考试指导：java多线程设计模式详解之一 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/180/2021_2022__E8_80_83_E8_AF_95_E6_8C_87_E5_c97_180996.htm 线程的创建和启动 java语言已经内置了多线程支持，所有实现Runnable接口的类都可被启动一个新线程，新线程会执行该实例的run()方法，当run()方法执行完毕后，线程就结束了。一旦一个线程执行完毕，这个实例就不能再重新启动，只能重新生成一个新实例，再启动一个新线程。 Thread类是实现了Runnable接口的一个实例，它代表一个线程的实例，并且，启动线程的唯一方法就是通过Thread类的start()实例方法： Thread t = new Thread(). t.start(). start()方法是一个native方法，它将启动一个新线程，并执行run()方法。Thread类默认的run()方法什么也不做就退出了。注意：直接调用run()方法并不会启动一个新线程，它和调用一个普通的java方法没有什么区别。 因此，有两个方法可以实现自己的线程： 方法1：自己的类extend Thread，并复写run()方法，就可以启动新线程并执行自己定义的run()方法。例如： public class MyThread extends Thread { public run() { System.out.println("MyThread.run()"). } } 在合适的地方启动线程：new MyThread().start(). 方法2：如果自己的类已经extends另一个类，就无法直接extends Thread，此时，必须实现一个Runnable接口： public class MyThread extends OtherClass implements Runnable { public run() { System.out.println("MyThread.run()"). } } 为了启动MyThread，需要首先实例化一个Thread，并传入自己的MyThread实例： MyThread myt = new MyThread(). Thread t = new Thread(myt).

t.start(). 事实上，当传入一个Runnable target参数给Thread后，Thread的run()方法就会调用target.run()，参考JDK源代码：public void run() { if (target != null) { target.run(). } } 线程还有一些Name, ThreadGroup, isDaemon等设置，由于和线程设计模式关联很少，这里就不多说了。 线程同步 由于同一进程内的多个线程共享内存空间，在Java中，就是共享实例，当多个线程试图同时修改某个实例的内容时，就会造成冲突，因此，线程必须实现共享互斥，使多线程同步。 最简单的同步是将一个方法标记为synchronized，对同一个实例来说，任一时刻只能有一个synchronized方法在执行。当一个方法正在执行某个synchronized方法时，其他线程如果想要执行这个实例的任意一个synchronized方法，都必须等待当前执行synchronized方法的线程退出此方法后，才能依次执行。 但是，非synchronized方法不受影响，不管当前有没有执行synchronized方法，非synchronized方法都可以被多个线程同时执行。 此外，必须注意，只有同一实例的synchronized方法同一时间只能被一个线程执行，不同实例的synchronized方法是可以并发的。例如，class A定义了synchronized方法sync()，则不同实例a1.sync()和a2.sync()可以同时由两个线程来执行。 Java锁机制 多线程同步的实现最终依赖锁机制。我们可以想象某一共享资源是一间屋子，每个人都是一个线程。当A希望进入房间时，他必须获得门锁，一旦A获得门锁，他进去后就立刻将门锁上，于是B,C,D...就不得不在门外等待，直到A释放锁出来后，B,C,D...中的某一人抢到了该锁（具体抢法依赖于JVM的实现，可以先到先得，也可以随机挑选），然后进屋又将门锁上。这样，任一时刻最多有一人在屋内（

使用共享资源）。 Java语言规范内置了对多线程的支持。对于Java程序来说，每一个对象实例都有一把“ 锁 ”，一旦某个线程获得了该锁，别的线程如果希望获得该锁，只能等待这个线程释放锁之后。获得锁的方法只有一个，就是synchronized关键字。例如： public class SharedResource { private int count = 0. public int getCount() { return count. } public synchronized void setCount(int count) { this.count = count. } } 同步方法public synchronized void setCount(int count) { this.count = count. } 事实上相当于： public void setCount(int count) { synchronized(this) { // 在此获得this锁 this.count = count. } // 在此释放this锁 } 红色部分表示需要同步的代码段，该区域为“ 危险区域 ”，如果两个以上的线程同时执行，会引发冲突，因此，要更改SharedResource的内部状态，必须先获得SharedResource实例的锁。 退出synchronized块时，线程拥有的锁自动释放，于是，别的线程又可以获取该锁了。 为了提高性能，不一定要锁定this，例如，SharedResource有两个独立变化的变量： public class SharedResouce { private int a = 0. private int b = 0. public synchronized void setA(int a) { this.a = a. } public synchronized void setB(int b) { this.b = b. } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com