package com.twitter.servo.repository

import com.twitter.finagle.mux.ClientDiscardedRequestException

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.{CancelledConnectionException, CancelledRequestException}

import com.twitter.servo.util.{Gate, SuccessRateTracker}

import com.twitter.util.Throwables.RootCause

import java.util.concurrent.CancellationException

object SuccessRateTrackingRepository {

/\*\*

\* (successes, failures)

\*/

type SuccessRateObserver = (Int, Int) => Unit

/\*\*

\* Identifies [[Throwable]]s that should not be counted as failures.

\*

\* This is a total function instead of a partial function so it can reliably recurse on itself

\* to find a root cause.

\*/

def isCancellation(t: Throwable): Boolean =

t match {

// We don't consider CancelledRequestExceptions or CancelledConnectionExceptions to be

// failures in order not to tarnish our success rate on upstream request cancellations.

case \_: CancelledRequestException => true

case \_: CancelledConnectionException => true

// non-finagle backends can throw CancellationExceptions when their futures are cancelled.

case \_: CancellationException => true

// Mux servers can return ClientDiscardedRequestException.

case \_: ClientDiscardedRequestException => true

// Most of these exceptions can be wrapped in com.twitter.finagle.Failure

case RootCause(t) => isCancellation(t)

case \_ => false

}

/\*\*

\* Return a Success Rate (SR) tracking repository along with the gate controlling it.

\*

\* @param stats Provides availability gauge

\* @param availabilityFromSuccessRate function to calculate availability given SR

\* @param tracker strategy for tracking (usually recent) SR

\* @param shouldIgnore don't count certain exceptions as failures, e.g. cancellations

\* @return tuple of (SR tracking repo, gate closing if SR drops too far)

\*/

def withGate[Q <: Seq[K], K, V](

stats: StatsReceiver,

availabilityFromSuccessRate: Double => Double,

tracker: SuccessRateTracker,

shouldIgnore: Throwable => Boolean = isCancellation

): (KeyValueRepository[Q, K, V] => KeyValueRepository[Q, K, V], Gate[Unit]) = {

val successRateGate = tracker.observedAvailabilityGate(availabilityFromSuccessRate, stats)

(new SuccessRateTrackingRepository[Q, K, V](\_, tracker.record, shouldIgnore), successRateGate)

}

}

/\*\*

\* A KeyValueRepository that provides feedback on query success rate to

\* a SuccessRateObserver. Both found and not found are considered successful

\* responses, while failures are not. Cancellations are ignored by default.

\*/

class SuccessRateTrackingRepository[Q <: Seq[K], K, V](

underlying: KeyValueRepository[Q, K, V],

observer: SuccessRateTrackingRepository.SuccessRateObserver,

shouldIgnore: Throwable => Boolean = SuccessRateTrackingRepository.isCancellation)

extends KeyValueRepository[Q, K, V] {

def apply(query: Q) =

underlying(query) onSuccess { kvr =>

val nonIgnoredFailures = kvr.failed.values.foldLeft(0) {

case (count, t) if shouldIgnore(t) => count

case (count, \_) => count + 1

}

observer(kvr.found.size + kvr.notFound.size, nonIgnoredFailures)

} onFailure { t =>

if (!shouldIgnore(t)) {

observer(0, query.size)

}

}

}