package com.twitter.follow\_recommendations.common.clients.user\_state

import com.google.inject.name.Named

import com.twitter.conversions.DurationOps.\_

import com.twitter.core\_workflows.user\_model.thriftscala.CondensedUserState

import com.twitter.core\_workflows.user\_model.thriftscala.UserState

import com.twitter.decider.Decider

import com.twitter.decider.RandomRecipient

import com.twitter.finagle.Memcached.Client

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.finagle.util.DefaultTimer

import com.twitter.follow\_recommendations.common.base.StatsUtil

import com.twitter.follow\_recommendations.common.clients.cache.MemcacheClient

import com.twitter.follow\_recommendations.common.clients.cache.ThriftEnumOptionBijection

import com.twitter.follow\_recommendations.common.constants.GuiceNamedConstants

import com.twitter.follow\_recommendations.configapi.deciders.DeciderKey

import com.twitter.stitch.Stitch

import com.twitter.strato.client.Fetcher

import com.twitter.util.Duration

import javax.inject.Inject

import javax.inject.Singleton

import java.lang.{Long => JLong}

@Singleton

class UserStateClient @Inject() (

@Named(GuiceNamedConstants.USER\_STATE\_FETCHER) userStateFetcher: Fetcher[

Long,

Unit,

CondensedUserState

],

client: Client,

statsReceiver: StatsReceiver,

decider: Decider = Decider.False) {

private val stats: StatsReceiver = statsReceiver.scope("user\_state\_client")

// client to memcache cluster

val bijection = new ThriftEnumOptionBijection[UserState](UserState.apply)

val memcacheClient = MemcacheClient[Option[UserState]](

client = client,

dest = "/s/cache/follow\_recos\_service:twemcaches",

valueBijection = bijection,

ttl = UserStateClient.CacheTTL,

statsReceiver = stats.scope("twemcache")

)

def getUserState(userId: Long): Stitch[Option[UserState]] = {

val deciderKey: String = DeciderKey.EnableDistributedCaching.toString

val enableDistributedCaching: Boolean = decider.isAvailable(deciderKey, Some(RandomRecipient))

val userStateStitch: Stitch[Option[UserState]] =

enableDistributedCaching match {

// read from memcache

case true => memcacheClient.readThrough(

// add a key prefix to address cache key collisions

key = "UserStateClient" + userId.toString,

underlyingCall = () => fetchUserState(userId)

)

case false => fetchUserState(userId)

}

val userStateStitchWithTimeout: Stitch[Option[UserState]] =

userStateStitch

// set a 150ms timeout limit for user state fetches

.within(150.milliseconds)(DefaultTimer)

.rescue {

case e: Exception =>

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

Stitch(None)

}

// profile the latency of stitch call and return the result

StatsUtil.profileStitch(

userStateStitchWithTimeout,

stats.scope("getUserState")

)

}

def fetchUserState(userId: JLong): Stitch[Option[UserState]] = {

userStateFetcher.fetch(userId).map(\_.v.flatMap(\_.userState))

}

}

object UserStateClient {

val CacheTTL: Duration = Duration.fromHours(6)

}