多线程支持和线程安全（3） PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/141/2021_2022__E5_A4_9A_E7_BA_BF_E7_A8_8B_E6_c29_141241.htm 它这里有一个非常重要的概念，“ 对象锁 ”。当系统中某个线程调用一个被同步的方法（临界区）访问某个对象时，这个对象就会被锁住。我们可以把一个对象比喻成一扇门，并且门锁的钥匙被放在门旁。当某个线程调用一个对象被同步（synchronized）了的方法时，它取走钥匙，进入房间里面，然后把门锁上，这相当于线程锁住了这个对象。此时，如果有其它线程试图调用这个对象的被同步的方法（可能不是刚才的那个方法）而访问这个对象时，它就不能找到打开这扇门的钥匙，于是它就处于不断地监测与等待当中（查找锁状态）。直到房间里面的线程（也即拥有这个对象锁的线程）退出被同步的方法时，并且，它按照规则把钥匙再次放在门旁，于是后来的其它线程便拥有了执行的机会。 上面描述的过程是线程间的互斥过程，同样线程间的同步控制也离不开对象锁，这一点从图10-59中也可以很好地被反映出。例如，线程A处于运行状态时，如果调用某个对象的wait()操作，线程A则进入阻塞的等待状态，并且线程A同时放弃拥有这个对象锁（注意，线程A调用对象的wait()操作时，必须已经拥有了该对象锁）。此时系统中其它线程因此便可以有机会来获取到这个对象锁，并访问这个共享对象。如果线程B调用了这个对象的notify()方法时，那么线程A将被唤醒，并且进入到查找锁状态，如此不断循环往复，以实现线程之间的同步控制。当然处于阻塞等待状态中的线程，如果被外部中断（interrupt）或被终止

（stop）运行时，它也将会退出阻塞等待状态。 从以上可以看出，JDK平台中，线程之间互斥与同步的实现，以及它的状态转换模型是非常严谨的，同时对于程序员而言，它也是比较容易理解掌握和易于使用的。另外在JDK平台中线程库模块中，也有一个负责管理线程的创建、运行和销毁等功能的实现类，那就是Thread类型。2、 线程安全性设计 基础库系统中对多线程有了很好的支持，这无疑是一件很好的事情，但这同时也带来了另一个比较棘手的设计问题。那就是基础库系统中，各组件模块是否应该很好地支持线程安全，这非常非常重要，如果处理不当，轻则导致程序运行时数据信息的不一致，或业务的执行错误；严重的话，将导致系统崩溃，而且设计不当也会很大程序上影响到程序的执行效率。比较典型例子，如C运行库中的errno变量，这是一个在C库中定义的全局性变量，用它来设置并指示，一些在系统调用时（或库中的功能函数的执行过程中）发生的错误信息。这在单线程工作环境的应用程序系统中并没有什么问题，但是如果多线程工作环境下，则可能出现不一致的现象，有可能线程A刚设置的errno值被线程B覆盖掉。怎样很好地解决这个问题？使得C库能很好地支持多线程的安全。较好的方法是利用线程本地存储数据机制，也即利用每个线程都有一些只与自己线程相关的存储信息。例如，每个线程的环境中都有一个errno变量，当线程切换时，线程本地存储数据也随之切换。 当然基础库中有其它更多不同种类的线程安全问题。如前面的章节中所阐述过的I/O平台的线程安全等，还有JDK平台中的SWING组件库。这在设计上的确是遇到了一个两难选择的尴尬境地。如果基础库中的组件支持很好的线程安全性，

那么它将会牺牲掉很多的运行效率；反过来也一样存在很大的问题。怎么办？现在大家取得基本共识的设计方案是比较倾向于后者，也即运行的效率。在保证基础库系统中的组件拥有最基本线程安全特性外，一般都把线程间的互斥与同步访问控制权，交给用户来掌握。