JavaThread应该注意的问题 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_JavaThread_c104_145287.htm Java的线程编程非常简单。但有时会看到一些关于线程的错误用法。下面列出一些应该注意的问题。 1. 同步对象的恒定性 All java objects are references. 对于局部变量和参数来说，java里面的int, float, double, boolean等基本数据类型，都在栈上。这些基本类型是无法同步的；java里面的对象（根对象是Object），全都在堆里，指向对象的reference在栈上。 java中的同步对象，实际上是对于reference所指的“对象地址”进行同步。 需要注意的问题是，千万不要对同步对象重新赋值。举个例子。 class A implements Runnable { Object lock = new Object(). void run() { for(...) { synchronized(lock) { // do something ... lock = new Object(). } } } } run函数里面的这段同步代码实际上是毫无意义的。因为每一次lock都给重新分配了新的对象的reference，每个线程都在新的reference同步。大家可能觉得奇怪，怎么会举这么一个例子。因为我见过这样的代码，同步对象在其它的函数里被重新赋了新值。 这种问题很难查出来。所以，一般应该把同步对象声明为final. final Object lock = new Object(). 使用Singleton Pattern 设计模式来获取同步对象，也是一种很好的选择。 2．如何放置共享数据 实现线程，有两种方法，一种是继承Thread类，一种是实现Runnable接口。 上面举的例子，采用实现Runnable接口的方法。本文推荐这种方法。 首先，把需要共享的数据放在一个实现Runnable接口的类里面，然后，把这个类的实例传给多个Thread的构造方法。这样，新创建的多个Thread，都共

同拥有一个Runnable实例，共享同一份数据。 如果采用继承Thread类的方法，就只好使用static静态成员了。如果共享的数据比较多，就需要大量的static静态成员，令程序数据结构混乱，难以扩展。这种情况应该尽量避免。 编写一段多线程代码，处理一个稍微复杂点的问题。两种方法的优劣，一试便知。 3．同步的粒度 线程同步的粒度越小越好，即，线程同步的代码块越小越好。尽量避免用synchronized修饰符来声明方法。尽量使用synchronized(anObject)的方式，如果不想引入新的同步对象，使用synchronized(this)的方式。而且，synchronized代码块越小越好。 4．线程之间的通知 这里使用“ 通知 ”这个词，而不用“ 通信 ”这个词，是为了避免词义的扩大化。 线程之间的通知，通过Object对象的wait() 和notify() 或notifyAll() 方法实现。 下面用一个例子，来说明其工作原理： 假设有两个线程，A和B。共同拥有一个同步对象，lock。 1．首先，线程A通过synchronized(lock) 获得lock同步对象，然后调用lock.wait()函数，放弃lock同步对象，线程A停止运行，进入等待队列。 2．线程B通过synchronized(lock) 获得线程A放弃的lock同步对象，做完一定的处理，然后调用lock.notify() 或者lock.notifyAll() 通知等待队列里面的线程A。 3．线程A从等待队列里面出来，进入ready队列，等待调度。 4．线程B继续处理，出了synchronized(lock)块之后，放弃lock同步对象。 5．线程A获得lock同步对象，继续运行。 