package com.twitter.servo.cache

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.{Duration, Future}

case class ByteCountingMemcacheFactory(

memcacheFactory: MemcacheFactory,

statsReceiver: StatsReceiver,

delimiter: String = constants.Colon,

checksumSize: Int = 8) // memcached checksums are u64s

extends MemcacheFactory {

def apply() =

new ByteCountingMemcache(memcacheFactory(), statsReceiver, delimiter, checksumSize)

}

/\*\*

\* A decorator around a Memcache that counts the rough number

\* of bytes transferred, bucketed & rolled up by in/out, method name,

\* and key prefix

\*/

class ByteCountingMemcache(

underlying: Memcache,

statsReceiver: StatsReceiver,

delimiter: String,

checksumSize: Int)

extends Memcache {

val scopedReceiver = statsReceiver.scope("memcache").scope("bytes")

val outStat = scopedReceiver.stat("out")

val outReceiver = scopedReceiver.scope("out")

val inStat = scopedReceiver.stat("in")

val inReceiver = scopedReceiver.scope("in")

val getOutStat = outReceiver.stat("get")

val getOutReceiver = outReceiver.scope("get")

val getInStat = inReceiver.stat("get")

val getInReceiver = inReceiver.scope("get")

val getInHitsStat = getInReceiver.stat("hits")

val getInHitsReceiver = getInReceiver.scope("hits")

val getInMissesStat = getInReceiver.stat("misses")

val getInMissesReceiver = getInReceiver.scope("misses")

val gwcOutStat = outReceiver.stat("get\_with\_checksum")

val gwcOutReceiver = outReceiver.scope("get\_with\_checksum")

val gwcInStat = inReceiver.stat("get\_with\_checksum")

val gwcInReceiver = inReceiver.scope("get\_with\_checksum")

val gwcInHitsStat = gwcOutReceiver.stat("hits")

val gwcInHitsReceiver = gwcOutReceiver.scope("hits")

val gwcInMissesStat = gwcOutReceiver.stat("misses")

val gwcInMissesReceiver = gwcOutReceiver.scope("misses")

val addStat = outReceiver.stat("add")

val addReceiver = outReceiver.scope("add")

val setStat = outReceiver.stat("set")

val setReceiver = outReceiver.scope("set")

val replaceStat = outReceiver.stat("replace")

val replaceReceiver = outReceiver.scope("replace")

val casStat = outReceiver.stat("check\_and\_set")

val casReceiver = outReceiver.scope("check\_and\_set")

def release() = underlying.release()

// get namespace from key

protected[this] def ns(key: String) = {

val idx = math.min(key.size - 1, math.max(key.lastIndexOf(delimiter), 0))

key.substring(0, idx).replaceAll(delimiter, "\_")

}

override def get(keys: Seq[String]): Future[KeyValueResult[String, Array[Byte]]] = {

keys foreach { key =>

val size = key.size

outStat.add(size)

getOutStat.add(size)

getOutReceiver.stat(ns(key)).add(size)

}

underlying.get(keys) onSuccess { lr =>

lr.found foreach {

case (key, bytes) =>

val size = key.size + bytes.length

inStat.add(size)

getInStat.add(size)

getInHitsStat.add(size)

getInHitsReceiver.stat(ns(key)).add(size)

}

lr.notFound foreach { key =>

val size = key.size

inStat.add(size)

getInStat.add(size)

getInMissesStat.add(size)

getInMissesReceiver.stat(ns(key)).add(size)

}

}

}

override def getWithChecksum(

keys: Seq[String]

): Future[CsKeyValueResult[String, Array[Byte]]] = {

keys foreach { key =>

val size = key.size

outStat.add(size)

gwcOutStat.add(size)

gwcOutReceiver.stat(ns(key)).add(size)

}

underlying.getWithChecksum(keys) onSuccess { lr =>

lr.found foreach {

case (key, (bytes, \_)) =>

val size = key.size + (bytes map { \_.length } getOrElse (0)) + checksumSize

inStat.add(size)

gwcInStat.add(size)

gwcInHitsStat.add(size)

gwcInHitsReceiver.stat(ns(key)).add(size)

}

lr.notFound foreach { key =>

val size = key.size

inStat.add(size)

gwcInStat.add(size)

gwcInMissesStat.add(size)

gwcInMissesReceiver.stat(ns(key)).add(size)

}

}

}

override def add(key: String, value: Array[Byte], ttl: Duration): Future[Boolean] = {

val size = key.size + value.size

outStat.add(size)

addStat.add(size)

addReceiver.stat(ns(key)).add(size)

underlying.add(key, value, ttl)

}

override def checkAndSet(

key: String,

value: Array[Byte],

checksum: Checksum,

ttl: Duration

): Future[Boolean] = {

val size = key.size + value.size + checksumSize

outStat.add(size)

casStat.add(size)

casReceiver.stat(ns(key)).add(size)

underlying.checkAndSet(key, value, checksum, ttl)

}

override def set(key: String, value: Array[Byte], ttl: Duration): Future[Unit] = {

val size = key.size + value.size

outStat.add(size)

setStat.add(size)

setReceiver.stat(ns(key)).add(size)

underlying.set(key, value, ttl)

}

override def replace(key: String, value: Array[Byte], ttl: Duration): Future[Boolean] = {

val size = key.size + value.size

outStat.add(size)

replaceStat.add(size)

replaceReceiver.stat(ns(key)).add(size)

underlying.replace(key, value, ttl)

}

override def delete(key: String): Future[Boolean] = {

outStat.add(key.size)

underlying.delete(key)

}

override def incr(key: String, delta: Long = 1): Future[Option[Long]] = {

val size = key.size + 8

outStat.add(size)

underlying.incr(key, delta)

}

override def decr(key: String, delta: Long = 1): Future[Option[Long]] = {

val size = key.size + 8

outStat.add(size)

underlying.decr(key, delta)

}

}