package com.twitter.servo.repository

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.logging.Logger

import com.twitter.servo.cache.{InProcessCache, StatsReceiverCacheObserver}

import com.twitter.servo.util.FrequencyCounter

import com.twitter.util.Future

/\*\*

\* A KeyValueRepository which uses a sliding window to track

\* the frequency at which keys are requested and diverts requests

\* for keys above the promotionThreshold through an in-memory request cache.

\*

\* @param underlyingRepo

\* the underlying KeyValueRepository

\* @param newQuery

\* a function for converting a subset of the keys of the original query into a new query.

\* @param windowSize

\* the number of previous requests to include in the window

\* @param promotionThreshold

\* the number of requests for the same key in the window required

\* to divert the request through the request cache

\* @param cacheFactory

\* a function which constructs a future response cache of the given size

\* @param statsReceiver

\* records stats on the cache

\* @param disableLogging

\* disables logging in token cache for pdp purposes

\*/

object HotKeyCachingKeyValueRepository {

def apply[Q <: Seq[K], K, V](

underlyingRepo: KeyValueRepository[Q, K, V],

newQuery: SubqueryBuilder[Q, K],

windowSize: Int,

promotionThreshold: Int,

cacheFactory: Int => InProcessCache[K, Future[Option[V]]],

statsReceiver: StatsReceiver,

disableLogging: Boolean = false

): KeyValueRepository[Q, K, V] = {

val log = Logger.get(getClass.getSimpleName)

val promotionsCounter = statsReceiver.counter("promotions")

val onPromotion = { (k: K) =>

log.debug("key %s promoted to HotKeyCache", k.toString)

promotionsCounter.incr()

}

val frequencyCounter = new FrequencyCounter[K](windowSize, promotionThreshold, onPromotion)

// Maximum cache size occurs in the event that every key in the buffer occurs

// `promotionThreshold` times. We apply a failure-refreshing filter to avoid

// caching failed responses.

val cache =

InProcessCache.withFilter(

cacheFactory(windowSize / promotionThreshold)

)(

ResponseCachingKeyValueRepository.refreshFailures

)

val observer =

new StatsReceiverCacheObserver(statsReceiver, windowSize, "request\_cache", disableLogging)

val cachingRepo =

new ResponseCachingKeyValueRepository[Q, K, V](underlyingRepo, cache, newQuery, observer)

KeyValueRepository.selected(

frequencyCounter.incr,

cachingRepo,

underlyingRepo,

newQuery

)

}

}