToHydrate, ref)

}

QueryFeatureHydratorExecutor.Result(

individualFeatureMaps = results.toMap,

featureMap = combinedFeatureMap,

asyncFeatureMap = AsyncFeatureMap.fromFeatureMaps(asyncFeatureMaps)

)

}

}

def handleConditionally[Query <: PipelineQuery](

hydrator: BaseQueryFeatureHydrator[Query, \_],

arrow: Arrow[

Query,

BaseIndividualFeatureHydratorResult

]

): Arrow[

Query,

(FeatureHydratorIdentifier, BaseIndividualFeatureHydratorResult)

] = {

val conditionallyRunArrow = hydrator match {

case hydrator: BaseQueryFeatureHydrator[Query, \_] with Conditionally[Query @unchecked] =>

Arrow.ifelse[Query, BaseIndividualFeatureHydratorResult](

hydrator.onlyIf,

arrow,

Arrow.value(FeatureHydratorDisabled)

)

case \_ => arrow

}

Arrow.join(

Arrow.value(hydrator.identifier),

conditionallyRunArrow

)

}

def handleAsyncHydrator[Query <: PipelineQuery](

hydrator: BaseQueryFeatureHydrator[Query, \_],

validPipelineSteps: Set[PipelineStepIdentifier],

arrow: Arrow[

Query,

IndividualFeatureHydratorResult

]

): Arrow[Query, BaseIndividualFeatureHydratorResult] = {

hydrator match {

case hydrator: BaseQueryFeatureHydrator[

Query,

\_

] with AsyncHydrator =>

validateAsyncQueryFeatureHydrator(hydrator, validPipelineSteps)

startArrowAsync(arrow.map(\_.featureMap))

.map { ref =>

AsyncIndividualFeatureHydratorResult(

hydrator.hydrateBefore,

hydrator.features.asInstanceOf[Set[Feature[\_, \_]]],

ref

)

}

case \_ => arrow

}

}

def getQueryHydratorArrow[Query <: PipelineQuery](

hydrator: BaseQueryFeatureHydrator[Query, \_],

context: Executor.Context,

queryFeatureHydratorObserver: FeatureHydratorObserver

): Arrow[Query, IndividualFeatureHydratorResult] = {

val componentExecutorContext = context.pushToComponentStack(hydrator.identifier)

val hydratorArrow: Arrow[Query, FeatureMap] =

Arrow.flatMap { query: Query => hydrator.hydrate(query) }

val validationFn: FeatureMap => FeatureMap = hydrator match {

// Feature store query hydrators store the resulting PredictionRecord and not

// the features, so we cannot validate the same way

case \_: FeatureStoreV1QueryFeatureHydrator[Query] =>

identity

case \_ =>

validateFeatureMap(

hydrator.features.asInstanceOf[Set[Feature[\_, \_]]],

\_,

componentExecutorContext)

}

// record the component-level stats

val observedArrow =

wrapComponentWithExecutorBookkeeping[Query, FeatureMap](

context,

hydrator.identifier

)(hydratorArrow.map(validationFn))

// store non-configuration errors in the FeatureMap

val liftNonValidationFailuresToFailedFeatures = Arrow.handle[FeatureMap, FeatureMap] {

case NotAMisconfiguredFeatureMapFailure(e) =>

featureMapWithFailuresForFeatures(

hydrator.features.asInstanceOf[Set[Feature[\_, \_]]],

e,

componentExecutorContext)

}

val observedLiftedAndWrapped = observedArrow

.andThen(liftNonValidationFailuresToFailedFeatures)

.applyEffect(Arrow.map[FeatureMap, Unit](featureMap =>

// record per-feature stats, this is separate from the component stats handled by `wrapWithExecutorBookkeeping`

queryFeatureHydratorObserver.observeFeatureSuccessAndFailures(hydrator, Seq(featureMap))))

.map(IndividualFeatureHydratorResult)

observedLiftedAndWrapped

}

}

object QueryFeatureHydratorExecutor {

case class Result(

individualFeatureMaps: Map[

FeatureHydratorIdentifier,

BaseIndividualFeatureHydratorResult

],

featureMap: FeatureMap,

asyncFeatureMap: AsyncFeatureMap)

extends ExecutorResult

sealed trait BaseIndividualFeatureHydratorResult

case object FeatureHydratorDisabled extends BaseIndividualFeatureHydratorResult

case class IndividualFeatureHydratorResult(featureMap: FeatureMap)

extends BaseIndividualFeatureHydratorResult

/\*\* Async result, serializes without the [[Stitch]] field since it's not serializable \*/

@JsonSerialize(using = classOf[AsyncIndividualFeatureHydratorResultSerializer])

case class AsyncIndividualFeatureHydratorResult(

hydrateBefore: PipelineStepIdentifier,

features: Set[Feature[\_, \_]],

ref: Stitch[FeatureMap])

extends BaseIndividualFeatureHydratorResult

/\*\*

\* Validates whether the [[AsyncHydrator.hydrateBefore]] [[PipelineStepIdentifier]] is valid

\*

\* @param asyncQueryFeatureHydrator the hydrator to validate

\* @param validPipelineSteps a Set of [[PipelineStepIdentifier]]s which are valid places to populate async

\* [[Feature]]s in a [[com.twitter.product\_mixer.core.pipeline.Pipeline]]

\*/

def validateAsyncQueryFeatureHydrator(

asyncQueryFeatureHydrator: AsyncHydrator,

validPipelineSteps: Set[PipelineStepIdentifier]

): Unit =

require(

validPipelineSteps.contains(asyncQueryFeatureHydrator.hydrateBefore),

s"`AsyncHydrator.hydrateBefore` contained ${asyncQueryFeatureHydrator.hydrateBefore} which was not in the parent pipeline's " +

s"`PipelineConfig` Companion object field `stepsAsyncFeatureHydrationCanBeCompletedBy = $validPipelineSteps`."

)

}