package com.twitter.representationscorer.twistlyfeatures

import com.github.benmanes.caffeine.cache.Caffeine

import com.twitter.stitch.cache.EvictingCache

import com.google.inject.Provides

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.inject.TwitterModule

import com.twitter.representationscorer.common.RepresentationScorerDecider

import com.twitter.stitch.Stitch

import com.twitter.stitch.cache.ConcurrentMapCache

import com.twitter.stitch.cache.MemoizeQuery

import com.twitter.strato.client.Client

import com.twitter.strato.generated.client.recommendations.user\_signal\_service.SignalsClientColumn

import java.util.concurrent.ConcurrentMap

import java.util.concurrent.TimeUnit

import javax.inject.Singleton

object UserSignalServiceRecentEngagementsClientModule extends TwitterModule {

@Singleton

@Provides

def provide(

client: Client,

decider: RepresentationScorerDecider,

statsReceiver: StatsReceiver

): Long => Stitch[Engagements] = {

val stratoClient = new SignalsClientColumn(client)

/\*

This cache holds a users recent engagements for a short period of time, such that batched requests

for multiple (userid, tweetid) pairs don't all need to fetch them.

[1] Caffeine cache keys/values must be objects, so we cannot use the `Long` primitive directly.

The boxed java.lang.Long works as a key, since it is an object. In most situations the compiler

can see where auto(un)boxing can occur. However, here we seem to need some wrapper functions

with explicit types to allow the boxing to happen.

\*/

val mapCache: ConcurrentMap[java.lang.Long, Stitch[Engagements]] =

Caffeine

.newBuilder()

.expireAfterWrite(5, TimeUnit.SECONDS)

.maximumSize(

1000 // We estimate 5M unique users in a 5m period - with 2k RSX instances, assume that one will see < 1k in a 5s period

)

.build[java.lang.Long, Stitch[Engagements]]

.asMap

statsReceiver.provideGauge("ussRecentEngagementsClient", "cache\_size") { mapCache.size.toFloat }

val engagementsClient =

new UserSignalServiceRecentEngagementsClient(stratoClient, decider, statsReceiver)

val f = (l: java.lang.Long) => engagementsClient.get(l) // See note [1] above

val cachedCall = MemoizeQuery(f, EvictingCache.lazily(new ConcurrentMapCache(mapCache)))

(l: Long) => cachedCall(l) // see note [1] above

}

}