package com.twitter.servo.cache

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

/\*\*

\* a Cache that takes a TTL per set

\*/

trait TtlCache[K, V] extends ReadCache[K, V] {

def add(key: K, value: V, ttl: Duration): Future[Boolean]

def checkAndSet(key: K, value: V, checksum: Checksum, ttl: Duration): Future[Boolean]

def set(key: K, value: V, ttl: Duration): Future[Unit]

/\*\*

\* Replaces the value for an existing key. If the key doesn't exist, this has no effect.

\* @return true if replaced, false if not found

\*/

def replace(key: K, value: V, ttl: Duration): Future[Boolean]

/\*\*

\* Deletes a value from cache.

\* @return true if deleted, false if not found

\*/

def delete(key: K): Future[Boolean]

}

/\*\*

\* allows one TtlCache to wrap another

\*/

trait TtlCacheWrapper[K, V] extends TtlCache[K, V] with ReadCacheWrapper[K, V, TtlCache[K, V]] {

override def add(key: K, value: V, ttl: Duration) = underlyingCache.add(key, value, ttl)

override def checkAndSet(key: K, value: V, checksum: Checksum, ttl: Duration) =

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

override def set(key: K, value: V, ttl: Duration) = underlyingCache.set(key, value, ttl)

override def replace(key: K, value: V, ttl: Duration) = underlyingCache.replace(key, value, ttl)

override def delete(key: K) = underlyingCache.delete(key)

}

class PerturbedTtlCache[K, V](

override val underlyingCache: TtlCache[K, V],

perturbTtl: Duration => Duration)

extends TtlCacheWrapper[K, V] {

override def add(key: K, value: V, ttl: Duration) =

underlyingCache.add(key, value, perturbTtl(ttl))

override def checkAndSet(key: K, value: V, checksum: Checksum, ttl: Duration) =

underlyingCache.checkAndSet(key, value, checksum, perturbTtl(ttl))

override def set(key: K, value: V, ttl: Duration) =

underlyingCache.set(key, value, perturbTtl(ttl))

override def replace(key: K, value: V, ttl: Duration) =

underlyingCache.replace(key, value, perturbTtl(ttl))

}

/\*\*

\* an adaptor to wrap a Cache[K, V] interface around a TtlCache[K, V]

\*/

class TtlCacheToCache[K, V](override val underlyingCache: TtlCache[K, V], ttl: (K, V) => Duration)

extends Cache[K, V]

with ReadCacheWrapper[K, V, TtlCache[K, V]] {

override def add(key: K, value: V) = underlyingCache.add(key, value, ttl(key, value))

override def checkAndSet(key: K, value: V, checksum: Checksum) =

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

override def set(key: K, value: V) = underlyingCache.set(key, value, ttl(key, value))

override def replace(key: K, value: V) = underlyingCache.replace(key, value, ttl(key, value))

override def delete(key: K) = underlyingCache.delete(key)

}

/\*\*

\* use a single TTL for all objects

\*/

class SimpleTtlCacheToCache[K, V](underlyingTtlCache: TtlCache[K, V], ttl: Duration)

extends TtlCacheToCache[K, V](underlyingTtlCache, (k: K, v: V) => ttl)

/\*\*

\* use a value-based TTL function

\*/

class ValueBasedTtlCacheToCache[K, V](underlyingTtlCache: TtlCache[K, V], ttl: V => Duration)

extends TtlCacheToCache[K, V](underlyingTtlCache, (k: K, v: V) => ttl(v))

/\*\*

\* use a key-based TTL function

\*/

class KeyBasedTtlCacheToCache[K, V](underlyingTtlCache: TtlCache[K, V], ttl: K => Duration)

extends TtlCacheToCache[K, V](underlyingTtlCache, (k: K, v: V) => ttl(k))