package com.twitter.product\_mixer.shared\_library.memcached\_client

import com.twitter.finagle.memcached.Client

import com.twitter.finagle.memcached.protocol.Command

import com.twitter.finagle.memcached.protocol.Response

import com.twitter.finagle.mtls.client.MtlsStackClient.\_

import com.twitter.finagle.service.RetryExceptionsFilter

import com.twitter.finagle.service.RetryPolicy

import com.twitter.finagle.service.TimeoutFilter

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.finagle.GlobalRequestTimeoutException

import com.twitter.finagle.Memcached

import com.twitter.finagle.liveness.FailureAccrualFactory

import com.twitter.finagle.liveness.FailureAccrualPolicy

import com.twitter.finagle.mtls.authentication.ServiceIdentifier

import com.twitter.hashing.KeyHasher

import com.twitter.util.Duration

object MemcachedClientBuilder {

/\*\*

\* Build a Finagle Memcached [[Client]].

\*

\* @param destName Destination as a Wily path e.g. "/s/sample/sample".

\* @param numTries Maximum number of times to try.

\* @param requestTimeout Thrift client timeout per request. The Finagle default

\* is unbounded which is almost never optimal.

\* @param globalTimeout Thrift client total timeout. The Finagle default is

\* unbounded which is almost never optimal.

\* @param connectTimeout Thrift client transport connect timeout. The Finagle

\* default of one second is reasonable but we lower this

\* to match acquisitionTimeout for consistency.

\* @param acquisitionTimeout Thrift client session acquisition timeout. The Finagle

\* default is unbounded which is almost never optimal.

\* @param serviceIdentifier Service ID used to S2S Auth.

\* @param statsReceiver Stats.

\* @param failureAccrualPolicy Policy to determine when to mark a cache server as dead.

\* Memcached client will use default failure accrual policy

\* if it is not set.

\* @param keyHasher Hash algorithm that hashes a key into a 32-bit or 64-bit

\* number. Memcached client will use default hash algorithm

\* if it is not set.

\*

\* @see [[https://confluence.twitter.biz/display/CACHE/Finagle-memcached+User+Guide user guide]]

\* @return Finagle Memcached [[Client]]

\*/

def buildMemcachedClient(

destName: String,

numTries: Int,

requestTimeout: Duration,

globalTimeout: Duration,

connectTimeout: Duration,

acquisitionTimeout: Duration,

serviceIdentifier: ServiceIdentifier,

statsReceiver: StatsReceiver,

failureAccrualPolicy: Option[FailureAccrualPolicy] = None,

keyHasher: Option[KeyHasher] = None

): Client = {

buildRawMemcachedClient(

numTries,

requestTimeout,

globalTimeout,

connectTimeout,

acquisitionTimeout,

serviceIdentifier,

statsReceiver,

failureAccrualPolicy,

keyHasher

).newRichClient(destName)

}

def buildRawMemcachedClient(

numTries: Int,

requestTimeout: Duration,

globalTimeout: Duration,

connectTimeout: Duration,

acquisitionTimeout: Duration,

serviceIdentifier: ServiceIdentifier,

statsReceiver: StatsReceiver,

failureAccrualPolicy: Option[FailureAccrualPolicy] = None,

keyHasher: Option[KeyHasher] = None

): Memcached.Client = {

val globalTimeoutFilter = new TimeoutFilter[Command, Response](

timeout = globalTimeout,

exception = new GlobalRequestTimeoutException(globalTimeout),

timer = DefaultTimer)

val retryFilter = new RetryExceptionsFilter[Command, Response](

RetryPolicy.tries(numTries),

DefaultTimer,

statsReceiver)

val client = Memcached.client.withTransport

.connectTimeout(connectTimeout)

.withMutualTls(serviceIdentifier)

.withSession

.acquisitionTimeout(acquisitionTimeout)

.withRequestTimeout(requestTimeout)

.withStatsReceiver(statsReceiver)

.filtered(globalTimeoutFilter.andThen(retryFilter))

(keyHasher, failureAccrualPolicy) match {

case (Some(hasher), Some(policy)) =>

client

.withKeyHasher(hasher)

.configured(FailureAccrualFactory.Param(() => policy))

case (Some(hasher), None) =>

client

.withKeyHasher(hasher)

case (None, Some(policy)) =>

client

.configured(FailureAccrualFactory.Param(() => policy))

case \_ =>

client

}

}

}