package com.twitter.follow\_recommendations.common.base

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.product\_mixer.core.functional\_component.candidate\_source.CandidateSource

import com.twitter.stitch.Stitch

import com.twitter.util.Duration

import com.twitter.util.TimeoutException

import scala.language.implicitConversions

class EnrichedCandidateSource[Target, Candidate](original: CandidateSource[Target, Candidate]) {

/\*\*

\* Gate the candidate source based on the Predicate of target.

\* It returns results only if the predicate returns Valid.

\*

\* @param predicate

\* @return

\*/

def gate(predicate: Predicate[Target]): CandidateSource[Target, Candidate] = {

throw new UnsupportedOperationException()

}

def observe(statsReceiver: StatsReceiver): CandidateSource[Target, Candidate] = {

val originalIdentifier = original.identifier

val stats = statsReceiver.scope(originalIdentifier.name)

new CandidateSource[Target, Candidate] {

val identifier = originalIdentifier

override def apply(target: Target): Stitch[Seq[Candidate]] = {

StatsUtil.profileStitchSeqResults[Candidate](original(target), stats)

}

}

}

/\*\*

\* Map target type into new target type (1 to optional mapping)

\*/

def stitchMapKey[Target2](

targetMapper: Target2 => Stitch[Option[Target]]

): CandidateSource[Target2, Candidate] = {

val targetsMapper: Target2 => Stitch[Seq[Target]] = { target =>

targetMapper(target).map(\_.toSeq)

}

stitchMapKeys(targetsMapper)

}

/\*\*

\* Map target type into new target type (1 to many mapping)

\*/

def stitchMapKeys[Target2](

targetMapper: Target2 => Stitch[Seq[Target]]

): CandidateSource[Target2, Candidate] = {

new CandidateSource[Target2, Candidate] {

val identifier = original.identifier

override def apply(target: Target2): Stitch[Seq[Candidate]] = {

for {

mappedTargets <- targetMapper(target)

results <- Stitch.traverse(mappedTargets)(original(\_))

} yield results.flatten

}

}

}

/\*\*

\* Map target type into new target type (1 to many mapping)

\*/

def mapKeys[Target2](

targetMapper: Target2 => Seq[Target]

): CandidateSource[Target2, Candidate] = {

val stitchMapper: Target2 => Stitch[Seq[Target]] = { target =>

Stitch.value(targetMapper(target))

}

stitchMapKeys(stitchMapper)

}

/\*\*

\* Map candidate types to new type based on candidateMapper

\*/

def mapValues[Candidate2](

candidateMapper: Candidate => Stitch[Option[Candidate2]]

): CandidateSource[Target, Candidate2] = {

new CandidateSource[Target, Candidate2] {

val identifier = original.identifier

override def apply(target: Target): Stitch[Seq[Candidate2]] = {

original(target).flatMap { candidates =>

val results = Stitch.traverse(candidates)(candidateMapper(\_))

results.map(\_.flatten)

}

}

}

}

/\*\*

\* Map candidate types to new type based on candidateMapper

\*/

def mapValue[Candidate2](

candidateMapper: Candidate => Candidate2

): CandidateSource[Target, Candidate2] = {

val stitchMapper: Candidate => Stitch[Option[Candidate2]] = { c =>

Stitch.value(Some(candidateMapper(c)))

}

mapValues(stitchMapper)

}

/\*\*

\* This method wraps the candidate source in a designated timeout so that a single candidate

\* source does not result in a timeout for the entire flow

\*/

def within(

candidateTimeout: Duration,

statsReceiver: StatsReceiver

): CandidateSource[Target, Candidate] = {

val originalIdentifier = original.identifier

val timeoutCounter =

statsReceiver.counter(originalIdentifier.name, "timeout")

new CandidateSource[Target, Candidate] {

val identifier = originalIdentifier

override def apply(target: Target): Stitch[Seq[Candidate]] = {

original

.apply(target)

.within(candidateTimeout)(com.twitter.finagle.util.DefaultTimer)

.rescue {

case \_: TimeoutException =>

timeoutCounter.incr()

Stitch.Nil

}

}

}

}

def failOpenWithin(

candidateTimeout: Duration,

statsReceiver: StatsReceiver

): CandidateSource[Target, Candidate] = {

val originalIdentifier = original.identifier

val timeoutCounter =

statsReceiver.counter(originalIdentifier.name, "timeout")

new CandidateSource[Target, Candidate] {

val identifier = originalIdentifier

override def apply(target: Target): Stitch[Seq[Candidate]] = {

original

.apply(target)

.within(candidateTimeout)(com.twitter.finagle.util.DefaultTimer)

.handle {

case \_: TimeoutException =>

timeoutCounter.incr()

Seq.empty

case e: Exception =>

statsReceiver

.scope("candidate\_source\_error").scope(originalIdentifier.name).counter(

e.getClass.getSimpleName).incr

Seq.empty

}

}

}

}

}

object EnrichedCandidateSource {

implicit def toEnriched[K, V](original: CandidateSource[K, V]): EnrichedCandidateSource[K, V] =

new EnrichedCandidateSource(original)

}