package com.twitter.product\_mixer.core.service.candidate\_source\_executor

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.BaseCandidateSource

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSourceWithExtractedFeatures

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidatesWithSourceFeatures

import com.twitter.product\_mixer.core.functional\_component.transformer.BaseCandidatePipelineQueryTransformer

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidateFeatureTransformer

import com.twitter.product\_mixer.core.functional\_component.transformer.CandidatePipelineResultsTransformer

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

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

import com.twitter.product\_mixer.core.model.common.presentation.CandidateSourcePosition

import com.twitter.product\_mixer.core.model.common.presentation.CandidateSources

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

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.ExecutionFailed

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.PipelineFailure

import com.twitter.product\_mixer.core.pipeline.pipeline\_failure.UnexpectedCandidateResult

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

import com.twitter.product\_mixer.core.service.candidate\_feature\_transformer\_executor.CandidateFeatureTransformerExecutor

import com.twitter.product\_mixer.core.service.transformer\_executor.PerCandidateTransformerExecutor

import com.twitter.product\_mixer.core.service.transformer\_executor.TransformerExecutor

import com.twitter.stitch.Arrow

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Try

import com.twitter.util.logging.Logging

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.immutable.ListSet

/\*\*

\* [[CandidateSourceExecutor]]:

\* - Executes a [[BaseCandidateSource]], using a [[BaseCandidatePipelineQueryTransformer]] and a [[CandidatePipelineResultsTransformer]]

\* - in parallel, uses a [[CandidateFeatureTransformer]] to optionally extract [[com.twitter.product\_mixer.core.feature.Feature]]s from the result

\* - Handles [[UnexpectedCandidateResult]] [[PipelineFailure]]s returned from [[CandidatePipelineResultsTransformer]] failures by removing those candidates from the result

\*/

@Singleton

class CandidateSourceExecutor @Inject() (

override val statsReceiver: StatsReceiver,

candidateFeatureTransformerExecutor: CandidateFeatureTransformerExecutor,

transformerExecutor: TransformerExecutor,

perCandidateTransformerExecutor: PerCandidateTransformerExecutor)

extends Executor

with Logging {

def arrow[

Query <: PipelineQuery,

CandidateSourceQuery,

CandidateSourceResult,

Candidate <: UniversalNoun[Any]

](

candidateSource: BaseCandidateSource[CandidateSourceQuery, CandidateSourceResult],

queryTransformer: BaseCandidatePipelineQueryTransformer[

Query,

CandidateSourceQuery

],

resultTransformer: CandidatePipelineResultsTransformer[CandidateSourceResult, Candidate],

resultFeaturesTransformers: Seq[CandidateFeatureTransformer[CandidateSourceResult]],

context: Executor.Context

): Arrow[Query, CandidateSourceExecutorResult[Candidate]] = {

val candidateSourceArrow: Arrow[CandidateSourceQuery, CandidatesWithSourceFeatures[

CandidateSourceResult

]] =

candidateSource match {

case regularCandidateSource: CandidateSource[CandidateSourceQuery, CandidateSourceResult] =>

Arrow.flatMap(regularCandidateSource.apply).map { candidates =>

CandidatesWithSourceFeatures(candidates, FeatureMap.empty)

}

case candidateSourceWithExtractedFeatures: CandidateSourceWithExtractedFeatures[

CandidateSourceQuery,

CandidateSourceResult

] =>

Arrow.flatMap(candidateSourceWithExtractedFeatures.apply)

}

val resultsTransformerArrow: Arrow[Seq[CandidateSourceResult], Seq[Try[Candidate]]] =

perCandidateTransformerExecutor.arrow(resultTransformer, context)

val featureMapTransformersArrow: Arrow[

Seq[CandidateSourceResult],

Seq[FeatureMap]

] =

candidateFeatureTransformerExecutor

.arrow(resultFeaturesTransformers, context).map(\_.featureMaps)

val candidatesResultArrow: Arrow[CandidatesWithSourceFeatures[CandidateSourceResult], Seq[

(Candidate, FeatureMap)

]] = Arrow

.map[CandidatesWithSourceFeatures[CandidateSourceResult], Seq[CandidateSourceResult]](

\_.candidates)

.andThen(Arrow

.joinMap(resultsTransformerArrow, featureMapTransformersArrow) {

case (transformed, features) =>

if (transformed.length != features.length)

throw PipelineFailure(

ExecutionFailed,

s"Found ${transformed.length} candidates and ${features.length} FeatureMaps, expected their lengths to be equal")

transformed.iterator

.zip(features.iterator)

.collect { case ErrorHandling(result) => result }

.toSeq

})

// Build the final CandidateSourceExecutorResult

val executorResultArrow: Arrow[

(FeatureMap, Seq[(Candidate, FeatureMap)]),

CandidateSourceExecutorResult[

Candidate

]

] = Arrow.map {

case (queryFeatures: FeatureMap, results: Seq[(Candidate, FeatureMap)]) =>

val candidatesWithFeatures: Seq[FetchedCandidateWithFeatures[Candidate]] =

results.zipWithIndex.map {

case ((candidate, featureMap), index) =>

FetchedCandidateWithFeatures(

candidate,

featureMap + (CandidateSourcePosition, index) + (CandidateSources, ListSet(

candidateSource.identifier))

)

}

CandidateSourceExecutorResult(

candidates = candidatesWithFeatures,

candidateSourceFeatureMap = queryFeatures

)

}

val queryTransformerArrow =

transformerExecutor.arrow[Query, CandidateSourceQuery](queryTransformer, context)

val combinedArrow =

queryTransformerArrow

.andThen(candidateSourceArrow)

.andThen(

Arrow

.join(

Arrow.map[CandidatesWithSourceFeatures[CandidateSourceResult], FeatureMap](

\_.features),

candidatesResultArrow

))

.andThen(executorResultArrow)

wrapComponentWithExecutorBookkeepingWithSize[Query, CandidateSourceExecutorResult[Candidate]](

context,

candidateSource.identifier,

result => result.candidates.size

)(combinedArrow)

}

object ErrorHandling {

/\*\* Silently drop [[UnexpectedCandidateResult]] \*/

def unapply[Candidate](

candidateTryAndFeatureMap: (Try[Candidate], FeatureMap)

): Option[(Candidate, FeatureMap)] = {

val (candidateTry, featureMap) = candidateTryAndFeatureMap

val candidateOpt = candidateTry match {

case Throw(PipelineFailure(UnexpectedCandidateResult, \_, \_, \_)) => None

case Throw(ex) => throw ex

case Return(r) => Some(r)

}

candidateOpt.map { candidate => (candidate, featureMap) }

}

}

}

case class FetchedCandidateWithFeatures[Candidate <: UniversalNoun[Any]](

candidate: Candidate,

features: FeatureMap)

extends CandidateWithFeatures[Candidate]