package com.twitter.unified\_user\_actions.enricher.hcache

import com.twitter.cache.FutureCache

import com.twitter.cache.FutureCacheProxy

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.Future

import scala.annotation.nowarn

/\*\*

\* Adds stats and reuse the main logic of the EvictingCache.

\*/

class ObservedEvictingCache[K, V](underlying: FutureCache[K, V], scopedStatsReceiver: StatsReceiver)

extends FutureCacheProxy[K, V](underlying) {

import ObservedEvictingCache.\_

private[this] val getsCounter = scopedStatsReceiver.counter(StatsNames.Gets)

private[this] val setsCounter = scopedStatsReceiver.counter(StatsNames.Sets)

private[this] val hitsCounter = scopedStatsReceiver.counter(StatsNames.Hits)

private[this] val missesCounter = scopedStatsReceiver.counter(StatsNames.Misses)

private[this] val evictionsCounter = scopedStatsReceiver.counter(StatsNames.Evictions)

private[this] val failedFuturesCounter = scopedStatsReceiver.counter(StatsNames.FailedFutures)

@nowarn("cat=unused")

private[this] val cacheSizeGauge = scopedStatsReceiver.addGauge(StatsNames.Size)(underlying.size)

private[this] def evictOnFailure(k: K, f: Future[V]): Future[V] = {

f.onFailure { \_ =>

failedFuturesCounter.incr()

evict(k, f)

}

f // we return the original future to make evict(k, f) easier to work with.

}

override def set(k: K, v: Future[V]): Unit = {

setsCounter.incr()

super.set(k, v)

evictOnFailure(k, v)

}

override def getOrElseUpdate(k: K)(v: => Future[V]): Future[V] = {

getsCounter.incr()

var computeWasEvaluated = false

def computeWithTracking: Future[V] = v.onSuccess { \_ =>

computeWasEvaluated = true

missesCounter.incr()

}

evictOnFailure(

k,

super.getOrElseUpdate(k)(computeWithTracking).onSuccess { \_ =>

if (!computeWasEvaluated) hitsCounter.incr()

}

).interruptible()

}

override def get(key: K): Option[Future[V]] = {

getsCounter.incr()

val value = super.get(key)

value match {

case Some(\_) => hitsCounter.incr()

case \_ => missesCounter.incr()

}

value

}

override def evict(key: K, value: Future[V]): Boolean = {

val evicted = super.evict(key, value)

if (evicted) evictionsCounter.incr()

evicted

}

}

object ObservedEvictingCache {

object StatsNames {

val Gets = "gets"

val Hits = "hits"

val Misses = "misses"

val Sets = "sets"

val Evictions = "evictions"

val FailedFutures = "failed\_futures"

val Size = "size"

}

/\*\*

\* Wraps an underlying FutureCache, ensuring that failed Futures that are set in

\* the cache are evicted later.

\*/

def apply[K, V](underlying: FutureCache[K, V], statsReceiver: StatsReceiver): FutureCache[K, V] =

new ObservedEvictingCache[K, V](underlying, statsReceiver)

}