package com.twitter.follow\_recommendations.common.feature\_hydration.sources

import com.google.inject.Provides

import com.google.inject.Singleton

import com.twitter.escherbird.util.stitchcache.StitchCache

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

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.inject.TwitterModule

import com.twitter.stitch.Stitch

import com.twitter.storage.client.manhattan.bijections.Bijections.BinaryCompactScalaInjection

import com.twitter.storage.client.manhattan.bijections.Bijections.LongInjection

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

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

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

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

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

import com.twitter.storage.client.manhattan.kv.impl.Component

import com.twitter.storage.client.manhattan.kv.impl.Component0

import com.twitter.storage.client.manhattan.kv.impl.KeyDescriptor

import com.twitter.storage.client.manhattan.kv.impl.ValueDescriptor

import com.twitter.strato.generated.client.ml.featureStore.McUserCountingOnUserClientColumn

import com.twitter.strato.generated.client.ml.featureStore.onboarding.TimelinesAuthorFeaturesOnUserClientColumn

import com.twitter.timelines.author\_features.v1.thriftscala.AuthorFeatures

import com.twitter.conversions.DurationOps.\_

import com.twitter.onboarding.relevance.features.thriftscala.MCUserCountingFeatures

import java.lang.{Long => JLong}

import scala.util.Random

object HydrationSourcesModule extends TwitterModule {

val readFromManhattan = flag(

"feature\_hydration\_enable\_reading\_from\_manhattan",

false,

"Whether to read the data from Manhattan or Strato")

val manhattanAppId =

flag("frs\_readonly.appId", "ml\_features\_athena", "RO App Id used by the RO FRS service")

val manhattanDestName = flag(

"frs\_readonly.destName",

"/s/manhattan/athena.native-thrift",

"manhattan Dest Name used by the RO FRS service")

@Provides

@Singleton

def providesAthenaManhattanClient(

serviceIdentifier: ServiceIdentifier

): ManhattanKVEndpoint = {

val client = ManhattanKVClient(

manhattanAppId(),

manhattanDestName(),

ManhattanKVClientMtlsParams(serviceIdentifier)

)

ManhattanKVEndpointBuilder(client)

.defaultGuarantee(Guarantee.Weak)

.build()

}

val manhattanAuthorDataset = "timelines\_author\_features"

private val defaultCacheMaxKeys = 60000

private val cacheTTL = 12.hours

private val earlyExpiration = 0.2

val authorKeyDesc = KeyDescriptor(Component(LongInjection), Component0)

val authorDatasetKey = authorKeyDesc.withDataset(manhattanAuthorDataset)

val authorValDesc = ValueDescriptor(BinaryCompactScalaInjection(AuthorFeatures))

@Provides

@Singleton

def timelinesAuthorStitchCache(

manhattanReadOnlyEndpoint: ManhattanKVEndpoint,

timelinesAuthorFeaturesColumn: TimelinesAuthorFeaturesOnUserClientColumn,

stats: StatsReceiver

): StitchCache[JLong, Option[AuthorFeatures]] = {

val stitchCacheStats =

stats

.scope("direct\_ds\_source\_feature\_hydration\_module").scope("timelines\_author")

val stStat = stitchCacheStats.counter("readFromStrato-each")

val mhtStat = stitchCacheStats.counter("readFromManhattan-each")

val timelinesAuthorUnderlyingCall = if (readFromManhattan()) {

stitchCacheStats.counter("readFromManhattan").incr()

val authorCacheUnderlyingManhattanCall: JLong => Stitch[Option[AuthorFeatures]] = id => {

mhtStat.incr()

val key = authorDatasetKey.withPkey(id)

manhattanReadOnlyEndpoint

.get(key = key, valueDesc = authorValDesc).map(\_.map(value =>

clearUnsedFieldsForAuthorFeature(value.contents)))

}

authorCacheUnderlyingManhattanCall

} else {

stitchCacheStats.counter("readFromStrato").incr()

val authorCacheUnderlyingStratoCall: JLong => Stitch[Option[AuthorFeatures]] = id => {

stStat.incr()

val timelinesAuthorFeaturesFetcher = timelinesAuthorFeaturesColumn.fetcher

timelinesAuthorFeaturesFetcher

.fetch(id).map(result => result.v.map(clearUnsedFieldsForAuthorFeature))

}

authorCacheUnderlyingStratoCall

}

StitchCache[JLong, Option[AuthorFeatures]](

underlyingCall = timelinesAuthorUnderlyingCall,

maxCacheSize = defaultCacheMaxKeys,

ttl = randomizedTTL(cacheTTL.inSeconds).seconds,

statsReceiver = stitchCacheStats

)

}

// Not adding manhattan since it didn't seem useful for Author Data, we can add in another phab

// if deemed helpful

@Provides

@Singleton

def metricCenterUserCountingStitchCache(

mcUserCountingFeaturesColumn: McUserCountingOnUserClientColumn,

stats: StatsReceiver

): StitchCache[JLong, Option[MCUserCountingFeatures]] = {

val stitchCacheStats =

stats

.scope("direct\_ds\_source\_feature\_hydration\_module").scope("mc\_user\_counting")

val stStat = stitchCacheStats.counter("readFromStrato-each")

stitchCacheStats.counter("readFromStrato").incr()

val mcUserCountingCacheUnderlyingCall: JLong => Stitch[Option[MCUserCountingFeatures]] = id => {

stStat.incr()

val mcUserCountingFeaturesFetcher = mcUserCountingFeaturesColumn.fetcher

mcUserCountingFeaturesFetcher.fetch(id).map(\_.v)

}

StitchCache[JLong, Option[MCUserCountingFeatures]](

underlyingCall = mcUserCountingCacheUnderlyingCall,

maxCacheSize = defaultCacheMaxKeys,

ttl = randomizedTTL(cacheTTL.inSeconds).seconds,

statsReceiver = stitchCacheStats

)

}

// clear out fields we don't need to save cache space

private def clearUnsedFieldsForAuthorFeature(entry: AuthorFeatures): AuthorFeatures = {

entry.unsetUserTopics.unsetUserHealth.unsetAuthorCountryCodeAggregates.unsetOriginalAuthorCountryCodeAggregates

}

// To avoid a cache stampede. See https://en.wikipedia.org/wiki/Cache\_stampede

private def randomizedTTL(ttl: Long): Long = {

(ttl - ttl \* earlyExpiration \* Random.nextDouble()).toLong

}

}