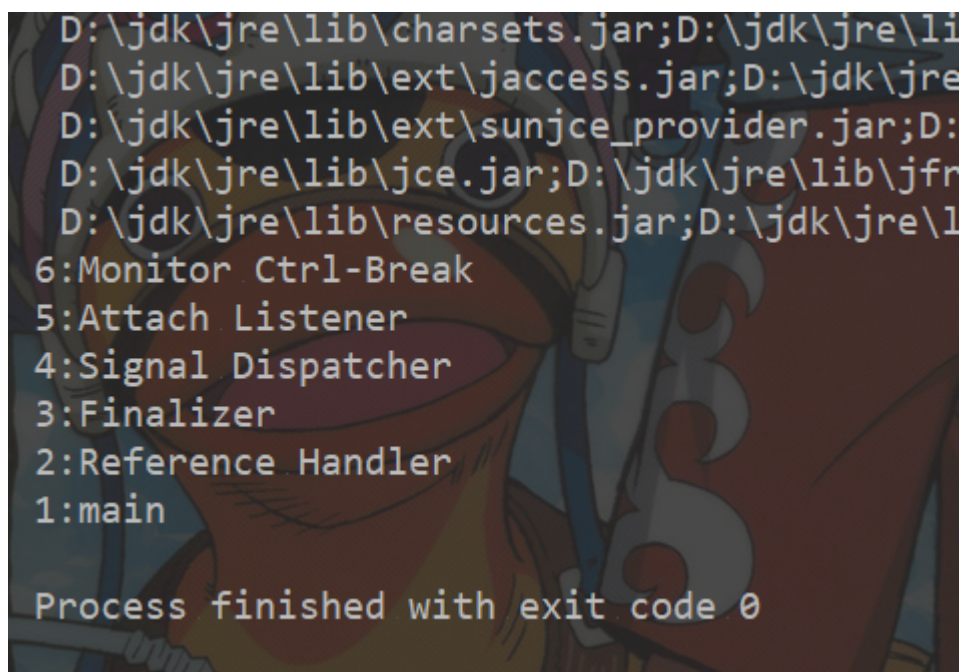


# 一、什么是线程

一个普通的Java程序包含哪些线程?

```
1 package com.example.thread;
2
3 import java.lang.management.ManagementFactory;
4 import java.lang.management.ThreadInfo;
5 import java.lang.management.ThreadMXBean;
6
7 /**
8  * @Author: 98050
9  * @Time: 2019-05-29 15:41
10  * @Feature:
11  */
12 public class MultiThread {
13
14     public static void main(String[] args) throws InterruptedException {
15         //1.获取Java线程管理MXBean
16         ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();
17         //2.不需要获取同步的monitor和synchronizer信息, 仅获取线程和线程堆栈信息
18         ThreadInfo[] threadInfos = threadMXBean.dumpAllThreads(false, false);
19         //3.输出
20         for (ThreadInfo threadInfo : threadInfos){
21             System.out.println(threadInfo.getThreadId() + ":" +
22 threadInfo.getThreadName());
23         }
24     }
25 }
```



```
D:\jdk\jre\lib\charsets.jar;D:\jdk\jre\li
D:\jdk\jre\lib\ext\jaccess.jar;D:\jdk\jre
D:\jdk\jre\lib\ext\sunjce_provider.jar;D:
D:\jdk\jre\lib\jce.jar;D:\jdk\jre\lib\jfr
D:\jdk\jre\lib\resources.jar;D:\jdk\jre\l
6:Monitor Ctrl-Break
5:Attach Listener
4:Signal Dispatcher
3:Finalizer
2:Reference Handler
1:main

Process finished with exit code 0
```

## 二、为什么使用多线程

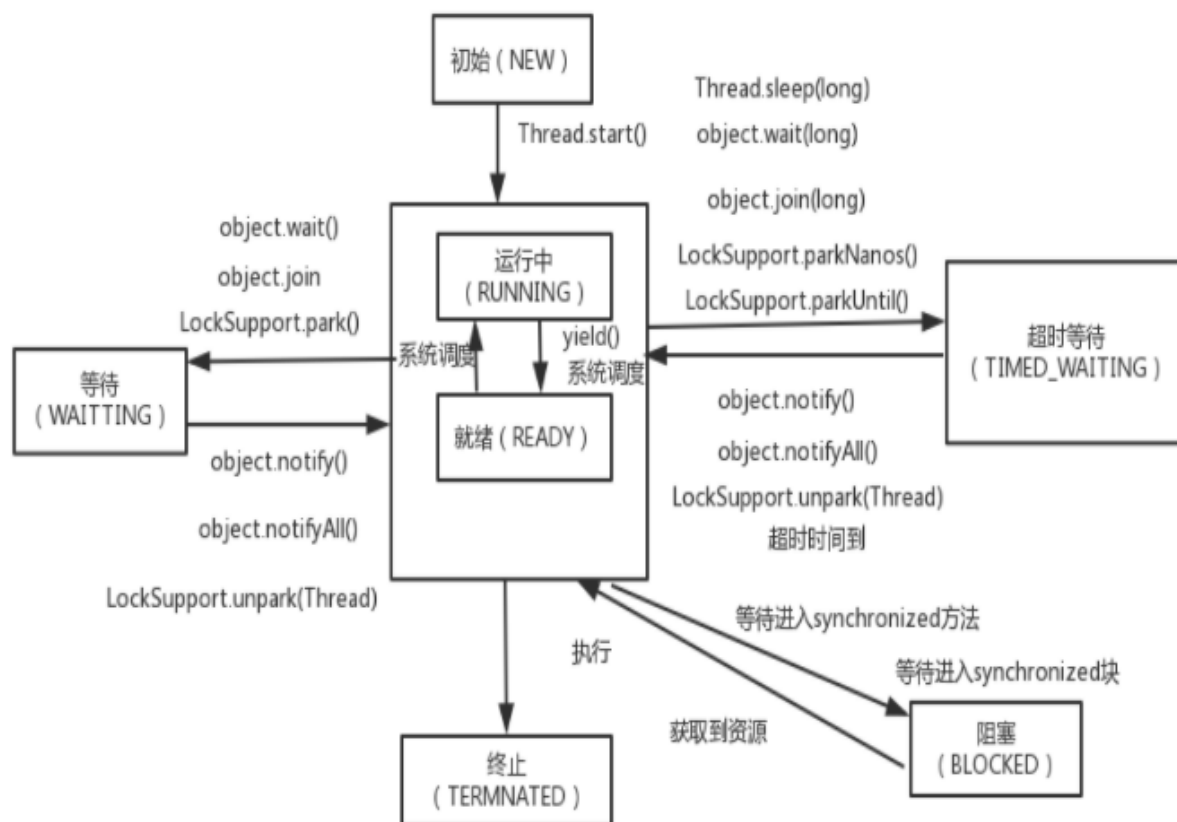
- 多个处理器核心
- 更快的响应时间
- 更好的编程模型

## 三、线程的优先级

通过`setPriority(int)`方法来修改优先级，默认优先级为5，优先级高的线程分配时间片的数量要多于优先级低的线程。

高IO的线程优先级设置较高，高CPU的优先级低

## 四、线程的状态



[https://blog.csdn.net/qq\\_32459653](https://blog.csdn.net/qq_32459653)

状态名称	说明
NEW	初试状态，线程被创建，但还没调用start()方法
RUNNABLE	运行状态，Java线程将操作系统中的就绪和运行统一为“运行中”
BLOCKED	阻塞状态，表示线程阻塞于锁
WAITING	等待装，表示线程进入等待状态
TIME_WAITING	超时等待状态，可以在指定的时间自行返回
TERMINATED	终止状态

```

1  package com.example.threadstate;
2
3  import java.util.concurrent.ThreadFactory;
4
5  /**
6   * @Author: 98050
7   * @Time: 2019-05-29 16:33
8   * @Feature:
9   */
10 public class ThreadState {
11
12     /**
13      * 该线程不断进行睡眠
14      */
15     static class Timewaiting implements Runnable{
16
17         @Override
18         public void run() {
19             while (true){
20                 try {
21                     Thread.sleep(100);
22                 } catch (InterruptedException e) {
23                     e.printStackTrace();
24                 }
25             }
26         }
27     }
28
29     /**
30      * 该线程一直在等待锁
31      */
32     static class waiting implements Runnable{
33
34         @Override
35         public void run() {
36             while (true){
37                 synchronized (waiting.class){
38                     try {
39                         waiting.class.wait();

```

```

40         } catch (InterruptedException e) {
41             e.printStackTrace();
42         }
43     }
44 }
45 }
46 }
47
48 /**
49  * 该线程加锁，但是不释放锁
50  */
51 static class Blocked implements Runnable{
52
53     @Override
54     public void run() {
55         synchronized (Blocked.class){
56             while (true){
57                 try {
58                     Thread.sleep(100);
59                 } catch (InterruptedException e) {
60                     e.printStackTrace();
61                 }
62             }
63         }
64     }
65 }
66
67 public static void main(String[] args) {
68     new Thread(new Timewaiting(), "TimewaitingThread").start();
69     new Thread(new waiting(), "waitingThread").start();
70     //两个Blocked线程，一个获取锁，另外一个阻塞
71     new Thread(new Blocked(), "BlockedThread-1").start();
72     new Thread(new Blocked(), "BlockedThread-2").start();
73 }
74 }

```

```

D:\jdk\bin>jps
21748 ThreadState
18024 Jps
20840 Launcher
7864 Launcher
14188

D:\jdk\bin>jstack 21748
2019-05-29 16:52:49
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.161-b12 mixed mode):

"DestroyJavaVM" #16 prio=5 os_prio=0 tid=0x00000000029de000 nid=0x5c44 waiting on condition [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE

"BlockedThread-2" #15 prio=5 os_prio=0 tid=0x0000000001eb4800 nid=0x5f10 waiting on condition [0x0000000001fb5f000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at com.example.threadstate.ThreadState$Blocked.run(ThreadState.java:58)
    - locked <0x0000000076b1a29c0> (a java.lang.Class for com.example.threadstate.ThreadState$Blocked)
    at java.lang.Thread.run(Thread.java:748)

"BlockedThread-1" #14 prio=5 os_prio=0 tid=0x0000000001eb47800 nid=0x3e70 waiting for monitor entry [0x0000000001fa5f000]
  java.lang.Thread.State: BLOCKED (on object monitor)
    at com.example.threadstate.ThreadState$Blocked.run(ThreadState.java:58)
    - waiting to lock <0x0000000076b1a29c0> (a java.lang.Class for com.example.threadstate.ThreadState$Blocked)
    at java.lang.Thread.run(Thread.java:748)

"WaitingThread" #13 prio=5 os_prio=0 tid=0x0000000001eb46800 nid=0x60d4 in Object.wait() [0x0000000001f95f000]
  java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    - waiting on <0x0000000076b19f818> (a java.lang.Class for com.example.threadstate.ThreadState$Waiting)
    at java.lang.Object.wait(Object.java:502)
    at com.example.threadstate.ThreadState$Waiting.run(ThreadState.java:39)
    - locked <0x0000000076b19f818> (a java.lang.Class for com.example.threadstate.ThreadState$Waiting)
    at java.lang.Thread.run(Thread.java:748)

"TimeWaitingThread" #12 prio=5 os_prio=0 tid=0x0000000001eb11000 nid=0x5278 waiting on condition [0x0000000001f85f000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at com.example.threadstate.ThreadState$TimeWaiting.run(ThreadState.java:21)
    at java.lang.Thread.run(Thread.java:748)

```

BlockedThread-2获取到了Blocked.class锁

BlockedThread-1阻塞在获取锁上

WaitingThread线程在Waiting实例上等待

TimeWaitingThread处于超时状态

## 五、启动和终止线程

### 5.1 启动

线程初始化:

```

1 private void init(ThreadGroup g, Runnable target, String name,
2                   long stackSize, AccessControlContext acc,
3                   boolean inheritThreadLocals) {
4     if (name == null) {
5         throw new NullPointerException("name cannot be null");
6     }
7
8     this.name = name;
9
10    Thread parent = currentThread();
11    SecurityManager security = System.getSecurityManager();
12    if (g == null) {
13        /* Determine if it's an applet or not */
14
15        /* If there is a security manager, ask the security manager
16         what to do. */
17        if (security != null) {
18            g = security.getThreadGroup();
19        }
20
21        /* If the security doesn't have a strong opinion of the matter

```

```

22         use the parent thread group. */
23         if (g == null) {
24             g = parent.getThreadGroup();
25         }
26     }
27
28     /* checkAccess regardless of whether or not threadgroup is
29        explicitly passed in. */
30     g.checkAccess();
31
32     /*
33      * Do we have the required permissions?
34      */
35     if (security != null) {
36         if (isCCLOverridden(getClass())) {
37             security.checkPermission(SUBCLASS_IMPLEMENTATION_PERMISSION);
38         }
39     }
40
41     g.addUnstarted();
42
43     this.group = g;
44     this.daemon = parent.isDaemon();
45     this.priority = parent.getPriority();
46     if (security == null || isCCLOverridden(parent.getClass()))
47         this.contextClassLoader = parent.getContextClassLoader();
48     else
49         this.contextClassLoader = parent.contextClassLoader;
50     this.inheritedAccessControlContext =
51         acc != null ? acc : AccessController.getContext();
52     this.target = target;
53     setPriority(priority);
54     if (inheritThreadLocals && parent.inheritableThreadLocals != null)
55         this.inheritableThreadLocals =
56             ThreadLocal.createInheritedMap(parent.inheritableThreadLocals);
57     /* Stash the specified stack size in case the VM cares */
58     this.stackSize = stackSize;
59
60     /* Set thread ID */
61     tid = nextThreadID();
62 }

```

启动:

```

1 public synchronized void start() {
2     /**
3      * This method is not invoked for the main method thread or "system"
4      * group threads created/set up by the VM. Any new functionality added
5      * to this method in the future may have to also be added to the VM.
6      *
7      * A zero status value corresponds to state "NEW".
8      */
9     if (threadStatus != 0)

```

```

10         throw new IllegalStateException();
11
12         /* Notify the group that this thread is about to be started
13          * so that it can be added to the group's list of threads
14          * and the group's unstarted count can be decremented. */
15         group.add(this);
16
17         boolean started = false;
18         try {
19             start0();
20             started = true;
21         } finally {
22             try {
23                 if (!started) {
24                     group.threadStartFailed(this);
25                 }
26             } catch (Throwable ignore) {
27                 /* do nothing. If start0 threw a Throwable then
28                  * it will be passed up the call stack */
29             }
30         }
31     }

```

## 5.2 安全的终止线程

中断或者boolean变量来控制

中断：线程的一个标识位

```

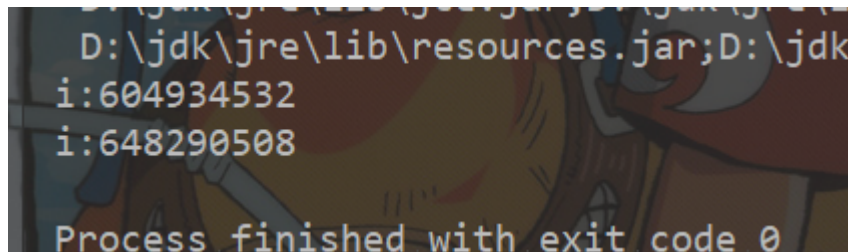
1  package com.example.threadstop;
2
3  /**
4   * @Author: 98050
5   * @Time: 2019-05-29 17:47
6   * @Feature:
7   */
8  public class Shutdown {
9
10     private static class Runner implements Runnable{
11
12         private long i;
13
14         private volatile boolean on = true;
15
16         @Override
17         public void run() {
18             while (on && !Thread.currentThread().isInterrupted()){
19                 i++;
20             }
21             System.out.println("i:" + i);
22         }
23     }

```

```

24     public void cancel(){
25         on = false;
26     }
27 }
28
29 public static void main(String[] args) throws InterruptedException {
30     Runner one = new Runner();
31     Thread thread1 = new Thread(one, "CountThread");
32     thread1.start();
33     Thread.sleep(1000);
34     thread1.interrupt();
35
36     Runner two = new Runner();
37     Thread thread2 = new Thread(two, "CountThread");
38     thread2.start();
39     Thread.sleep(1000);
40     two.cancel();
41 }
42 }

```



```

D:\jdk\jre\lib\resources.jar;D:\jdk\jre\lib\rt.jar;D:\jdk\jre\lib\sunrsync.jar
i:604934532
i:648290508
Process finished with exit code 0

```

## 六、线程之间的通讯

### 6.1 等待/通知经典范式

```

1  package com.example.communication;
2
3  import java.text.SimpleDateFormat;
4  import java.util.Date;
5
6  /**
7   * @Author: 98050
8   * @Time: 2019-05-30 16:05
9   * @Feature:
10  */
11  public class waitNotify {
12
13      static boolean flag = true;
14      static Object lock = new Object();
15
16      static class wait implements Runnable{
17          @Override
18          public void run() {
19              synchronized (lock){

```



```

20         while (flag){
21             System.out.println(Thread.currentThread() + "flag is true wait" +
new SimpleDateFormat("HH:mm:ss").format(new Date()));
22             try {
23                 lock.wait();
24             } catch (InterruptedException e) {
25                 e.printStackTrace();
26             }
27         }
28         System.out.println(Thread.currentThread() + "flag is false running" +
new SimpleDateFormat("HH:mm:ss").format(new Date()));
29     }
30 }
31 }
32
33
34 static class Notify implements Runnable{
35
36     @Override
37     public void run() {
38         synchronized (lock){
39             System.out.println(Thread.currentThread() + "hold lock notify" + new
SimpleDateFormat("HH:mm:ss").format(new Date()));
40             lock.notifyAll();
41             flag = false;
42             try {
43                 Thread.sleep(5000);
44             } catch (InterruptedException e) {
45                 e.printStackTrace();
46             }
47         }
48         synchronized (lock){
49             System.out.println(Thread.currentThread() + "hold lock again sleep" +
new SimpleDateFormat("HH:mm:ss").format(new Date()));
50             try {
51                 Thread.sleep(5000);
52             } catch (InterruptedException e) {
53                 e.printStackTrace();
54             }
55         }
56     }
57 }
58
59 public static void main(String[] args) throws InterruptedException {
60     Thread waitThread = new Thread(new wait(),"waitThread");
61     waitThread.start();
62
63     Thread.sleep(1000);
64
65     Thread notifyThread = new Thread(new Notify(),"NotifyThread");
66     notifyThread.start();
67 }
68 }

```

```
D:\jdk\jre\lib\jce.jar;D:\jdk\jre\lib\jfr.jar;D:\jdk\jre\lib\
D:\jdk\jre\lib\resources.jar;D:\jdk\jre\lib\rt.jar;E:\Java_De
Thread[WaitThread,5,main]flag is true wait16:27:46
Thread[NotifyThread,5,main]hold lock notify16:27:47
Thread[NotifyThread,5,main]hold lock again sleep16:27:52
Thread[WaitThread,5,main]flag is false running16:27:57
```

## 消费者

1. 获取对象的锁
2. 如果条件不满足，那么调用对象的wait()方法，被通知后仍要检查条件
3. 条件满足则执行对应的逻辑

```
1 synchronized (对象){
2     while (条件不满足){
3         对象.wait();
4     }
5     对应的处理逻辑
6 }
```

## 生产者

1. 获得对象锁
2. 改变条件
3. 通知所有等待在对象上的线程

```
1 synchronized (对象){
2     改变条件;
3     对象.notifyAll();
4 }
```

## 6.2 等待超时模式

由于经典的等待/通知范式无法做到超时等待，也就是说，当消费者在获得锁后，如果条件不满足，等待生产者改变条件之前会一直处于等待状态，在一些实际应用中，会浪费资源，降低运行效率。

事实上，只要对经典范式做出非常小的改动，就可以加入超时等待。

假设超时时间段是T，那么可以推断出，在当前时间now+T之后就会超时。

定义如下变量：

等待持续时间remaining = T;

超时时间future = now + T。

```

1 public synchronized Object get(long mills) throws InterruptedException {
2     long future = System.currentTimeMillis() + mills;
3     long remaining = mills;
4     while ((result == null) && remaining > 0){
5         wait(remaining);
6         remaining = future - System.currentTimeMillis();
7     }
8     return result;
9 }

```

例子:

```

1 package com.example.communication;
2
3 import java.util.LinkedList;
4
5 /**
6  * @Author: 98050
7  * @Time: 2019-06-02 17:34
8  * @Feature:
9  */
10 public class Test3 {
11
12     static Operator operator = new Operator(1);
13     public static void main(String[] args) {
14         for (int i = 0; i < 5; i++) {
15             new Thread(new MyThread(10)).start();
16         }
17     }
18
19     static class MyThread implements Runnable{
20
21         private int count;
22
23         public MyThread(int count) {
24             this.count = count;
25         }
26
27         @Override
28         public void run() {
29             while (count > 0){
30                 try {
31                     Object o = operator.get(1000);
32                     if (o != null) {
33                         /**
34                          * 实际操作, 睡眠2秒
35                          */
36                         Thread.sleep(2000);
37                         operator.find();
38                         System.out.println(o);
39                     }else {
40                         System.out.println("超时");

```

```

41         }
42     } catch (InterruptedException e) {
43         e.printStackTrace();
44     } finally {
45         count--;
46     }
47 }
48 }
49 }
50
51
52 static class Operator{
53
54     private final LinkedList<Object> result = new LinkedList<>();
55
56     public Operator(int size) {
57         for (int i = 0; i < size; i++) {
58             result.addLast(i);
59         }
60     }
61
62     public Object get(long mills) throws InterruptedException {
63         synchronized (result) {
64             long future = System.currentTimeMillis() + mills;
65             long remaining = mills;
66             while ((result.size() == 0) && remaining > 0) {
67                 result.wait(remaining);
68                 remaining = future - System.currentTimeMillis();
69             }
70             Object o = null;
71             if (result.size() != 0){
72                 o = result.removeFirst();
73             }
74             return o;
75         }
76     }
77
78     private void find(){
79         synchronized (result) {
80             result.add(2);
81             result.notifyAll();
82         }
83     }
84 }
85 }

```

## 6.3 面试题

### 6.3.1 三个线程轮流打印

1、有三个线程分别打印A、B、C,请用多线程编程实现，在屏幕上循环打印10次ABCABC...

Lock+Condition实现细粒度的控制

```

1 package com.example.join;
2
3 import java.util.concurrent.locks.Condition;
4 import java.util.concurrent.locks.Lock;
5 import java.util.concurrent.locks.ReentrantLock;
6
7 /**
8  * @Author: 98050
9  * @Time: 2019-05-30 17:02
10  * @Feature:
11  */
12 public class Test {
13
14     final static Lock lock = new ReentrantLock();
15     final static Condition conditionA = lock.newCondition();
16     final static Condition conditionB = lock.newCondition();
17     final static Condition conditionC = lock.newCondition();
18     volatile static String now = "A";
19
20
21     static class MyThreadPrintA implements Runnable{
22         @Override
23         public void run() {
24             for (int i = 0; i < 10; i++) {
25                 try {
26                     lock.lock();
27                     while (!now.equals("A")) {
28                         conditionA.await();
29                     }
30                     System.out.println(now);
31                     now = "B";
32                     conditionB.signal();
33                 } catch (Exception e){
34                     e.printStackTrace();
35                 } finally {
36                     lock.unlock();
37                 }
38             }
39         }
40     }
41
42     static class MyThreadPrintB implements Runnable{
43
44         @Override
45         public void run() {
46             for (int i = 0; i < 10; i++) {
47                 try {
48                     lock.lock();
49                     while (!now.equals("B")) {
50                         conditionB.await();
51                     }
52                     System.out.println(now);
53                     now = "C";

```

```

54         conditionC.signal();
55     }catch (Exception e){
56         e.printStackTrace();
57     }finally {
58         lock.unlock();
59     }
60 }
61 }
62 }
63
64 static class MyThreadPrintC implements Runnable{
65
66     @Override
67     public void run() {
68         for (int i = 0; i < 10; i++) {
69             try {
70                 lock.lock();
71                 while (!now.equals("C")) {
72                     conditionC.await();
73                 }
74                 System.out.println(now);
75                 now = "A";
76                 conditionA.signal();
77             }catch (Exception e){
78                 e.printStackTrace();
79             }finally {
80                 lock.unlock();
81             }
82         }
83     }
84 }
85
86
87 public static void main(String[] args) throws InterruptedException {
88     MyThreadPrintA myThreadPrintA = new MyThreadPrintA();
89     MyThreadPrintB myThreadPrintB = new MyThreadPrintB();
90     MyThreadPrintC myThreadPrintC = new MyThreadPrintC();
91
92     Thread t1 = new Thread(myThreadPrintA);
93     Thread t2 = new Thread(myThreadPrintB);
94     Thread t3 = new Thread(myThreadPrintC);
95
96     t1.start();
97     t2.start();
98     t3.start();
99 }
100
101 }

```

synchronized + wait、notifyAll

```

1 package com.example.communication;
2

```

```

3  /**
4   * @Author: 98050
5   * @Time: 2019-05-30 20:21
6   * @Feature:
7   */
8  public class Test2 {
9
10     static Object lock = new Object();
11
12     static volatile String name = "A";
13
14     static class MyThreadA implements Runnable{
15         @Override
16         public void run() {
17             for (int i = 0; i < 10; i++) {
18                 synchronized (lock){
19                     while (!name.equals("A")){
20                         try {
21                             lock.wait();
22                         } catch (InterruptedException e) {
23                             e.printStackTrace();
24                         }
25                     }
26                     System.out.println("A");
27                     name = "B";
28                     lock.notifyAll();
29                 }
30             }
31         }
32     }
33
34     static class MyThreadB implements Runnable{
35         @Override
36         public void run() {
37             for (int i = 0; i < 10; i++) {
38                 synchronized (lock){
39                     while (!name.equals("B")){
40                         try {
41                             lock.wait();
42                         } catch (InterruptedException e) {
43                             e.printStackTrace();
44                         }
45                     }
46                     System.out.println("B");
47                     name = "C";
48                     lock.notifyAll();
49                 }
50             }
51         }
52     }
53 }
54
55

```

```

56     static class MyThreadC implements Runnable{
57         @Override
58         public void run() {
59             for (int i = 0; i < 10; i++) {
60                 synchronized (lock){
61                     while (!name.equals("C")){
62                         try {
63                             lock.wait();
64                         } catch (InterruptedException e) {
65                             e.printStackTrace();
66                         }
67                     }
68                     System.out.println("C");
69                     name = "A";
70                     lock.notifyAll();
71                 }
72             }
73         }
74     }
75 }
76
77
78 public static void main(String[] args) {
79     new Thread(new MyThreadA()).start();
80     new Thread(new MyThreadB()).start();
81     new Thread(new MyThreadC()).start();
82 }
83 }

```

在多线程操作中，我们常常会遇到需要先判断信号量状态是否就绪，然后执行后续操作的场景。这里对状态的判断使用的是while而不是单线程下常用的if。

原因：

在线程中notify或者notifyAll会唤醒一个或多个线程，当线程被唤醒后，被唤醒的线程继续执行阻塞后的操作。

这里分析一下get操纵：当某个线程得到锁时storage为空，此时它应该wait，下次被唤醒时（任意线程调用notify），storage可能还是空的。因为有可能其他线程清空了storage。如果此时用的是if它将不再判断storage是否为空，直接继续，这样就引起了错误。但如果用while则每次被唤醒时都会先检查storage是否为空再继续，这样才是正确的操作；生产也是同一个道理。

## 6.3.2 两个线程轮流打印

### 2、两个线程轮流打印1~100

思路一：synchronized+wait/notify

```

1 package com.thread.test;
2
3 /**
4  * @Author: 98050
5  * @Time: 2019-06-11 21:03
6  * @Feature:
7  */

```



```

8 public class Test2 {
9
10     private static volatile int i = 1;
11
12     private static Object object = new Object();
13
14     private static boolean flag = false;
15
16
17     static class MyThread implements Runnable{
18
19         public void run() {
20             synchronized (object){
21                 while (i <= 100){
22                     if (flag) {
23                         System.out.println(Thread.currentThread().getName() + "打印: "
+ i++);
24                         flag = false;
25                     }else {
26                         object.notifyAll();
27                         try {
28                             object.wait();
29                         } catch (InterruptedException e) {
30                             e.printStackTrace();
31                         }
32                     }
33                 }
34                 if (i > 100){
35                     System.exit(0);
36                 }
37             }
38         }
39     }
40
41     static class MyThread2 implements Runnable{
42
43         public void run() {
44             synchronized (object){
45                 while (i <= 100) {
46                     if (!flag) {
47                         System.out.println(Thread.currentThread().getName() + "打印: "
+ i++);
48                         flag = true;
49                     } else{
50                         object.notifyAll();
51                         try {
52                             object.wait();
53                         } catch (InterruptedException e) {
54                             e.printStackTrace();
55                         }
56                     }
57                 }
58                 if (i > 100){

```

```

59         System.exit(0);
60     }
61 }
62 }
63 }
64
65 public static void main(String[] args) {
66     Thread t1 = new Thread(new MyThread());
67     Thread t2 = new Thread(new MyThread2());
68
69     t1.start();
70     t2.start();
71 }
72 }

```

```

1 package com.thread.test;
2
3 /**
4  * @Author: 98050
5  * @Time: 2019-06-11 21:47
6  * @Feature:
7  */
8 public class Test5 {
9
10     static volatile int count = 1;
11     static volatile Object object = new Object();
12
13     static class MyThread implements Runnable{
14
15         @Override
16         public void run() {
17             synchronized (object) {
18                 while (count <= 100) {
19                     System.out.println(Thread.currentThread().getName() + "打印: " +
count++);
20                     try {
21                         object.wait();
22                     } catch (InterruptedException e) {
23                         e.printStackTrace();
24                     }
25                 }
26             }
27         }
28     }
29
30     static class MyThread2 implements Runnable{
31
32         @Override
33         public void run() {
34             synchronized (object) {
35                 while (count <= 100) {

```

```

36         System.out.println(Thread.currentThread().getName() + "打印: " +
count++);
37         object.notify();
38     }
39 }
40 }
41 }
42
43 public static void main(String[] args) {
44     Thread t1 = new Thread(new Test2.MyThread());
45     Thread t2 = new Thread(new Test2.MyThread2());
46
47     t1.start();
48     t2.start();
49 }
50 }

```

思路二: while+boolean变量

```

1 package com.thread.test;
2
3 /**
4  * @Author: 98050
5  * @Time: 2019-06-11 21:27
6  * @Feature:
7  */
8 public class Test3 {
9
10     static volatile boolean tag = false;
11
12     static volatile int i = 1;
13
14     public static void main(String[] args) {
15         new Thread(() -> {
16             while (i <= 100){
17                 if (tag){
18                     System.out.println(Thread.currentThread().getName() + "打印: " +
i++);
19                     tag = false;
20                 }
21             }
22         }).start();
23
24         new Thread(() -> {
25             while (i <= 100){
26                 if (!tag){
27                     System.out.println(Thread.currentThread().getName() + "打印: " +
i++);
28                     tag = true;
29                 }
30             }
31         }).start();
32     }

```

思路三：lock + condition

```

1  package com.thread.test;
2
3  import java.util.concurrent.locks.Condition;
4  import java.util.concurrent.locks.Lock;
5  import java.util.concurrent.locks.ReentrantLock;
6
7  /**
8   * @Author: 98050
9   * @Time: 2019-06-11 21:31
10  * @Feature:
11  */
12  public class Test4 {
13
14      static Lock lock = new ReentrantLock();
15      static Condition condition1 = lock.newCondition();
16      static Condition condition2 = lock.newCondition();
17      static volatile int count = 1;
18
19      static class MyThread implements Runnable{
20
21          @Override
22          public void run() {
23              lock.lock();
24              try {
25                  while (count <= 100){
26                      System.out.println(Thread.currentThread().getName() + "打印: " +
count++);
27                      condition1.await();
28                      condition2.signal();
29                  }
30              }catch (Exception e ){
31
32              }finally {
33                  lock.unlock();
34              }
35          }
36      }
37
38      static class MyThread2 implements Runnable{
39
40          @Override
41          public void run() {
42              lock.lock();
43              try {
44                  while (count <= 100){
45                      System.out.println(Thread.currentThread().getName() + "打印: " +
count++);
46                      condition2.await();
47                      condition1.signal();

```

```
48         }
49     }catch (Exception e ){
50
51     }finally {
52         lock.unlock();
53     }
54 }
55 }
56
57 public static void main(String[] args) {
58     Thread t1 = new Thread(new Test2.MyThread());
59     Thread t2 = new Thread(new Test2.MyThread2());
60
61     t1.start();
62     t2.start();
63 }
64 }
```