package com.twitter.tweetypie

package backends

import com.twitter.finagle.Backoff

import com.twitter.finagle.service.RetryPolicy

import com.twitter.service.talon.thriftscala.ExpandRequest

import com.twitter.service.talon.thriftscala.ExpandResponse

import com.twitter.service.talon.thriftscala.ResponseCode

import com.twitter.service.talon.thriftscala.ShortenRequest

import com.twitter.service.talon.thriftscala.ShortenResponse

import com.twitter.service.talon.{thriftscala => talon}

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.core.OverCapacity

import com.twitter.tweetypie.util.RetryPolicyBuilder

object Talon {

import Backend.\_

type Expand = FutureArrow[talon.ExpandRequest, talon.ExpandResponse]

type Shorten = FutureArrow[talon.ShortenRequest, talon.ShortenResponse]

case object TransientError extends Exception()

case object PermanentError extends Exception()

def fromClient(client: talon.Talon.MethodPerEndpoint): Talon =

new Talon {

val shorten = FutureArrow(client.shorten \_)

val expand = FutureArrow(client.expand \_)

def ping(): Future[Unit] = client.serviceInfo().unit

}

case class Config(

shortenTimeout: Duration,

expandTimeout: Duration,

timeoutBackoffs: Stream[Duration],

transientErrorBackoffs: Stream[Duration]) {

def apply(svc: Talon, ctx: Backend.Context): Talon =

new Talon {

val shorten: FutureArrow[ShortenRequest, ShortenResponse] =

policy("shorten", shortenTimeout, shortenResponseCode, ctx)(svc.shorten)

val expand: FutureArrow[ExpandRequest, ExpandResponse] =

policy("expand", expandTimeout, expandResponseCode, ctx)(svc.expand)

def ping(): Future[Unit] = svc.ping()

}

private[this] def policy[A, B](

name: String,

requestTimeout: Duration,

getResponseCode: B => talon.ResponseCode,

ctx: Context

): Builder[A, B] =

handleResponseCodes(name, getResponseCode, ctx) andThen

defaultPolicy(name, requestTimeout, retryPolicy, ctx)

private[this] def retryPolicy[B]: RetryPolicy[Try[B]] =

RetryPolicy.combine[Try[B]](

RetryPolicyBuilder.timeouts[B](timeoutBackoffs),

RetryPolicy.backoff(Backoff.fromStream(transientErrorBackoffs)) {

case Throw(TransientError) => true

}

)

private[this] def handleResponseCodes[A, B](

name: String,

extract: B => talon.ResponseCode,

ctx: Context

): Builder[A, B] = {

val scopedStats = ctx.stats.scope(name)

val responseCodeStats = scopedStats.scope("response\_code")

\_ andThen FutureArrow[B, B] { res =>

val responseCode = extract(res)

responseCodeStats.counter(responseCode.toString).incr()

responseCode match {

case talon.ResponseCode.TransientError => Future.exception(TransientError)

case talon.ResponseCode.PermanentError => Future.exception(PermanentError)

case talon.ResponseCode.ServerOverloaded => Future.exception(OverCapacity("talon"))

case \_ => Future.value(res)

}

}

}

}

def shortenResponseCode(res: talon.ShortenResponse): ResponseCode = res.responseCode

def expandResponseCode(res: talon.ExpandResponse): ResponseCode = res.responseCode

implicit val warmup: Warmup[Talon] = Warmup[Talon]("talon")(\_.ping())

}

trait Talon {

import Talon.\_

val shorten: Shorten

val expand: Expand

def ping(): Future[Unit]

}