import io

import logging

import subprocess

from threading import Lock

"""

This module provides a binary data record reader for EventBus data.

It starts a EventBus subscriber in a separate process to receive EventBus streaming data.

The subscriber is supposed to outputs received data through PIPE to this module.

This module parses input and output binary data record to serve as a record reader.

"""

class BinaryRecordReader(object):

def initialize(self):

pass

def read(self):

"""Read raw bytes for one record

"""

raise NotImplementedError

def close(self):

pass

class ReadableWrapper(object):

def \_\_init\_\_(self, internal):

self.internal = internal

def \_\_getattr\_\_(self, name):

return getattr(self.internal, name)

def readable(self):

return True

class EventBusPipedBinaryRecordReader(BinaryRecordReader):

JAVA = '/usr/lib/jvm/java-11-twitter/bin/java'

RECORD\_SEPARATOR\_HEX = [

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

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

]

RECORD\_SEPARATOR = ''.join([chr(i) for i in RECORD\_SEPARATOR\_HEX])

RECORD\_SEPARATOR\_LENGTH = len(RECORD\_SEPARATOR)

CHUNK\_SIZE = 8192

def \_\_init\_\_(self, jar\_file, num\_eb\_threads, subscriber\_id,

filter\_str=None, buffer\_size=32768, debug=False):

self.jar\_file = jar\_file

self.num\_eb\_threads = num\_eb\_threads

self.subscriber\_id = subscriber\_id

self.filter\_str = filter\_str if filter\_str else '""'

self.buffer\_size = buffer\_size

self.lock = Lock()

self.\_pipe = None

self.\_buffered\_reader = None

self.\_bytes\_buffer = None

self.debug = debug

def initialize(self):

if not self.\_pipe:

self.\_pipe = subprocess.Popen(

[

self.JAVA, '-jar', self.jar\_file,

'-subscriberId', self.subscriber\_id,

'-numThreads', str(self.num\_eb\_threads),

'-dataFilter', self.filter\_str,

'-debug' if self.debug else ''

],

stdout=subprocess.PIPE

)

self.\_buffered\_reader = io.BufferedReader(

ReadableWrapper(self.\_pipe.stdout), self.buffer\_size)

self.\_bytes\_buffer = io.BytesIO()

else:

logging.warning('Already initialized')

def \_find\_next\_record(self):

tail = ['']

while True:

chunk = tail[0] + self.\_buffered\_reader.read(self.CHUNK\_SIZE)

index = chunk.find(self.RECORD\_SEPARATOR)

if index < 0:

self.\_bytes\_buffer.write(chunk[:-self.RECORD\_SEPARATOR\_LENGTH])

tail[0] = chunk[-self.RECORD\_SEPARATOR\_LENGTH:]

else:

self.\_bytes\_buffer.write(chunk[:index])

return chunk[(index + self.RECORD\_SEPARATOR\_LENGTH):]

def \_read(self):

with self.lock:

remaining = self.\_find\_next\_record()

record = self.\_bytes\_buffer.getvalue()

# clean up buffer

self.\_bytes\_buffer.close()

self.\_bytes\_buffer = io.BytesIO()

self.\_bytes\_buffer.write(remaining)

return record

def read(self):

while True:

try:

return self.\_read()

except Exception as e:

logging.error("Error reading bytes for next record: {}".format(e))

if self.debug:

raise

def close(self):

try:

self.\_bytes\_buffer.close()

self.\_buffered\_reader.close()

self.\_pipe.terminate()

except Exception as e:

logging.error("Error closing reader: {}".format(e))