# java多线程工作总结(推荐7篇)

来源：网络 作者：轻吟低唱 更新时间：2025-01-22

*java多线程工作总结1前台线程：是指接受后台线程服务的线程，其实前台后台线程是联系在一起，就像傀儡和幕后操纵者一样的关系。傀儡是前台线程、幕后操纵者是后台线程。由前台线程创建的线程默认也是前台线程。可以通过isDaemon()和setDa...*

**java多线程工作总结1**

前台线程：是指接受后台线程服务的线程，其实前台后台线程是联系在一起，就像傀儡和幕后操纵者一样的关系。傀儡是前台线程、幕后操纵者是后台线程。由前台线程创建的线程默认也是前台线程。可以通过isDaemon()和setDaemon()方法来判断和设置一个线程是否为后台线程。线程类的一些常用方法：

sleep(): 强迫一个线程睡眠Ｎ毫秒。isAlive(): 判断一个线程是否存活。join(): 等待线程终止。activeCount(): 程序中活跃的线程数。enumerate(): 枚举程序中的线程。currentThread(): 得到当前线程。isDaemon(): 一个线程是否为守护线程。setDaemon(): 设置一个线程为守护线程。(用户线程和守护线程的区别在于，是否等待主线程依赖于主线程结束而结束)setName(): 为线程设置一个名称。wait(): 强迫一个线程等待。notify(): 通知一个线程继续运行。setPriority(): 设置一个线程的优先级。

**java多线程工作总结2**

如果一个类继承Thread，则不适合资源共享。但是如果实现了Runable接口的话，则很容易的实现资源共享。

总结：实现Runnable接口比继承Thread类所具有的优势：1）：适合多个相同的程序代码的线程去处理同一个资源2）：可以避免java中的单继承的限制3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立

有个问题需要说明一下，main方法其实也是一个线程。在java中所有的线程都是同时启动的，至于什么时候，哪个先执行，完全看谁先得到CPU的资源。

在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个JVM，每一个JVM实际上就是在操作系统中启动了一个进程。

**java多线程工作总结3**

启动类

实现Runable的优点

名词解释

让当前cpu执行的线程等待  直到调用join( ) 的那个线程执行完成后 继续执行等待的线程

不传入时间默认join(0) 传入0 当调用join() 的线程isAlive活着的时候  是让调用join的线程执行完成后 再执行其他线程

synchronized（this）  其中this 指的是锁住当前类的对象， 所以当是继承Thread时  实例化多个Thread对象并不能 锁住多个对象的方法

synchronize 被修饰在方法和代码块上  使用在代码块上更加能提高效率

结论：

>线程调度算法：

实例

**java多线程工作总结4**

1、synchronized关键字的作用域有二种：

1）是某个对象实例内，synchronized aMethod(){}可以防止多个线程同时访问这个对象的synchronized方法（如果一个对象有多个synchronized方法，只要一个线程访问了其中的一个synchronized方法，其它线程不能同时访问这个对象中任何一个synchronized方法）。这时，不同的对象实例的synchronized方法是不相干扰的。也就是说，其它线程照样可以同时访问相同类的另一个对象实例中的synchronized方法；

2）是某个类的范围，synchronized static aStaticMethod{}防止多个线程同时访问这个类中的synchronized static 方法。它可以对类的所有对象实例起作用。

2、除了方法前用synchronized关键字，synchronized关键字还可以用于方法中的某个区块中，表示只对这个区块的资源实行互斥访问。用法是: synchronized(this){/区块/}，它的作用域是当前对象；

3、synchronized关键字是不能继承的，也就是说，基类的方法synchronized f(){} 在继承类中并不自动是synchronized f(){}，而是变成了f(){}。继承类需要你显式的指定它的某个方法为synchronized方法；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：

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：

(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：

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：

注：零长度的byte数组对象创建起来将比任何对象都经济――查看编译后的字节码：生成零长度的byte[]对象只需3条操作码，而Object lock = new Object()则需要7行操作码。3．将synchronized作用于static 函数，示例代码如下：

代码中的methodBBB()方法是把class literal作为锁的情况，它和同步的static函数产生的效果是一样的，取得的锁很特别，是当前调用这个方法的对象所属的类（Class，而不再是由这个Class产生的某个具体对象了）。记得在《Effective Java》一书中看到过将 和 ()用于作同步锁还不一样，不能用()来达到锁这个Class的目的。P1指的是由Foo类产生的对象。可以推断：如果一个类中定义了一个synchronized的static函数A，也定义了一个synchronized 的instance函数B，那么这个类的同一对象Obj在多线程中分别访问A和B两个方法时，不会构成同步，因为它们的锁都不一样。A方法的锁是Obj这个对象，而B的锁是Obj所属的那个Class。

1、线程同步的目的是为了保护多个线程反问一个资源时对资源的破坏。

2、线程同步方法是通过锁来实现，每个对象都有切仅有一个锁，这个锁与一个特定的对象关联，线程一旦获取了对象锁，其他访问该对象的线程就无法再访问该对象的其他非同步方法。

3、对于静态同步方法，锁是针对这个类的，锁对象是该类的Class对象。静态和非静态方法的锁互不干预。一个线程获得锁，当在一个同步方法中访问另外对象上的同步方法时，会获取这两个对象锁。

4、对于同步，要时刻清醒在哪个对象上同步，这是关键。

5、编写线程安全的类，需要时刻注意对多个线程竞争访问资源的逻辑和安全做出正确的判断，对“原子”操作做出分析，并保证原子操作期间别的线程无法访问竞争资源。

6、当多个线程等待一个对象锁时，没有获取到锁的线程将发生阻塞。

7、死锁是线程间相互等待锁锁造成的，在实际中发生的概率非常的小。真让你写个死锁程序，不一定好使，呵呵。但是，一旦程序发生死锁，程序将死掉。

**java多线程工作总结5**

CAS compare and swap

结构图：

解释：先读取当前值 a  假设  a = 1,修改后再次读取a的值 ，假如两次一样 则可以修改，不一样不能修改。

应用：java concurrent 包下面的 原子类的操作。如： AtomicInteger

底层：调用汇编指令 lock cmpxchg

volatile

1 :  保持线程可见性。

解释： 线程会读取常量池的数据，修改的话会同步到常量池中，因为上面的例子中，子线程睡眠2秒导致主线程执行了 while的内存，所以不会再主动修改a的数值。

加上volatile 关键字后： 相当于每一次获取值是直接从常量池 直接重新拿生成的新的副本。

2：禁止指令重排 加上 lock

在单例设计模式中，双重check加锁实现多线程的单例模式，在定义class时，在上volatile防止在指令重排时，对象new 一半的时候返回。

synchronized

在>JVM 有关于对象的内存结构的详细分析，我们知道锁信息是存放在对象头信息中的。

synchronized :过程分析，

      假如现在头信息存储的线程id是1，线程1去获取锁的时候首选读取是1，然修改成1，然后再读一次对象头的线程id还是1，则算获取到锁。线程2也执行类型过程：首先读取线程id是1，然后准备修改为2，修改前再去读一次还是1，则修改成功，这个过程只有一个线程能获取到当前锁。 这个过程叫做>自旋锁。

就是jvm给synchronized这个方法做了一些优化，并不是最开始就使用指令把同步代码块锁住，当一段同步代码没有被线程访问的时候，这个时候是处于无锁状态（0），如果这个同步代码块一只被同一个线程访问，那么jvm会给这段代码块加上偏量锁（1），如果不断的有线程进来，那么其他线程开始竞争的时候使用的是cas方式去竞争，这个时候jvm又会把锁升级成自旋锁（00），如果10次都没获取到锁，则进入重量级锁（11），开启指令。

用户态：大部分操作程序叫 用户态，但是一些特殊操作需要调用内核。比如加锁。

内核态：指的是和硬件的内核操作。

AQS

AbstractQueuedSynchronizer（以下简写AQS）这个抽象类

sync : 上面说了 是 偏向锁  自旋锁 重量级锁的演变

ReentrantLock ：jvm实现

加锁过程：判断是否有锁，没有 -> 判断是否需要排队，不需要就直接返回。有锁，入队，

>TODO

1.使用park 实现锁

**java多线程工作总结6**

在传统的同步开发模式下，当我们调用一个函数时，通过这个函数的参数将数据传入，并通过这个函数的返回值来返回最终的计算结果。但在多线程的异步开发模式下，数据的传递和返回和同步开发模式有很大的区别。由于线程的运行和结束是不可预料的，因此，在传递和返回数据时就无法象函数一样通过函数参数和return语句来返回数据。

、通过构造方法传递数据在创建线程时，必须要建立一个Thread类的或其子类的实例。因此，我们不难想到在调用start方法之前通过线程类的构造方法将数据传入线程。并将传入的数据使用类变量保存起来，以便线程使用(其实就是在run方法中使用)。下面的代码演示了如何通过构造方法来传递数据：

由于这种方法是在创建线程对象的同时传递数据的，因此，在线程运行之前这些数据就就已经到位了，这样就不会造成数据在线程运行后才传入的现象。如果要传递更复杂的数据，可以使用集合、类等数据结构。使用构造方法来传递数据虽然比较安全，但如果要传递的数据比较多时，就会造成很多不便。由于Java没有默认参数，要想实现类似默认参数的效果，就得使用重载，这样不但使构造方法本身过于复杂，又会使构造方法在数量上大增。因此，要想避免这种情况，就得通过类方法或类变量来传递数据。

、通过变量和方法传递数据向对象中传入数据一般有两次机会，第一次机会是在建立对象时通过构造方法将数据传入，另外一次机会就是在类中定义一系列的public的方法或变量（也可称之为字段）。然后在建立完对象后，通过对象实例逐个赋值。下面的代码是对MyThread1类的改版，使用了一个setName方法来设置 name变量：

、通过回调函数传递数据上面讨论的两种向线程中传递数据的方法是最常用的。但这两种方法都是main方法中主动将数据传入线程类的。这对于线程来说，是被动接收这些数据的。然而，在有些应用中需要在线程运行的过程中动态地获取数据，如在下面代码的run方法中产生了3个随机数，然后通过Work类的process方法求这三个随机数的和，并通过Data类的value将结果返回。从这个例子可以看出，在返回value之前，必须要得到三个随机数。也就是说，这个 value是无法事先就传入线程类的。

**java多线程工作总结7**

1、新建状态（New）：新创建了一个线程对象。

2、就绪状态（Runnable）：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态（Running）：就绪状态的线程获取了CPU的使用权限，执行程序代码。

4、阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

（一）、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。（二）、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。（三）、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5、死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

本DOCX文档由 www.zciku.com/中词库网 生成，海量范文文档任你选，，为你的工作锦上添花,祝你一臂之力！