package com.twitter.cr\_mixer.module

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.bijection.Bufferable

import com.twitter.bijection.Injection

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.cr\_mixer.model.ModuleNames

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.memcached.{Client => MemcachedClient}

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.hermit.store.common.ObservedMemcachedReadableStore

import com.twitter.inject.TwitterModule

import com.twitter.simclusters\_v2.common.UserId

import com.twitter.snowflake.id.SnowflakeId

import com.twitter.storage.client.manhattan.kv.ManhattanKVClientMtlsParams

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.manhattan.ManhattanRO

import com.twitter.storehaus\_internal.manhattan.ManhattanROConfig

import com.twitter.storehaus\_internal.util.HDFSPath

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

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

import com.twitter.cr\_mixer.config.TimeoutConfig

import com.twitter.cr\_mixer.param.decider.CrMixerDecider

import com.twitter.cr\_mixer.param.decider.DeciderKey

import com.twitter.hermit.store.common.DeciderableReadableStore

import com.twitter.storehaus\_internal.manhattan.Apollo

import com.twitter.storehaus\_internal.util.ApplicationID

import com.twitter.storehaus\_internal.util.DatasetName

import com.twitter.util.Duration

import com.twitter.util.Future

import com.twitter.util.JavaTimer

import com.twitter.util.Time

import com.twitter.util.TimeoutException

import com.twitter.util.Timer

import javax.inject.Named

object UserStateStoreModule extends TwitterModule {

implicit val timer: Timer = new JavaTimer(true)

final val NewUserCreateDaysThreshold = 7

final val DefaultUnknownUserStateValue = 100

// Convert CondensedUserState to UserState Enum

// If CondensedUserState is None, back fill by checking whether the user is new user

class UserStateStore(

userStateStore: ReadableStore[UserId, CondensedUserState],

timeout: Duration,

statsReceiver: StatsReceiver)

extends ReadableStore[UserId, UserState] {

override def get(userId: UserId): Future[Option[UserState]] = {

userStateStore

.get(userId).map(\_.flatMap(\_.userState)).map {

case Some(userState) => Some(userState)

case None =>

val isNewUser = SnowflakeId.timeFromIdOpt(userId).exists { userCreateTime =>

Time.now - userCreateTime < Duration.fromDays(NewUserCreateDaysThreshold)

}

if (isNewUser) Some(UserState.New)

else Some(UserState.EnumUnknownUserState(DefaultUnknownUserStateValue))

}.raiseWithin(timeout)(timer).rescue {

case \_: TimeoutException =>

statsReceiver.counter("TimeoutException").incr()

Future.None

}

}

}

@Provides

@Singleton

def providesUserStateStore(

crMixerDecider: CrMixerDecider,

statsReceiver: StatsReceiver,

manhattanKVClientMtlsParams: ManhattanKVClientMtlsParams,

@Named(ModuleNames.UnifiedCache) crMixerUnifiedCacheClient: MemcachedClient,

timeoutConfig: TimeoutConfig

): ReadableStore[UserId, UserState] = {

val underlyingStore = new UserStateStore(

ManhattanRO

.getReadableStoreWithMtls[UserId, CondensedUserState](

ManhattanROConfig(

HDFSPath(""),

ApplicationID("cr\_mixer\_apollo"),

DatasetName("condensed\_user\_state"),

Apollo),

manhattanKVClientMtlsParams

)(

implicitly[Injection[Long, Array[Byte]]],

BinaryScalaCodec(CondensedUserState)

),

timeoutConfig.userStateStoreTimeout,

statsReceiver.scope("UserStateStore")

).mapValues(\_.value) // Read the value of Enum so that we only caches the Int

val memCachedStore = ObservedMemcachedReadableStore

.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 24.hours,

)(

valueInjection = Bufferable.injectionOf[Int], // Cache Value is Enum Value for UserState

statsReceiver = statsReceiver.scope("memCachedUserStateStore"),

keyToString = { k: UserId => s"uState/$k" }

).mapValues(value => UserState.getOrUnknown(value))

DeciderableReadableStore(

memCachedStore,

crMixerDecider.deciderGateBuilder.idGate(DeciderKey.enableUserStateStoreDeciderKey),

statsReceiver.scope("UserStateStore")

)

}

}