**进程与线程：**

1. 多线程是实现并发机制的一种手段。
2. 进程和线程一样，都是实现并发执行的一个基本单位。
3. 线程是比进程更小的的执行单位，是在进程的基础上进一步划分。
4. 多线程是指一个进程在执行过程中可以产生多个更小的程序单元，即线程，这些线 程可以同时存在、同时运行，而一个进程可能包含了多个同时执行的线程。
5. 线程是指程序的运行流程，多线程的进制则是可以同时运行多个程序块，使程序运 行的效率变得更高。

**线程的实现：**

在Java中，实现多线程有两种方式：一是继承Thread类，二是实现Runnable接口。

**继承Thread类实现多线程：**

1）Thread类定义在java.lang包中，一个类只要继承了Thread类，此类就称为多线程操 作类，此类的对象表示线程对象，每一个线程对象可以表示一个线程。

2）继承Thread类后，必须覆写Thread类中的run( )方法，此方法是线程的主体。

**class 类名称 extends Thread{**

**属性...;**

**方法...;**

**public void run( ){  
 线程主体**

**}**

**}**

3）通过线程对象调用Thread类中的start( )方法启动线程

1. 如果通过继承Thread类实现多线程的话，那么每个线程对象只能调用一次start( ) 方法，否则会出现IlleglThreadStateException异常

**实现Runnable接口实现多线程：**

1. 也可以通过Runnable接口实现多线程，一个类只要是实现了Runnable接口，此 类就称为多线程操作类，每一个该类对象可以表示一个线程。
2. 实现Runnable接口后，必须覆写Runnable接口中的run( )方法，此方法是线程的主 体。

**class 类名称 extends Runnable{**

**属性...;**

**方法...;**

**public void run( ){  
 线程主体；  
 }**

**}**

3）需要依靠Thread启动多线程，在Thread类中存在构造方法public Thread(Runnable target)，此方法可以接受Runnable接口的子类对象。

**两种实现多线程方式的区别：**

1. Thread类是Runnable接口的子类，通过start( )方法默认启动多线程时调用的run( ) 方法实际上是调用的Runnable接口中的run( )方法
2. 继承0Thread类，可以直接调用start( )方法启动多线程；实现Runnable接口，需要 通过Thread类对象启动多线程。
3. 如果一个类继承Thread类，则不适合多个线程共享同一资源，而实现了Runnable 接口，可以方便的实现资源共享。

**线程的运行原则：**

在Java中所有的线程都是同时启动的，线程之间是交错运行的，哪个线程对象抢到了 CPU资源，哪个线程就可以运行。但线程抢到CPU资源的速度是极快的，这就让多 个线程运行时看上去好像都是同时运行的。

**线程的状态：**

线程具有5种状态，即新建状态、就绪状态、运行状态、阻塞状态、死亡状态。

1. 新建状态：创建线程类对象后，线程对象便处于新建状态。此时，它已经有了相应 的内存空间和其他资源。
2. 就绪状态：当线程对象调用start( )方法启动线程时，线程进入就绪状态。此时，线 程将进入线程队列排队，等待CPU服务。
3. 运行状态：当就绪转态的线程被调用后并获得处理器资源时，线程进入运行状态。 此时将自动调用线程对象的run( )方法。
4. 阻塞状态：一个正在执行的线程在某些特殊情况下，如果被人为挂起或需要执行耗 时的输入输出操作时，CPU会暂停该线程的执行，线程进入阻塞状态。阻塞状态下 的线程不能进入排队队列，当引起阻塞的原因被消除后，线程又进入了就绪转态。
5. 线程调用stop( )方法时或run( )方法执行结束后，即处于死亡状态。处于死亡状态下 的线程不具有继续运行的能力。

**线程常用的操作：**

操作线程的主要方法都定义在Thread类中。

1. 设置和取得线程的名称：在Thread类中，可以通过getName( )方法取得线程的名称， 通过setName( )方法或者Thread类的构造方法设置线程的名称；线程的名称一般在 启动线程前设置，但也允许为已经允许的线程设置名称；允许两个Thread类对象设 置相同的名称，但应避免这种情况；在线程中，如果没有为线程设置名称，则系统在 使用时会为线程自动分配一个名称，分配的名称很有规律，格式为Thread-编号，由 此可得知在Thread中必然存在一个static属性，用于为线程编号命名。
2. 判断线程是否启动：可以通过Thread类中的isAlive( )方法来判断一个线程是否已经 启动并且没有死亡（或在运行）。
3. 线程的强制运行：在线程操作中，可以使用Thread类中的join( )方法让一个线程强 制运行，在线程的强制运行期间，其他线程无法运行，必须等待 此线程完成之后才可以继续执行。

4）线程休眠：可以通过Thread类名称调用sleep( )方法让一个线程进行暂时的休眠， sleep( )方法参数为毫秒。

5）中断线程：当一个线程在运行的时候，另外一个线程可以使用interrupt( )方法中断 其运行。

6）后台线程：在Java程序中，只要后台有一个线程在运行，则整个Java进程都不会消 失，所以此时可以设置一个后台线程，这样即使Java进程结束了，此后 台线程依然会继续执行。要想实现这样的功能，可以使用setDaemon( ) 方法即可，此方法的参数为布尔值。

7）线程的优先级：线程在启动前，可以通过setPriority( )方法设置线程的优先级。此时， 如果哪个线程的优先级高，哪个线程就可能会先被执行。线程的优先 级一共有三种，但并不定线程的优先级高就一定会先执行，哪个线程 最先执行还得由CPU的调度决定。

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | 描述 | 表示的常量 |
| 1 | putlic static final int MIN\_PRIORITY | 低 | 1 |
| 2 | public static final int NORE\_PRIORITY | 中（默认的优先级） | 5 |
| 3 | public static final int MAX\_PRIORITY | 高 | 10 |

也可以通过getPriority( )方法得到一个线程的优先级，主线程的优先 级是中等级别。

8）线程的礼让：可以使用yield( )方法将一个线程的操作暂时让给其他线程执行。

**Java程序运行时中线程的最少数量：**

在Java程序运行时，至少会启动两个线程，即：

1. 一是main线程，因为主方法也是一个线程。如果存在多个线程的话，主 线程的有可能最先执行完。
2. 二是垃圾收集机制。

**为什么要使用同步：**

通过Runnable接口实现多线程，则意味着类中的属性可以被多个线程共享。但多个线 程共享同一资源时，就有可能出现资源的同步问题。如果需要解决这样的问题，就必须 使用同步。所谓的同步，是指多个线程在同一时间段内只有一个线程进行，其他线程线 程要等待此线程完成之后才可以继续执行。

**使用同步解决问题：**

要想解决多个线程共享同一资源而带来的资源问题，可以使用同步代码块或者同步方法 完成。

1. 同步代码块是使用synchronized声明的代码块，并且在使用同步代码块时必须指 定一个需要同步的对象，一般将当前对象（this）设置为同步对象。其格式为：

**synchronized (this){  
 需要同步的代码;  
 }**

1. 同步方法是使用了synchronized关键字声明了的方法，格式为：

**访问权限 synchronized 返回值类型 方法名称(参数列表){  
 方法体；  
 }**

**死锁：**

使用同步可以保证多个线程共享同一资源的正确性，但使用过多的同步也产生死锁。所 谓的死锁，是的指线程间都在等待彼此先完成，造成了程序的停滞状态。

死锁是一种需要回避的代码，并且在多线程的开发中，死锁都是需要经过大量的测 试后才可以被检测出来的一种程序非法状态。

**同步和死锁的使用：**

多个线程共享同一资源时，一定要使用同步操作，同步可以使用同步代码块或同步方法。 但过多的同步操作有可能带来死锁问题，导致程序进入停滞状态。

**线程的生命周期：**

在Java中每个线程对象都有着自己的生命周期。如下所示：

实例化线程对象后，便表示一个线程，线程对象调用start( )方法启动线程；也可以调用 sleep( )、suspend( )、wait( )方法暂停线程，使得线程进入阻塞状态；当run( )方法运行 完后或者调用了stop( )方法后，线程即死亡了。

**线程操作的其他方法：**

在线程操作中还存在suspend( )、resume( )方法、stop( )方法这三个方法，具体功能为：

1. suspend( )：暂时挂起线程
2. resume( )：恢复挂起的线程
3. stop( )：停止线程

但对于这三种方法并不建议使用，因为这三种方法操作时会产生死锁问题。

（在Java中，对于不建议的方法都会使用@Deprecated声明，这三个方法也是如此。）

**停止线程：**

使用stop( )方法可以停止一个线程，但它可能会在成死锁。所以在开发中，通常是 通 过设置标志位的方式来停止一个线程的运行。