 Java对多线程的支持与同步机制深受大家的喜爱，似乎看起来使用了synchronized关键字就可以轻松地解决多线程共享数据同步问题。到底如何？——还得对synchronized关键字的作用进行深入了解才可定论。 总的说来，synchronized关键字可以作为函数的修饰符，也可作为函数内的语句，也就是平时说的同步方法和同步语句块。如果再细的分类，synchronized可作用于instance变量、object reference（对象引用）、static函数和class literals(类名称字面常量)身上。 在进一步阐述之前，我们需要明确几点： A．无论synchronized关键字加在方法上还是对象上，它取得的锁都是对象，而不是把一段代码或函数当作锁——而且同步方法很可能还会被其他线程的对象访问。 B．每个对象只有一个锁（lock）与之相关联。 C．实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。 接着来讨论synchronized用到不同地方对代码产生的影响：假设P1、P2是同一个类的不同对象，这个类中定义了以下几种情况的同步块或同步方法，P1、P2就都可以调用它们。 1． 把synchronized当作函数修饰符时，示例代码如下： Public synchronized void methodAAA() { //…. } 这也就是同步方法，那这时synchronized锁定的是哪个对象呢？它锁定的是调用这个同步方法对象。也就是说，当一个对象P1在不同的线程中执行这个同步方法时，它们之间会形成互斥，达到同步的效果。但是这个对象所属的Class所产生的另一对象P2却可以任

意调用这个被加了synchronized关键字的方法。 上边的示例代码等同于如下代码： public void methodAAA() { synchronized (this) // (1) { //…..} } (1)处的this指的是什么呢？它指的就是调用这个方法的对象，如P1。可见同步方法实质是将synchronized作用于object reference。——那个拿到了P1对象锁的线程，才可以调用P1的同步方法，而对P2而言，P1这个锁与它毫不相干，程序也可能在这种情形下摆脱同步机制的控制，造成数据混乱：（ 2. 同步块，示例代码如下： public void method3(SomeObject so) { synchronized(so) { //…..} } 这时，锁就是so这个对象，谁拿到这个锁谁就可以运行它所控制的那段代码。当有一个明确的对象作为锁时，就可以这样写程序，但当没有明确的对象作为锁，只是想让一段代码同步时，可以创建一个特殊的instance变量（它得是一个对象）来充当锁： class Foo implements Runnable { private byte[] lock = new byte[0]. // 特殊的instance变量 