package com.twitter.search.earlybird;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import java.util.Optional;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicBoolean;

import com.google.common.collect.Lists;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.BuildInfo;

import com.twitter.search.earlybird.partition.SearchIndexingMetricSet;

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

import com.twitter.util.Duration;

/\*\*

\* High level status of an Earlybird server. SEARCH-28016

\*/

public final class EarlybirdStatus {

private static final Logger LOG = LoggerFactory.getLogger(EarlybirdStatus.class);

private static final String BUILD\_SHA = getBuildShaFromVars();

protected static long startTime;

protected static EarlybirdStatusCode statusCode;

protected static String statusMessage;

protected static final AtomicBoolean THRIFT\_PORT\_OPEN = new AtomicBoolean(false);

protected static final AtomicBoolean WARMUP\_THRIFT\_PORT\_OPEN = new AtomicBoolean(false);

protected static final AtomicBoolean THRIFT\_SERVICE\_STARTED = new AtomicBoolean(false);

private static final List<EarlybirdEvent> EARLYBIRD\_SERVER\_EVENTS = Lists.newArrayList();

private static class EarlybirdEvent {

private final String eventName;

private final long timestampMillis;

private final long timeSinceServerStartMillis;

private final long durationMillis;

public EarlybirdEvent(String eventName, long timestampMillis) {

this(eventName, timestampMillis, -1);

}

public EarlybirdEvent(

String eventName,

long timestampMillis,

long eventDurationMillis) {

this.eventName = eventName;

this.timestampMillis = timestampMillis;

this.timeSinceServerStartMillis = timestampMillis - startTime;

this.durationMillis = eventDurationMillis;

}

public String getEventLogString() {

String result = String.format(

"%s %s",

new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS").format(new Date(timestampMillis)),

eventName);

if (durationMillis > 0) {

result += String.format(

", took: %s", Duration.apply(durationMillis, TimeUnit.MILLISECONDS).toString());

}

result += String.format(

", time since server start: %s",

Duration.apply(timeSinceServerStartMillis, TimeUnit.MILLISECONDS).toString()

);

return result;

}

}

private EarlybirdStatus() {

}

public static synchronized void setStartTime(long time) {

startTime = time;

LOG.info("startTime set to " + time);

}

public static synchronized void setStatus(EarlybirdStatusCode code) {

setStatus(code, null);

}

public static synchronized void setStatus(EarlybirdStatusCode code, String message) {

statusCode = code;

statusMessage = message;

LOG.info("status set to " + code + (message != null ? " with message " + message : ""));

}

public static synchronized long getStartTime() {

return startTime;

}

public static synchronized boolean isStarting() {

return statusCode == EarlybirdStatusCode.STARTING;

}

public static synchronized boolean hasStarted() {

return statusCode == EarlybirdStatusCode.CURRENT;

}

public static boolean isThriftServiceStarted() {

return THRIFT\_SERVICE\_STARTED.get();

}

public static synchronized EarlybirdStatusCode getStatusCode() {

return statusCode;

}

public static synchronized String getStatusMessage() {

return (statusMessage == null ? "" : statusMessage + ", ")

+ "warmup thrift port is " + (WARMUP\_THRIFT\_PORT\_OPEN.get() ? "OPEN" : "CLOSED")

+ ", production thrift port is " + (THRIFT\_PORT\_OPEN.get() ? "OPEN" : "CLOSED");

}

public static synchronized void recordEarlybirdEvent(String eventName) {

long timeMillis = System.currentTimeMillis();

EARLYBIRD\_SERVER\_EVENTS.add(new EarlybirdEvent(eventName, timeMillis));

}

private static String getBeginEventMessage(String eventName) {

return "[Begin Event] " + eventName;

}

private static String getEndEventMessage(String eventName) {

return "[ End Event ] " + eventName;

}

/\*\*

\* Records the beginning of the given event.

\*

\* @param eventName The event name.

\* @param startupMetric The metric that will be used to keep track of the time for this event.

\*/

public static synchronized void beginEvent(String eventName,

SearchIndexingMetricSet.StartupMetric startupMetric) {

long timeMillis = System.currentTimeMillis();

String eventMessage = getBeginEventMessage(eventName);

LOG.info(eventMessage);

EARLYBIRD\_SERVER\_EVENTS.add(new EarlybirdEvent(eventMessage, timeMillis));

startupMetric.begin();

}

/\*\*

\* Records the end of the given event.

\*

\* @param eventName The event name.

\* @param startupMetric The metric used to keep track of the time for this event.

\*/

public static synchronized void endEvent(String eventName,

SearchIndexingMetricSet.StartupMetric startupMetric) {

long timeMillis = System.currentTimeMillis();

String beginEventMessage = getBeginEventMessage(eventName);

Optional<EarlybirdEvent> beginEventOpt = EARLYBIRD\_SERVER\_EVENTS.stream()

.filter(event -> event.eventName.equals(beginEventMessage))

.findFirst();

String eventMessage = getEndEventMessage(eventName);

LOG.info(eventMessage);

EarlybirdEvent endEvent = new EarlybirdEvent(

eventMessage,

timeMillis,

beginEventOpt.map(e -> timeMillis - e.timestampMillis).orElse(-1L));

EARLYBIRD\_SERVER\_EVENTS.add(endEvent);

startupMetric.end(endEvent.durationMillis);

}

public static synchronized void clearAllEvents() {

EARLYBIRD\_SERVER\_EVENTS.clear();

}

public static String getBuildSha() {

return BUILD\_SHA;

}

/\*\*

\* Returns the list of all earlybird events that happened since the server started.

\*/

public static synchronized Iterable<String> getEarlybirdEvents() {

List<String> eventLog = Lists.newArrayListWithCapacity(EARLYBIRD\_SERVER\_EVENTS.size());

for (EarlybirdEvent event : EARLYBIRD\_SERVER\_EVENTS) {

eventLog.add(event.getEventLogString());

}

return eventLog;

}

private static String getBuildShaFromVars() {

BuildInfo buildInfo = new BuildInfo();

String buildSha = buildInfo.getProperties().getProperty(BuildInfo.Key.GIT\_REVISION.value);

if (buildSha != null) {

return buildSha;

} else {

return "UNKNOWN";

}

}

}