package com.twitter.servo.cache

import com.google.common.cache.{CacheBuilder, RemovalListener}

import com.twitter.util.Duration

import java.util.concurrent.TimeUnit

object InProcessCache {

/\*\*

\* Apply a read filter to exclude items in an InProcessCache

\*/

def withFilter[K, V](

underlying: InProcessCache[K, V]

)(

shouldFilter: (K, V) => Boolean

): InProcessCache[K, V] =

new InProcessCache[K, V] {

def get(key: K): Option[V] = underlying.get(key) filterNot { shouldFilter(key, \_) }

def set(key: K, value: V) = underlying.set(key, value)

}

}

/\*\*

\* An in-process cache interface. It is distinct from a map in that:

\* 1) All methods must be threadsafe

\* 2) A value set in cache is not guaranteed to remain in the cache.

\*/

trait InProcessCache[K, V] {

def get(key: K): Option[V]

def set(key: K, value: V): Unit

}

/\*\*

\* In-process implementation of a cache with LRU semantics and a TTL.

\*/

class ExpiringLruInProcessCache[K, V](

ttl: Duration,

maximumSize: Int,

removalListener: Option[RemovalListener[K, V]] = None: None.type)

extends InProcessCache[K, V] {

private[this] val cacheBuilder =

CacheBuilder.newBuilder

.asInstanceOf[CacheBuilder[K, V]]

.expireAfterWrite(ttl.inMilliseconds, TimeUnit.MILLISECONDS)

.initialCapacity(maximumSize)

.maximumSize(maximumSize)

private[this] val cache =

removalListener match {

case Some(listener) =>

cacheBuilder

.removalListener(listener)

.build[K, V]()

case None =>

cacheBuilder

.build[K, V]()

}

def get(key: K): Option[V] = Option(cache.getIfPresent(key))

def set(key: K, value: V): Unit = cache.put(key, value)

}