线程

一.线程创建

1.继承Thread类，重写run()方法

2.实现Runnable接口，并实现该接口的run()方法（创建Thread对象，用实现Runnable接口的对象作为参数实例化该Thread对象）

3.实现Callable接口，重写call()方法

Callable接口是属于Executor框架的功能类，运行Callable可以拿到一个Future对象，Future对象表示异步运算的结果。

Thread与Runnable的二者区别：实现Runnable接口可以支持多继承

二.线程状态

- New： 初始状态
- RUNNABLE： 运行状态，操作系统中运行与就绪两种状态统称运行中
- BLOCKED： 阻塞状态
- WAITING： 等待状态
- TIME_WAITING： 超时等待
- TERMINATED： 终止状态

三.线程中断

中断表示一个运行中的线程是否被其他线程进行了中断操作（通过调用其他线程的interrupt()方法对其进行中断操作）

可以通过isInterrupted()来进行判断是否被中断，但要注意许多声明抛出InterruptedException的方法（例如sleep）这些方法在抛出异常之前，会先将该线程的中断标识位清除

Thread中中断标志位检查方法：

this.interrupted()：测试当前线程是否已经是中断状态，执行后具有将标志位置清除为false的功能（连续两次调用，第二次返回false）

this.isInterrupted()：测试线程Thread对象是否已经是中断状态，但不清除状态标志

单独调用它可以使得处于阻塞状态的线程抛出一个异常，它可以中断一个正处于阻塞状态的线程，但不能中断处于非租塞状态的线程，通过interrupt方法和isInterrupt()方法来停止正在运行的线程

守护线程与用户线程的区别：守护线程依赖于创建它的线程，而用户线程则不依赖。举个简单的例子：如果在main线程中创建了一个守护线程，当main方法运行完毕之后，守护线程也会随着消亡。而用户线程则不会，用户线程会一直运行直到其运行完毕。在JVM中，像垃圾收集器线程就是守护线程。

四.Thread中的方法

sleep()：让当前正在执行的线程休眠，但是有一点要非常注意，sleep方法不会释放锁，也就是说如果当前线程持有对某个对象的锁，则即使调用sleep方法，其他线程也无法访问这个对象，只能等到线程执行结束释放锁，如果调用了sleep方法，必须捕获InterruptedException异常或者将该异常向上层抛出，所以说调用sleep方法相当于让线程进入阻塞状态。

yied(): 调用yield方法会让当前线程交出CPU权限，让CPU去执行其他的线程。它跟sleep方法类似，同样不会释放锁。但是yield不能控制具体的交出CPU的时间，另外，yield方法只能让拥有相同优先级的线程有获取CPU执行时间的机会。注意，调用yield方法并不会让线程进入阻塞状态，而是让线程重回就绪状态，它只需要等待重新获取CPU执行时间，这一点是和sleep方法不一样的。

join(): 假如在main线程中，调用thread.join方法，则main方法会等待thread线程执行完毕或者等待一定的时间。如果调用的是无参join方法，则等待thread执行完毕，如果调用的是指定了时间参数的join方法，则等待一定的时间再继续执行。

由于join(long)方法实际是在内部调用了wait(long)方法，所以join(long)方法具有释放锁的特点（线程释放对象锁）

五.线程间通信

等待通知机制：

wait()、notify()、notifyAll()都是Object类中的方法，方法为本地方法，为final方法，无法重写

- 调用某个对象的wait()方法能让当前线程阻塞，并且在调用之前当前线程必须拥有此对象的锁，在执行wait()方法之后当前线程释放锁

- 调用某个对象的notify()方法能够唤醒其中一个正在等待这个对象锁的线程，notifyAll()唤醒所有
- 调用wait()、notify()都方法必须在同步块或者同步方法中进行
- notify()和notifyAll()只是唤醒正在等待该对象monitor的线程，并不决定哪个线程能够获取到monitor（一个线程被唤醒不等于立刻获取了对象的monitor,只有等调完notify()或者notifyAll()执行完并退出synchronized块后，释放对象锁后才可以收到）
- 如果线程调用了对象的wait（）方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。当有线程调用了对象的notifyAll()方法（唤醒所有wait线程）或notify（）方法（只随机唤醒一个wait线程），被唤醒的的线程便会进入该对象的锁池中，锁池中的线程会去竞争该对象锁。
- 当线程呈wait()状态，调用该线程对象的interrupt()方法会出现InterruptedException异常
- 为什么wait, notify 和 notifyAll这些方法不在thread类里面： JAVA提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得,由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象.

通过管道进行线程间通信：

管道流（pipeStream)是一种特殊的流用于在不同线程间直接传送数据，一个线程发送数据到输出管道，另一个线程从输入管道中读数据

JDK中提供了4个类来使线程间可以进行通信：

PipedInputStream和PipedOutStream

PipedReader和PipedWriter

六.多线程实现同步的方法：

- 使用synchronized同步代码块
- 使用ReentrantLock加锁
- 使用ThreadLocal为变量在每个线程中都创建一个副本
- 使用wait()与notify()方法来进行线程间通信

七.实现线程安全的方式：

- 互斥同步（同步阻塞）：采用synchronized关键字和可重入锁（一种悲观的并发策略）
- 非阻塞同步：基于冲突检测的乐观并发策略，就是先进行操作，如果没有其他线程争用共享资源，那操作就成功，如果有争用，产生冲突，再采取其他补救措施
- 无同步方案：如果不涉及任何共享数据，则就无需任何同步措施

八.Runnable和Callable的区别

- Callable规定的方法是call()，Runnable规定的方法是run()
- call()方法可以返回任务执行结果，run()方法不返回
- call()方法可以抛出异常，run()不可以
- 加入线程池运行，Runnable使用ExecutorService的execute方法，Callable使用submit方法

九.一个进程中最多能开辟几个线程？是否是有限的？为什么能开辟那么多，怎么计算的?

默认情况下，一个线程的栈要预留1M的内存空间,而一个进程中可用的内存空间只有2G，所以理论上一个进程中最多可以开2048个线程,但是内存当然不可能完全拿来作线程的栈，所以实际数目要比这个值要小。

十.sleep()与wait()的区别

- sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复（线程回到就绪状态）
- wait()是Object类的方法，用于线程间的通信，调用对象的wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify()方法（或notifyAll()方法）时才能唤醒等

待池中的线程进入锁池（lock pool），如果线程重新获得对象的锁就可以进入就绪状态。

- 使用区域不同:wait()方法必须在同步控制方法或者同步语句块中使用，而sleep(0方法则可以放在任何地方使用
- sleep()方法必须捕获异常，而wait()方法不需要

十一.sleep()方法与yied()方法有什么区别

- sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会，而yied()方法只会给相同优先级或者更高优先级的线程以运行的机会
- 线程执行sleep()方法后会转入阻塞状态，即在指定的时间内线程肯定不会被执行，而yied()方法只是使当前线程重新回到就绪状态，很有可能在进入到就绪状态后又马上被执行
- sleep()方法抛出异常，而yied()方法没有声明任何异常