package com.twitter.product\_mixer.component\_library.selector

import com.twitter.product\_mixer.core.functional\_component.common.CandidateScope

import com.twitter.product\_mixer.core.functional\_component.selector.Selector

import com.twitter.product\_mixer.core.functional\_component.selector.SelectorResult

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

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

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

import scala.reflect.ClassTag

sealed trait SubpoolIncludeTypes

trait IncludeInSubpool[-Query <: PipelineQuery] extends SubpoolIncludeTypes {

/\*\*

\* Given the `query`, current `remainingCandidate`, and the `result`,

\* returns whether the specific `remainingCandidate` should be passed into the

\* [[SelectFromSubpoolCandidates]]'s [[SelectFromSubpoolCandidates.selector]]

\*

\* @note the `result` contains the [[SelectorResult.result]] that was passed into this selector,

\* so each `remainingCandidate` will get the same `result` Seq.

\*/

def apply(

query: Query,

remainingCandidate: CandidateWithDetails,

result: Seq[CandidateWithDetails]

): Boolean

}

case class IncludeCandidateTypeInSubpool[CandidateType <: UniversalNoun[\_]](

)(

implicit tag: ClassTag[CandidateType])

extends IncludeInSubpool[PipelineQuery] {

override def apply(

query: PipelineQuery,

remainingCandidate: CandidateWithDetails,

result: Seq[CandidateWithDetails]

): Boolean = remainingCandidate.isCandidateType[CandidateType]()

}

trait IncludeSetInSubpool[-Query <: PipelineQuery] extends SubpoolIncludeTypes {

/\*\*

\* Given the `query`, all `remainingCandidates`` and `results`,

\* returns a Set of which candidates should be included in the subpool.

\*

\* @note the returned set is only used to determine subpool membership. Mutating the candidates

\* is invalid and won't work. The order of the candidates will be preserved from the current

\* order of the remaining candidates sequence.

\*/

def apply(

query: Query,

remainingCandidate: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): Set[CandidateWithDetails]

}

sealed trait SubpoolRemainingCandidatesHandler

/\*\*

\* Candidates remaining in the subpool after running the selector will be

\* prepended to the beginning of the [[SelectorResult.remainingCandidates]]

\*/

case object PrependToBeginningOfRemainingCandidates extends SubpoolRemainingCandidatesHandler

/\*\*

\* Candidates remaining in the subpool after running the selector will be

\* appended to the end of the [[SelectorResult.remainingCandidates]]

\*/

case object AppendToEndOfRemainingCandidates extends SubpoolRemainingCandidatesHandler

/\*\*

\* Creates a subpool of all `remainingCandidates` for which [[subpoolInclude]] resolves to true

\* (in the same order as the original `remainingCandidates`) and runs the [[selector]] with the

\* subpool passed in as the `remainingCandidates`.

\*

\* Most customers want to use a IncludeInSubpool that chooses if each candidate should be included

\* in the subpool.

\* Where necessary, IncludeSetInSubpool allows you to define them in bulk w/ a Set.

\*

\* @note any candidates in the subpool which are not added to the [[SelectorResult.result]]

\* will be treated according to the [[SubpoolRemainingCandidatesHandler]]

\*/

class SelectFromSubpoolCandidates[-Query <: PipelineQuery] private[selector] (

val selector: Selector[Query],

subpoolInclude: SubpoolIncludeTypes,

subpoolRemainingCandidatesHandler: SubpoolRemainingCandidatesHandler =

AppendToEndOfRemainingCandidates)

extends Selector[Query] {

override val pipelineScope: CandidateScope = selector.pipelineScope

override def apply(

query: Query,

remainingCandidates: Seq[CandidateWithDetails],

result: Seq[CandidateWithDetails]

): SelectorResult = {

val (selectedCandidates, otherCandidates) = subpoolInclude match {

case includeInSubpool: IncludeInSubpool[Query] =>

remainingCandidates.partition(candidate =>

pipelineScope.contains(candidate) && includeInSubpool(query, candidate, result))

case includeSetInSubpool: IncludeSetInSubpool[Query] =>

val includeSet =

includeSetInSubpool(query, remainingCandidates.filter(pipelineScope.contains), result)

remainingCandidates.partition(candidate => includeSet.contains(candidate))

}

val underlyingSelectorResult = selector.apply(query, selectedCandidates, result)

val remainingCandidatesWithSubpoolRemainingCandidates =

subpoolRemainingCandidatesHandler match {

case AppendToEndOfRemainingCandidates =>

otherCandidates ++ underlyingSelectorResult.remainingCandidates

case PrependToBeginningOfRemainingCandidates =>

underlyingSelectorResult.remainingCandidates ++ otherCandidates

}

underlyingSelectorResult.copy(remainingCandidates =

remainingCandidatesWithSubpoolRemainingCandidates)

}

override def toString: String = s"SelectFromSubpoolCandidates(${selector.toString}))"

}

object SelectFromSubpoolCandidates {

def apply[Query <: PipelineQuery](

selector: Selector[Query],

includeInSubpool: IncludeInSubpool[Query],

subpoolRemainingCandidatesHandler: SubpoolRemainingCandidatesHandler =

AppendToEndOfRemainingCandidates

) = new SelectFromSubpoolCandidates[Query](

selector,

includeInSubpool,

subpoolRemainingCandidatesHandler

)

def includeSet[Query <: PipelineQuery](

selector: Selector[Query],

includeSetInSubpool: IncludeSetInSubpool[Query],

subpoolRemainingCandidatesHandler: SubpoolRemainingCandidatesHandler =

AppendToEndOfRemainingCandidates

) = new SelectFromSubpoolCandidates[Query](

selector,

includeSetInSubpool,

subpoolRemainingCandidatesHandler

)

}