package com.twitter.servo.util

import com.twitter.finagle.{Backoff, Service, TimeoutException, WriteException}

import com.twitter.finagle.service.{RetryExceptionsFilter, RetryPolicy}

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.util.{Duration, Future, Throw, Timer, Try}

/\*\*

\* Allows an action to be retried according to a backoff strategy.

\* This is an adaption of the Finagle RetryExceptionsFilter, but with an

\* arbitrary asynchronous computation.

\*/

class Retry(

statsReceiver: StatsReceiver,

backoffs: Backoff,

private[this] val timer: Timer = DefaultTimer) {

/\*\*

\* retry on specific exceptions

\*/

def apply[T](

f: () => Future[T]

)(

shouldRetry: PartialFunction[Throwable, Boolean]

): Future[T] = {

val policy = RetryPolicy.backoff[Try[Nothing]](backoffs) {

case Throw(t) if shouldRetry.isDefinedAt(t) => shouldRetry(t)

}

val service = new Service[Unit, T] {

override def apply(u: Unit): Future[T] = f()

}

val retrying = new RetryExceptionsFilter(policy, timer, statsReceiver) andThen service

retrying()

}

@deprecated("release() has no function and will be removed", "2.8.2")

def release(): Unit = {}

}

/\*\*

\* Use to configure separate backoffs for WriteExceptions, TimeoutExceptions,

\* and service-specific exceptions

\*/

class ServiceRetryPolicy(

writeExceptionBackoffs: Backoff,

timeoutBackoffs: Backoff,

serviceBackoffs: Backoff,

shouldRetryService: PartialFunction[Throwable, Boolean])

extends RetryPolicy[Try[Nothing]] {

override def apply(r: Try[Nothing]) = r match {

case Throw(t) if shouldRetryService.isDefinedAt(t) =>

if (shouldRetryService(t))

onServiceException

else

None

case Throw(\_: WriteException) => onWriteException

case Throw(\_: TimeoutException) => onTimeoutException

case \_ => None

}

def copy(

writeExceptionBackoffs: Backoff = writeExceptionBackoffs,

timeoutBackoffs: Backoff = timeoutBackoffs,

serviceBackoffs: Backoff = serviceBackoffs,

shouldRetryService: PartialFunction[Throwable, Boolean] = shouldRetryService

) =

new ServiceRetryPolicy(

writeExceptionBackoffs,

timeoutBackoffs,

serviceBackoffs,

shouldRetryService

)

private[this] def onWriteException = consume(writeExceptionBackoffs) { tail =>

copy(writeExceptionBackoffs = tail)

}

private[this] def onTimeoutException = consume(timeoutBackoffs) { tail =>

copy(timeoutBackoffs = tail)

}

private[this] def onServiceException = consume(serviceBackoffs) { tail =>

copy(serviceBackoffs = tail)

}

private[this] def consume(b: Backoff)(f: Backoff => ServiceRetryPolicy) = {

if (b.isExhausted) None

else Some((b.duration, f(b.next)))

}

override val toString = "ServiceRetryPolicy(%s, %s, %s)".format(

writeExceptionBackoffs,

timeoutBackoffs,

serviceBackoffs

)

}