package com.twitter.product\_mixer.core.service.transport\_marshaller\_executor

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

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

import com.twitter.stitch.Arrow

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Executes a [[TransportMarshaller]].

\*

\* @note This is a synchronous transform, so we don't observe it directly. Failures and such

\* can be observed at the parent pipeline.

\*/

@Singleton

class TransportMarshallerExecutor @Inject() (override val statsReceiver: StatsReceiver)

extends Executor {

def arrow[DomainResponseType <: HasMarshalling, TransportResponseType](

marshaller: TransportMarshaller[DomainResponseType, TransportResponseType],

context: Executor.Context

): Arrow[Inputs[DomainResponseType], Result[TransportResponseType]] = {

val arrow =

Arrow.map[Inputs[DomainResponseType], Result[TransportResponseType]] {

case Inputs(domainResponse) => Result(marshaller(domainResponse))

}

wrapComponentWithExecutorBookkeeping(context, marshaller.identifier)(arrow)

}

}

object TransportMarshallerExecutor {

case class Inputs[DomainResponseType <: HasMarshalling](domainResponse: DomainResponseType)

case class Result[TransportResponseType](result: TransportResponseType) extends ExecutorResult

}