package com.twitter.product\_mixer.core.pipeline.step.transport\_marshaller

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

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.pipeline\_failure.IllegalStateFailure

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

import com.twitter.product\_mixer.core.pipeline.state.HasExecutorResults

import com.twitter.product\_mixer.core.pipeline.step.Step

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

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

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

import com.twitter.stitch.Arrow

import javax.inject.Inject

/\*\*

\* A transport marshaller step, it takes domain marshalled result as input and returns trasnport

\* ready marshalled object.

\* The [[State]] object is responsible for keeping a reference of the built marshalled response.

\*

\* @param transportMarshallerExecutor Domain Marshaller executor.

\* @tparam Query Type of PipelineQuery domain model

\* @tparam DomainResponseType the domain marshalling type used as input

\* @tparam TransportResponseType the expected returned transport type

\* @tparam State The pipeline state domain model.

\*/

case class TransportMarshallerStep[

DomainResponseType <: HasMarshalling,

TransportResponseType,

State <: HasExecutorResults[State]] @Inject() (

transportMarshallerExecutor: TransportMarshallerExecutor)

extends Step[

State,

TransportMarshallerConfig[DomainResponseType, TransportResponseType],

TransportMarshallerExecutor.Inputs[DomainResponseType],

TransportMarshallerExecutor.Result[TransportResponseType]

] {

override def isEmpty(

config: TransportMarshallerConfig[DomainResponseType, TransportResponseType]

): Boolean = false

override def adaptInput(

state: State,

config: TransportMarshallerConfig[DomainResponseType, TransportResponseType]

): TransportMarshallerExecutor.Inputs[DomainResponseType] = {

val domainMarshallerResult = state.executorResultsByPipelineStep

.getOrElse(

config.domainMarshallerStepIdentifier,

throw PipelineFailure(

IllegalStateFailure,

"Missing Domain Marshaller in Transport Marshaller Step")).asInstanceOf[

DomainMarshallerExecutor.Result[DomainResponseType]]

TransportMarshallerExecutor.Inputs(domainMarshallerResult.result)

}

// Noop as platform updates executor result

override def updateState(

state: State,

executorResult: TransportMarshallerExecutor.Result[TransportResponseType],

config: TransportMarshallerConfig[DomainResponseType, TransportResponseType]

): State = state

override def arrow(

config: TransportMarshallerConfig[DomainResponseType, TransportResponseType],

context: Executor.Context

): Arrow[TransportMarshallerExecutor.Inputs[

DomainResponseType

], TransportMarshallerExecutor.Result[TransportResponseType]] =

transportMarshallerExecutor.arrow(config.transportMarshaller, context)

}

case class TransportMarshallerConfig[DomainResponseType <: HasMarshalling, TransportResponseType](

transportMarshaller: TransportMarshaller[DomainResponseType, TransportResponseType],

domainMarshallerStepIdentifier: PipelineStepIdentifier)