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

import com.twitter.finagle.stats.StatsReceiver

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

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

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

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

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

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

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

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

import com.twitter.stitch.Arrow

import javax.inject.Inject

import javax.inject.Singleton

/\*\*

\* Executes a [[DomainMarshaller]].

\*

\* @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 DomainMarshallerExecutor @Inject() (override val statsReceiver: StatsReceiver)

extends Executor {

def arrow[Query <: PipelineQuery, DomainResponseType <: HasMarshalling](

marshaller: DomainMarshaller[Query, DomainResponseType],

context: Executor.Context

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

val arrow = Arrow

.map[Inputs[Query], DomainMarshallerExecutor.Result[DomainResponseType]] {

case Inputs(query, candidates) =>

DomainMarshallerExecutor.Result(marshaller(query, candidates))

}

wrapComponentWithExecutorBookkeeping(context, marshaller.identifier)(arrow)

}

}

object DomainMarshallerExecutor {

case class Inputs[Query <: PipelineQuery](

query: Query,

candidatesWithDetails: Seq[CandidateWithDetails])

case class Result[+DomainResponseType](result: DomainResponseType) extends ExecutorResult

}