**Java多线程 之 同步队列BlockingQueue与管道（十五）**

一.同步队列BlockingQueue

前面的两篇博文：

Java多线程 之 生产者、消费者（十三）

Java多线程 之 lock与condition的使用（十四）

详细阐述了多个任务之间的协同合作，需要使用wait、notify、notifyAll或者lock、condition、await、signal、signalAll方法来进行同步。实现起来比较复杂。因此java提供了同步队列（阻塞队列BlockingQueue）。

同步队列要求只能有一个任务对其进行操作（因此无需再对其使用同步操作，同步队列内部是同步的。）。当队列是空时，会导致取该队列的线程阻塞；当队列满（设置固定大小的队列）时，会导致写该队列的线程阻塞。

java的JUC（java.util.concurrent）包提供了BlockingQueue接口，并为其提供了三个实现：LinkedBlockingQueue（无界队列）、ArrayBlockingQueue（有固定大小的队列）、 SynchronousQueue（大小为1的队列）。

下面请看代码：

package org.fan.learn.thread.blockqueue;

/\*\*

\* Created by fan on 2016/6/1.

\*/

public class LiftOff implements Runnable {

private static int taskCunnt = 0;

private final int id = taskCunnt++;

private int countDown = 10;

private void status() {

System.out.print("Task(" + id + ")#" + (countDown > 0 ? countDown : "liftOff") + " ");

}

public void run() {

while (countDown-- > 0) {

status();

Thread.yield();

}

System.out.println();

}

}

package org.fan.learn.thread.blockqueue;

import java.util.concurrent.\*;

/\*\*

\* Created by fan on 2016/7/17.

\*/

class LiftOffRunner implements Runnable {

private BlockingQueue<LiftOff> blockingQueue;

public LiftOffRunner(BlockingQueue<LiftOff> blockingQueue) {

this.blockingQueue = blockingQueue;

}

public void add(LiftOff liftOff) {

try {

//add方法，如果是ArrayBlockingQueue会抛 队列满的异常

//blockingQueue.add(liftOff);

blockingQueue.put(liftOff);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public void run() {

try {

while (!Thread.interrupted()) {

LiftOff liftOff = blockingQueue.take();

liftOff.run();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Exiting Liftoff Runner");

}

}

public class TestBlockingQueue {

static void test(BlockingQueue<LiftOff> blockingQueue) throws InterruptedException {

LiftOffRunner liftOffRunner = new LiftOffRunner(blockingQueue);

Thread thread = new Thread(liftOffRunner);

thread.start();

for (int i = 0; i < 5; i++) {

liftOffRunner.add(new LiftOff());

}

TimeUnit.SECONDS.sleep(2);

thread.interrupt();

System.out.println("Finished Test");

}

public static void main(String[] args) {

try {

//test(new ArrayBlockingQueue<LiftOff>(3));

//test(new LinkedBlockingQueue<LiftOff>());

test(new SynchronousQueue<LiftOff>());

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

注意：

//add方法，如果是ArrayBlockingQueue会抛 队列满的异常

//blockingQueue.add(liftOff);

1.add会造成异常。

Exception in thread "main" java.lang.IllegalStateException: Queue full

Task(0)#9 Task(0)#8 Task(0)#7 Task(0)#6 Task(0)#5 Task(0)#4 Task(0)#3 Task(0)#2 Task(0)#1 Task(0)#liftOff Task(1)#9 Task(1)#8 Task(1)#7 Task(1)#6 Task(1)#5 Task(1)#4 Task(1)#3 Task(1)#2 Task(1)#1 Task(1)#liftOff Task(2)#9 Task(2)#8 Task(2)#7 Task(2)#6 Task(2)#5 Task(2)#4 Task(2)#3 Task(2)#2 Task(2)#1 Task(2)#liftOff at java.util.AbstractQueue.add(AbstractQueue.java:98)

at java.util.concurrent.ArrayBlockingQueue.add(ArrayBlockingQueue.java:312)

at org.fan.learn.thread.blockqueue.LiftOffRunner.add(TestBlockingQueue.java:20)

at org.fan.learn.thread.blockqueue.TestBlockingQueue.test(TestBlockingQueue.java:42)

at org.fan.learn.thread.blockqueue.TestBlockingQueue.main(TestBlockingQueue.java:51)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:497)

at com.intellij.rt.execution.application.AppMain.main(AppMain.java:144)

Process finished with exit code -1

2.正常结束

Task(0)#9 Task(0)#8 Task(0)#7 Task(0)#6 Task(0)#5 Task(0)#4 Task(0)#3 Task(0)#2 Task(0)#1 Task(0)#liftOff

Task(1)#9 Task(1)#8 Task(1)#7 Task(1)#6 Task(1)#5 Task(1)#4 Task(1)#3 Task(1)#2 Task(1)#1 Task(1)#liftOff

Task(2)#9 Task(2)#8 Task(2)#7 Task(2)#6 Task(2)#5 Task(2)#4 Task(2)#3 Task(2)#2 Task(2)#1 Task(2)#liftOff

Task(3)#9 Task(3)#8 Task(3)#7 Task(3)#6 Task(3)#5 Task(3)#4 Task(3)#3 Task(3)#2 Task(3)#1 Task(3)#liftOff

Task(4)#9 Task(4)#8 Task(4)#7 Task(4)#6 Task(4)#5 Task(4)#4 Task(4)#3 Task(4)#2 Task(4)#1 Task(4)#liftOff

Finished Test

java.lang.InterruptedException

Exiting Liftoff Runner

at java.util.concurrent.SynchronousQueue.take(SynchronousQueue.java:928)

at org.fan.learn.thread.blockqueue.LiftOffRunner.run(TestBlockingQueue.java:28)

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

Process finished with exit code 0

二.管道

管道跟队列差不多，但是比起BlockingQueue来，不如BlockingQueue更健壮。当管道是空时，它会阻塞读取管道的线程。

两个任务操作同一个管道时，一个写端（PipeWriter）、一个读端（PipeReader）。注意，只能其中一个任务创建管道，而另一个管道与刚才创建的管道进行关联。管道这种IO方式可以被interrrupt()中断，而普通的IO是不能被interrupt中断的。

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

作者：fan2012huan

来源：CSDN

原文：https://blog.csdn.net/fan2012huan/article/details/51932593

版权声明：本文为博主原创文章，转载请附上博文链接！