**线程的基本概念：**

1）线程是比进程更小的执行单位，是在进程的基础上进一步划分的。线程可以同时存在、同时运行，一个进程可能包含多个同时执行的线程。

2）线程是指程序的运行流程，多线程是指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中实现多线程的方式有两种：一是继承Thread类，二是实现Runnable接口。

**Thread类：**

Thread类在lang包中，一个类只要继承了Thread类，此类就称为多线程操作类。

1. 继承Thread类后，必须覆写Thread类中的run( )方法，run( )方法是线程的主方法。
2. 要想启动线程，必须通过线程对象调用从Thread类中继承而来的start( )方法。并且，一个线程对象只能调用一次start( )方法。每调用几次start( )方法就相当于有几个线程。
3. 多个线程是交错运行的，哪个对象抢到了CPU资源，哪个线程就可以运行，所以程序每次的运行结果是不一样的。

**Runnable接口：**

一个类只要实现了Runnable接口，该类就称为多线程操作类。

1. 实现Runnable接口后，必须覆写run( )方法；
2. 实现Runnable接口后，最终还是要依靠Thread方法中的start( )方法启动线程。将线程类对象通过Thread的构造方法送入Thread类中，再由Thread类的实例化对象启动线程。

**Thread类与Runnable接口的区别：**

1. Thread类是Runnable接口的子类。在Runnable接口中，存在抽象方法run( )，所以Thread类中覆写了Runnable接口中的抽象方法run( )。在Thread类中的run( )方法会调用Runnable接口中的run( )方法，但此Runnable接口中的run( )方法是抽象的，所以此时Thread类中的run( )方中调用的将是Runnable接口子类中的run( )方法。因此，实现多线程时必须在多线程操作类中覆写run( )方法。
2. 继承Thread类实现多线程时，拥有单继承的局限。而实现Runnable接口实现多线程时，不会有单继承的局限。
3. 实现Runnable接口比继承Thread类可以更好地达到资源共享的目的。

**线程的状态：**

线程一般具有5种状态，即创建、就绪、运行、阻塞、终止。

1. 创建状态：创建线程对象后，已经拥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，已经具备了运行条件。

1. 运行状态：当就绪状态的线程被调用并获得处理器资源时，线程就进入了运行状态，

此时将自动调用线程对象的run方法进行相应的操作。

1. 堵塞状态：一个正在执行的线程如果被人为挂起或需要执行耗时的输入输出操作时，

会让CPU暂时中止自己的执行，进入堵塞状态。阻塞状态下，线程不能

进入队列排队，只有当引起阻塞的原因被消除后，线程才可以转入就绪

状态。

1. 死亡状态：线程调用stop( )方法时或run( )方法执行结束后，即处于死亡状态。

**线程操作的相关方法：**

在Java线程操作中，操作线程的主要方法在Thread类中。如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 方法名称 | 类型 | 描述 |
| 1 | public Thread(Runnable target) | 构造 | 接收Runnable接口的子类对象，实例化Thread对象 |
| 2 | public Thread(Runnable target,String name) | 构造 | 接收Runnable接口子类对象，实例化Thread对象，并设置线程名称 |
| 3 | public Thread(String name) | 构造 | 实例化Thread对象，并设置线程名称 |
| 4 | public static Thread currentThread( ) | 普通 | 返回目前正在执行的线程 |
| 5 | public final String getName( ) | 普通 | 返回线程的名称 |
| 6 | public final int getPriority( ) | 普通 | 返回线程的优先级 |
| 7 | public boolean isInterrupted( ) | 普通 | 判断线程是否被中断，如果是，返回true，否则，返回false |
| 8 | public final boolean isAlive( ) | 普通 | 判断线程是否启动或运行，结果返回布尔值 |
| 9 | public final void join( ) throws InterruptedException | 普通 | 等待线程死亡 |
| 10 | public final synchronized void join(long millis) throws InterruptedException | 普通 | 等待millis毫秒后，线程死亡 |
| 11 | public void run( ) | 普通 | 运行线程 |
| 12 | public final void setName(String name) | 普通 | 设置线程名称 |
| 13 | public final void setPriority(int newPriority) | 普通 | 设置线程的优先级 |
| 14 | public static void sleep(long millis) throws InterruptedException | 普通 | 使目前正在执行的线程休眠millis毫秒 |
| 15 | public void start( ) | 普通 | 启动线程 |
| 16 | public String toString( ) | 普通 | 返回代表线程的字符串 |
| 17 | public static void yield( ) | 普通 | 将正在执行的线程暂停，其他线程先执行 |
| 18 | public final void setDaemon(boolean on) | 普通 | 将一个线程设置成后台运行 |

**取得和设置线程名称：**

1）在Thread类中，可以通过setName( )方法设置线程的名称，通过getName( )方法取得线程的名称。

2）如果没有为一个线程设置名称，系统会自动分配名称，名称格式为Thread-编号。由此可知在Thread中必然存在一个static类型的属性用于为线程自动命名。

1. 主方法本身也是一个线程。通过在主方法中使用线程对象调用run( )方法打印当前线程名称即可验证。
2. 在Java中所有的线程都是同时启动的，哪个线程抢占了CPU资源，哪个就先运行。
3. 一个Java程序运行时至少会启动两个线程：一个是main线程；另外一个是垃圾收集线程。

**判断线程是否启动：**

1. 在线程操作中可以使用isAlive( )方法来判断线程是否启动或者线程是否依然在运行，结果返回布尔值。
2. 因为线程操作的不确定性，所以主方法有可能比其他线程最先执行完，但其他线程不会受到影响，也不会随着主线程的结束而结束。

**线程的强制运行：**

在线程操作中，可以使用join( )方法让一个线程强制运行。在线程强制运行期间，其他线程必须等待此线程运行完之后才可以运行。此语句须进行异常处理。

**线程的休眠：**

在线程操作中可以使用sleep( )方法让线程休眠。此语句须进行异常处理。

**中断线程：**

当一个线程运行时，另外一个线程可以直接通过interrupt( )方法，中断其运行。

**后台线程：**

在线程操作中可以使用setDaemon( )方法设置一个后台线程。

**线程的优先级：**

线程在运行前都会保持就绪状态，如果哪个线程的优先级高，哪个线程就有可能会

被先执行。在线程操作中，可以使用线程对象调用setPriority( )方法设置该线程的优先级，调用getPriority( )方法查看线程优先级的常量表示。在Java的线程中一共有三种优先级：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | 描述 | 表示的常量 |
| 1 | public static final MIN\_PRIORITY | 最低优先级 | 1 |
| 2 | public static final NORM\_PRIORITY | 中等优先级，是线程的默认优先级 | 5 |
| 3 | public static final MAX\_PRIORITY | 最高优先级 | 10 |

但并不一定线程的优先级最高就一定会先执行，哪个线程先执行将由CPU的调度决定。

主线程的优先级是中等级别。

**线程的礼让：**

在线程操作中，可以调用yield( )方法将一个线程的操作暂时让给其他线程执行。

**线程操作实例：**

设计一个线程操作类，要求产生三个线程对象，并可以分别设置3个线程的休眠时间，

1. 线程A，休眠10s
2. 线程B，休眠20s
3. 线程C，休眠30s

通过继承Thread类和实现Runnable接口两种方式完成。

**同步：**

通过Runnable接口实现线程，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，有可能出现问题，但使用同步可以解决。所谓同步，是指多个线程操作在一个时间段内只能有一个线程操作进行，其他线程操作要等待此线程完成之后才可以继续执行。要想解决资源共享带来的同步问题，可以使用同步代码块和同步方法两种方式完成。

1. 同步代码块是使用synchronized关键字声明的代码块，在使用同步代码块时必

须指定一个需要同步的对象，一般将当前对象设置为（this）同步对象。

1. 同步方法是使用synchronized关键字声明的方法，在线程主方法中通过调用同步方法解决资源共享带来同步问题

**死锁：**

同步可以保证资源共享操作的正确性，但是过多的同步会产生死锁，死锁是指两个线程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。在多线程的开发中，死锁是需要通过大量测试后才能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需要进行同步，以保证资源操作的正确性，但过多的同步可能

会产生死锁。

**生产者和消费者：**

生产者与消费者是线程操作中的经典案例程序，即生产者不断生产，消费者不断取走生产者生产的产品。以生产信息产品为例（包括信息名称和信息内容）为例。由于线程运行的不确定性。其基本模型会存在两个问题：

1. 添加了信息名称，但为没有添加信息的内容，程序就切换到了消费者线程，消费者

线程把这个信息的名称和上一个信息的内容联系到一起了。

1. 生产者放了若干次信息，消费者才开始取信息，或者是消费者取完一个信息后，还没等到生产者放入新的信息，又重复取出已去过的信息。

对于以上问题，第一个问题可以使用同步来解决，第二个问题可以使用Object类中的方法来解决

**Object类对线程的支持：**

Object类对线程的支持主要是等待和唤醒。

|  |  |  |
| --- | --- | --- |
| 序号 | 方法 | 功能 |
| 1 | public final void wait( )throws InterruptedException | 线程等待 |
| 2 | public final void wait(long timeout)throws InterruptException | 线程等待，并指定等待的最长时间，单位为毫秒 |
| 3 | public final void wait(long timeout,int nanos)throws InterruptedException | 线程等待，并指定等待的最长毫秒及纳秒 |
| 4 | public final void notify( ) | 唤醒第一个等待的线程 |
| 5 | public final void notifyAll( ) | 唤醒全部等待的线程 |

**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

**Thread中的其他方法：**

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

对于以上方法不推荐使用，因为使用这三种方法在操作时会产生死锁问题。

打开Thread源代码可以发现以上方法的声明上加入了一条“@Deprecated”的注释。这属于Annotation的语法，表示此操作不建议使用，一旦使用了这些方法将出现警告信息。

**停止线程：**

在多线程的开发中通过设置标志位的方式停止一个线程的运行。这种停止方式在开发中是比较常用的。