package com.twitter.servo.cache

import com.twitter.logging.Logger

import com.twitter.util.{Future, Return, Throw, Try}

object SecondaryIndexingCache {

type IndexMapping[S, V] = V => Try[Option[S]]

}

/\*\*

\* Stores a secondary index whenever set is called,

\* using a mapping from value to secondary index

\*/

class SecondaryIndexingCache[K, S, V](

override val underlyingCache: Cache[K, Cached[V]],

secondaryIndexCache: Cache[S, Cached[K]],

secondaryIndex: SecondaryIndexingCache.IndexMapping[S, V])

extends CacheWrapper[K, Cached[V]] {

protected[this] val log = Logger.get(getClass.getSimpleName)

protected[this] def setSecondaryIndex(key: K, cachedValue: Cached[V]): Future[Unit] =

cachedValue.value match {

case Some(value) =>

secondaryIndex(value) match {

case Return(Some(index)) =>

val cachedKey = cachedValue.copy(value = Some(key))

secondaryIndexCache.set(index, cachedKey)

case Return.None =>

Future.Done

case Throw(t) =>

log.error(t, "failed to determine secondary index for: %s", cachedValue)

Future.Done

}

// if we're storing a tombstone, no secondary index can be made

case None => Future.Done

}

override def set(key: K, cachedValue: Cached[V]): Future[Unit] =

super.set(key, cachedValue) flatMap { \_ =>

setSecondaryIndex(key, cachedValue)

}

override def checkAndSet(key: K, cachedValue: Cached[V], checksum: Checksum): Future[Boolean] =

super.checkAndSet(key, cachedValue, checksum) flatMap { wasStored =>

if (wasStored)

// do a straight set of the secondary index, but only if the CAS succeeded

setSecondaryIndex(key, cachedValue) map { \_ =>

true

}

else

Future.value(false)

}

override def add(key: K, cachedValue: Cached[V]): Future[Boolean] =

super.add(key, cachedValue) flatMap { wasAdded =>

if (wasAdded)

// do a straight set of the secondary index, but only if the add succeeded

setSecondaryIndex(key, cachedValue) map { \_ =>

true

}

else

Future.value(false)

}

override def replace(key: K, cachedValue: Cached[V]): Future[Boolean] =

super.replace(key, cachedValue) flatMap { wasReplaced =>

if (wasReplaced)

setSecondaryIndex(key, cachedValue) map { \_ =>

true

}

else

Future.value(false)

}

override def release(): Unit = {

underlyingCache.release()

secondaryIndexCache.release()

}

def withSecondaryIndex[T](

secondaryIndexingCache: Cache[T, Cached[K]],

secondaryIndex: SecondaryIndexingCache.IndexMapping[T, V]

): SecondaryIndexingCache[K, T, V] =

new SecondaryIndexingCache[K, T, V](this, secondaryIndexingCache, secondaryIndex)

}