package com.twitter.product\_mixer.core.util

import com.twitter.concurrent.NamedPoolThreadFactory

import com.twitter.finagle.stats.StatsReceiver

import com.twitter.util.Duration

import com.twitter.util.FuturePool

import java.util.concurrent.ArrayBlockingQueue

import java.util.concurrent.BlockingQueue

import java.util.concurrent.LinkedBlockingQueue

import java.util.concurrent.ThreadPoolExecutor

import java.util.concurrent.TimeUnit

/\*\*

\* Utility for making [[FuturePool]] with finite thread counts and different work queue options

\* while also monitoring the size of the work queue that's used.

\*

\* This only handles the cases where the number of threads are bounded.

\* For unbounded numbers of threads in a [[FuturePool]] use [[FuturePool.interruptibleUnboundedPool]] instead.

\*/

object FuturePools {

/\*\*

\* Makes a [[FuturePool]] with a fixed number of threads and a work queue

\* with a maximum size of `maxWorkQueueSize`.

\*

\* @note the [[FuturePool]] returns a failed [[com.twitter.util.Future]]s containing

\* [[java.util.concurrent.RejectedExecutionException]] when trying to enqueue

\* work when the work queue is full.

\*/

def boundedFixedThreadPool(

name: String,

fixedThreadCount: Int,

workQueueSize: Int,

statsReceiver: StatsReceiver

): FuturePool =

futurePool(

name = name,

minThreads = fixedThreadCount,

maxThreads = fixedThreadCount,

keepIdleThreadsAlive = Duration.Zero,

workQueue = new ArrayBlockingQueue[Runnable](workQueueSize),

statsReceiver = statsReceiver

)

/\*\*

\* Makes a [[FuturePool]] with a fix number of threads and an unbounded work queue

\*

\* @note Since the work queue is unbounded, this will fill up memory if the available worker threads can't keep up

\*/

def unboundedFixedThreadPool(

name: String,

fixedThreadCount: Int,

statsReceiver: StatsReceiver

): FuturePool =

futurePool(

name = name,

minThreads = fixedThreadCount,

maxThreads = fixedThreadCount,

keepIdleThreadsAlive = Duration.Zero,

workQueue = new LinkedBlockingQueue[Runnable],

statsReceiver = statsReceiver

)

/\*\*

\* Makes a [[FuturePool]] with the provided thread configuration and

\* who's `workQueue` is monitored by a [[com.twitter.finagle.stats.Gauge]]

\*

\* @note if `minThreads` == `maxThreads` then the threadpool has a fixed size

\*

\* @param name name of the threadpool

\* @param minThreads minimum number of threads in the pool

\* @param maxThreads maximum number of threads in the pool

\* @param keepIdleThreadsAlive threads that are idle for this long will be removed from

\* the pool if there are more than `minThreads` in the pool.

\* If the pool size is fixed this is ignored.

\*/

private def futurePool(

name: String,

minThreads: Int,

maxThreads: Int,

keepIdleThreadsAlive: Duration,

workQueue: BlockingQueue[Runnable],

statsReceiver: StatsReceiver

): FuturePool = {

val gaugeReference = statsReceiver.addGauge("workQueueSize")(workQueue.size())

val threadFactory = new NamedPoolThreadFactory(name, makeDaemons = true)

val executorService =

new ThreadPoolExecutor(

minThreads,

maxThreads, // ignored by ThreadPoolExecutor when an unbounded queue is provided

keepIdleThreadsAlive.inMillis,

TimeUnit.MILLISECONDS,

workQueue,

threadFactory)

FuturePool.interruptible(executorService)

}

}