package com.twitter.tweetypie

package backends

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.Backoff

import com.twitter.finagle.service.RetryPolicy

import com.twitter.service.gen.scarecrow.thriftscala.CheckTweetResponse

import com.twitter.service.gen.scarecrow.thriftscala.Retweet

import com.twitter.service.gen.scarecrow.thriftscala.TieredAction

import com.twitter.service.gen.scarecrow.thriftscala.TweetContext

import com.twitter.service.gen.scarecrow.thriftscala.TweetNew

import com.twitter.service.gen.scarecrow.{thriftscala => scarecrow}

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.util.RetryPolicyBuilder

object Scarecrow {

import Backend.\_

type CheckTweet2 =

FutureArrow[(scarecrow.TweetNew, scarecrow.TweetContext), scarecrow.CheckTweetResponse]

type CheckRetweet = FutureArrow[scarecrow.Retweet, scarecrow.TieredAction]

def fromClient(client: scarecrow.ScarecrowService.MethodPerEndpoint): Scarecrow =

new Scarecrow {

val checkTweet2 = FutureArrow((client.checkTweet2 \_).tupled)

val checkRetweet = FutureArrow(client.checkRetweet \_)

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

}

case class Config(

readTimeout: Duration,

writeTimeout: Duration,

timeoutBackoffs: Stream[Duration],

scarecrowExceptionBackoffs: Stream[Duration]) {

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

new Scarecrow {

val checkTweet2: FutureArrow[(TweetNew, TweetContext), CheckTweetResponse] =

writePolicy("checkTweet2", ctx)(svc.checkTweet2)

val checkRetweet: FutureArrow[Retweet, TieredAction] =

writePolicy("checkRetweet", ctx)(svc.checkRetweet)

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

}

private[this] def readPolicy[A, B](name: String, ctx: Context): Builder[A, B] =

defaultPolicy(name, readTimeout, readRetryPolicy, ctx)

private[this] def writePolicy[A, B](name: String, ctx: Context): Builder[A, B] =

defaultPolicy(name, writeTimeout, nullRetryPolicy, ctx)

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

RetryPolicy.combine[Try[B]](

RetryPolicyBuilder.timeouts[B](timeoutBackoffs),

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

case Throw(ex: scarecrow.InternalServerError) => true

}

)

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

// retry policy that runs once, and will not retry on any exception

RetryPolicy.backoff(Backoff.fromStream(Stream(0.milliseconds))) {

case Throw(\_) => false

}

}

implicit val warmup: Warmup[Scarecrow] = Warmup[Scarecrow]("scarecrow")(\_.ping())

}

trait Scarecrow {

import Scarecrow.\_

val checkTweet2: CheckTweet2

val checkRetweet: CheckRetweet

def ping(): Future[Unit]

}