package com.twitter.cr\_mixer.module

import com.google.inject.Module

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.bijection.scrooge.BinaryScalaCodec

import com.twitter.contentrecommender.thriftscala.TweetInfo

import com.twitter.conversions.DurationOps.\_

import com.twitter.finagle.stats.StatsReceiver

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

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

import com.twitter.frigate.common.store.health.TweetHealthModelStore

import com.twitter.frigate.common.store.health.TweetHealthModelStore.TweetHealthModelStoreConfig

import com.twitter.frigate.common.store.health.UserHealthModelStore

import com.twitter.frigate.thriftscala.TweetHealthScores

import com.twitter.frigate.thriftscala.UserAgathaScores

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

import com.twitter.hermit.store.common.ObservedCachedReadableStore

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

import com.twitter.hermit.store.common.ObservedReadableStore

import com.twitter.inject.TwitterModule

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

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

import com.twitter.storehaus.ReadableStore

import com.twitter.strato.client.{Client => StratoClient}

import com.twitter.contentrecommender.store.TweetInfoStore

import com.twitter.contentrecommender.store.TweetyPieFieldsStore

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

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

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

import com.twitter.frigate.data\_pipeline.scalding.thriftscala.BlueVerifiedAnnotationsV2

import com.twitter.recos.user\_tweet\_graph\_plus.thriftscala.UserTweetGraphPlus

import com.twitter.recos.user\_tweet\_graph\_plus.thriftscala.TweetEngagementScores

import com.twitter.relevance\_platform.common.health\_store.UserMediaRepresentationHealthStore

import com.twitter.relevance\_platform.common.health\_store.MagicRecsRealTimeAggregatesStore

import com.twitter.relevance\_platform.thriftscala.MagicRecsRealTimeAggregatesScores

import com.twitter.relevance\_platform.thriftscala.UserMediaRepresentationScores

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

import com.twitter.tweetypie.thriftscala.TweetService

import com.twitter.util.Future

import com.twitter.util.JavaTimer

import com.twitter.util.Timer

import javax.inject.Named

object TweetInfoStoreModule extends TwitterModule {

implicit val timer: Timer = new JavaTimer(true)

override def modules: Seq[Module] = Seq(UnifiedCacheClient)

@Provides

@Singleton

def providesTweetInfoStore(

statsReceiver: StatsReceiver,

serviceIdentifier: ServiceIdentifier,

stratoClient: StratoClient,

@Named(ModuleNames.UnifiedCache) crMixerUnifiedCacheClient: MemcachedClient,

manhattanKVClientMtlsParams: ManhattanKVClientMtlsParams,

tweetyPieService: TweetService.MethodPerEndpoint,

userTweetGraphPlusService: UserTweetGraphPlus.MethodPerEndpoint,

@Named(ModuleNames.BlueVerifiedAnnotationStore) blueVerifiedAnnotationStore: ReadableStore[

String,

BlueVerifiedAnnotationsV2

],

decider: CrMixerDecider

): ReadableStore[TweetId, TweetInfo] = {

val tweetEngagementScoreStore: ReadableStore[TweetId, TweetEngagementScores] = {

val underlyingStore =

ObservedReadableStore(new ReadableStore[TweetId, TweetEngagementScores] {

override def get(

k: TweetId

): Future[Option[TweetEngagementScores]] = {

userTweetGraphPlusService.tweetEngagementScore(k).map {

Some(\_)

}

}

})(statsReceiver.scope("UserTweetGraphTweetEngagementScoreStore"))

DeciderableReadableStore(

underlyingStore,

decider.deciderGateBuilder.idGate(

DeciderKey.enableUtgRealTimeTweetEngagementScoreDeciderKey),

statsReceiver.scope("UserTweetGraphTweetEngagementScoreStore")

)

}

val tweetHealthModelStore: ReadableStore[TweetId, TweetHealthScores] = {

val underlyingStore = TweetHealthModelStore.buildReadableStore(

stratoClient,

Some(

TweetHealthModelStoreConfig(

enablePBlock = true,

enableToxicity = true,

enablePSpammy = true,

enablePReported = true,

enableSpammyTweetContent = true,

enablePNegMultimodal = true,

))

)(statsReceiver.scope("UnderlyingTweetHealthModelStore"))

DeciderableReadableStore(

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 2.hours

)(

valueInjection = BinaryScalaCodec(TweetHealthScores),

statsReceiver = statsReceiver.scope("memCachedTweetHealthModelStore"),

keyToString = { k: TweetId => s"tHMS/$k" }

),

decider.deciderGateBuilder.idGate(DeciderKey.enableHealthSignalsScoreDeciderKey),

statsReceiver.scope("TweetHealthModelStore")

) // use s"tHMS/$k" instead of s"tweetHealthModelStore/$k" to differentiate from CR cache

}

val userHealthModelStore: ReadableStore[UserId, UserAgathaScores] = {

val underlyingStore = UserHealthModelStore.buildReadableStore(stratoClient)(

statsReceiver.scope("UnderlyingUserHealthModelStore"))

DeciderableReadableStore(

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 18.hours

)(

valueInjection = BinaryScalaCodec(UserAgathaScores),

statsReceiver = statsReceiver.scope("memCachedUserHealthModelStore"),

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

),

decider.deciderGateBuilder.idGate(DeciderKey.enableUserAgathaScoreDeciderKey),

statsReceiver.scope("UserHealthModelStore")

)

}

val userMediaRepresentationHealthStore: ReadableStore[UserId, UserMediaRepresentationScores] = {

val underlyingStore =

UserMediaRepresentationHealthStore.buildReadableStore(

manhattanKVClientMtlsParams,

statsReceiver.scope("UnderlyingUserMediaRepresentationHealthStore")

)

DeciderableReadableStore(

ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 12.hours

)(

valueInjection = BinaryScalaCodec(UserMediaRepresentationScores),

statsReceiver = statsReceiver.scope("memCacheUserMediaRepresentationHealthStore"),

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

),

decider.deciderGateBuilder.idGate(DeciderKey.enableUserMediaRepresentationStoreDeciderKey),

statsReceiver.scope("UserMediaRepresentationHealthStore")

)

}

val magicRecsRealTimeAggregatesStore: ReadableStore[

TweetId,

MagicRecsRealTimeAggregatesScores

] = {

val underlyingStore =

MagicRecsRealTimeAggregatesStore.buildReadableStore(

serviceIdentifier,

statsReceiver.scope("UnderlyingMagicRecsRealTimeAggregatesScores")

)

DeciderableReadableStore(

underlyingStore,

decider.deciderGateBuilder.idGate(DeciderKey.enableMagicRecsRealTimeAggregatesStore),

statsReceiver.scope("MagicRecsRealTimeAggregatesStore")

)

}

val tweetInfoStore: ReadableStore[TweetId, TweetInfo] = {

val underlyingStore = TweetInfoStore(

TweetyPieFieldsStore.getStoreFromTweetyPie(tweetyPieService),

userMediaRepresentationHealthStore,

magicRecsRealTimeAggregatesStore,

tweetEngagementScoreStore,

blueVerifiedAnnotationStore

)(statsReceiver.scope("tweetInfoStore"))

val memcachedStore = ObservedMemcachedReadableStore.fromCacheClient(

backingStore = underlyingStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 15.minutes,

// Hydrating tweetInfo is now a required step for all candidates,

// hence we needed to tune these thresholds.

asyncUpdate = serviceIdentifier.environment == "prod"

)(

valueInjection = BinaryScalaCodec(TweetInfo),

statsReceiver = statsReceiver.scope("memCachedTweetInfoStore"),

keyToString = { k: TweetId => s"tIS/$k" }

)

ObservedCachedReadableStore.from(

memcachedStore,

ttl = 15.minutes,

maxKeys = 8388607, // Check TweetInfo definition. size~92b. Around 736 MB

windowSize = 10000L,

cacheName = "tweet\_info\_cache",

maxMultiGetSize = 20

)(statsReceiver.scope("inMemoryCachedTweetInfoStore"))

}

tweetInfoStore

}

}