package com.twitter.product\_mixer.core.pipeline.step.query\_transformer

import com.twitter.product\_mixer.core.model.marshalling.request.Request

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

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

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

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

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

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

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

import com.twitter.stitch.Arrow

import com.twitter.timelines.configapi.Params

/\*\*

\* Query Transformation Step that takes an incoming thrift request model object and returns a

\* pipeline query. The pipeline state is responsible for keeping the updated query.

\*

\* @tparam TRequest Thrift request domain model

\* @tparam Query PipelineQuery type to transform to h

\* @tparam State The request domain model

\*/

case class QueryTransformerStep[

TRequest <: Request,

Query <: PipelineQuery,

State <: HasQuery[Query, State] with HasRequest[TRequest] with HasParams

]() extends Step[State, (TRequest, Params) => Query, (TRequest, Params), QueryTransformerResult[

Query

]] {

override def isEmpty(config: (TRequest, Params) => Query): Boolean = false

override def arrow(

config: (TRequest, Params) => Query,

context: Executor.Context

): Arrow[(TRequest, Params), QueryTransformerResult[Query]] = Arrow.map {

case (request: TRequest @unchecked, params: Params) =>

QueryTransformerResult(config(request, params))

}

override def updateState(

state: State,

executorResult: QueryTransformerResult[Query],

config: (TRequest, Params) => Query

): State = state.updateQuery(executorResult.query)

override def adaptInput(

state: State,

config: (TRequest, Params) => Query

): (TRequest, Params) = (state.request, state.params)

}

case class QueryTransformerResult[Query <: PipelineQuery](query: Query) extends ExecutorResult