## 一、java线程安全

线程安全类：当多个线程访问某个类时，这个类始终保持正确行为那么这个类是线程安全的

无状态对象:既不包含任何域，也不包含对任何其他域的引用。使用的局部变量存于栈中的局部变量里，只能由正在执行的线程访问

Java提供了一种内置的锁机制来支持原子性：Synchronized关键字。同步代码块包括两部分：一个作为锁的对象引用，一个由该锁保护的代码块。以synchronized修饰的方法的锁就是方法调用所在的对象，静态的synchronized方法以Class对象作为锁。

线程进入同步代码块之前会自动获得锁，并在退出同步代码块时自动释放锁。

重入：如果某个线程视图获得一个已经由它自己持有的锁，那么这个请求就会成功。实现方法：为每个锁关联一个获取计数值和一个所有者线程；当计数值为0时这个锁被认为是没有被任何线程持有。当线程请求一个未被持有的锁时，jvm将记下锁的持有者并将获取计数值置位1.如果同一个线程再次获取这个锁，计数值将+1，当线程退出同步代码块时计数器会相应递减，计数值为0时这个锁会被释放。

当执行较长时间的计算或者可能无法快速完成的操作时（网络I/O或者控制台I/O）,一定不要持有锁。

Volatile变量：确保将变量的更新操作通知其他线程.当变量声明为volatile类型后，编译器与运行时都会注意到这个变量是共享的，因此不会讲该变量上的操作与其他内存操作一起重排序。Volatile变量不会缓存在寄存器或者其他处理器不可见的地方，因此读取volatile类型的变量时总会返回最新写入的值。访问volatile变量时不会执行加锁操作，也不会使执行线程阻塞，因此volatile是一种比sychronized关键字更轻量级的同步机制

当且仅当满足以下所有条件时，才应该使用volatile变量：

对变量的写入操作不依赖变量当前的值，或者只有单个线程更新变量的值

该变量不会与其他状态变量一起纳入不变性条件中

访问变量时不需要加锁

ThreadLocal类:维持线程封闭性的方法。这个类能使线程中的某个值与保存值的对象关联起来，ThreadLocal类提供了get与set等访问接口或方法，这些方法为每个使用该变量的线程都存有一份独立的副本，因为get总返回由当前执行线程在调用set时设置的最新值。

ThreadLocal对象通常用于放置对可变的单实例变量或全局变量进行共享。比如将JDBC链接保存到ThreadLocal对象中，每个线程都会拥有属于自己的连接。

## 二、java克隆

浅克隆 对象中其他对象的成员变量会克隆引用；

深克隆 会对对象成员变量也克隆(实现深克隆的方法：序列化与反序列化(需要实现Serializable接口))

public Object deepClone()

{

//将对象写到流里

ByteArrayOutoutStream bo=new ByteArrayOutputStream();

ObjectOutputStream oo=new ObjectOutputStream(bo);

oo.writeObject(this);

//从流里读出来

ByteArrayInputStream bi=new ByteArrayInputStream(bo.toByteArray());

ObjectInputStream oi=new ObjectInputStream(bi);

return(oi.readObject());

}

这样做的前提是对象以及对象内部所有引用到的对象都是可串行化的