package com.twitter.graph\_feature\_service.server.modules

import com.google.inject.Provides

import com.twitter.bijection.scrooge.CompactScalaCodec

import com.twitter.conversions.DurationOps.\_

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.graph\_feature\_service.common.Configs.\_

import com.twitter.graph\_feature\_service.server.stores.GetIntersectionStore

import com.twitter.graph\_feature\_service.server.stores.GetIntersectionStore.GetIntersectionQuery

import com.twitter.graph\_feature\_service.thriftscala.CachedIntersectionResult

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

import com.twitter.inject.TwitterModule

import com.twitter.inject.annotations.Flag

import com.twitter.storehaus.ReadableStore

import com.twitter.storehaus\_internal.memcache.MemcacheStore

import com.twitter.storehaus\_internal.util.{ClientName, ZkEndPoint}

import com.twitter.util.Duration

import javax.inject.{Named, Singleton}

/\*\*

\* Initialize the MemCache based GetIntersectionStore.

\* The Key of MemCache is UserId~CandidateId~FeatureTypes~IntersectionIdLimit.

\*/

object GetIntersectionStoreModule extends TwitterModule {

private[this] val requestTimeout: Duration = 25.millis

private[this] val retries: Int = 0

@Provides

@Named("ReadThroughGetIntersectionStore")

@Singleton

def provideReadThroughGetIntersectionStore(

graphFeatureServiceWorkerClients: GraphFeatureServiceWorkerClients,

serviceIdentifier: ServiceIdentifier,

@Flag(ServerFlagNames.MemCacheClientName) memCacheName: String,

@Flag(ServerFlagNames.MemCachePath) memCachePath: String

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[GetIntersectionQuery, CachedIntersectionResult] = {

buildMemcacheStore(

graphFeatureServiceWorkerClients,

memCacheName,

memCachePath,

serviceIdentifier)

}

@Provides

@Named("BypassCacheGetIntersectionStore")

@Singleton

def provideReadOnlyGetIntersectionStore(

graphFeatureServiceWorkerClients: GraphFeatureServiceWorkerClients,

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[GetIntersectionQuery, CachedIntersectionResult] = {

// Bypass the Memcache.

GetIntersectionStore(graphFeatureServiceWorkerClients, statsReceiver)

}

private[this] def buildMemcacheStore(

graphFeatureServiceWorkerClients: GraphFeatureServiceWorkerClients,

memCacheName: String,

memCachePath: String,

serviceIdentifier: ServiceIdentifier,

)(

implicit statsReceiver: StatsReceiver

): ReadableStore[GetIntersectionQuery, CachedIntersectionResult] = {

val backingStore = GetIntersectionStore(graphFeatureServiceWorkerClients, statsReceiver)

val cacheClient = MemcacheStore.memcachedClient(

name = ClientName(memCacheName),

dest = ZkEndPoint(memCachePath),

timeout = requestTimeout,

retries = retries,

serviceIdentifier = serviceIdentifier,

statsReceiver = statsReceiver

)

ObservedMemcachedReadableStore.fromCacheClient[GetIntersectionQuery, CachedIntersectionResult](

backingStore = backingStore,

cacheClient = cacheClient,

ttl = MemCacheTTL

)(

valueInjection = LZ4Injection.compose(CompactScalaCodec(CachedIntersectionResult)),

statsReceiver = statsReceiver.scope("mem\_cache"),

keyToString = { key =>

s"L~${key.userId}~${key.candidateId}~${key.featureTypesString}~${key.intersectionIdLimit}"

}

)

}

}