**Java多线程 之 lock与condition的使用（十四）**

在博文 [Java多线程 之 wait、notifyAll（十二）](http://blog.csdn.net/fan2012huan/article/details/51824339) 中使用wait、notify使“打蜡”和“抛光”两个任务能够协同工作，本文阐述使用lock、condition来实现。而且使用signalAll要比使用notifyAll更安全。

1.使用抛出异常

package org.fan.learn.thread.testCondition;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.locks.AbstractQueuedSynchronizer;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\* Created by fan on 2016/7/4.

\*/

class Car {

private ReentrantLock reentrantLock = new ReentrantLock();

private Condition condition = reentrantLock.newCondition();

private boolean waxOn = false;

//同步方法

//打蜡

void waxing() {

reentrantLock.lock();

try {

waxOn = true;

condition.signalAll();

} finally {

reentrantLock.unlock();

}

}

//同步方法

//抛光

void buffing() {

reentrantLock.lock();

try {

waxOn = false;

condition.signalAll();

} finally {

reentrantLock.unlock();

}

}

//同步方法

//等待打蜡完成

void waitForWaxing() throws InterruptedException {

reentrantLock.lock();

try {

while (!waxOn) {

condition.await();

}

} finally {

reentrantLock.unlock();

}

}

//同步方法

//等待抛光完成

void waitForBuffing() throws InterruptedException {

reentrantLock.lock();

try {

while (waxOn) {

condition.await();

}

} finally {

reentrantLock.unlock();

}

}

}

//打蜡任务

//一次打蜡完成，需要等待抛光完成之后才能继续打蜡

class WaxingTask implements Runnable {

private Car car;

public WaxingTask(Car car) {

this.car = car;

}

public void run() {

try {

while (!Thread.interrupted()) {

System.out.println("waxing");

TimeUnit.MILLISECONDS.sleep(200);

car.waxing();

car.waitForBuffing();

}

} catch (InterruptedException e) {

//打印异常堆栈

e.printStackTrace();

}

System.out.println("Ending Waxing Task");

}

}

//抛光任务

//在抛光之前必须要先打蜡，需要等待打蜡完成才能抛光

class BuffingTask implements Runnable {

private Car car;

public BuffingTask(Car car) {

this.car = car;

}

public void run() {

try {

while (!Thread.interrupted()) {

car.waitForWaxing();

System.out.println("Buffing");

TimeUnit.MILLISECONDS.sleep(200);

car.buffing();

}

} catch (InterruptedException e) {

//打印异常堆栈

e.printStackTrace();

}

System.out.println("Ending Buffering Task");

}

}

public class WaxingBuffing {

public static void main(String[] args) throws InterruptedException {

Car car = new Car();

ExecutorService exe = Executors.newCachedThreadPool();

exe.execute(new WaxingTask(car));

exe.execute(new BuffingTask(car));

TimeUnit.SECONDS.sleep(5);

//向所有任务发送interrupt()信号

exe.shutdownNow();

System.out.println("main exit");

}

}

使用抛出异常的输出：

waxing

Buffing

waxing

Buffing

waxing

Buffing

waxing

Buffing

waxing

main exit

Ending Waxing Task

java.lang.InterruptedException: sleep interrupted

Ending Buffering Task

at java.lang.Thread.sleep(Native Method)

at java.lang.Thread.sleep(Thread.java:340)

at java.util.concurrent.TimeUnit.sleep(TimeUnit.java:386)

at org.fan.learn.thread.testCondition.WaxingTask.run(WaxingBuffing.java:110)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

at java.lang.Thread.run(Thread.java:745)

java.lang.InterruptedException

at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.reportInterruptAfterWait(AbstractQueuedSynchronizer.java:2014)

at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2048)

at org.fan.learn.thread.testCondition.Car.waitForWaxing(WaxingBuffing.java:49)

at org.fan.learn.thread.testCondition.BuffingTask.run(WaxingBuffing.java:134)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

at java.lang.Thread.run(Thread.java:745)

Process finished with exit code 0

2.使用catch异常：

package org.fan.learn.thread.testCondition;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.locks.AbstractQueuedSynchronizer;

import java.util.concurrent.locks.Condition;

import java.util.concurrent.locks.ReentrantLock;

/\*\*

\* Created by fan on 2016/7/4.

\*/

class Car {

private ReentrantLock reentrantLock = new ReentrantLock();

private Condition condition = reentrantLock.newCondition();

private boolean waxOn = false;

//同步方法

//打蜡

void waxing() {

reentrantLock.lock();

try {

waxOn = true;

condition.signalAll();

} finally {

reentrantLock.unlock();

}

}

//同步方法

//抛光

void buffing() {

reentrantLock.lock();

try {

waxOn = false;

condition.signalAll();

} finally {

reentrantLock.unlock();

}

}

//同步方法

//等待打蜡完成

void waitForWaxing() {

reentrantLock.lock();

try {

while (!waxOn) {

condition.await();

}

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

reentrantLock.unlock();

}

}

//同步方法

//等待抛光完成

void waitForBuffing() {

reentrantLock.lock();

try {

while (waxOn) {

condition.await();

}

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

reentrantLock.unlock();

}

}

}

//打蜡任务

//一次打蜡完成，需要等待抛光完成之后才能继续打蜡

class WaxingTask implements Runnable {

private Car car;

public WaxingTask(Car car) {

this.car = car;

}

public void run() {

try {

while (!Thread.interrupted()) {

System.out.println("waxing");

TimeUnit.MILLISECONDS.sleep(200);

car.waxing();

car.waitForBuffing();

}

} catch (InterruptedException e) {

//打印异常堆栈

e.printStackTrace();

}

System.out.println("Ending Waxing Task");

}

}

//抛光任务

//在抛光之前必须要先打蜡，需要等待打蜡完成才能抛光

class BuffingTask implements Runnable {

private Car car;

public BuffingTask(Car car) {

this.car = car;

}

public void run() {

try {

while (!Thread.interrupted()) {

car.waitForWaxing();

System.out.println("Buffing");

TimeUnit.MILLISECONDS.sleep(200);

car.buffing();

}

} catch (InterruptedException e) {

//打印异常堆栈

e.printStackTrace();

}

System.out.println("Ending Buffering Task");

}

}

public class WaxingBuffing {

public static void main(String[] args) throws InterruptedException {

Car car = new Car();

ExecutorService exe = Executors.newCachedThreadPool();

exe.execute(new WaxingTask(car));

exe.execute(new BuffingTask(car));

TimeUnit.SECONDS.sleep(5);

//向所有任务发送interrupt()信号

exe.shutdownNow();

System.out.println("main exit");

}

}

这个无法正常结束。。

在waitForWaxing中catch异常之后，无法继续执行了。因此需要往外抛。

对于异常：如果一个方法A中捕获异常，调用该方法Ａ的方法Ｂ则无法正常退出，需要Ａ方法抛出异常，由方法Ｂ捕获，这时执行完catch之后可以继续执行方法Ｂ中的语句。

使用catch异常的输出：

waxing

Buffing

waxing

Buffing

waxing

main exit

java.lang.InterruptedException: sleep interrupted

at java.lang.Thread.sleep(Native Method)

at java.lang.Thread.sleep(Thread.java:340)

at java.util.concurrent.TimeUnit.sleep(TimeUnit.java:386)

at org.fan.learn.thread.testCondition.WaxingTask.run(WaxingBuffing.java:110)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

at java.lang.Thread.run(Thread.java:745)

java.lang.InterruptedException

at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.reportInterruptAfterWait(AbstractQueuedSynchronizer.java:2014)

at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2048)

at org.fan.learn.thread.testCondition.Car.waitForWaxing(WaxingBuffing.java:61)

at org.fan.learn.thread.testCondition.BuffingTask.run(WaxingBuffing.java:134)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

at java.lang.Thread.run(Thread.java:745)

Ending Waxing Task

Buffing

---------------------

作者：fan2012huan

来源：CSDN

原文：https://blog.csdn.net/fan2012huan/article/details/51931735

版权声明：本文为博主原创文章，转载请附上博文链接！