package com.twitter.tweetypie.caching

import com.twitter.io.Buf

import com.twitter.scrooge.CompactThriftSerializer

import com.twitter.scrooge.ThriftStruct

import com.twitter.scrooge.ThriftStructCodec

import com.twitter.servo.cache.thriftscala.CachedValue

import com.twitter.servo.cache.thriftscala.CachedValueStatus

import com.twitter.stitch.NotFound

import com.twitter.util.Return

import com.twitter.util.Throw

import com.twitter.util.Time

import com.twitter.util.Try

import java.nio.ByteBuffer

object ServoCachedValueSerializer {

/\*\*

\* Thrown when the fields of the servo CachedValue struct do not

\* satisfy the invariants expected by this serialization code.

\*/

case class UnexpectedCachedValueState(cachedValue: CachedValue) extends Exception {

def message: String = s"Unexpected state for CachedValue. Value was: $cachedValue"

}

val CachedValueThriftSerializer: CompactThriftSerializer[CachedValue] = CompactThriftSerializer(

CachedValue)

}

/\*\*

\* A [[ValueSerializer]] that is compatible with the use of

\* Servo's [[CachedValue]] struct by tweetypie:

\*

\* - The only [[CachedValueStatus]] values that are cacheable are

\* [[CachedValueStatus.Found]] and [[CachedValueStatus.NotFound]].

\*

\* - We only track the `cachedAtMsec` field, because tweetypie's cache

\* interaction does not use the other fields, and the values that

\* are cached this way are never updated, so storing readThroughAt

\* or writtenThroughAt would not add any information.

\*

\* - When values are present, they are serialized using

\* [[org.apache.thrift.protocol.TCompactProtocol]].

\*

\* - The CachedValue struct itself is also serialized using TCompactProtocol.

\*

\* The serializer operates on [[Try]] values and will cache [[Return]]

\* and `Throw(NotFound)` values.

\*/

case class ServoCachedValueSerializer[V <: ThriftStruct](

codec: ThriftStructCodec[V],

expiry: Try[V] => Time,

softTtl: SoftTtl[Try[V]])

extends ValueSerializer[Try[V]] {

import ServoCachedValueSerializer.UnexpectedCachedValueState

import ServoCachedValueSerializer.CachedValueThriftSerializer

private[this] val ValueThriftSerializer = CompactThriftSerializer(codec)

/\*\*

\* Return an expiry based on the value and a

\* TCompactProtocol-encoded servo CachedValue struct with the

\* following fields defined:

\*

\* - `value`: [[None]]

\* for {{{Throw(NotFound)}}, {{{Some(encodedStruct)}}} for

\* [[Return]], where {{{encodedStruct}}} is a

\* TCompactProtocol-encoding of the value inside of the Return.

\*

\* - `status`: [[CachedValueStatus.Found]] if the value is Return,

\* and [[CachedValueStatus.NotFound]] if it is Throw(NotFound)

\*

\* - `cachedAtMsec`: The current time, accoring to [[Time.now]]

\*

\* No other fields will be defined.

\*

\* @throws IllegalArgumentException if called with a value that

\* should not be cached.

\*/

override def serialize(value: Try[V]): Option[(Time, Buf)] = {

def serializeCachedValue(payload: Option[ByteBuffer]) = {

val cachedValue = CachedValue(

value = payload,

status = if (payload.isDefined) CachedValueStatus.Found else CachedValueStatus.NotFound,

cachedAtMsec = Time.now.inMilliseconds)

val serialized = Buf.ByteArray.Owned(CachedValueThriftSerializer.toBytes(cachedValue))

(expiry(value), serialized)

}

value match {

case Throw(NotFound) =>

Some(serializeCachedValue(None))

case Return(struct) =>

val payload = Some(ByteBuffer.wrap(ValueThriftSerializer.toBytes(struct)))

Some(serializeCachedValue(payload))

case \_ =>

None

}

}

/\*\*

\* Deserializes values serialized by [[serializeValue]]. The

\* value will be [[CacheResult.Fresh]] or [[CacheResult.Stale]]

\* depending on the result of {{{softTtl.isFresh}}}.

\*

\* @throws UnexpectedCachedValueState if the state of the

\* [[CachedValue]] could not be produced by [[serialize]]

\*/

override def deserialize(buf: Buf): CacheResult[Try[V]] = {

val cachedValue = CachedValueThriftSerializer.fromBytes(Buf.ByteArray.Owned.extract(buf))

val hasValue = cachedValue.value.isDefined

val isValid =

(hasValue && cachedValue.status == CachedValueStatus.Found) ||

(!hasValue && cachedValue.status == CachedValueStatus.NotFound)

if (!isValid) {

// Exceptions thrown by deserialization are recorded and treated

// as a cache miss by CacheOperations, so throwing this

// exception will cause the value in cache to be

// overwritten. There will be stats recorded whenever this

// happens.

throw UnexpectedCachedValueState(cachedValue)

}

val value =

cachedValue.value match {

case Some(valueBuffer) =>

val valueBytes = new Array[Byte](valueBuffer.remaining)

valueBuffer.duplicate.get(valueBytes)

Return(ValueThriftSerializer.fromBytes(valueBytes))

case None =>

Throw(NotFound)

}

softTtl.toCacheResult(value, Time.fromMilliseconds(cachedValue.cachedAtMsec))

}

}