package com.twitter.product\_mixer.core.feature.featuremap.asyncfeaturemap

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

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

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

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

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

import com.twitter.stitch.Stitch

import scala.collection.immutable.Queue

/\*\*

\* An internal representation of an async [[FeatureMap]] containing [[Stitch]]s of [[FeatureMap]]s

\* which are already running in the background.

\*

\* Async features are added by providing the [[PipelineStepIdentifier]] of the [[com.twitter.product\_mixer.core.pipeline.PipelineBuilder.Step Step]]

\* before which the async [[Feature]]s are needed, and a [[Stitch]] of the async [[FeatureMap]].

\* It's expected that the [[Stitch]] has already been started and is running in the background.

\*

\* While not essential to it's core behavior, [[AsyncFeatureMap]] also keeps track of the [[FeatureHydratorIdentifier]]

\* and the Set of [[Feature]]s which will be hydrated for each [[Stitch]] of a [[FeatureMap]] it's given.

\*

\* @param asyncFeatureMaps the [[FeatureMap]]s for [[PipelineStepIdentifier]]s which have not been reached yet

\*

\* @note [[PipelineStepIdentifier]]s must only refer to [[com.twitter.product\_mixer.core.pipeline.PipelineBuilder.Step Step]]s

\* in the current [[com.twitter.product\_mixer.core.pipeline.Pipeline Pipeline]].

\* Only plain [[FeatureMap]]s are passed into underlying [[com.twitter.product\_mixer.core.model.common.Component Component]]s and

\* [[com.twitter.product\_mixer.core.pipeline.Pipeline Pipeline]]s so [[AsyncFeatureMap]]s are scoped

\* for a specific [[com.twitter.product\_mixer.core.pipeline.Pipeline Pipeline]] only.

\*/

@JsonSerialize(using = classOf[AsyncFeatureMapSerializer])

private[core] case class AsyncFeatureMap(

asyncFeatureMaps: Map[PipelineStepIdentifier, Queue[

(FeatureHydratorIdentifier, Set[Feature[\_, \_]], Stitch[FeatureMap])

]]) {

def ++(right: AsyncFeatureMap): AsyncFeatureMap = {

val map = Map.newBuilder[

PipelineStepIdentifier,

Queue[(FeatureHydratorIdentifier, Set[Feature[\_, \_]], Stitch[FeatureMap])]]

(asyncFeatureMaps.keysIterator ++ right.asyncFeatureMaps.keysIterator).foreach { key =>

val currentThenRightAsyncFeatureMaps =

asyncFeatureMaps.getOrElse(key, Queue.empty) ++

right.asyncFeatureMaps.getOrElse(key, Queue.empty)

map += (key -> currentThenRightAsyncFeatureMaps)

}

AsyncFeatureMap(map.result())

}

/\*\*

\* Returns a new [[AsyncFeatureMap]] which now keeps track of the provided `features`

\* and will make them available when calling [[hydrate]] with `hydrateBefore`.

\*

\* @param featureHydratorIdentifier the [[FeatureHydratorIdentifier]] of the [[com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.FeatureHydrator FeatureHydrator]]

\* which these [[Feature]]s are from

\* @param hydrateBefore the [[PipelineStepIdentifier]] before which the [[Feature]]s need to be hydrated

\* @param featuresToHydrate a Set of the [[Feature]]s which will be hydrated

\* @param features a [[Stitch]] of the [[FeatureMap]]

\*/

def addAsyncFeatures(

featureHydratorIdentifier: FeatureHydratorIdentifier,

hydrateBefore: PipelineStepIdentifier,

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

features: Stitch[FeatureMap]

): AsyncFeatureMap = {

val featureMapList =

asyncFeatureMaps.getOrElse(hydrateBefore, Queue.empty) :+

((featureHydratorIdentifier, featuresToHydrate, features))

AsyncFeatureMap(asyncFeatureMaps + (hydrateBefore -> featureMapList))

}

/\*\*

\* The current state of the [[AsyncFeatureMap]] excluding the [[Stitch]]s.

\*/

def features: Map[PipelineStepIdentifier, Seq[(FeatureHydratorIdentifier, Set[Feature[\_, \_]])]] =

asyncFeatureMaps.mapValues(\_.map {

case (featureHydratorIdentifier, features, \_) => (featureHydratorIdentifier, features)

})

/\*\*

\* Returns a [[Some]] containing a [[Stitch]] with a [[FeatureMap]] holding the [[Feature]]s that are

\* supposed to be hydrated at `identifier` if there are [[Feature]]s to hydrate at `identifier`

\*

\* Returns [[None]] if there are no [[Feature]]s to hydrate at the provided `identifier`,

\* this allows for determining if there is work to do without running a [[Stitch]].

\*

\* @note this only hydrates the [[Feature]]s for the specific `identifier`, it does not hydrate

\* [[Feature]]s for earlier Steps.

\* @param identifier the [[PipelineStepIdentifier]] to hydrate [[Feature]]s for

\*/

def hydrate(

identifier: PipelineStepIdentifier

): Option[Stitch[FeatureMap]] =

asyncFeatureMaps.get(identifier) match {

case Some(Queue((\_, \_, featureMap))) =>

// if there is only 1 `FeatureMap` we dont need to do a collect so just return that Stitch

Some(featureMap)

case Some(featureMapList) =>

// if there are multiple `FeatureMap`s we need to collect and merge them together

Some(

Stitch

.collect(featureMapList.map { case (\_, \_, featureMap) => featureMap })

.map { featureMapList => FeatureMap.merge(featureMapList) })

case None =>

// No results for the provided `identifier` so return `None`

None

}

}

private[core] object AsyncFeatureMap {

val empty: AsyncFeatureMap = AsyncFeatureMap(Map.empty)

/\*\*

\* Builds the an [[AsyncFeatureMap]] from a Seq of [[Stitch]] of [[FeatureMap]]

\* tupled with the relevant metadata we use to build the necessary state.

\*

\* This is primarily for convenience, since in most cases an [[AsyncFeatureMap]]

\* will be built from the result of individual [[com.twitter.product\_mixer.core.functional\_component.feature\_hydrator.FeatureHydrator FeatureHydrator]]s

\* and combining them into the correct internal state.

\*/

def fromFeatureMaps(

asyncFeatureMaps: Seq[

(FeatureHydratorIdentifier, PipelineStepIdentifier, Set[Feature[\_, \_]], Stitch[FeatureMap])

]

): AsyncFeatureMap =

AsyncFeatureMap(

asyncFeatureMaps

.groupBy { case (\_, hydrateBefore, \_, \_) => hydrateBefore }

.mapValues(featureMaps =>

Queue(featureMaps.map {

case (hydratorIdentifier, \_, featuresToHydrate, stitch) =>

(hydratorIdentifier, featuresToHydrate, stitch)

}: \_\*)))

}