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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.stitch.Stitch

import com.twitter.util.Duration

import com.twitter.util.TimeoutException

/\*\*

\* Ranker is a special kind of transform that would only change the order of a list of items.

\* If a single item is given, it "may" attach additional scoring information to the item.

\*

\* @tparam Target target to recommend the candidates

\* @tparam Candidate candidate type to rank

\*/

trait Ranker[Target, Candidate] extends Transform[Target, Candidate] { ranker =>

def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]]

override def transform(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] = {

rank(target, candidates)

}

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

val originalRanker = this

new Ranker[Target, Candidate] {

override def rank(target: Target, items: Seq[Candidate]): Stitch[Seq[Candidate]] = {

statsReceiver.counter(Transform.InputCandidatesCount).incr(items.size)

statsReceiver.stat(Transform.InputCandidatesStat).add(items.size)

StatsUtil.profileStitchSeqResults(originalRanker.rank(target, items), statsReceiver)

}

}

}

def reverse: Ranker[Target, Candidate] = new Ranker[Target, Candidate] {

def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] =

ranker.rank(target, candidates).map(\_.reverse)

}

def andThen(other: Ranker[Target, Candidate]): Ranker[Target, Candidate] = {

val original = this

new Ranker[Target, Candidate] {

def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] = {

original.rank(target, candidates).flatMap { results => other.rank(target, results) }

}

}

}

/\*\*

\* This method wraps the Ranker in a designated timeout.

\* If the ranker timeouts, it would return the original candidates directly,

\* instead of failing the whole recommendation flow

\*/

def within(timeout: Duration, statsReceiver: StatsReceiver): Ranker[Target, Candidate] = {

val timeoutCounter = statsReceiver.counter("timeout")

val original = this

new Ranker[Target, Candidate] {

override def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] = {

original

.rank(target, candidates)

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

.rescue {

case \_: TimeoutException =>

timeoutCounter.incr()

Stitch.value(candidates)

}

}

}

}

}

object Ranker {

def chain[Target, Candidate](

transformer: Transform[Target, Candidate],

ranker: Ranker[Target, Candidate]

): Ranker[Target, Candidate] = {

new Ranker[Target, Candidate] {

def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] = {

transformer

.transform(target, candidates)

.flatMap { results => ranker.rank(target, results) }

}

}

}

}

class IdentityRanker[Target, Candidate] extends Ranker[Target, Candidate] {

def rank(target: Target, candidates: Seq[Candidate]): Stitch[Seq[Candidate]] =

Stitch.value(candidates)

}