#include <twml/BlockFormatReader.h>

#include <cstring>

#include <stdexcept>

#define OFFSET\_CHUNK (32768)

#define RECORDS\_PER\_BLOCK (100)

#define WIRE\_TYPE\_VARINT (0)

#define WIRE\_TYPE\_64BIT (1)

#define WIRE\_TYPE\_LENGTH\_PREFIXED (2)

/\*

This was all extracted from the ancient elephant bird scrolls

https://github.com/twitter/elephant-bird/blob/master/core/src/main/java/com/twitter/elephantbird/mapreduce/io/BinaryBlockReader.java

\*/

#define MARKER\_SIZE (16)

static uint8\_t \_marker[MARKER\_SIZE] = {

0x29, 0xd8, 0xd5, 0x06, 0x58, 0xcd, 0x4c, 0x29,

0xb2, 0xbc, 0x57, 0x99, 0x21, 0x71, 0xbd, 0xff

};

namespace twml {

BlockFormatReader::BlockFormatReader():

record\_size\_(0), block\_pos\_(0), block\_end\_(0) {

memset(classname\_, 0, sizeof(classname\_));

}

bool BlockFormatReader::next() {

record\_size\_ = read\_one\_record\_size();

if (record\_size\_ < 0) {

record\_size\_ = 0;

return false;

}

return true;

}

int BlockFormatReader::read\_int() {

uint8\_t buff[4];

if (read\_bytes(buff, 1, 4) != 4)

return -1;

return static\_cast<int>(buff[0])

| (static\_cast<int>(buff[1] << 8))

| (static\_cast<int>(buff[2] << 16))

| (static\_cast<int>(buff[3] << 24));

}

int BlockFormatReader::consume\_marker(int scan) {

uint8\_t buff[MARKER\_SIZE];

if (read\_bytes(buff, 1, MARKER\_SIZE) != MARKER\_SIZE)

return 0;

while (memcmp(buff, \_marker, MARKER\_SIZE) != 0) {

if (!scan) return 0;

memmove(buff, buff + 1, MARKER\_SIZE - 1);

if (read\_bytes(buff + MARKER\_SIZE - 1, 1, 1) != 1)

return 0;

}

return 1;

}

int BlockFormatReader::unpack\_varint\_i32() {

int value = 0;

for (int i = 0; i < 10; i++) {

uint8\_t x;

if (read\_bytes(&x, 1, 1) != 1)

return -1;

block\_pos\_++;

value |= (static\_cast<int>(x & 0x7F)) << (i \* 7);

if ((x & 0x80) == 0) break;

}

return value;

}

int BlockFormatReader::unpack\_tag\_and\_wiretype(uint32\_t \*tag, uint32\_t \*wiretype) {

uint8\_t x;

if (read\_bytes(&x, 1, 1) != 1)

return -1;

block\_pos\_++;

\*tag = (x & 0x7f) >> 3;

\*wiretype = x & 7;

if ((x & 0x80) == 0)

return 0;

return -1;

}

int BlockFormatReader::unpack\_string(char \*out, uint64\_t max\_out\_len) {

int len = unpack\_varint\_i32();

if (len < 0) return -1;

uint64\_t slen = len;

if (slen + 1 > max\_out\_len) return -1;

uint64\_t n = read\_bytes(out, 1, slen);

if (n != slen) return -1;

block\_pos\_ += n;

out[n] = 0;

return 0;

}

int BlockFormatReader::read\_one\_record\_size() {

for (int i = 0; i < 2; i++) {

if (block\_end\_ == 0) {

while (consume\_marker(1)) {

int block\_size = read\_int();

if (block\_size > 0) {

block\_pos\_ = 0;

block\_end\_ = block\_size;

uint32\_t tag, wiretype;

if (unpack\_tag\_and\_wiretype(&tag, &wiretype))

throw std::invalid\_argument("unsupported tag and wiretype");

if (tag != 1 && wiretype != WIRE\_TYPE\_VARINT)

throw std::invalid\_argument("unexpected tag and wiretype");

int version = unpack\_varint\_i32();

if (version != 1)

throw std::invalid\_argument("unsupported version");

if (unpack\_tag\_and\_wiretype(&tag, &wiretype))

throw std::invalid\_argument("unsupported tag and wiretype");

if (tag != 2 && wiretype != WIRE\_TYPE\_LENGTH\_PREFIXED)

throw std::invalid\_argument("unexpected tag and wiretype");

if (unpack\_string(classname\_, sizeof(classname\_)-1))

throw std::invalid\_argument("unsupported class name");

break;

}

}

}

if (block\_pos\_ < block\_end\_) {

uint32\_t tag, wiretype;

if (unpack\_tag\_and\_wiretype(&tag, &wiretype))

throw std::invalid\_argument("unsupported tag and wiretype");

if (tag != 3 && wiretype != WIRE\_TYPE\_LENGTH\_PREFIXED)

throw std::invalid\_argument("unexpected tag and wiretype");

int record\_size = unpack\_varint\_i32();

block\_pos\_ += record\_size;

return record\_size;

} else {

block\_end\_ = 0;

}

}

return -1;

}

} // namespace twml