package com.twitter.servo.cache

import com.google.common.base.Charsets

import com.twitter.util.Try

/\*\*

\* Fast implementation of dealing with memcached counters.

\*

\* Memcache is funkytown for incr and decr. Basically, you store a number,

\* as a STRING, and then incr and decr that. This abstracts over that detail.

\*

\* This implementation was quite a bit faster than the simple implementation

\* of `new String(bytes, Charsets.US\_ASCII).toLong()`

\* and `Long.toString(value).getBytes()`

\*

\* Thread-safe.

\*/

object CounterSerializer extends Serializer[Long] {

private[this] val Minus = '-'.toByte

// The lower bound

private[this] val Zero = '0'.toByte

// The upper bound

private[this] val Nine = '9'.toByte

// Max length for our byte arrays that'll fit all positive longs

private[this] val MaxByteArrayLength = 19

override def to(long: Long): Try[Array[Byte]] = Try {

// NOTE: code based on Long.toString(value), but it avoids creating the

// intermediate String object and the charset encoding in String.getBytes

// This was about 12% faster than calling Long.toString(long).getBytes

if (long == Long.MinValue) {

"-9223372036854775808".getBytes(Charsets.US\_ASCII)

} else {

val size = if (long < 0) stringSize(-long) + 1 else stringSize(long)

val bytes = new Array[Byte](size)

var isNegative = false

var endAt = 0

var currentLong = if (long < 0) {

isNegative = true

endAt = 1

-long

} else {

long

}

// Note: look at the implementation in Long.getChars(long, int, char[])

// They can do 2 digits at a time for this, so we could speed this up

// See: Division by Invariant Integers using Multiplication

// http://gmplib.org/~tege/divcnst-pldi94.pdf

// starting at the least significant digit and working our way up...

var pos = size - 1

do {

val byte = currentLong % 10

bytes(pos) = (Zero + byte).toByte

currentLong /= 10

pos -= 1

} while (currentLong != 0)

if (isNegative) {

assert(pos == 0, "For value " + long + ", pos " + pos)

bytes(0) = Minus

}

bytes

}

}

override def from(bytes: Array[Byte]): Try[Long] = Try {

// This implementation was about 4x faster than the simple:

// new String(bytes, Charsets.US\_ASCII).toLong

if (bytes.length < 1)

throw new NumberFormatException("Empty byte arrays are unsupported")

val isNegative = bytes(0) == Minus

if (isNegative && bytes.length == 1)

throw new NumberFormatException(bytes.mkString(","))

// we count in negative numbers so we don't have problems at Long.MaxValue

var total = 0L

val endAt = bytes.length

var i = if (isNegative) 1 else 0

while (i < endAt) {

val b = bytes(i)

if (b < Zero || b > Nine)

throw new NumberFormatException(bytes.mkString(","))

val int = b - Zero

total = (total \* 10L) - int

i += 1

}

if (isNegative) total else -total

}

/\*\*

\* @param long must be non-negative

\*/

private[this] def stringSize(long: Long): Int = {

var p = 10

var i = 1

while (i < MaxByteArrayLength) {

if (long < p) return i

p \*= 10

i += 1

}

MaxByteArrayLength

}

}