package com.twitter.servo.repository

import com.twitter.finagle.stats.{StatsReceiver, Stat}

import com.twitter.servo.util.{ExceptionCounter, LogarithmicallyBucketedTimer}

import com.twitter.util.{Future, Return, Throw, Try}

class RepositoryObserver(

statsReceiver: StatsReceiver,

bucketBySize: Boolean,

exceptionCounter: ExceptionCounter) {

protected[this] lazy val timer = new LogarithmicallyBucketedTimer(statsReceiver)

protected[this] val sizeStat = statsReceiver.stat("size")

protected[this] val foundStat = statsReceiver.counter("found")

protected[this] val notFoundStat = statsReceiver.counter("not\_found")

protected[this] val total = statsReceiver.counter("total")

private[this] val timeStat = statsReceiver.stat(LogarithmicallyBucketedTimer.LatencyStatName)

def this(statsReceiver: StatsReceiver, bucketBySize: Boolean = true) =

this(statsReceiver, bucketBySize, new ExceptionCounter(statsReceiver))

def time[T](size: Int = 1)(f: => Future[T]) = {

sizeStat.add(size)

if (bucketBySize)

timer(size)(f)

else

Stat.timeFuture(timeStat)(f)

}

private[this] def total(size: Int = 1): Unit = total.incr(size)

def found(size: Int = 1): Unit = {

foundStat.incr(size)

total(size)

}

def notFound(size: Int = 1): Unit = {

notFoundStat.incr(size)

total(size)

}

def exception(ts: Throwable\*): Unit = {

exceptionCounter(ts)

total(ts.size)

}

def exceptions(ts: Seq[Throwable]): Unit = {

exception(ts: \_\*)

}

def observeTry[V](tryObj: Try[V]): Unit = {

tryObj.respond {

case Return(\_) => found()

case Throw(t) => exception(t)

}

}

def observeOption[V](optionTry: Try[Option[V]]): Unit = {

optionTry.respond {

case Return(Some(\_)) => found()

case Return(None) => notFound()

case Throw(t) => exception(t)

}

}

def observeKeyValueResult[K, V](resultTry: Try[KeyValueResult[K, V]]): Unit = {

resultTry.respond {

case Return(result) =>

found(result.found.size)

notFound(result.notFound.size)

exceptions(result.failed.values.toSeq)

case Throw(t) =>

exception(t)

}

}

/\*\*

\* observeSeq observes the result of a fetch against a key-value repository

\* when the returned value is a Seq of type V. When the fetch is completed,

\* observes whether or not the returned Seq is empty, contains some number of

\* items, or has failed in some way.

\*/

def observeSeq[V](seqTry: Try[Seq[V]]): Unit = {

seqTry.respond {

case Return(seq) if seq.isEmpty => notFound()

case Return(seq) => found(seq.length)

case Throw(t) => exception(t)

}

}

}