* **释放锁的操作**
* 当前线程的同步方法、同步代码块执行结束。
* 当前线程在同步代码块、同步方法中遇到break、return终止了该代码块、该方法的继续执行。
* 当前线程在同步代码块、同步方法中出现了未处理的Error或Exception，导致异常结束
* 当前线程在同步代码块、同步方法中执行了线程对象的**wait()方法**，当前线程暂停，并释放锁。
* **不会释放锁的操作**
* 线程执行同步代码块或同步方法时，程序调用**Thread.sleep()、Thread.yield()**方法暂停当前线程的执行。
* 线程执行同步代码块时，其他线程调用了该线程的**suspend()**方法将该线程挂起，该线程不会释放锁（同步监视器）。
* 应尽量避免使用suspend()和resume()来控制线程。
* **线程的死锁问题**
* 死锁：
* 不同的线程分别占用对方需要的同步资源不放弃，都在等待对方放弃自己需要的同步资源，就形成了线程的死锁。
* 解决办法：
* 专门的算法、原则；
* 尽量减少同步资源的定义。

public class TestDeadLock {  
 static StringBuffer sb1 = new StringBuffer();  
 static StringBuffer sb2 = new StringBuffer();  
  
 public static void main(String[] args) {  
 new Thread() {  
 @Override  
 public void run() {  
 synchronized (sb1) {  
 try {  
 Thread.sleep(10);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 sb1.append("A");  
 synchronized (sb2) {  
 sb2.append("B");  
 System.out.println(sb1);  
 System.out.println(sb2);  
 }  
 }  
 }  
 }.start();  
  
 new Thread() {  
 @Override  
 public void run() {  
 synchronized (sb2) {  
 try {  
 Thread.sleep(10);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 sb1.append("C");  
 synchronized (sb1) {  
 sb2.append("D");  
 System.***out***.println(sb1);  
 System.***out***.println(sb2);  
 }  
 }  
 }  
 }.start();  
  
 }  
}