## 1、什么是线程？

线程是操作系统中能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位

## 2、线程和进程有什么区别？

线程是进程的子集。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。每个线程都有单独的栈内存来存储本地数据。

## 3、如何在Java中实现线程

Thread类的实例就是一个线程但是它需要调用java.lang.Runnable接口来执行，由于线程类本身就是调用的Runnable接口所以可以继承Thread类或者实现Runnable接口。

## 4、用Runnable还是Thread

如果是要继承其他类，那么调用Runnable接口就好了

## 5、Thread类中的start()和run()有什么区别？

start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()才会启动新的线程。

## ============6、java中Runnable和Callable有什么区别？

Callable的call()可以返回值和抛出异常，而Runnable的run()没有这些功能。Callable可以返回装载有计算结果的Future对象

## 8 Java中CyclicBarrier 和 CountDownLatch有什么不同？

两者都可以让一组线程等待其他线程。CountDownLatch强调一个线程等多个线程，CyclicBarrier强调多个线程相互等待。

CountDownLatch，一个同步辅助类，在完成一组正在其他线程中执行的操作之前，它允许一个或多个线程一直等待。

主要方法

public CountDownLatch(int count); //参数指定了计数的次数

public void countDown(); //当线程调用此方法计数减一

public void await throws InterruptedException; //调用此方法会一直阻塞当前线程，直到计数器的值为0

CountDownLatch类使用如下：

public class CountDownLatchDemo {  
 final static SimpleDateFormat *sdf* = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  
  
 public static void main (String[] args) throws InterruptedException {  
 CountDownLatch count = new CountDownLatch(2);  
 Worker w1 = new Worker("zhang san", 5000, count);  
 Worker w2 = new Worker("li si", 8000, count);  
 w1.start();  
 w2.start();  
 count.await(); //表示在这里等待(判断哪个线程等待就看count在哪里定义的，这里是主线程)

System.*out*.println("all work completed at " + *sdf*.format(new Date()));  
 }  
  
 static class Worker extends Thread {  
 String workName;  
 int workTime;  
 CountDownLatch latch;  
 public Worker (String workName, int workTime,CountDownLatch latch) {  
 this.workName = workName;  
 this.workTime = workTime;  
 this.latch = latch;  
 }  
 public void run () {  
 System.*out*.println("Worker " + workName + " do work begin at " + *sdf*.format(new Date()));  
 doWork();  
 System.*out*.println("Worker " + workName + " do work end at " + *sdf*.format(new Date()));  
 latch.countDown();  
 }  
  
 private void doWork () {  
 try{  
 Thread.*sleep*(workTime);  
 }catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }

输出结果：

Worker zhang san do work begin at 2017-04-07 13:51:00

Worker li si do work begin at 2017-04-07 13:51:00

Worker zhang san do work end at 2017-04-07 13:51:05

Worker li si do work end at 2017-04-07 13:51:08

all work completed at 2017-04-07 13:51:08

如果将main方法中的count.await();注释掉，结果就会变成：

all work completed at 2017-04-07 13:53:56

Worker zhang san do work begin at 2017-04-07 13:53:56

Worker li si do work begin at 2017-04-07 13:53:56

Worker zhang san do work end at 2017-04-07 13:54:01

Worker li si do work end at 2017-04-07 13:54:04

可见count.await()方法的作用就是阻塞当前线程(当前线程的意思就是count.await()在哪个线程调用那个线程就是当前线程，在本例中就是main方法的主线程)直到latch.countDown()的计数变为0，其他线程完成各自的任务，然后恢复主线程。

该类通常用在如下场景：主线程开始执行前要等待n个线程完成各自的任务。

CyclicBarrier类使用实例如下：

public class TestCyclicBarrier {  
 private static final int *THREAD\_NUM* = 5;  
 static class WorkThread implements Runnable {  
 CyclicBarrier barrier;  
 public WorkThread (CyclicBarrier barrier) {  
 this.barrier = barrier;  
 }  
 public void run () {  
 try {  
 System.*out*.println("is waiting...");  
 barrier.await(); //线程在这里等待，直到所有线程到达barrier  
 System.*out*.println("ID " + Thread.*currentThread*().getId() + " is working...");  
 } catch (BrokenBarrierException e) {  
 e.printStackTrace();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
 public static void main (String[] args) {  
 CyclicBarrier barrier = new CyclicBarrier(*THREAD\_NUM*, new Runnable() {  
 @Override //当所有线程到达barrier时执行  
 public void run() {  
 System.*out*.println("Completed...");  
 }  
 });  
 for (int i = 0; i < *THREAD\_NUM*; i++) {  
 new Thread(new WorkThread(barrier)).start();  
 }  
 }  
}

输出结果：

is waiting...

is waiting...

is waiting...

is waiting...

is waiting...

Completed...

ID 13 is working...

ID 12 is working...

ID 15 is working...

ID 16 is working...

ID 14 is working...

## 9) Java内存模型是什么？

《Java并发编程实战》第十六章

## 10) Java中的volatile变量是什么？

使变量在各个线程之间透明

## 11) 什么是线程安全？Vector是一个线程安全类吗？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码，如果每次运行的结果和单线程运行的结果是一样的，而且其他变量的值也和预期是一样的，就是线程安全的。Vector是使用同步方法实现线程安全的，而和它相似的ArrayList不是线程安全的。

12) Java中什么是竞态条件？ 举个例子说明

13) Java中如何停止一个线程？

14) 一个线程运行时发生异常会怎样？

15） 如何在两个线程间共享数据？

16) Java中notify 和 notifyAll有什么区别？

17) 为什么wait, notify 和 notifyAll这些方法不在thread类里面？

18) 什么是ThreadLocal变量？

19) 什么是FutureTask？

20) Java中interrupted 和 isInterruptedd方法的区别？

21) 为什么wait和notify方法要在同步块中调用？

22) 为什么你应该在循环中检查等待条件?

23) Java中的同步集合与并发集合有什么区别？

24） Java中堆和栈有什么不同？

25） 什么是线程池？ 为什么要使用它？

26） 如何写代码来解决生产者消费者问题？

27） 如何避免死锁？

28) Java中活锁和死锁有什么区别？

29） 怎么检测一个线程是否拥有锁？

30) 你如何在Java中获取线程堆栈？

31) JVM中哪个参数是用来控制线程的栈堆栈小的

32） Java中synchronized 和 ReentrantLock 有什么不同？

33） 有三个线程T1，T2，T3，怎么确保它们按顺序执行？

34) Thread类中的yield方法有什么作用？

35） Java中ConcurrentHashMap的并发度是什么？

36） Java中Semaphore是什么？

37）如果你提交任务时，线程池队列已满。会时发会生什么？

38) Java线程池中submit() 和 execute()方法有什么区别？

39) 什么是阻塞式方法？

40) Swing是线程安全的吗？ 为什么？

41） Java中invokeAndWait 和 invokeLater有什么区别？

42) Swing API中那些方法是线程安全的？

43) 如何在Java中创建Immutable对象？

44） Java中的ReadWriteLock是什么？

45) 多线程中的忙循环是什么?

46）volatile 变量和 atomic 变量有什么不同？

47) 如果同步块内的线程抛出异常会发生什么？

48） 单例模式的双检锁是什么？

49） 如何在Java中创建线程安全的Singleton？

50) 写出3条你遵循的多线程最佳实践

51) 如何强制启动一个线程？

52) Java中的fork join框架是什么？

53） Java多线程中调用wait() 和 sleep()方法有什么不同？