package com.twitter.search.earlybird.util;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.ScheduledFuture;

import java.util.concurrent.TimeUnit;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.util.Clock;

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

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

import com.twitter.search.earlybird.exception.CriticalExceptionHandler;

/\*\*

\* Base class for classes that run periodic tasks.

\*/

public abstract class ScheduledExecutorManager {

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

private static final long SHUTDOWN\_WAIT\_INTERVAL\_SEC = 30;

public static final String SCHEDULED\_EXECUTOR\_TASK\_PREFIX = "scheduled\_executor\_task\_";

private final String name;

private final ScheduledExecutorService executor;

private final ShutdownWaitTimeParams shutdownWaitTimeParams;

private final SearchCounter iterationCounter;

private final SearchStatsReceiver searchStatsReceiver;

protected final CriticalExceptionHandler criticalExceptionHandler;

private final Clock clock;

protected boolean shouldLog = true;

public ScheduledExecutorManager(

ScheduledExecutorService executor,

ShutdownWaitTimeParams shutdownWaitTimeParams,

SearchStatsReceiver searchStatsReceiver,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

this(executor, shutdownWaitTimeParams, searchStatsReceiver, null,

criticalExceptionHandler, clock);

}

ScheduledExecutorManager(

ScheduledExecutorService executor,

ShutdownWaitTimeParams shutdownWaitTimeParams,

SearchStatsReceiver searchStatsReceiver,

SearchCounter iterationCounter,

CriticalExceptionHandler criticalExceptionHandler,

Clock clock) {

this.name = getClass().getSimpleName();

this.executor = executor;

this.criticalExceptionHandler = criticalExceptionHandler;

this.shutdownWaitTimeParams = shutdownWaitTimeParams;

if (iterationCounter != null) {

this.iterationCounter = iterationCounter;

} else {

this.iterationCounter = searchStatsReceiver.getCounter(SCHEDULED\_EXECUTOR\_TASK\_PREFIX + name);

}

this.searchStatsReceiver = searchStatsReceiver;

this.clock = clock;

}

/\*\*

\* Schedule a task.

\*/

protected final ScheduledFuture scheduleNewTask(

ScheduledExecutorTask task,

PeriodicActionParams periodicActionParams) {

long interval = periodicActionParams.getIntervalDuration();

TimeUnit timeUnit = periodicActionParams.getIntervalUnit();

long initialDelay = periodicActionParams.getInitialDelayDuration();

if (interval <= 0) {

String message = String.format(

"Not scheduling manager %s for wrong interval %d %s", name, interval, timeUnit);

LOG.error(message);

throw new UnsupportedOperationException(message);

}

if (shouldLog) {

LOG.info("Scheduling to run {} every {} {} with {}", name, interval, timeUnit,

periodicActionParams.getDelayType());

}

final ScheduledFuture scheduledFuture;

if (periodicActionParams.isFixedDelay()) {

scheduledFuture = executor.scheduleWithFixedDelay(task, initialDelay, interval, timeUnit);

} else {

scheduledFuture = executor.scheduleAtFixedRate(task, initialDelay, interval, timeUnit);

}

return scheduledFuture;

}

/\*\*

\* Shutdown everything that's running with the executor.

\*/

public boolean shutdown() throws InterruptedException {

LOG.info("Start shutting down {}.", name);

executor.shutdownNow();

boolean terminated = false;

long waitSeconds = shutdownWaitTimeParams.getWaitUnit().toSeconds(

shutdownWaitTimeParams.getWaitDuration()

);

if (waitSeconds == 0) {

LOG.info("Not waiting at all for {}, wait time is set to zero.", name);

} else {

while (!terminated && waitSeconds > 0) {

long waitTime = Math.min(waitSeconds, SHUTDOWN\_WAIT\_INTERVAL\_SEC);

terminated = executor.awaitTermination(waitTime, TimeUnit.SECONDS);

waitSeconds -= waitTime;

if (!terminated) {

LOG.info("Still shutting down {} ...", name);

}

}

}

LOG.info("Done shutting down {}, terminated: {}", name, terminated);

shutdownComponent();

return terminated;

}

protected ScheduledExecutorService getExecutor() {

return executor;

}

public final String getName() {

return name;

}

public SearchCounter getIterationCounter() {

return iterationCounter;

}

protected final SearchStatsReceiver getSearchStatsReceiver() {

return searchStatsReceiver;

}

// Override if you need to shutdown additional services.

protected void shutdownComponent() {

}

}