package com.twitter.follow\_recommendations.common.predicates.gizmoduck

import java.util.concurrent.TimeUnit

import com.google.common.base.Ticker

import com.google.common.cache.CacheBuilder

import com.google.common.cache.Cache

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.Time

import com.twitter.util.Duration

/\*\*

\* In-memory cache used for caching GizmoduckPredicate query calls in

\* com.twitter.follow\_recommendations.common.predicates.gizmoduck.GizmoduckPredicate.

\*

\* References the cache implementation in com.twitter.escherbird.util.stitchcache,

\* but without the underlying Stitch call.

\*/

object GizmoduckPredicateCache {

private[GizmoduckPredicateCache] class TimeTicker extends Ticker {

override def read(): Long = Time.now.inNanoseconds

}

def apply[K, V](

maxCacheSize: Int,

ttl: Duration,

statsReceiver: StatsReceiver

): Cache[K, V] = {

val cache: Cache[K, V] =

CacheBuilder

.newBuilder()

.maximumSize(maxCacheSize)

.asInstanceOf[CacheBuilder[K, V]]

.expireAfterWrite(ttl.inSeconds, TimeUnit.SECONDS)

.recordStats()

.ticker(new TimeTicker())

.build()

// metrics for tracking cache usage

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

statsReceiver.provideGauge("cache\_hits") { cache.stats.hitCount.toFloat }

statsReceiver.provideGauge("cache\_misses") { cache.stats.missCount.toFloat }

statsReceiver.provideGauge("cache\_hit\_rate") { cache.stats.hitRate.toFloat }

statsReceiver.provideGauge("cache\_evictions") { cache.stats.evictionCount.toFloat }

cache

}

}