package com.twitter.follow\_recommendations.common.clients.cache

import com.twitter.bijection.Bijection

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.Memcached.Client

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.io.Buf

import com.twitter.stitch.Stitch

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.Time

import java.security.MessageDigest

object MemcacheClient {

def apply[V](

client: Client,

dest: String,

valueBijection: Bijection[Buf, V],

ttl: Duration,

statsReceiver: StatsReceiver

): MemcacheClient[V] = {

new MemcacheClient(client, dest, valueBijection, ttl, statsReceiver)

}

}

class MemcacheClient[V](

client: Client,

dest: String,

valueBijection: Bijection[Buf, V],

ttl: Duration,

statsReceiver: StatsReceiver) {

val cache = client.newRichClient(dest).adapt[V](valueBijection)

val cacheTtl = Time.fromSeconds(ttl.inSeconds)

/\*\*

\* If cache contains key, return value from cache. Otherwise, run the underlying call

\* to fetch the value, store it in cache, and then return the value.

\*/

def readThrough(

key: String,

underlyingCall: () => Stitch[V]

): Stitch[V] = {

val cachedResult: Stitch[Option[V]] = Stitch

.callFuture(getIfPresent(key))

.within(70.millisecond)(DefaultTimer)

.rescue {

case e: Exception =>

statsReceiver.scope("rescued").counter(e.getClass.getSimpleName).incr()

Stitch(None)

}

val resultStitch = cachedResult.map { resultOption =>

resultOption match {

case Some(cacheValue) => Stitch.value(cacheValue)

case None =>

val underlyingCallStitch = profileStitch(

underlyingCall(),

statsReceiver.scope("underlyingCall")

)

underlyingCallStitch.map { result =>

put(key, result)

result

}

}

}.flatten

// profile the overall Stitch, and return the result

profileStitch(resultStitch, statsReceiver.scope("readThrough"))

}

def getIfPresent(key: String): Future[Option[V]] = {

cache

.get(hashString(key))

.onSuccess {

case Some(value) => statsReceiver.counter("cache\_hits").incr()

case None => statsReceiver.counter("cache\_misses").incr()

}

.onFailure {

case e: Exception =>

statsReceiver.counter("cache\_misses").incr()

statsReceiver.scope("rescued").counter(e.getClass.getSimpleName).incr()

}

.rescue {

case \_ => Future.None

}

}

def put(key: String, value: V): Future[Unit] = {

cache.set(hashString(key), 0, cacheTtl, value)

}

/\*\*

\* Hash the input key string to a fixed length format using SHA-256 hash function.

\*/

def hashString(input: String): String = {

val bytes = MessageDigest.getInstance("SHA-256").digest(input.getBytes("UTF-8"))

bytes.map("%02x".format(\_)).mkString

}

/\*\*

\* Helper function for timing a stitch, returning the original stitch.

\*

\* Defining the profiling function here to keep the dependencies of this class

\* generic and easy to export (i.e. copy-and-paste) into other services or packages.

\*/

def profileStitch[T](stitch: Stitch[T], stat: StatsReceiver): Stitch[T] = {

Stitch

.time(stitch)

.map {

case (response, stitchRunDuration) =>

stat.counter("requests").incr()

stat.stat("latency\_ms").add(stitchRunDuration.inMilliseconds)

response

.onSuccess { \_ => stat.counter("success").incr() }

.onFailure { e =>

stat.counter("failures").incr()

stat.scope("failures").counter(e.getClass.getSimpleName).incr()

}

}

.lowerFromTry

}

}