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

import com.google.common.cache.Cache

import com.google.common.cache.CacheBuilder

import com.twitter.finagle.stats.InMemoryStatsReceiver

import com.twitter.inject.Test

import com.twitter.util.Await

import com.twitter.util.Future

import com.twitter.util.Time

import java.util.concurrent.TimeUnit

import java.lang.{Integer => JInt}

class LocalCacheTest extends Test {

trait Fixture {

val time = Time.fromMilliseconds(123456L)

val ttl = 5

val maxSize = 10

val underlying: Cache[JInt, Future[JInt]] = CacheBuilder

.newBuilder()

.expireAfterWrite(ttl, TimeUnit.SECONDS)

.maximumSize(maxSize)

.build[JInt, Future[JInt]]()

val stats = new InMemoryStatsReceiver

val cache = new LocalCache[JInt, JInt](

underlying = underlying,

statsReceiver = stats

)

def getCounts(counterName: String\*): Long = stats.counter(counterName: \_\*)()

}

test("simple local cache works") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

(1 to maxSize + 1).foreach { id =>

cache.getOrElseUpdate(id)(Future.value(id))

val actual = Await.result(cache.get(id).get)

assert(actual === id)

}

assert(cache.size === maxSize)

assert(getCounts("gets") === 2 \* (maxSize + 1))

assert(getCounts("hits") === maxSize + 1)

assert(getCounts("misses") === maxSize + 1)

assert(getCounts("sets", "evictions", "failed\_futures") === 0)

cache.reset()

assert(cache.size === 0)

}

}

}

test("getOrElseUpdate successful futures") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

(1 to maxSize + 1).foreach { \_ =>

cache.getOrElseUpdate(1) {

Future.value(1)

}

}

assert(cache.size === 1)

assert(getCounts("gets") === maxSize + 1)

assert(getCounts("hits") === maxSize)

assert(getCounts("misses") === 1)

assert(getCounts("sets", "evictions", "failed\_futures") === 0)

cache.reset()

assert(cache.size === 0)

}

}

}

test("getOrElseUpdate Failed Futures") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

(1 to maxSize + 1).foreach { id =>

cache.getOrElseUpdate(id)(Future.exception(new IllegalArgumentException("")))

assert(cache.get(id).map {

Await.result(\_)

} === None)

}

assert(cache.size === 0)

assert(getCounts("gets") === 2 \* (maxSize + 1))

assert(getCounts("hits", "misses", "sets") === 0)

assert(getCounts("evictions") === maxSize + 1)

assert(getCounts("failed\_futures") === maxSize + 1)

}

}

}

test("Set successful Future") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

cache.set(1, Future.value(2))

assert(Await.result(cache.get(1).get) === 2)

assert(getCounts("gets") === 1)

assert(getCounts("hits") === 1)

assert(getCounts("misses") === 0)

assert(getCounts("sets") === 1)

assert(getCounts("evictions", "failed\_futures") === 0)

}

}

}

test("Evict") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

// need to use reference here!!!

val f1 = Future.value(int2Integer(1))

val f2 = Future.value(int2Integer(2))

cache.set(1, f2)

cache.evict(1, f1)

cache.evict(1, f2)

assert(getCounts("gets", "hits", "misses") === 0)

assert(getCounts("sets") === 1)

assert(getCounts("evictions") === 1) // not 2

assert(getCounts("failed\_futures") === 0)

}

}

}

test("Set Failed Futures") {

new Fixture {

Time.withTimeAt(time) { \_ =>

assert(cache.size === 0)

cache.set(1, Future.exception(new IllegalArgumentException("")))

assert(cache.size === 0)

assert(getCounts("gets", "hits", "misses", "sets") === 0)

assert(getCounts("evictions") === 1)

assert(getCounts("failed\_futures") === 1)

}

}

}

}