package com.twitter.product\_mixer.core.service.filter\_executor

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.functional\_component.filter.Filter

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

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

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

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

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

import com.twitter.product\_mixer.core.service.filter\_executor.FilterExecutor.FilterState

import com.twitter.stitch.Arrow

import com.twitter.stitch.Arrow.Iso

import javax.inject.Inject

import javax.inject.Singleton

import scala.collection.immutable.Queue

/\*\*

\* Applies a `Seq[Filter]` in sequential order.

\* Returns the results and a detailed Seq of each filter's results (for debugging / coherence).

\*

\* Note that each successive filter is only passed the 'kept' Seq from the previous filter, not the full

\* set of candidates.

\*/

@Singleton

class FilterExecutor @Inject() (override val statsReceiver: StatsReceiver) extends Executor {

private val Kept = "kept"

private val Removed = "removed"

def arrow[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]](

filters: Seq[Filter[Query, Candidate]],

context: Executor.Context

): Arrow[(Query, Seq[CandidateWithFeatures[Candidate]]), FilterExecutorResult[Candidate]] = {

val filterArrows = filters.map(getIsoArrowForFilter(\_, context))

val combinedArrow = isoArrowsSequentially(filterArrows)

Arrow

.map[(Query, Seq[CandidateWithFeatures[Candidate]]), FilterState[Query, Candidate]] {

case (query, filterCandidates) =>

// transform the input to the initial state of a `FilterExecutorResult`

val initialFilterExecutorResult =

FilterExecutorResult(filterCandidates.map(\_.candidate), Queue.empty)

val allCandidates: Map[Candidate, CandidateWithFeatures[Candidate]] =

filterCandidates.map { fc =>

(fc.candidate, fc)

}.toMap

FilterState(query, allCandidates, initialFilterExecutorResult)

}

.flatMapArrow(combinedArrow)

.map {

case FilterState(\_, \_, filterExecutorResult) =>

filterExecutorResult.copy(individualFilterResults =

// materialize the Queue into a List for faster future iterations

filterExecutorResult.individualFilterResults.toList)

}

}

/\*\*

\* Adds filter specific stats, generic [[wrapComponentWithExecutorBookkeeping]] stats, and wraps with failure handling

\*

\* If the filter is a [[Conditionally]] ensures that we dont record stats if its turned off

\*

\* @note For performance, the [[FilterExecutorResult.individualFilterResults]] is build backwards - the head being the most recent result.

\* @param filter the filter to make an [[Arrow]] out of

\* @param context the [[Executor.Context]] for the pipeline this is a part of

\*/

private def getIsoArrowForFilter[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]](

filter: Filter[Query, Candidate],

context: Executor.Context

): Iso[FilterState[Query, Candidate]] = {

val broadcastStatsReceiver =

Executor.broadcastStatsReceiver(context, filter.identifier, statsReceiver)

val keptCounter = broadcastStatsReceiver.counter(Kept)

val removedCounter = broadcastStatsReceiver.counter(Removed)

val filterArrow = Arrow.flatMap[

(Query, Seq[CandidateWithFeatures[Candidate]]),

FilterExecutorIndividualResult[Candidate]

] {

case (query, candidates) =>

filter.apply(query, candidates).map { result =>

FilterExecutorIndividualResult(

identifier = filter.identifier,

kept = result.kept,

removed = result.removed)

}

}

val observedArrow: Arrow[

(Query, Seq[CandidateWithFeatures[Candidate]]),

FilterExecutorIndividualResult[

Candidate

]

] = wrapComponentWithExecutorBookkeeping(

context = context,

currentComponentIdentifier = filter.identifier,

onSuccess = { result: FilterExecutorIndividualResult[Candidate] =>

keptCounter.incr(result.kept.size)

removedCounter.incr(result.removed.size)

}

)(

filterArrow

)

val conditionallyRunArrow: Arrow[

(Query, Seq[CandidateWithFeatures[Candidate]]),

IndividualFilterResults[

Candidate

]

] =

filter match {

case filter: Filter[Query, Candidate] with Conditionally[

Filter.Input[Query, Candidate] @unchecked

] =>

Arrow.ifelse(

{

case (query, candidates) =>

filter.onlyIf(Filter.Input(query, candidates))

},

observedArrow,

Arrow.value(ConditionalFilterDisabled(filter.identifier))

)

case \_ => observedArrow

}

// return an `Iso` arrow for easier composition later

Arrow

.zipWithArg(

Arrow

.map[FilterState[Query, Candidate], (Query, Seq[CandidateWithFeatures[Candidate]])] {

case FilterState(query, candidateToFeaturesMap, FilterExecutorResult(result, \_)) =>

(query, result.flatMap(candidateToFeaturesMap.get))

}.andThen(conditionallyRunArrow))

.map {

case (

FilterState(query, allCandidates, filterExecutorResult),

filterResult: FilterExecutorIndividualResult[Candidate]) =>

val resultWithCurrentPrepended =

filterExecutorResult.individualFilterResults :+ filterResult

val newFilterExecutorResult = FilterExecutorResult(

result = filterResult.kept,

individualFilterResults = resultWithCurrentPrepended)

FilterState(query, allCandidates, newFilterExecutorResult)

case (filterState, filterDisabledResult: ConditionalFilterDisabled) =>

filterState.copy(

executorResult = filterState.executorResult.copy(

individualFilterResults =

filterState.executorResult.individualFilterResults :+ filterDisabledResult

))

}

}

}

object FilterExecutor {

/\*\*

\* FilterState is an internal representation of the state that is passed between each individual filter arrow.

\*

\* @param query: The query

\* @param candidateToFeaturesMap: A lookup mapping from Candidate -> FilterCandidate, to rebuild the inputs quickly for the next filter

\* @param executorResult: The in-progress executor result

\*/

private case class FilterState[Query <: PipelineQuery, Candidate <: UniversalNoun[Any]](

query: Query,

candidateToFeaturesMap: Map[Candidate, CandidateWithFeatures[Candidate]],

executorResult: FilterExecutorResult[Candidate])

}