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

import com.twitter.finagle.stats.StatsReceiver

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.common.alert.Alert

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

import com.twitter.product\_mixer.core.functional\_component.gate.Gate

import com.twitter.product\_mixer.core.functional\_component.premarshaller.DomainMarshaller

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.side\_effect.PipelineResultSideEffect

import com.twitter.product\_mixer.core.functional\_component.marshaller.TransportMarshaller

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

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

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

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

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

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

import com.twitter.product\_mixer.core.model.marshalling.HasMarshalling

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

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.candidate.CandidatePipeline

import com.twitter.product\_mixer.core.pipeline.candidate.CandidatePipelineBuilderFactory

import com.twitter.product\_mixer.core.pipeline.candidate.CandidatePipelineConfig

import com.twitter.product\_mixer.core.pipeline.candidate.DependentCandidatePipelineConfig

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

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

import com.twitter.product\_mixer.core.quality\_factor.HasQualityFactorStatus

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorObserver

import com.twitter.product\_mixer.core.quality\_factor.QualityFactorStatus

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.product\_mixer.core.service.candidate\_pipeline\_executor.CandidatePipelineExecutor

import com.twitter.product\_mixer.core.service.candidate\_pipeline\_executor.CandidatePipelineExecutorResult

import com.twitter.product\_mixer.core.service.domain\_marshaller\_executor.DomainMarshallerExecutor

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.pipeline\_result\_side\_effect\_executor.PipelineResultSideEffectExecutor

import com.twitter.product\_mixer.core.service.quality\_factor\_executor.QualityFactorExecutorResult

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

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

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

import com.twitter.product\_mixer.core.service.transport\_marshaller\_executor.TransportMarshallerExecutor

import com.twitter.stitch.Arrow

import com.twitter.util.logging.Logging

/\*\*

\* MixerPipelineBuilder builds [[MixerPipeline]]s from [[MixerPipelineConfig]]s.

\*

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

\*

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

\*/

class MixerPipelineBuilder[Query <: PipelineQuery, DomainResultType <: HasMarshalling, Result](

candidatePipelineExecutor: CandidatePipelineExecutor,

gateExecutor: GateExecutor,

selectorExecutor: SelectorExecutor,

queryFeatureHydratorExecutor: QueryFeatureHydratorExecutor,

asyncFeatureMapExecutor: AsyncFeatureMapExecutor,

domainMarshallerExecutor: DomainMarshallerExecutor,

transportMarshallerExecutor: TransportMarshallerExecutor,

pipelineResultSideEffectExecutor: PipelineResultSideEffectExecutor,

candidatePipelineBuilderFactory: CandidatePipelineBuilderFactory,

override val statsReceiver: StatsReceiver)

extends PipelineBuilder[Query]

with Logging {

override type UnderlyingResultType = Result

override type PipelineResultType = MixerPipelineResult[Result]

def qualityFactorStep(

qualityFactorStatus: QualityFactorStatus

): Step[Query, QualityFactorExecutorResult] =

new Step[Query, QualityFactorExecutorResult] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.qualityFactorStep

override def executorArrow: Arrow[Query, QualityFactorExecutorResult] =

Arrow

.map[Query, QualityFactorExecutorResult] { \_ =>

QualityFactorExecutorResult(

pipelineQualityFactors =

qualityFactorStatus.qualityFactorByPipeline.mapValues(\_.currentValue)

)

}

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): Query = query

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: QualityFactorExecutorResult

): MixerPipelineResult[Result] =

previousPipelineResult.copy(qualityFactorResult = Some(executorResult))

override def queryUpdater(

query: Query,

executorResult: QualityFactorExecutorResult

): Query = {

query match {

case queryWithQualityFactor: HasQualityFactorStatus =>

queryWithQualityFactor

.withQualityFactorStatus(

queryWithQualityFactor.qualityFactorStatus.getOrElse(QualityFactorStatus.empty) ++

qualityFactorStatus

).asInstanceOf[Query]

case \_ =>

query

}

}

}

def gatesStep(

gates: Seq[Gate[Query]],

context: Executor.Context

): Step[Query, GateExecutorResult] = new Step[Query, GateExecutorResult] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.gatesStep

override def executorArrow: Arrow[Query, GateExecutorResult] =

gateExecutor.arrow(gates, context)

override def inputAdaptor(query: Query, previousResult: MixerPipelineResult[Result]): Query =

query

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: GateExecutorResult

): MixerPipelineResult[Result] =

previousPipelineResult.copy(gateResult = Some(executorResult))

}

def fetchQueryFeaturesStep(

queryFeatureHydrators: Seq[QueryFeatureHydrator[Query]],

stepIdentifier: PipelineStepIdentifier,

updater: ResultUpdater[MixerPipelineResult[Result], QueryFeatureHydratorExecutor.Result],

context: Executor.Context

): Step[Query, QueryFeatureHydratorExecutor.Result] =

new Step[Query, QueryFeatureHydratorExecutor.Result] {

override def identifier: PipelineStepIdentifier = stepIdentifier

override def executorArrow: Arrow[Query, QueryFeatureHydratorExecutor.Result] =

queryFeatureHydratorExecutor.arrow(

queryFeatureHydrators,

MixerPipelineConfig.stepsAsyncFeatureHydrationCanBeCompletedBy,

context)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): Query = query

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: QueryFeatureHydratorExecutor.Result

): MixerPipelineResult[Result] =

updater(previousPipelineResult, executorResult)

override def queryUpdater(

query: Query,

executorResult: QueryFeatureHydratorExecutor.Result

): Query =

query

.withFeatureMap(

query.features

.getOrElse(FeatureMap.empty) ++ executorResult.featureMap).asInstanceOf[Query]

}

def asyncFeaturesStep(

stepToHydrateFor: PipelineStepIdentifier,

context: Executor.Context

): Step[AsyncFeatureMap, AsyncFeatureMapExecutorResults] =

new Step[AsyncFeatureMap, AsyncFeatureMapExecutorResults] {

override def identifier: PipelineStepIdentifier =

MixerPipelineConfig.asyncFeaturesStep(stepToHydrateFor)

override def executorArrow: Arrow[AsyncFeatureMap, AsyncFeatureMapExecutorResults] =

asyncFeatureMapExecutor.arrow(

stepToHydrateFor,

identifier,

context

)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): AsyncFeatureMap =

previousResult.mergedAsyncQueryFeatures

.getOrElse(

throw InvalidStepStateException(identifier, "MergedAsyncQueryFeatures")

)

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: AsyncFeatureMapExecutorResults

): MixerPipelineResult[Result] = previousPipelineResult.copy(

asyncFeatureHydrationResults = previousPipelineResult.asyncFeatureHydrationResults match {

case Some(existingResults) => Some(existingResults ++ executorResult)

case None => Some(executorResult)

})

override def queryUpdater(

query: Query,

executorResult: AsyncFeatureMapExecutorResults

): Query =

if (executorResult.featureMapsByStep

.getOrElse(stepToHydrateFor, FeatureMap.empty).isEmpty) {

query

} else {

query

.withFeatureMap(

query.features

.getOrElse(FeatureMap.empty) ++ executorResult.featureMapsByStep(

stepToHydrateFor)).asInstanceOf[Query]

}

}

def candidatePipelinesStep(

candidatePipelines: Seq[CandidatePipeline[Query]],

defaultFailOpenPolicy: FailOpenPolicy,

failOpenPolicies: Map[CandidatePipelineIdentifier, FailOpenPolicy],

qualityFactorObserverByPipeline: Map[ComponentIdentifier, QualityFactorObserver],

context: Executor.Context

): Step[CandidatePipeline.Inputs[Query], CandidatePipelineExecutorResult] =

new Step[CandidatePipeline.Inputs[Query], CandidatePipelineExecutorResult] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.candidatePipelinesStep

override def executorArrow: Arrow[CandidatePipeline.Inputs[

Query

], CandidatePipelineExecutorResult] =

candidatePipelineExecutor

.arrow(

candidatePipelines,

defaultFailOpenPolicy,

failOpenPolicies,

qualityFactorObserverByPipeline,

context

)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): CandidatePipeline.Inputs[Query] = CandidatePipeline.Inputs[Query](query, Seq.empty)

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: CandidatePipelineExecutorResult

): MixerPipelineResult[Result] =

previousPipelineResult.copy(candidatePipelineResults = Some(executorResult))

override def queryUpdater(

query: Query,

executorResult: CandidatePipelineExecutorResult

): Query = {

val updatedFeatureMap = query.features

.getOrElse(FeatureMap.empty) ++ executorResult.queryFeatureMap

query

.withFeatureMap(updatedFeatureMap).asInstanceOf[Query]

}

}

def dependentCandidatePipelinesStep(

candidatePipelines: Seq[CandidatePipeline[Query]],

defaultFailOpenPolicy: FailOpenPolicy,

failOpenPolicies: Map[CandidatePipelineIdentifier, FailOpenPolicy],

qualityFactorObserverByPipeline: Map[ComponentIdentifier, QualityFactorObserver],

context: Executor.Context

): Step[CandidatePipeline.Inputs[Query], CandidatePipelineExecutorResult] =

new Step[CandidatePipeline.Inputs[Query], CandidatePipelineExecutorResult] {

override def identifier: PipelineStepIdentifier =

MixerPipelineConfig.dependentCandidatePipelinesStep

override def executorArrow: Arrow[CandidatePipeline.Inputs[

Query

], CandidatePipelineExecutorResult] =

candidatePipelineExecutor

.arrow(

candidatePipelines,

defaultFailOpenPolicy,

failOpenPolicies,

qualityFactorObserverByPipeline,

context

)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): CandidatePipeline.Inputs[Query] = {

val previousCandidates = previousResult.candidatePipelineResults

.getOrElse {

throw InvalidStepStateException(identifier, "Candidates")

}.candidatePipelineResults.flatMap(\_.result.getOrElse(Seq.empty))

CandidatePipeline.Inputs[Query](query, previousCandidates)

}

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: CandidatePipelineExecutorResult

): MixerPipelineResult[Result] =

previousPipelineResult.copy(dependentCandidatePipelineResults = Some(executorResult))

override def queryUpdater(

query: Query,

executorResult: CandidatePipelineExecutorResult

): Query = {

val updatedFeatureMap = query.features

.getOrElse(FeatureMap.empty) ++ executorResult.queryFeatureMap

query

.withFeatureMap(updatedFeatureMap).asInstanceOf[Query]

}

}

def resultSelectorsStep(

selectors: Seq[Selector[Query]],

context: Executor.Context

): Step[SelectorExecutor.Inputs[Query], SelectorExecutorResult] =

new Step[SelectorExecutor.Inputs[Query], SelectorExecutorResult] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.resultSelectorsStep

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

selectorExecutor.arrow(selectors, context)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): SelectorExecutor.Inputs[Query] = {

val candidates = previousResult.candidatePipelineResults

.getOrElse {

throw InvalidStepStateException(identifier, "Candidates")

}.candidatePipelineResults.flatMap(\_.result.getOrElse(Seq.empty))

val dependentCandidates =

previousResult.dependentCandidatePipelineResults

.getOrElse {

throw InvalidStepStateException(identifier, "DependentCandidates")

}.candidatePipelineResults.flatMap(\_.result.getOrElse(Seq.empty))

SelectorExecutor.Inputs(

query = query,

candidatesWithDetails = candidates ++ dependentCandidates

)

}

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: SelectorExecutorResult

): MixerPipelineResult[Result] =

previousPipelineResult.copy(resultSelectorResults = Some(executorResult))

}

def domainMarshallingStep(

domainMarshaller: DomainMarshaller[Query, DomainResultType],

context: Executor.Context

): Step[DomainMarshallerExecutor.Inputs[Query], DomainMarshallerExecutor.Result[

DomainResultType

]] =

new Step[DomainMarshallerExecutor.Inputs[Query], DomainMarshallerExecutor.Result[

DomainResultType

]] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.domainMarshallerStep

override def executorArrow: Arrow[

DomainMarshallerExecutor.Inputs[Query],

DomainMarshallerExecutor.Result[DomainResultType]

] =

domainMarshallerExecutor.arrow(domainMarshaller, context)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): DomainMarshallerExecutor.Inputs[Query] = {

val selectorResults = previousResult.resultSelectorResults.getOrElse {

throw InvalidStepStateException(identifier, "SelectorResults")

}

DomainMarshallerExecutor.Inputs(

query = query,

candidatesWithDetails = selectorResults.selectedCandidates

)

}

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: DomainMarshallerExecutor.Result[DomainResultType]

): MixerPipelineResult[Result] = previousPipelineResult.copy(

domainMarshallerResults = Some(executorResult)

)

}

def resultSideEffectsStep(

sideEffects: Seq[PipelineResultSideEffect[Query, DomainResultType]],

context: Executor.Context

): Step[

PipelineResultSideEffect.Inputs[Query, DomainResultType],

PipelineResultSideEffectExecutor.Result

] = new Step[

PipelineResultSideEffect.Inputs[Query, DomainResultType],

PipelineResultSideEffectExecutor.Result

] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.resultSideEffectsStep

override def executorArrow: Arrow[

PipelineResultSideEffect.Inputs[Query, DomainResultType],

PipelineResultSideEffectExecutor.Result

] = pipelineResultSideEffectExecutor.arrow(sideEffects, context)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): PipelineResultSideEffect.Inputs[Query, DomainResultType] = {

val selectorResults = previousResult.resultSelectorResults.getOrElse {

throw InvalidStepStateException(identifier, "SelectorResults")

}

val domainMarshallerResults = previousResult.domainMarshallerResults.getOrElse {

throw InvalidStepStateException(identifier, "DomainMarshallerResults")

}

PipelineResultSideEffect.Inputs[Query, DomainResultType](

query = query,

selectedCandidates = selectorResults.selectedCandidates,

remainingCandidates = selectorResults.remainingCandidates,

droppedCandidates = selectorResults.droppedCandidates,

response = domainMarshallerResults.result.asInstanceOf[DomainResultType]

)

}

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: PipelineResultSideEffectExecutor.Result

): MixerPipelineResult[Result] =

previousPipelineResult.copy(resultSideEffectResults = Some(executorResult))

}

def transportMarshallingStep(

transportMarshaller: TransportMarshaller[DomainResultType, Result],

context: Executor.Context

): Step[

TransportMarshallerExecutor.Inputs[DomainResultType],

TransportMarshallerExecutor.Result[Result]

] = new Step[TransportMarshallerExecutor.Inputs[

DomainResultType

], TransportMarshallerExecutor.Result[Result]] {

override def identifier: PipelineStepIdentifier = MixerPipelineConfig.transportMarshallerStep

override def executorArrow: Arrow[TransportMarshallerExecutor.Inputs[

DomainResultType

], TransportMarshallerExecutor.Result[Result]] =

transportMarshallerExecutor.arrow(transportMarshaller, context)

override def inputAdaptor(

query: Query,

previousResult: MixerPipelineResult[Result]

): TransportMarshallerExecutor.Inputs[DomainResultType] = {

val domainMarshallingResults = previousResult.domainMarshallerResults.getOrElse {

throw InvalidStepStateException(identifier, "DomainMarshallerResults")

}

// Since the PipelineResult just uses HasMarshalling

val domainResult = domainMarshallingResults.result.asInstanceOf[DomainResultType]

TransportMarshallerExecutor.Inputs(domainResult)

}

override def resultUpdater(

previousPipelineResult: MixerPipelineResult[Result],

executorResult: TransportMarshallerExecutor.Result[Result]

): MixerPipelineResult[Result] = previousPipelineResult.copy(

transportMarshallerResults = Some(executorResult),

result = Some(executorResult.result)

)

}

def build(

parentComponentIdentifierStack: ComponentIdentifierStack,

config: MixerPipelineConfig[Query, DomainResultType, Result]

): MixerPipeline[Query, Result] = {

val pipelineIdentifier = config.identifier

val context = Executor.Context(

PipelineFailureClassifier(

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

parentComponentIdentifierStack.push(pipelineIdentifier)

)

val qualityFactorStatus: QualityFactorStatus =

QualityFactorStatus.build(config.qualityFactorConfigs)

val qualityFactorObserverByPipeline =

qualityFactorStatus.qualityFactorByPipeline.mapValues { qualityFactor =>

qualityFactor.buildObserver()

}

buildGaugesForQualityFactor(pipelineIdentifier, qualityFactorStatus, statsReceiver)

val candidatePipelines: Seq[CandidatePipeline[Query]] = config.candidatePipelines.map {

pipelineConfig: CandidatePipelineConfig[Query, \_, \_, \_] =>

pipelineConfig.build(context.componentStack, candidatePipelineBuilderFactory)

}

val dependentCandidatePipelines: Seq[CandidatePipeline[Query]] =

config.dependentCandidatePipelines.map {

pipelineConfig: DependentCandidatePipelineConfig[Query, \_, \_, \_] =>

pipelineConfig.build(context.componentStack, candidatePipelineBuilderFactory)

}

val builtSteps = Seq(

qualityFactorStep(qualityFactorStatus),

gatesStep(config.gates, context),

fetchQueryFeaturesStep(

config.fetchQueryFeatures,

MixerPipelineConfig.fetchQueryFeaturesStep,

(previousPipelineResult, executorResult) =>

previousPipelineResult.copy(queryFeatures = Some(executorResult)),

context

),

fetchQueryFeaturesStep(

config.fetchQueryFeaturesPhase2,

MixerPipelineConfig.fetchQueryFeaturesPhase2Step,

(previousPipelineResult, executorResult) =>

previousPipelineResult.copy(

queryFeaturesPhase2 = Some(executorResult),

mergedAsyncQueryFeatures = Some(

previousPipelineResult.queryFeatures

.getOrElse(throw InvalidStepStateException(

MixerPipelineConfig.fetchQueryFeaturesPhase2Step,

"QueryFeatures"))

.asyncFeatureMap ++ executorResult.asyncFeatureMap)

),

context

),

asyncFeaturesStep(MixerPipelineConfig.candidatePipelinesStep, context),

candidatePipelinesStep(

candidatePipelines,

config.defaultFailOpenPolicy,

config.failOpenPolicies,

qualityFactorObserverByPipeline,

context),

asyncFeaturesStep(MixerPipelineConfig.dependentCandidatePipelinesStep, context),

dependentCandidatePipelinesStep(

dependentCandidatePipelines,

config.defaultFailOpenPolicy,

config.failOpenPolicies,

qualityFactorObserverByPipeline,

context),

asyncFeaturesStep(MixerPipelineConfig.resultSelectorsStep, context),

resultSelectorsStep(config.resultSelectors, context),

domainMarshallingStep(config.domainMarshaller, context),

asyncFeaturesStep(MixerPipelineConfig.resultSideEffectsStep, context),

resultSideEffectsStep(config.resultSideEffects, context),

transportMarshallingStep(config.transportMarshaller, context)

)

val finalArrow = buildCombinedArrowFromSteps(

steps = builtSteps,

context = context,

initialEmptyResult = MixerPipelineResult.empty,

stepsInOrderFromConfig = MixerPipelineConfig.stepsInOrder

)

val configFromBuilder = config

new MixerPipeline[Query, Result] {

override private[core] val config: MixerPipelineConfig[Query, \_, Result] = configFromBuilder

override val arrow: Arrow[Query, MixerPipelineResult[Result]] = finalArrow

override val identifier: MixerPipelineIdentifier = pipelineIdentifier

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

override val children: Seq[Component] =

config.gates ++

config.fetchQueryFeatures ++

candidatePipelines ++

dependentCandidatePipelines ++

config.resultSideEffects ++

Seq(config.domainMarshaller, config.transportMarshaller)

}

}

}