package com.twitter.search.earlybird.common;

import org.apache.commons.codec.binary.Base64;

import org.apache.thrift.TException;

import org.apache.thrift.TSerializer;

import org.apache.thrift.protocol.TBinaryProtocol;

import org.slf4j.Logger;

import com.twitter.search.earlybird.thrift.EarlybirdRequest;

import com.twitter.search.earlybird.thrift.EarlybirdResponse;

public final class Base64RequestResponseForLogging {

private static final Logger GENERAL\_LOG = org.slf4j.LoggerFactory.getLogger(

Base64RequestResponseForLogging.class);

private static final Logger FAILED\_REQUEST\_LOG = org.slf4j.LoggerFactory.getLogger(

Base64RequestResponseForLogging.class.getName() + ".FailedRequests");

private static final Logger RANDOM\_REQUEST\_LOG = org.slf4j.LoggerFactory.getLogger(

Base64RequestResponseForLogging.class.getName() + ".RandomRequests");

private static final Logger SLOW\_REQUEST\_LOG = org.slf4j.LoggerFactory.getLogger(

Base64RequestResponseForLogging.class.getName() + ".SlowRequests");

private enum LogType {

FAILED,

RANDOM,

SLOW,

};

private final LogType logtype;

private final String logLine;

private final EarlybirdRequest request;

private final EarlybirdResponse response;

private final Base64 base64 = new Base64();

// TSerializer is not threadsafe, so create a new one for each request

private final TSerializer serializer = new TSerializer(new TBinaryProtocol.Factory());

private Base64RequestResponseForLogging(

LogType logType, String logLine, EarlybirdRequest request, EarlybirdResponse response) {

this.logtype = logType;

this.logLine = logLine;

this.request = request;

this.response = response;

}

public static Base64RequestResponseForLogging randomRequest(

String logLine, EarlybirdRequest request, EarlybirdResponse response) {

return new Base64RequestResponseForLogging(LogType.RANDOM, logLine, request, response);

}

public static Base64RequestResponseForLogging failedRequest(

String logLine, EarlybirdRequest request, EarlybirdResponse response) {

return new Base64RequestResponseForLogging(LogType.FAILED, logLine, request, response);

}

public static Base64RequestResponseForLogging slowRequest(

String logLine, EarlybirdRequest request, EarlybirdResponse response) {

return new Base64RequestResponseForLogging(LogType.SLOW, logLine, request, response);

}

private String asBase64(EarlybirdRequest clearedRequest) {

try {

// The purpose of this log is to make it easy to re-issue requests in formz to reproduce

// issues. If queries are re-issued as is they will be treated as late-arriving queries and

// dropped due to the clientRequestTimeMs being set to the original query time. For ease of

// use purposes we clear clientRequestTimeMs and log it out separately for the rare case it

// is needed.

clearedRequest.unsetClientRequestTimeMs();

return base64.encodeToString(serializer.serialize(clearedRequest));

} catch (TException e) {

GENERAL\_LOG.error("Failed to serialize request for logging.", e);

return "failed\_to\_serialize";

}

}

private String asBase64(EarlybirdResponse earlybirdResponse) {

try {

return base64.encodeToString(serializer.serialize(earlybirdResponse));

} catch (TException e) {

GENERAL\_LOG.error("Failed to serialize response for logging.", e);

return "failed\_to\_serialize";

}

}

private String getFormattedMessage() {

String base64Request = asBase64(

EarlybirdRequestUtil.copyAndClearUnnecessaryValuesForLogging(request));

String base64Response = asBase64(response);

return logLine + ", clientRequestTimeMs: " + request.getClientRequestTimeMs()

+ ", " + base64Request + ", " + base64Response;

}

/\*\*

\* Logs the Base64-encoded request and response to the success or failure log.

\*/

public void log() {

// Do the serializing/concatting this way so it happens on the background thread for

// async logging

Object logObject = new Object() {

@Override

public String toString() {

return getFormattedMessage();

}

};

switch (logtype) {

case FAILED:

FAILED\_REQUEST\_LOG.info("{}", logObject);

break;

case RANDOM:

RANDOM\_REQUEST\_LOG.info("{}", logObject);

break;

case SLOW:

SLOW\_REQUEST\_LOG.info("{}", logObject);

break;

default:

// Not logging anything for other log types.

break;

}

}

}