1. 线程八锁的解释;
   1. 首先我们必须明确synchronized 同步锁的原理: synchronized 锁的不是方法，而是synchronized所在对象的整个对象；

|  |
| --- |
| class ThreadResource{  public synchronized void sendEmial(){  System.out.println("---发送邮件");  }    public synchronized void endSMS(){  System.out.println("----发送短信");  }  } |

在该类中，synchronized 不是锁的单个的方法而是，锁的整个ThreadResource对象。

\*即：一个对象里面如果与多个synchronized 方法，某一时刻内，只要有一个线程去调用了其中的一个synchronized 的方法，其它线程就只能等待，换句话说某一时刻内只能有唯一一个线程去访问这些synchronized 方法，锁的是当前对象this ，被锁定后，其它的线程都不能进入到当前对象的synchronized 方法里面。 （1，2锁的答案）

\*加一个普通方法后发现与同步锁无关，换成两个对象后不是同一把锁了，情况也变了。（3，4锁的答案）

\*换成静态方法后，情况又变化了，所有的非静态同步方法用的是同一把锁——实例化对象本身：即: static synchronized 锁的是类，与实例化对象无关，无论实例化了多少对象，静态类方法都只能有一个线程进入。（5，6，锁的答案）（7，8锁的答案有以上几点综合得出）

总结:

|  |
| --- |
| 所有的非静态同步方法用的都是同一把锁—实例对象本身，  synchronized实现同步的基础: Java中的每一个对象都可以作为锁。具体表现为以下3种形式。  对于普通同步方法，锁是当前实例对象。  对于静态同步方法，锁是当前类的class对象。  对于同步方法块，锁是Synchonized括号里配置的对象。  当一个线程试图访问同步代码块时，它首先必须得到锁，退出或抛出异常时必须释放锁。  也就是说如果一个实例对象的非静态同步方法获取锁后，该实例对象的其他非静态同步方法必须等待获取锁的方法释放锁后才能获取锁，可是别的实例对象的非静态同步方法因为跟该实例对象的非静态同步方法用的是不同的锁，  所以毋须等待该实例对象已获取锁的非静态同步方法释放锁就可以获取他们自己的锁。  所有的静态同步方法用的也是同一把锁—类对象本身，这两把锁（this/Class）是两个不同的对象，所以静态同步方法和非静态同步方法里面是不存在竞争关系的。 |

* 1. 首先我们先将八锁的举例列出：

·标准访问，请问使先打印邮件还是短信；/邮件

·邮件暂停四秒，请问使先打印邮件还是短信；/邮件

·新增一个普通方法，请问使先打印邮件还是普通方法；

/普通方法

·两个实例化对象，请问使先打印邮件还是短信 /短信

·两个静态同步方法，同一个对象，请问使先打印邮件还是短信

·两个静态同步方法，两个对象，请问使先打印邮件还是短信

·一个普通方法，一个静态方法，一个对象，请问使先打印邮件还是短信

·一个普通方法，一个静态同步方法，两个对象，请问使先打印邮件还是短信

（二锁到八锁发送邮件都有一个4秒延迟）