package com.twitter.search.earlybird.common;

import java.util.EnumMap;

import java.util.Map;

import scala.Option;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.collect.Maps;

import com.twitter.context.TwitterContext;

import com.twitter.context.thriftscala.Viewer;

import com.twitter.decider.Decider;

import com.twitter.finagle.thrift.ClientId;

import com.twitter.finagle.thrift.ClientId$;

import com.twitter.search.TwitterContextPermit;

import com.twitter.search.common.constants.thriftjava.ThriftQuerySource;

import com.twitter.search.common.decider.DeciderUtil;

import com.twitter.search.common.logging.RPCLogger;

import com.twitter.search.common.metrics.FailureRatioCounter;

import com.twitter.search.common.metrics.Timer;

import com.twitter.search.common.util.earlybird.TermStatisticsUtil;

import com.twitter.search.common.util.earlybird.ThriftSearchResultUtil;

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

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

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

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

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

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

import static com.twitter.search.common.util.earlybird.EarlybirdResponseUtil

.responseConsideredFailed;

public class EarlybirdRequestLogger extends RPCLogger {

protected enum ExtraFields {

QUERY\_MAX\_HITS\_TO\_PROCESS,

COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS,

RELEVANCE\_OPTIONS\_MAX\_HITS\_TO\_PROCESS,

NUM\_HITS\_PROCESSED,

QUERY\_COST,

CPU\_TOTAL,

QUERY\_SOURCE,

CLIENT\_ID,

FINAGLE\_CLIENT\_ID

}

protected enum ShardOnlyExtraFields {

NUM\_SEARCHED\_SEGMENTS,

SCORING\_TIME\_NANOS

}

protected enum RootOnlyExtraFields {

CACHING\_ALLOWED,

DEBUG\_MODE,

CACHE\_HIT,

USER\_AGENT,

// See JIRA APPSEC-2303 for IP addresses logging

}

private static final String LOG\_FULL\_REQUEST\_DETAILS\_ON\_ERROR\_DECIDER\_KEY =

"log\_full\_request\_details\_on\_error";

private static final String LOG\_FULL\_REQUEST\_DETAILS\_RANDOM\_FRACTION\_DECIDER\_KEY =

"log\_full\_request\_details\_random\_fraction";

private static final String LOG\_FULL\_SLOW\_REQUEST\_DETAILS\_RANDOM\_FRACTION\_DECIDER\_KEY =

"log\_full\_slow\_request\_details\_random\_fraction";

private static final String SLOW\_REQUEST\_LATENCY\_THRESHOLD\_MS\_DECIDER\_KEY =

"slow\_request\_latency\_threshold\_ms";

private final Decider decider;

private final boolean enableLogUnknownClientRequests;

private static final Map<ThriftQuerySource, FailureRatioCounter>

FAILURE\_RATIO\_COUNTER\_BY\_QUERY\_SOURCE = preBuildFailureRatioCounters();

private static final FailureRatioCounter NO\_QUERY\_SOURCE\_FAILURE\_RATIO\_COUNTER =

new FailureRatioCounter("earlybird\_logger", "query\_source", "not\_set");

static EarlybirdRequestLogger buildForRoot(

String loggerName, int latencyWarnThreshold, Decider decider) {

return new EarlybirdRequestLogger(loggerName, latencyWarnThreshold,

decider, true, RPCLogger.Fields.values(), ExtraFields.values(),

RootOnlyExtraFields.values());

}

static EarlybirdRequestLogger buildForShard(

String loggerName, int latencyWarnThreshold, Decider decider) {

return new EarlybirdRequestLogger(loggerName, latencyWarnThreshold,

decider, false, RPCLogger.Fields.values(), ExtraFields.values(),

ShardOnlyExtraFields.values());

}

@VisibleForTesting

EarlybirdRequestLogger(String loggerName, int latencyWarnThreshold, Decider decider) {

this(loggerName, latencyWarnThreshold, decider, false, RPCLogger.Fields.values(),

ExtraFields.values(), RootOnlyExtraFields.values(), ShardOnlyExtraFields.values());

}

private EarlybirdRequestLogger(String loggerName, int latencyWarnThreshold, Decider decider,

boolean enableLogUnknownClientRequests, Enum[]... fieldEnums) {

super(loggerName, fieldEnums);

this.decider = decider;

this.enableLogUnknownClientRequests = enableLogUnknownClientRequests;

setLatencyWarnThreshold(latencyWarnThreshold);

}

/\*\*

\* Logs the given earlybird request and response.

\*

\* @param request The earlybird request.

\* @param response The earlybird response.

\* @param timer The time it took to process this request.

\*/

public void logRequest(EarlybirdRequest request, EarlybirdResponse response, Timer timer) {

try {

LogEntry entry = newLogEntry();

setRequestLogEntries(entry, request);

setResponseLogEntries(entry, response);

if (timer != null) {

entry.setField(ExtraFields.CPU\_TOTAL, Long.toString(timer.getElapsedCpuTotal()));

}

boolean wasError = response != null && responseConsideredFailed(response.getResponseCode());

long responseTime = response != null ? response.getResponseTime() : 0L;

String logLine = writeLogLine(entry, responseTime, wasError);

// This code path is called for pre/post logging

// Prevent same request showing up twice by only logging on post logging

if (response != null && DeciderUtil.isAvailableForRandomRecipient(

decider, LOG\_FULL\_REQUEST\_DETAILS\_RANDOM\_FRACTION\_DECIDER\_KEY)) {

Base64RequestResponseForLogging.randomRequest(logLine, request, response).log();

}

// Unknown client request logging only applies to pre-logging.

if (enableLogUnknownClientRequests && response == null) {

UnknownClientRequestForLogging unknownClientRequestLogger =

UnknownClientRequestForLogging.unknownClientRequest(logLine, request);

if (unknownClientRequestLogger != null) {

unknownClientRequestLogger.log();

}

}

if (wasError

&& DeciderUtil.isAvailableForRandomRecipient(

decider, LOG\_FULL\_REQUEST\_DETAILS\_ON\_ERROR\_DECIDER\_KEY)) {

new RequestResponseForLogging(request, response).logFailedRequest();

Base64RequestResponseForLogging.failedRequest(logLine, request, response).log();

}

boolean wasSlow = response != null

&& responseTime >= DeciderUtil.getAvailability(

decider, SLOW\_REQUEST\_LATENCY\_THRESHOLD\_MS\_DECIDER\_KEY);

if (wasSlow

&& DeciderUtil.isAvailableForRandomRecipient(

decider, LOG\_FULL\_SLOW\_REQUEST\_DETAILS\_RANDOM\_FRACTION\_DECIDER\_KEY)) {

Base64RequestResponseForLogging.slowRequest(logLine, request, response).log();

}

FailureRatioCounter failureRatioCounter =

FAILURE\_RATIO\_COUNTER\_BY\_QUERY\_SOURCE.get(request.getQuerySource());

if (failureRatioCounter != null) {

failureRatioCounter.requestFinished(!wasError);

} else {

NO\_QUERY\_SOURCE\_FAILURE\_RATIO\_COUNTER.requestFinished(!wasError);

}

} catch (Exception e) {

LOG.error("Exception building log entry ", e);

}

}

private void setRequestLogEntries(LogEntry entry, EarlybirdRequest request) {

entry.setField(Fields.CLIENT\_HOST, request.getClientHost());

entry.setField(Fields.CLIENT\_REQUEST\_ID, request.getClientRequestID());

entry.setField(Fields.REQUEST\_TYPE, requestTypeForLog(request));

if (request.isSetSearchQuery()) {

ThriftSearchQuery searchQuery = request.getSearchQuery();

entry.setField(Fields.QUERY, searchQuery.getSerializedQuery());

if (searchQuery.isSetMaxHitsToProcess()) {

entry.setField(ExtraFields.QUERY\_MAX\_HITS\_TO\_PROCESS,

Integer.toString(searchQuery.getMaxHitsToProcess()));

}

if (searchQuery.isSetCollectorParams()

&& searchQuery.getCollectorParams().isSetTerminationParams()

&& searchQuery.getCollectorParams().getTerminationParams().isSetMaxHitsToProcess()) {

entry.setField(ExtraFields.COLLECTOR\_PARAMS\_MAX\_HITS\_TO\_PROCESS,

Integer.toString(searchQuery.getCollectorParams().getTerminationParams()

.getMaxHitsToProcess()));

}

if (searchQuery.isSetRelevanceOptions()

&& searchQuery.getRelevanceOptions().isSetMaxHitsToProcess()) {

entry.setField(ExtraFields.RELEVANCE\_OPTIONS\_MAX\_HITS\_TO\_PROCESS,

Integer.toString(searchQuery.getRelevanceOptions().getMaxHitsToProcess()));

}

}

entry.setField(Fields.NUM\_REQUESTED, Integer.toString(numRequestedForLog(request)));

if (request.isSetQuerySource()) {

entry.setField(ExtraFields.QUERY\_SOURCE, request.getQuerySource().name());

}

if (request.isSetClientId()) {

entry.setField(ExtraFields.CLIENT\_ID, request.getClientId());

}

entry.setField(RootOnlyExtraFields.CACHING\_ALLOWED,

Boolean.toString(EarlybirdRequestUtil.isCachingAllowed(request)));

entry.setField(RootOnlyExtraFields.DEBUG\_MODE, Byte.toString(request.getDebugMode()));

Option<ClientId> clientIdOption = ClientId$.MODULE$.current();

if (clientIdOption.isDefined()) {

entry.setField(ExtraFields.FINAGLE\_CLIENT\_ID, clientIdOption.get().name());

}

setLogEntriesFromTwitterContext(entry);

}

@VisibleForTesting

Option<Viewer> getTwitterContext() {

return TwitterContext.acquire(TwitterContextPermit.get()).apply();

}

private void setLogEntriesFromTwitterContext(LogEntry entry) {

Option<Viewer> viewerOption = getTwitterContext();

if (viewerOption.nonEmpty()) {

Viewer viewer = viewerOption.get();

if (viewer.userAgent().nonEmpty()) {

String userAgent = viewer.userAgent().get();

// we only replace the comma in the user-agent with %2C to make it easily parseable,

// specially with command line tools like cut/sed/awk

userAgent = userAgent.replace(",", "%2C");

entry.setField(RootOnlyExtraFields.USER\_AGENT, userAgent);

}

}

}

private void setResponseLogEntries(LogEntry entry, EarlybirdResponse response) {

if (response != null) {

entry.setField(Fields.NUM\_RETURNED, Integer.toString(numResultsForLog(response)));

entry.setField(Fields.RESPONSE\_CODE, String.valueOf(response.getResponseCode()));

entry.setField(Fields.RESPONSE\_TIME\_MICROS, Long.toString(response.getResponseTimeMicros()));

if (response.isSetSearchResults()) {

entry.setField(ExtraFields.NUM\_HITS\_PROCESSED,

Integer.toString(response.getSearchResults().getNumHitsProcessed()));

entry.setField(ExtraFields.QUERY\_COST,

Double.toString(response.getSearchResults().getQueryCost()));

if (response.getSearchResults().isSetScoringTimeNanos()) {

entry.setField(ShardOnlyExtraFields.SCORING\_TIME\_NANOS,

Long.toString(response.getSearchResults().getScoringTimeNanos()));

}

}

if (response.isSetCacheHit()) {

entry.setField(RootOnlyExtraFields.CACHE\_HIT, String.valueOf(response.isCacheHit()));

}

if (response.isSetNumSearchedSegments()) {

entry.setField(ShardOnlyExtraFields.NUM\_SEARCHED\_SEGMENTS,

Integer.toString(response.getNumSearchedSegments()));

}

}

}

private static int numRequestedForLog(EarlybirdRequest request) {

int num = 0;

if (request.isSetFacetRequest() && request.getFacetRequest().isSetFacetFields()) {

for (ThriftFacetFieldRequest field : request.getFacetRequest().getFacetFields()) {

num += field.getNumResults();

}

} else if (request.isSetTermStatisticsRequest()) {

num = request.getTermStatisticsRequest().getTermRequestsSize();

} else if (request.isSetSearchQuery()) {

num = request.getSearchQuery().isSetCollectorParams()

? request.getSearchQuery().getCollectorParams().getNumResultsToReturn() : 0;

if (request.getSearchQuery().getSearchStatusIdsSize() > 0) {

num = Math.max(num, request.getSearchQuery().getSearchStatusIdsSize());

}

}

return num;

}

/\*\*

\* Returns the number of results in the given response. If the response is a term stats response,

\* then the returned value will be the number of term results. If the response is a facet

\* response, then the returned value will be the number of facet results. Otherwise, the returned

\* value will be the number of search results.

\*/

public static int numResultsForLog(EarlybirdResponse response) {

if (response == null) {

return 0;

} else if (response.isSetFacetResults()) {

return ThriftSearchResultUtil.numFacetResults(response.getFacetResults());

} else if (response.isSetTermStatisticsResults()) {

return response.getTermStatisticsResults().getTermResultsSize();

} else {

return ThriftSearchResultUtil.numResults(response.getSearchResults());

}

}

private static String requestTypeForLog(EarlybirdRequest request) {

StringBuilder requestType = new StringBuilder(64);

if (request.isSetFacetRequest()) {

requestType.append("FACETS");

int numFields = request.getFacetRequest().getFacetFieldsSize();

if (numFields > 0) {

// For 1 or 2 fields, just put them in the request type. For more, just log the number.

if (numFields <= 2) {

for (ThriftFacetFieldRequest field : request.getFacetRequest().getFacetFields()) {

requestType.append(":").append(field.getFieldName().toUpperCase());

}

} else {

requestType.append(":MULTI-").append(numFields);

}

}

} else if (request.isSetTermStatisticsRequest()) {

ThriftTermStatisticsRequest termStatsRequest = request.getTermStatisticsRequest();

requestType.append("TERMSTATS-")

.append(termStatsRequest.getTermRequestsSize());

ThriftHistogramSettings histoSettings = termStatsRequest.getHistogramSettings();

if (histoSettings != null) {

String binSizeVal = String.valueOf(TermStatisticsUtil.determineBinSize(histoSettings));

String numBinsVal = String.valueOf(histoSettings.getNumBins());

requestType.append(":NUMBINS-").append(numBinsVal).append(":BINSIZE-").append(binSizeVal);

}

} else if (request.isSetSearchQuery()) {

requestType.append("SEARCH:");

requestType.append(request.getSearchQuery().getRankingMode().name());

// Denote when a from user id is present.

if (request.getSearchQuery().isSetFromUserIDFilter64()) {

requestType.append(":NETWORK-")

.append(request.getSearchQuery().getFromUserIDFilter64Size());

}

// Denote when required status ids are present.

if (request.getSearchQuery().getSearchStatusIdsSize() > 0) {

requestType.append(":IDS-").append(request.getSearchQuery().getSearchStatusIdsSize());

}

}

return requestType.toString();

}

private static Map<ThriftQuerySource, FailureRatioCounter> preBuildFailureRatioCounters() {

Map<ThriftQuerySource, FailureRatioCounter> counterByQuerySource =

new EnumMap<>(ThriftQuerySource.class);

for (ThriftQuerySource thriftQuerySource : ThriftQuerySource.values()) {

FailureRatioCounter counter = new FailureRatioCounter("earlybird\_logger", "query\_source",

thriftQuerySource.toString());

counterByQuerySource.put(thriftQuerySource, counter);

}

return Maps.immutableEnumMap(counterByQuerySource);

}

}