#### 1、创建线程有几种不同的方式？你喜欢哪种？为什么？

有三种方式可以用来创建线程：

* 继承Thread类
  + 继承Thread类
  + 重写run()方法
  + 调用start()方法启动线程

public class ThreadTest extends Thread {

public ThreadTest() {  
 }  
  
 */\*\*  
 \* 重写run方法  
 \*/* @Override  
 public void run() {  
 for (int i = 0; i < 20; i++) {  
 System.*out*.println("来自线程" + Thread.*currentThread*().getName());  
 }  
 }  
  
 public static class Test {  
 public static void main(String[] args) {  
 ThreadTest thread1 = new ThreadTest();  
 ThreadTest thread2 = new ThreadTest();  
 //启动线程  
 thread1.start();  
 thread2.start();  
 }  
 }  
}

* 实现Runnable接口
  + 定义Runnable接口的实现类，重写该接口的run()方法，该run()方法的方法体是该线程的线程执行体。
  + 创建Runnable实现类的实例，并以此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。
  + 调用线程对象的start()方法来启动该线程。

public class RunnableTest implements Runnable {

*/\*\*  
 \* 重写run方法，线程执行体  
 \*/* @Override  
 public void run() {  
 for (int i = 0; i < 100; i++) {  
 // 当线程类实现Runnable接口时，  
 // 如果想获取当前线程，只能用Thread.currentThread()方法。  
 System.*out*.println(Thread.*currentThread*().getName() + " " + i);  
 }  
 }  
  
 public static void main(String[] args) {  
 RunnableTest thread = new RunnableTest();  
 // 通过new Thread(target , name)方法创建新线程  
 new Thread(thread, "新线程1").start();  
 new Thread(thread, "新线程2").start();  
 }  
}

* 应用程序可以使用Executor框架来创建线程池

#### 2、概括的解释下线程的几种可用状态

新建（new）：新建了一个线程对象；

可运行（runnable）：线程对象创建后，其他线程（比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取CPU的使用权；

运行（running）：可运行状态（runnable）的线程获得了CPU时间片（timelice），执行程序代码；

阻塞（block）：阻塞状态是指线程因为某种原因放弃了CPU使用权，也即让出了CPU timeslice，暂时停止运行。直到线程进入可运行状态（runnable）状态，才有机会再次获得CPU timeslice转到运行（running）状态。

阻塞的情况分三种：

1. 等待阻塞：运行（running）的线程执行o.wait()方法，JVM会把该线程放入等待队列（waiting queue）中。
2. 同步阻塞：运行（running）的线程在获取对象的同步锁时，若该同步锁被别的线程被别的线程占用，则JVM会把该线程放入锁池（lock pool）中。
3. 其他阻塞：运行（running）的线程执行Thread.sleep(long ms)或t.join()方法，或者发出了I/O请求时，JVM会把该线程设置为阻塞状态。当sleep()状态超时、join()等待线程中值或者超时、或者I/O处理完毕时，线程重新转入可运行（runnable）状态。

死亡（dead）：线程run()、main()方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。

#### 3、同步方法和同步代码块的区别是什么？

区别：

1. 同步方法默认用this或者当前类class对象作为锁；
2. 同步代码块可以选择以什么来加锁，比同步方法要更细粒度，我们可以选择只同步会发生同步问题的部分代码而不是整个方法；

#### 4、在监视器（Monitor）内部，是如何做线程同步的？程序应该做哪种级别的同步？

监视器和锁在Java虚拟机中的一块使用的。监视器监视一块同步代码块，确保一次只有一个线程执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码。

java还提供了显示监视器（Lock）和隐式监视器（synchronized）两种锁方案。

#### 什么是死锁（deadlock）？

两个线程或两个以上线程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是这些线程都陷入了无线的等待中。

#### 如何确保N个线程可以访问N个资源同时又不导致死锁？

多线程产生死锁的四个必要条件：

* 互斥条件：一个资源每次只能被一个进程使用。
* 保持和请求条件：一个进程因请求资源而阻塞时，对已获得资源保持不放。
* 不可剥夺性：进程已获得资源，在未使用完成前，不能被剥夺。
* 循环等待条件（闭环）：若干进程之间形成一种头尾相接的循环等待资源关系。

只要破坏其中任意一个条件，就可以避免死锁。

一个非常简单避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺讯获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了。