package com.twitter.product\_mixer.core.pipeline.scoring

import com.twitter.finagle.stats.StatsReceiver

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

import com.twitter.product\_mixer.core.functional\_component.common.alert.Alert

import com.twitter.product\_mixer.core.functional\_component.scorer.ScoredCandidateResult

import com.twitter.product\_mixer.core.gate.ParamGate

import com.twitter.product\_mixer.core.gate.ParamGate.EnabledGateSuffix

import com.twitter.product\_mixer.core.gate.ParamGate.SupportedClientGateSuffix

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

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.product\_mixer.core.pipeline.scoring.ScoringPipeline.Inputs

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

import com.twitter.product\_mixer.core.service.candidate\_feature\_hydrator\_executor.CandidateFeatureHydratorExecutor

import com.twitter.product\_mixer.core.service.candidate\_feature\_hydrator\_executor.CandidateFeatureHydratorExecutorResult

import com.twitter.product\_mixer.core.service.gate\_executor.GateExecutor

import com.twitter.product\_mixer.core.service.gate\_executor.GateExecutorResult

import com.twitter.product\_mixer.core.service.gate\_executor.StoppedGateException

import com.twitter.product\_mixer.core.service.selector\_executor.SelectorExecutor

import com.twitter.product\_mixer.core.service.selector\_executor.SelectorExecutorResult

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* ScoringPipelineBuilder builds [[ScoringPipeline]]s from [[ScoringPipelineConfig]]s.

\*

\* You should inject a [[ScoringPipelineBuilderFactory]] and call `.get` to build these.

\*

\* @see [[ScoringPipelineConfig]] for the description of the type parameters

\* @tparam Query the type of query these accept.

\* @tparam Candidate the domain model for the candidate being scored

\*/

class ScoringPipelineBuilder[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]] @Inject() (

gateExecutor: GateExecutor,

selectorExecutor: SelectorExecutor,

candidateFeatureHydratorExecutor: CandidateFeatureHydratorExecutor,

override val statsReceiver: StatsReceiver)

extends PipelineBuilder[Inputs[Query]] {

override type UnderlyingResultType = Seq[ScoredCandidateResult[Candidate]]

override type PipelineResultType = ScoringPipelineResult[Candidate]

def build(

parentComponentIdentifierStack: ComponentIdentifierStack,

config: ScoringPipelineConfig[Query, Candidate]

): ScoringPipeline[Query, Candidate] = {

val pipelineIdentifier = config.identifier

val context = Executor.Context(

PipelineFailureClassifier(

config.failureClassifier.orElse(StoppedGateException.classifier(ClosedGate))),

parentComponentIdentifierStack.push(pipelineIdentifier)

)

val enabledGateOpt = config.enabledDeciderParam.map { deciderParam =>

ParamGate(pipelineIdentifier + EnabledGateSuffix, deciderParam)

}

val supportedClientGateOpt = config.supportedClientParam.map { param =>

ParamGate(pipelineIdentifier + SupportedClientGateSuffix, param)

}

/\*\*

\* Evaluate enabled decider gate first since if it's off, there is no reason to proceed

\* Next evaluate supported client feature switch gate, followed by customer configured gates

\*/

val allGates = enabledGateOpt.toSeq ++ supportedClientGateOpt.toSeq ++ config.gates

val GatesStep = new Step[Query, GateExecutorResult] {

override def identifier: PipelineStepIdentifier = ScoringPipelineConfig.gatesStep

override lazy val executorArrow: Arrow[Query, GateExecutorResult] =

gateExecutor.arrow(allGates, context)

override def inputAdaptor(

query: ScoringPipeline.Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): Query = {

query.query

}

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: GateExecutorResult

): ScoringPipelineResult[Candidate] =

previousPipelineResult.copy(gateResults = Some(executorResult))

}

val SelectorsStep = new Step[SelectorExecutor.Inputs[Query], SelectorExecutorResult] {

override def identifier: PipelineStepIdentifier = ScoringPipelineConfig.selectorsStep

override def executorArrow: Arrow[SelectorExecutor.Inputs[Query], SelectorExecutorResult] =

selectorExecutor.arrow(config.selectors, context)

override def inputAdaptor(

query: ScoringPipeline.Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): SelectorExecutor.Inputs[Query] = SelectorExecutor.Inputs(query.query, query.candidates)

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: SelectorExecutorResult

): ScoringPipelineResult[Candidate] =

previousPipelineResult.copy(selectorResults = Some(executorResult))

}

val PreScoringFeatureHydrationPhase1Step =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] {

override def identifier: PipelineStepIdentifier =

ScoringPipelineConfig.preScoringFeatureHydrationPhase1Step

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] =

candidateFeatureHydratorExecutor.arrow(config.preScoringFeatureHydrationPhase1, context)

override def inputAdaptor(

query: ScoringPipeline.Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): CandidateFeatureHydratorExecutor.Inputs[Query, Candidate] = {

val selectedCandidatesResult = previousResult.selectorResults.getOrElse {

throw InvalidStepStateException(identifier, "SelectorResults")

}.selectedCandidates

CandidateFeatureHydratorExecutor.Inputs(

query.query,

selectedCandidatesResult.asInstanceOf[Seq[CandidateWithFeatures[Candidate]]])

}

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: CandidateFeatureHydratorExecutorResult[Candidate]

): ScoringPipelineResult[Candidate] = previousPipelineResult.copy(

preScoringHydrationPhase1Result = Some(executorResult)

)

}

val PreScoringFeatureHydrationPhase2Step =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] {

override def identifier: PipelineStepIdentifier =

ScoringPipelineConfig.preScoringFeatureHydrationPhase2Step

override def executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] =

candidateFeatureHydratorExecutor.arrow(config.preScoringFeatureHydrationPhase2, context)

override def inputAdaptor(

query: ScoringPipeline.Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): CandidateFeatureHydratorExecutor.Inputs[Query, Candidate] = {

val preScoringHydrationPhase1FeatureMaps: Seq[FeatureMap] =

previousResult.preScoringHydrationPhase1Result

.getOrElse(

throw InvalidStepStateException(identifier, "PreScoringHydrationPhase1Result"))

.results.map(\_.features)

val itemCandidates = previousResult.selectorResults

.getOrElse(throw InvalidStepStateException(identifier, "SelectionResults"))

.selectedCandidates.collect {

case itemCandidate: ItemCandidateWithDetails => itemCandidate

}

// If there is no feature hydration (empty results), no need to attempt merging.

val candidates = if (preScoringHydrationPhase1FeatureMaps.isEmpty) {

itemCandidates

} else {

itemCandidates.zip(preScoringHydrationPhase1FeatureMaps).map {

case (itemCandidate, featureMap) =>

itemCandidate.copy(features = itemCandidate.features ++ featureMap)

}

}

CandidateFeatureHydratorExecutor.Inputs(

query.query,

candidates.asInstanceOf[Seq[CandidateWithFeatures[Candidate]]])

}

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: CandidateFeatureHydratorExecutorResult[Candidate]

): ScoringPipelineResult[Candidate] = previousPipelineResult.copy(

preScoringHydrationPhase2Result = Some(executorResult)

)

}

def getMergedPreScoringFeatureMap(

stepIdentifier: PipelineStepIdentifier,

previousResult: ScoringPipelineResult[Candidate]

): Seq[FeatureMap] = {

val preScoringHydrationPhase1FeatureMaps: Seq[FeatureMap] =

previousResult.preScoringHydrationPhase1Result

.getOrElse(

throw InvalidStepStateException(

stepIdentifier,

"PreScoringHydrationPhase1Result")).results.map(\_.features)

val preScoringHydrationPhase2FeatureMaps: Seq[FeatureMap] =

previousResult.preScoringHydrationPhase2Result

.getOrElse(

throw InvalidStepStateException(

stepIdentifier,

"PreScoringHydrationPhase2Result")).results.map(\_.features)

/\*

\* If either pre-scoring hydration phase feature map is empty, no need to merge them,

\* we can just take all non-empty ones.

\*/

if (preScoringHydrationPhase1FeatureMaps.isEmpty) {

preScoringHydrationPhase2FeatureMaps

} else if (preScoringHydrationPhase2FeatureMaps.isEmpty) {

preScoringHydrationPhase1FeatureMaps

} else {

// No need to check the size in both, since the inputs to both hydration phases are the

// same and each phase ensures the number of candidates and ordering matches the input.

preScoringHydrationPhase1FeatureMaps.zip(preScoringHydrationPhase2FeatureMaps).map {

case (preScoringHydrationPhase1FeatureMap, preScoringHydrationPhasesFeatureMap) =>

preScoringHydrationPhase1FeatureMap ++ preScoringHydrationPhasesFeatureMap

}

}

}

val ScorersStep =

new Step[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[Candidate]

] {

override def identifier: PipelineStepIdentifier = ScoringPipelineConfig.scorersStep

override def inputAdaptor(

query: ScoringPipeline.Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): CandidateFeatureHydratorExecutor.Inputs[Query, Candidate] = {

val mergedPreScoringFeatureHydrationFeatures: Seq[FeatureMap] =

getMergedPreScoringFeatureMap(ScoringPipelineConfig.scorersStep, previousResult)

val itemCandidates = previousResult.selectorResults

.getOrElse(throw InvalidStepStateException(identifier, "SelectionResults"))

.selectedCandidates.collect {

case itemCandidate: ItemCandidateWithDetails => itemCandidate

}

// If there was no pre-scoring features hydration, no need to re-merge feature maps

// and construct a new item candidate

val updatedCandidates = if (mergedPreScoringFeatureHydrationFeatures.isEmpty) {

itemCandidates

} else {

itemCandidates.zip(mergedPreScoringFeatureHydrationFeatures).map {

case (itemCandidate, preScoringFeatureMap) =>

itemCandidate.copy(features = itemCandidate.features ++ preScoringFeatureMap)

}

}

CandidateFeatureHydratorExecutor.Inputs(

query.query,

updatedCandidates.asInstanceOf[Seq[CandidateWithFeatures[Candidate]]])

}

override lazy val executorArrow: Arrow[

CandidateFeatureHydratorExecutor.Inputs[Query, Candidate],

CandidateFeatureHydratorExecutorResult[

Candidate

]

] = candidateFeatureHydratorExecutor.arrow(config.scorers.toSeq, context)

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: CandidateFeatureHydratorExecutorResult[Candidate]

): ScoringPipelineResult[Candidate] =

previousPipelineResult.copy(scorerResults = Some(executorResult))

}

val ResultStep =

new Step[Seq[CandidateWithFeatures[UniversalNoun[Any]]], Seq[

CandidateWithFeatures[UniversalNoun[Any]]

]] {

override def identifier: PipelineStepIdentifier = ScoringPipelineConfig.resultStep

override def executorArrow: Arrow[Seq[CandidateWithFeatures[UniversalNoun[Any]]], Seq[

CandidateWithFeatures[UniversalNoun[Any]]

]] = Arrow.identity

override def inputAdaptor(

query: Inputs[Query],

previousResult: ScoringPipelineResult[Candidate]

): Seq[CandidateWithFeatures[UniversalNoun[Any]]] = previousResult.selectorResults

.getOrElse(throw InvalidStepStateException(identifier, "SelectionResults"))

.selectedCandidates.collect {

case itemCandidate: ItemCandidateWithDetails => itemCandidate

}

override def resultUpdater(

previousPipelineResult: ScoringPipelineResult[Candidate],

executorResult: Seq[CandidateWithFeatures[UniversalNoun[Any]]]

): ScoringPipelineResult[Candidate] = {

val scorerResults: Seq[FeatureMap] = previousPipelineResult.scorerResults

.getOrElse(throw InvalidStepStateException(identifier, "ScorerResult")).results.map(

\_.features)

val mergedPreScoringFeatureHydrationFeatureMaps: Seq[FeatureMap] =

getMergedPreScoringFeatureMap(ScoringPipelineConfig.resultStep, previousPipelineResult)

val itemCandidates = executorResult.asInstanceOf[Seq[ItemCandidateWithDetails]]

val finalFeatureMap = if (mergedPreScoringFeatureHydrationFeatureMaps.isEmpty) {

scorerResults

} else {

scorerResults

.zip(mergedPreScoringFeatureHydrationFeatureMaps).map {

case (preScoringFeatureMap, scoringFeatureMap) =>

preScoringFeatureMap ++ scoringFeatureMap

}

}

val finalResults = itemCandidates.zip(finalFeatureMap).map {

case (itemCandidate, featureMap) =>

ScoredCandidateResult(itemCandidate.candidate.asInstanceOf[Candidate], featureMap)

}

previousPipelineResult.withResult(finalResults)

}

}

val builtSteps = Seq(

GatesStep,

SelectorsStep,

PreScoringFeatureHydrationPhase1Step,

PreScoringFeatureHydrationPhase2Step,

ScorersStep,

ResultStep

)

/\*\* The main execution logic for this Candidate Pipeline. \*/

val finalArrow: Arrow[ScoringPipeline.Inputs[Query], ScoringPipelineResult[Candidate]] =

buildCombinedArrowFromSteps(

steps = builtSteps,

context = context,

initialEmptyResult = ScoringPipelineResult.empty,

stepsInOrderFromConfig = ScoringPipelineConfig.stepsInOrder

)

val configFromBuilder = config

new ScoringPipeline[Query, Candidate] {

override private[core] val config: ScoringPipelineConfig[Query, Candidate] = configFromBuilder

override val arrow: Arrow[ScoringPipeline.Inputs[Query], ScoringPipelineResult[Candidate]] =

finalArrow

override val identifier: ScoringPipelineIdentifier = pipelineIdentifier

override val alerts: Seq[Alert] = config.alerts

override val children: Seq[Component] =

allGates ++ config.preScoringFeatureHydrationPhase1 ++ config.preScoringFeatureHydrationPhase2 ++ config.scorers

}

}

}