一、什么是线程

一个普通的Java程序包含哪些线程?

```java
package com.example.thread;

import java.lang.management.ManagementFactory;
import java.lang.management.ThreadInfo;
import java.lang.management.ThreadMXBean;

/**
 * @Author: 98050
 * @Time: 2019-05-29 15:41
 * @Feature:
 */
public class MultiThread {

    public static void main(String[] args) throws InterruptedException {
        //1.获取Java线程管理MXBean
        ThreadMXBean threadMXBean = ManagementFactory.getThreadMXBean();
        //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
        ThreadInfo[] threadInfos = threadMXBean.dumpAllThreads(false, false);
        //3.输出
        for (ThreadInfo threadInfo : threadInfos){
            System.out.println(threadInfo.getThreadId() + ":" + threadInfo.getThreadName());
        }
    }
}
```

二、为什么使用多线程

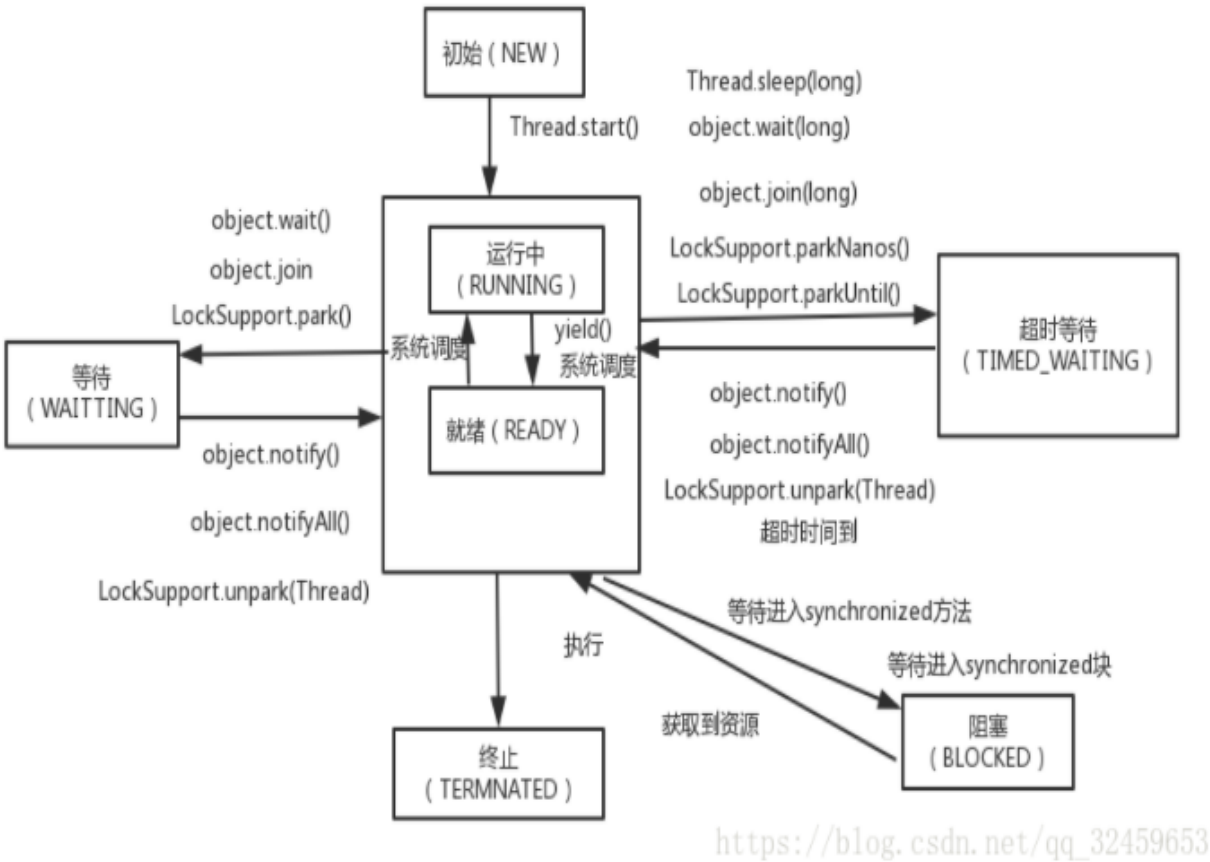
- 多个处理器核心
- 更快的响应时间
- 更好的编程模型

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



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

```java
package com.example.threadstate;

import java.util.concurrent.ThreadFactory;

/**
 * @Author: 98050
 * @Time: 2019-05-29 16:33
 * @Feature:
 */
public class ThreadState {

    /**
     * 该线程不断进行睡眠
     */
    static class TimeWaiting implements Runnable{

        @Override
        public void run() {
            while (true){
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    /**
     * 该线程一直在等待锁
     */
    static class Waiting implements Runnable{

        @Override
        public void run() {
            while (true){
                synchronized (Waiting.class){
                    try {
                        Waiting.class.wait();
```

```java
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }

    /**
     * 该线程加锁，但是不释放锁
     */
    static class Blocked implements Runnable{

        @Override
        public void run() {
            synchronized (Blocked.class){
                while (true){
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        new Thread(new TimeWaiting(),"TimeWaitingThread").start();
        new Thread(new Waiting(),"WaitingThread").start();
        //两个Blocked线程，一个获取锁，另外一个阻塞
        new Thread(new Blocked(),"BlockedThread-1").start();
        new Thread(new Blocked(),"BlockedThread-2").start();
    }
}
```

```
D:\jdk\bin>jps
21748 ThreadState
16024 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=0x000000001eb48800 nid=0x5f10 waiting on condition [0x000000001fb5f000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(Native Method)
        at com.example.threadstate.ThreadState$Blocked.run(ThreadState.java:58)
        - locked <0x000000076b1a29c0> (a java.lang.Class for com.example.threadstate.ThreadState$Blocked)
        at java.lang.Thread.run(Thread.java:748)

"BlockedThread-1" #14 prio=5 os_prio=0 tid=0x000000001eb47800 nid=0x3e70 waiting for monitor entry [0x000000001fa5f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at com.example.threadstate.ThreadState$Blocked.run(ThreadState.java:58)
        - waiting to lock <0x000000076b1a29c0> (a java.lang.Class for com.example.threadstate.ThreadState$Blocked)
        at java.lang.Thread.run(Thread.java:748)

"WaitingThread" #13 prio=5 os_prio=0 tid=0x000000001eb46800 nid=0x60d4 in Object.wait() [0x000000001f95f000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x000000076b19f818> (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 <0x000000076b19f818> (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=0x000000001eb11000 nid=0x5278 waiting on condition [0x000000001f85f000]
   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.clss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：

```java
private void init(ThreadGroup g, Runnable target, String name,
                  long stackSize, AccessControlContext acc,
                  boolean inheritThreadLocals) {
    if (name == null) {
        throw new NullPointerException("name cannot be null");
    }

    this.name = name;

    Thread parent = currentThread();
    SecurityManager security = System.getSecurityManager();
    if (g == null) {
        /* Determine if it's an applet or not */

        /* If there is a security manager, ask the security manager
           what to do. */
        if (security != null) {
            g = security.getThreadGroup();
        }

        /* If the security doesn't have a strong opinion of the matter
```

```
            use the parent thread group. */
        if (g == null) {
            g = parent.getThreadGroup();
        }
    }

    /* checkAccess regardless of whether or not threadgroup is
       explicitly passed in. */
    g.checkAccess();

    /*
     * Do we have the required permissions?
     */
    if (security != null) {
        if (isCCLOverridden(getClass())) {
            security.checkPermission(SUBCLASS_IMPLEMENTATION_PERMISSION);
        }
    }

    g.addUnstarted();

    this.group = g;
    this.daemon = parent.isDaemon();
    this.priority = parent.getPriority();
    if (security == null || isCCLOverridden(parent.getClass()))
        this.contextClassLoader = parent.getContextClassLoader();
    else
        this.contextClassLoader = parent.contextClassLoader;
    this.inheritedAccessControlContext =
            acc != null ? acc : AccessController.getContext();
    this.target = target;
    setPriority(priority);
    if (inheritThreadLocals && parent.inheritableThreadLocals != null)
        this.inheritableThreadLocals =
            ThreadLocal.createInheritedMap(parent.inheritableThreadLocals);
    /* Stash the specified stack size in case the VM cares */
    this.stackSize = stackSize;

    /* Set thread ID */
    tid = nextThreadID();
}
```

启动:

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

```
        throw new IllegalThreadStateException();

    /* Notify the group that this thread is about to be started
     * so that it can be added to the group's list of threads
     * and the group's unstarted count can be decremented. */
    group.add(this);

    boolean started = false;
    try {
        start0();
        started = true;
    } finally {
        try {
            if (!started) {
                group.threadStartFailed(this);
            }
        } catch (Throwable ignore) {
            /* do nothing. If start0 threw a Throwable then
              it will be passed up the call stack */
        }
    }
}
```

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

```
package com.example.threadstop;

/**
 * @Author: 98050
 * @Time: 2019-05-29 17:47
 * @Feature:
 */
public class Shutdown {

    private static class Runner implements Runnable{

        private long i;

        private volatile boolean on = true;

        @Override
        public void run() {
            while (on && !Thread.currentThread().isInterrupted()){
                i++;
            }
            System.out.println("i:" + i);
        }

```

```
        public void cancel(){
            on = false;
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Runner one = new Runner();
        Thread thread1 = new Thread(one,"CountThread");
        thread1.start();
        Thread.sleep(1000);
        thread1.interrupt();

        Runner two = new Runner();
        Thread thread2 = new Thread(two,"CountThread");
        thread2.start();
        Thread.sleep(1000);
        two.cancel();
    }
}
```

六、线程之间的通讯

6.1 等待/通知经典范式

```
package com.example.communication;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @Author: 98050
 * @Time: 2019-05-30 16:05
 * @Feature:
 */
public class WaitNotify {

    static boolean flag = true;
    static Object lock = new Object();

    static class Wait implements Runnable{
        @Override
        public void run() {
            synchronized (lock){
```

```java
                while (flag){
                    System.out.println(Thread.currentThread() + "flag is true wait" + new SimpleDateFormat("HH:mm:ss").format(new Date()));
                    try {
                        lock.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println(Thread.currentThread() + "flag is false running" + new SimpleDateFormat("HH:mm:ss").format(new Date()));
            }
        }
    }


    static class Notify implements Runnable{

        @Override
        public void run() {
            synchronized (lock){
                System.out.println(Thread.currentThread() + "hold lock notify" + new SimpleDateFormat("HH:mm:ss").format(new Date()));
                lock.notifyAll();
                flag = false;
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            synchronized (lock){
                System.out.println(Thread.currentThread() + "hold lock again sleep" + new SimpleDateFormat("HH:mm:ss").format(new Date()));
                try {
                    Thread.sleep(5000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        Thread waitThread = new Thread(new Wait(),"WaitThread");
        waitThread.start();

        Thread.sleep(1000);

        Thread notifyThread = new Thread(new Notify(),"NotifyThread");
        notifyThread.start();
    }
}
```

```
;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

```
synchronized (对象){
    while (条件不满足){
        对象.wait();
    }
    对应的处理逻辑
}
```

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

```
synchronized (对象){
    改变条件;
    对象.notifyAll();
}
```

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。

```java
public synchronized Object get(long mills) throws InterruptedException {
    long future = System.currentTimeMillis() + mills;
    long remaining = mills;
    while ((result == null) && remaining > 0){
        wait(remaining);
        remaining = future - System.currentTimeMillis();
    }
    return result;
}
```

例子:

```java
package com.example.communication;

import java.util.LinkedList;

/**
 * @Author: 98050
 * @Time: 2019-06-02 17:34
 * @Feature:
 */
public class Test3 {

    static Operator operator = new Operator(1);
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            new Thread(new MyThread(10)).start();
        }
    }

    static class MyThread implements Runnable{

        private int count;

        public MyThread(int count) {
            this.count = count;
        }

        @Override
        public void run() {
            while (count > 0){
                try {
                    Object o = operator.get(1000);
                    if (o != null) {
                        /**
                         * 实际操作，睡眠2秒
                         */
                        Thread.sleep(2000);
                        operator.find();
                        System.out.println(o);
                    }else {
                        System.out.println("超时");
```

```java
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }finally {
                    count--;
                }
            }
        }
    }


    static class Operator{

        private final LinkedList<Object> result = new LinkedList<>();

        public Operator(int size) {
            for (int i = 0; i < size; i++) {
                result.addLast(i);
            }
        }

        public Object get(long mills) throws InterruptedException {
            synchronized (result) {
                long future = System.currentTimeMillis() + mills;
                long remaining = mills;
                while ((result .size() == 0) && remaining > 0) {
                    result.wait(remaining);
                    remaining = future - System.currentTimeMillis();
                }
                Object o = null;
                if (result.size() != 0){
                    o = result.removeFirst();
                }
                return o;
            }
        }

        private void find(){
            synchronized (result) {
                result.add(2);
                result.notifyAll();
            }
        }
    }
}
```

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

```java
package com.example.join;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @Author: 98050
 * @Time: 2019-05-30 17:02
 * @Feature:
 */
public class Test {

    final static Lock lock = new ReentrantLock();
    final static Condition conditionA = lock.newCondition();
    final static Condition conditionB = lock.newCondition();
    final static Condition conditionC = lock.newCondition();
    volatile static String now = "A";


    static class MyThreadPrintA implements Runnable{
        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                try {
                    lock.lock();
                    while (!now.equals("A")) {
                        conditionA.await();
                    }
                    System.out.println(now);
                    now = "B";
                    conditionB.signal();
                }catch (Exception e){
                    e.printStackTrace();
                }finally {
                    lock.unlock();
                }
            }
        }
    }

    static class MyThreadPrintB implements Runnable{

        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                try {
                    lock.lock();
                    while (!now.equals("B")) {
                        conditionB.await();
                    }
                    System.out.println(now);
                    now = "C";
```

```java
                    conditionC.signal();
                }catch (Exception e){
                    e.printStackTrace();
                }finally {
                    lock.unlock();
                }
            }
        }
    }

    static class MyThreadPrintC implements Runnable{

        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                try {
                    lock.lock();
                    while (!now.equals("C")) {
                        conditionC.await();
                    }
                    System.out.println(now);
                    now = "A";
                    conditionA.signal();
                }catch (Exception e){
                    e.printStackTrace();
                }finally {
                    lock.unlock();
                }
            }
        }
    }


    public static void main(String[] args) throws InterruptedException {
        MyThreadPrintA myThreadPrintA = new MyThreadPrintA();
        MyThreadPrintB myThreadPrintB = new MyThreadPrintB();
        MyThreadPrintC myThreadPrintC = new MyThreadPrintC();

        Thread t1 = new Thread(myThreadPrintA);
        Thread t2 = new Thread(myThreadPrintB);
        Thread t3 = new Thread(myThreadPrintC);

        t1.start();
        t2.start();
        t3.start();
    }

}
```

synchronized + wait、notifyAll

```java
package com.example.communication;

```

```java
/**
 * @Author: 98050
 * @Time: 2019-05-30 20:21
 * @Feature:
 */
public class Test2 {

    static Object lock = new Object();

    static volatile String name = "A";

    static class MyThreadA implements Runnable{
        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                synchronized (lock){
                    while (!name.equals("A")){
                        try {
                            lock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }

                    }
                    System.out.println("A");
                    name = "B";
                    lock.notifyAll();
                }
            }
        }
    }

    static class MyThreadB implements Runnable{
        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                synchronized (lock){
                    while (!name.equals("B")){
                        try {
                            lock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }

                    }
                    System.out.println("B");
                    name = "C";
                    lock.notifyAll();
                }
            }
        }
    }

```

```java
    static class MyThreadC implements Runnable{
        @Override
        public void run() {
            for (int i = 0; i < 10; i++) {
                synchronized (lock){
                    while (!name.equals("C")){
                        try {
                            lock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }

                    }
                    System.out.println("C");
                    name = "A";
                    lock.notifyAll();
                }
            }
        }
    }


    public static void main(String[] args)  {
        new Thread(new MyThreadA()).start();
        new Thread(new MyThreadB()).start();
        new Thread(new MyThreadC()).start();
    }
}
```

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

```java
package com.thread.test;

/**
 * @Author: 98050
 * @Time: 2019-06-11 21:03
 * @Feature:
 */
```

```java
public class Test2 {

    private static volatile int i = 1;

    private static Object object = new Object();

    private static boolean flag = false;


    static class MyThread implements Runnable{

        public void run() {
            synchronized (object){
                while (i <= 100){
                    if (flag) {
                        System.out.println(Thread.currentThread().getName() + "打印: " + i++);
                        flag = false;
                    }else {
                        object.notifyAll();
                        try {
                            object.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
                if (i > 100){
                    System.exit(0);
                }
            }
        }
    }

    static class MyThread2 implements Runnable{

        public void run() {
            synchronized (object){
                while (i <= 100) {
                    if (!flag) {
                        System.out.println(Thread.currentThread().getName() + "打印: " + i++);
                        flag = true;
                    } else{
                        object.notifyAll();
                        try {
                            object.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
                if (i > 100){
```

```
                    System.exit(0);
                }
            }
        }
    }

    public static void main(String[] args) {
        Thread t1 = new Thread(new MyThread());
        Thread t2 = new Thread(new MyThread2());

        t1.start();
        t2.start();
    }
}
```

```
package com.thread.test;

/**
 * @Author: 98050
 * @Time: 2019-06-11 21:47
 * @Feature:
 */
public class Test5 {

    static volatile int count = 1;
    static volatile Object object = new Object();

    static class MyThread implements Runnable{

        @Override
        public void run() {
            synchronized (object) {
                while (count <= 100) {
                    System.out.println(Thread.currentThread().getName() + "打印: " + count++);
                    try {
                        object.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }

    static class MyThread2 implements Runnable{

        @Override
        public void run() {
            synchronized (object) {
                while (count <= 100) {
```

```java
                    System.out.println(Thread.currentThread().getName() + "打印: " + count++);
                    object.notify();
                }
            }
        }
    }

    public static void main(String[] args) {
        Thread t1 = new Thread(new Test2.MyThread());
        Thread t2 = new Thread(new Test2.MyThread2());

        t1.start();
        t2.start();
    }
}
```

思路二：while+boolea变量

```java
package com.thread.test;

/**
 * @Author: 98050
 * @Time: 2019-06-11 21:27
 * @Feature:
 */
public class Test3 {

    static volatile boolean tag = false;

    static volatile int i = 1;

    public static void main(String[] args) {
        new Thread(() -> {
            while (i <= 100){
                if (tag){
                    System.out.println(Thread.currentThread().getName() + "打印: " + i++);
                    tag = false;
                }
            }
        }).start();

        new Thread(() -> {
            while (i <= 100){
                if (!tag){
                    System.out.println(Thread.currentThread().getName() + "打印: " + i++);
                    tag = true;
                }
            }
        }).start();
    }
```

```
}
```

思路三：lock + condition

```java
package com.thread.test;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @Author: 98050
 * @Time: 2019-06-11 21:31
 * @Feature:
 */
public class Test4 {

    static Lock lock = new ReentrantLock();
    static Condition condition1 = lock.newCondition();
    static Condition condition2 = lock.newCondition();
    static volatile int count = 1;

    static class MyThread implements Runnable{

        @Override
        public void run() {
            lock.lock();
            try {
                while (count <= 100){
                    System.out.println(Thread.currentThread().getName() + "打印: " + count++);
                    condition1.await();
                    condition2.signal();
                }
            }catch (Exception e ){

            }finally {
                lock.unlock();
            }
        }
    }

    static class MyThread2 implements Runnable{

        @Override
        public void run() {
            lock.lock();
            try {
                while (count <= 100){
                    System.out.println(Thread.currentThread().getName() + "打印: " + count++);
                    condition2.await();
                    condition1.signal();
```

```
                }
            }catch (Exception e ){

            }finally {
                lock.unlock();
            }
        }
    }

    public static void main(String[] args) {
        Thread t1 = new Thread(new Test2.MyThread());
        Thread t2 = new Thread(new Test2.MyThread2());

        t1.start();
        t2.start();
    }
}
```