**[java之yield(),sleep(),wait()区别详解-备忘笔记](http://dylanxu.iteye.com/blog/1322066)**

**1、sleep()**

使当前线程（即调用该方法的线程）暂停执行一段时间，让其他线程有机会继续执行，但它并不释放对象锁。也就是说如果有synchronized同步快，其他线程仍然不能访问共享数据。注意该方法要捕捉异常。

例如有两个线程同时执行(没有synchronized)一个线程优先级为MAX\_PRIORITY，另一个为MIN\_PRIORITY，如果没有Sleep()方法，只有高优先级的线程执行完毕后，低优先级的线程才能够执行；但是高优先级的线程sleep(500)后，低优先级就有机会执行了。

总之，sleep()可以使低优先级的线程得到执行的机会，当然也可以让同优先级、高优先级的线程有执行的机会。

**2、join()**

join()方法使调用该方法的线程在此之前执行完毕，也就是等待该方法的线程执行完毕后再往下继续执行。注意该方法也需要捕捉异常。

**3、yield()**

该方法与sleep()类似，只是不能由用户指定暂停多长时间，并且yield（）方法只能让同优先级的线程有执行的机会。

**4、wait()和notify()、notifyAll()**

这三个方法用于协调多个线程对共享数据的存取，所以必须在synchronized语句块内使用。synchronized关键字用于保护共享数据，阻止其他线程对共享数据的存取，但是这样程序的流程就很不灵活了，如何才能在当前线程还没退出synchronized数据块时让其他线程也有机会访问共享数据呢？此时就用这三个方法来灵活控制。

wait()方法使当前线程暂停执行并释放对象锁标示，让其他线程可以进入synchronized数据块，当前线程被放入对象等待池中。当调用notify()方法后，将从对象的等待池中移走一个任意的线程并放到锁标志等待池中，只有锁标志等待池中线程能够获取锁标志；如果锁标志等待池中没有线程，则notify()不起作用。

notifyAll()则从对象等待池中移走所有等待那个对象的线程并放到锁标志等待池中。

**注意 这三个方法都是java.lang.Object的方法。**

**二、run和start()**

把需要处理的代码放到run()方法中，start()方法启动线程将自动调用run()方法，这个由java的内存机制规定的。并且run()方法必需是public访问权限，返回值类型为void。

**三、关键字synchronized**

该关键字用于保护共享数据，当然前提条件是要分清哪些数据是共享数据。每个对象都有一个锁标志，当一个线程访问到该对象，被Synchronized修饰的数据将被"上锁"，阻止其他线程访问。当前线程访问完这部分数据后释放锁标志，其他线程就可以访问了。

**四、wait()和notify(),notifyAll()是Object类的方法，sleep()和yield()是Thread类的方法。**

**(1)、常用的wait方法有wait()和wait(long timeout);**

void wait() 在其他线程调用此对象的 notify() 方法或者 notifyAll()方法前，导致当前线程等待。

void wait(long timeout)在其他线程调用此对象的notify() 方法 或者 notifyAll()方法，或者超过指定的时间量前，导致当前线程等待。

wait()后，线程会释放掉它所占有的“锁标志”，从而使线程所在对象中的其他shnchronized数据可被别的线程使用。

wait()h和notify()因为会对对象的“锁标志”进行操作，所以他们必需在Synchronized函数或者 synchronized block 中进行调用。如果在non-synchronized 函数或 non-synchronized block 中进行调用，虽然能编译通过，但在运行时会发生IllegalMonitorStateException的异常。。

**(2)、Thread.sleep(long millis)必须带有一个时间参数。**

sleep(long)使当前线程进入停滞状态，所以执行sleep()的线程在指定的时间内肯定不会被执行；

sleep(long)可使优先级低的线程得到执行的机会，当然也可以让同优先级的线程有执行的机会；

sleep(long)是不会释放锁标志的。

**(3)、yield()没有参数**

sleep 方法使当前运行中的线程睡眠一段时间，进入不可以运行状态，这段时间的长短是由程序设定的，yield方法使当前线程让出CPU占有权，但让出的时间是不可设定的。

yield()也不会释放锁标志。

实际上，yield()方法对应了如下操作；先检测当前是否有相同优先级的线程处于同可运行状态，如有，则把CPU的占有权交给次线程，否则继续运行原来的线程，所以yield()方法称为“退让”，它把运行机会让给了同等级的其他线程。

sleep 方法允许较低优先级的线程获得运行机会，但yield（）方法执行时，当前线程仍处在可运行状态，所以不可能让出较低优先级的线程此时获取CPU占有权。在一个运行系统中，如果较高优先级的线程没有调用sleep方法，也没有受到I/O阻塞，那么较低优先级线程只能等待所有较高优先级的线程运行结束，方可有机会运行。

yield()只是使当前线程重新回到可执行状态，所有执行yield()的线程有可能在进入到可执行状态后马上又被执行，所以yield()方法只能使同优先级的线程有执行的机会。