package com.twitter.cr\_mixer.module.similarity\_engine

import com.google.inject.Provides

import com.twitter.conversions.DurationOps.\_

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

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

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

import com.twitter.cr\_mixer.similarity\_engine.SimClustersANNSimilarityEngine

import com.twitter.cr\_mixer.similarity\_engine.SimClustersANNSimilarityEngine.Query

import com.twitter.cr\_mixer.similarity\_engine.SimilarityEngine.GatingConfig

import com.twitter.cr\_mixer.similarity\_engine.SimilarityEngine.SimilarityEngineConfig

import com.twitter.cr\_mixer.similarity\_engine.StandardSimilarityEngine

import com.twitter.cr\_mixer.thriftscala.SimilarityEngineType

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.hashing.KeyHasher

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

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

import com.twitter.inject.TwitterModule

import com.twitter.relevance\_platform.common.injection.LZ4Injection

import com.twitter.relevance\_platform.common.injection.SeqObjectInjection

import com.twitter.simclusters\_v2.candidate\_source.SimClustersANNCandidateSource.CacheableShortTTLEmbeddingTypes

import com.twitter.simclustersann.thriftscala.SimClustersANNService

import com.twitter.storehaus.ReadableStore

import com.twitter.util.Future

import javax.inject.Named

import javax.inject.Singleton

object SimClustersANNSimilarityEngineModule extends TwitterModule {

private val keyHasher: KeyHasher = KeyHasher.FNV1A\_64

@Provides

@Singleton

@Named(ModuleNames.SimClustersANNSimilarityEngine)

def providesProdSimClustersANNSimilarityEngine(

@Named(ModuleNames.UnifiedCache) crMixerUnifiedCacheClient: MemcachedClient,

simClustersANNServiceNameToClientMapper: Map[String, SimClustersANNService.MethodPerEndpoint],

timeoutConfig: TimeoutConfig,

statsReceiver: StatsReceiver

): StandardSimilarityEngine[Query, TweetWithScore] = {

val underlyingStore =

SimClustersANNSimilarityEngine(simClustersANNServiceNameToClientMapper, statsReceiver)

val observedReadableStore =

ObservedReadableStore(underlyingStore)(statsReceiver.scope("SimClustersANNServiceStore"))

val memCachedStore: ReadableStore[Query, Seq[TweetWithScore]] =

ObservedMemcachedReadableStore

.fromCacheClient(

backingStore = observedReadableStore,

cacheClient = crMixerUnifiedCacheClient,

ttl = 10.minutes

)(

valueInjection = LZ4Injection.compose(SeqObjectInjection[TweetWithScore]()),

statsReceiver = statsReceiver.scope("simclusters\_ann\_store\_memcache"),

keyToString = { k =>

//Example Query CRMixer:SCANN:1:2:1234567890ABCDEF:1234567890ABCDEF

f"CRMixer:SCANN:${k.simClustersANNQuery.sourceEmbeddingId.embeddingType.getValue()}%X" +

f":${k.simClustersANNQuery.sourceEmbeddingId.modelVersion.getValue()}%X" +

f":${keyHasher.hashKey(k.simClustersANNQuery.sourceEmbeddingId.internalId.toString.getBytes)}%X" +

f":${keyHasher.hashKey(k.simClustersANNQuery.config.toString.getBytes)}%X"

}

)

// Only cache the candidates if it's not Consumer-source. For example, TweetSource,

// ProducerSource, TopicSource

val wrapperStats = statsReceiver.scope("SimClustersANNWrapperStore")

val wrapperStore: ReadableStore[Query, Seq[TweetWithScore]] =

buildWrapperStore(memCachedStore, observedReadableStore, wrapperStats)

new StandardSimilarityEngine[

Query,

TweetWithScore

](

implementingStore = wrapperStore,

identifier = SimilarityEngineType.SimClustersANN,

globalStats = statsReceiver,

engineConfig = SimilarityEngineConfig(

timeout = timeoutConfig.similarityEngineTimeout,

gatingConfig = GatingConfig(

deciderConfig = None,

enableFeatureSwitch = None

)

)

)

}

def buildWrapperStore(

memCachedStore: ReadableStore[Query, Seq[TweetWithScore]],

underlyingStore: ReadableStore[Query, Seq[TweetWithScore]],

wrapperStats: StatsReceiver

): ReadableStore[Query, Seq[TweetWithScore]] = {

// Only cache the candidates if it's not Consumer-source. For example, TweetSource,

// ProducerSource, TopicSource

val wrapperStore: ReadableStore[Query, Seq[TweetWithScore]] =

new ReadableStore[Query, Seq[TweetWithScore]] {

override def multiGet[K1 <: Query](

queries: Set[K1]

): Map[K1, Future[Option[Seq[TweetWithScore]]]] = {

val (cacheableQueries, nonCacheableQueries) =

queries.partition { query =>

CacheableShortTTLEmbeddingTypes.contains(

query.simClustersANNQuery.sourceEmbeddingId.embeddingType)

}

memCachedStore.multiGet(cacheableQueries) ++

underlyingStore.multiGet(nonCacheableQueries)

}

}

wrapperStore

}

}