package com.twitter.tweetypie

package backends

import com.twitter.finagle.Backoff

import com.twitter.finagle.service.RetryPolicy

import com.twitter.gizmoduck.thriftscala.CountsUpdateField

import com.twitter.gizmoduck.thriftscala.LookupContext

import com.twitter.gizmoduck.thriftscala.ModifiedUser

import com.twitter.gizmoduck.thriftscala.UserResult

import com.twitter.gizmoduck.{thriftscala => gd}

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.core.OverCapacity

import com.twitter.tweetypie.util.RetryPolicyBuilder

object Gizmoduck {

import Backend.\_

type GetById = FutureArrow[(gd.LookupContext, Seq[UserId], Set[UserField]), Seq[gd.UserResult]]

type GetByScreenName =

FutureArrow[(gd.LookupContext, Seq[String], Set[UserField]), Seq[gd.UserResult]]

type IncrCount = FutureArrow[(UserId, gd.CountsUpdateField, Int), Unit]

type ModifyAndGet = FutureArrow[(gd.LookupContext, UserId, gd.ModifiedUser), gd.User]

def fromClient(client: gd.UserService.MethodPerEndpoint): Gizmoduck =

new Gizmoduck {

val getById = FutureArrow((client.get \_).tupled)

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

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

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

def ping(): Future[Unit] = client.get(gd.LookupContext(), Seq.empty, Set.empty).unit

}

case class Config(

readTimeout: Duration,

writeTimeout: Duration,

modifyAndGetTimeout: Duration,

modifyAndGetTimeoutBackoffs: Stream[Duration],

defaultTimeoutBackoffs: Stream[Duration],

gizmoduckExceptionBackoffs: Stream[Duration]) {

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

new Gizmoduck {

val getById: FutureArrow[(LookupContext, Seq[UserId], Set[UserField]), Seq[UserResult]] =

policy("getById", readTimeout, ctx)(svc.getById)

val getByScreenName: FutureArrow[(LookupContext, Seq[String], Set[UserField]), Seq[

UserResult

]] = policy("getByScreenName", readTimeout, ctx)(svc.getByScreenName)

val incrCount: FutureArrow[(UserId, CountsUpdateField, Int), Unit] =

policy("incrCount", writeTimeout, ctx)(svc.incrCount)

val modifyAndGet: FutureArrow[(LookupContext, UserId, ModifiedUser), User] = policy(

"modifyAndGet",

modifyAndGetTimeout,

ctx,

timeoutBackoffs = modifyAndGetTimeoutBackoffs

)(svc.modifyAndGet)

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

}

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

name: String,

requestTimeout: Duration,

ctx: Context,

timeoutBackoffs: Stream[Duration] = defaultTimeoutBackoffs

): Builder[A, B] =

translateExceptions andThen

defaultPolicy(name, requestTimeout, retryPolicy(timeoutBackoffs), ctx)

private[this] def translateExceptions[A, B]: Builder[A, B] =

\_.translateExceptions {

case gd.OverCapacity(msg) => OverCapacity(s"gizmoduck: $msg")

}

private[this] def retryPolicy[B](timeoutBackoffs: Stream[Duration]): RetryPolicy[Try[B]] =

RetryPolicy.combine[Try[B]](

RetryPolicyBuilder.timeouts[B](timeoutBackoffs),

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

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

}

)

}

implicit val warmup: Warmup[Gizmoduck] =

Warmup[Gizmoduck]("gizmoduck")(\_.ping())

}

trait Gizmoduck {

import Gizmoduck.\_

val getById: GetById

val getByScreenName: GetByScreenName

val incrCount: IncrCount

val modifyAndGet: ModifyAndGet

def ping(): Future[Unit]

}