package com.twitter.tweetypie

package backends

import com.twitter.finagle.service.RetryPolicy

import com.twitter.limiter.thriftscala.FeatureRequest

import com.twitter.limiter.thriftscala.Usage

import com.twitter.limiter.{thriftscala => ls}

import com.twitter.servo.util.FutureArrow

import com.twitter.tweetypie.util.RetryPolicyBuilder

object LimiterBackend {

import Backend.\_

type IncrementFeature = FutureArrow[(ls.FeatureRequest, Int), Unit]

type GetFeatureUsage = FutureArrow[ls.FeatureRequest, ls.Usage]

def fromClient(client: ls.LimitService.MethodPerEndpoint): LimiterBackend =

new LimiterBackend {

val incrementFeature: IncrementFeature =

FutureArrow {

case (featureReq, amount) => client.incrementFeature(featureReq, amount).unit

}

val getFeatureUsage: GetFeatureUsage =

FutureArrow(featureReq => client.getLimitUsage(None, Some(featureReq)))

}

case class Config(requestTimeout: Duration, timeoutBackoffs: Stream[Duration]) {

def apply(client: LimiterBackend, ctx: Backend.Context): LimiterBackend =

new LimiterBackend {

val incrementFeature: FutureArrow[(FeatureRequest, Int), Unit] =

policy("incrementFeature", requestTimeout, ctx)(client.incrementFeature)

val getFeatureUsage: FutureArrow[FeatureRequest, Usage] =

policy("getFeatureUsage", requestTimeout, ctx)(client.getFeatureUsage)

}

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

name: String,

requestTimeout: Duration,

ctx: Context

): Builder[A, B] =

defaultPolicy(name, requestTimeout, retryPolicy, ctx)

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

RetryPolicyBuilder.timeouts[Any](timeoutBackoffs)

}

}

trait LimiterBackend {

import LimiterBackend.\_

val incrementFeature: IncrementFeature

val getFeatureUsage: GetFeatureUsage

}