多线程 网络通信 反射 图形界面编程

多线程

1. 进程、线程、多线程、多线程的优缺点：

进程：是一个可执行程序一次运行的过程。系统给进程分配独立的资源。

线程：程序执行的最小单位。系统不会给线程分配独立的资源。线程使用

进程的资源。

线程本质：就是一个方法的执行过程。

关系：进程创建线程。进程开始执行时会创建主线程。线程中可以创建线

程。

特点：线程和进程可以并发执行。时间片，微观串行。

多线程：一个进程创建多个线程来完成不同的功能。

优点：可以最大限度的降低CPU的闲置时间，从而CPU的利用率。

缺点：

多线程需要协调和管理，所以需要CPU跟着线程。不是线程越动越好。

线程之间对共享资源的访问会相互影响，必须解决竟用资源的问题。

线程太多会导致控制太复杂，最终可能造成很多bug。

1. 创建线程类：
2. 继承Thread类，重写run方法。
3. 创建实现了Runable接口的类，重写run方法，使用该类对象创建Thread对象。
4. 创建实现了Callable接口的类，重写call方法，使用该类对象创建FuterTask对象，使用FuterTask对象创建Thread对象，通过FuterTask对象的get方法可以获取线程的返回值。
5. 创建线程池，使用线程池管理线程。

使用情况：

线程要返回一个值，则使用第三种方法。

两个线程执行的操做相同，则使用第二种方法。如：卖票

两个线程执行的操做不相同，则使用第一种方法。如：生产消费

1. Thread类中方法：

数据成员：MAX\_PRIORITY、MIN\_PRIORITY、NORM\_PRIORITY

构造方法：

Thread(Runnable target)

Thread(Runnable target, String name)

方法成员：

public static Thread currentThread()：获取当前执行该程序的线程。

getName()：获取线程名字

setName(String name)：设置线程名字

getId()：获取线程id值

setPriority()：获取优先级

getPriority()：设置优先级

getState()：获取线程状态

interrupt()：中断线程

setDaemon(boolean on)：设施是否为守护线程。

守护线程：在后台运行，并且不受任何终端控制的线程。

在执行start之前执行。

当最后一个非守护线程结束时，守护线程会随着JVM一同结束。

isDaemon()：判断是否为守护线程。

线程对象.join()：线程没有结束则处于阻塞状态。

static void sleep(long millis) ：使线程处于阻塞状态。

start()：启动线程。只能执行一次。

static void yield()：放弃CPU，进入可运行状态。用于调试程序。

1. 线程互斥：

互斥：间接相互制约。一个系统中的多个线程必然要共享某种系统资源。

共享资源----临界资源

同步块：

synchronized（对象）{

所有对共享资源操做的代码

不要把和共享资源操做无关的代码放同步块中，以提高效率。

}

每个线程中的同步对象要求是同一个对象。通常用this。

共享资源设为private。

工作原理：任意对象中都有一个 “锁”，有解锁状态，上锁状态两个状态。如果对象是解锁状态，则先上锁，然后执行同步代码块中的内容，同步块执行完后对对象解锁；如果对象是上锁状态，则处于阻塞状态。

同步方法：

访问权限 修饰符 返回类型 方法名(形参表){

对共享资源操做的代码。

}

修饰符：synchronized

非静态同步方法上锁的对象是this。

同一个对象在不同的线程中调用同步方法，某一时刻只能有一个线

程可以执行同步方法

注意：静态方法可以是同步方法。对所在的类对应的Class对象上锁解锁操做。类对应的Class对象只有一个。

1. 线程同步：

同步：线程之间的合作。如：生产消费问题。

wait()：等待。解锁、进入阻塞状态。

notify()：唤醒某一个处于等待的线程。被唤醒的线程进入锁池，进入阻

塞状态，解锁后进入可运行是状态。

notifyAll()：唤醒所有处于等待的线程。被唤醒的线程进入锁池，进入阻

塞状态，解锁后进入可运行是状态。

注意：要在同步块或同步方法种调用该方法，同时调用该方法的对象要

求是同步对象。

面试题：

wait和sleep的区别？

等待时 wait 会释放锁，而 sleep 一直持有锁。

wait 通常被用于线程间交互，sleep 通常被用于暂停执行。

wait是Object类中的方法；sleep是Thread类中的方法；

执行wait后进入等待状态；执行sleep后进入休眠状态；

子线程运行执行 10 次后，主线程再运行 5 次。这样交替执行三遍。

线程1：500毫秒生产一个鸡蛋筒，

线程2：100毫秒使用鸡蛋筒生产一个冰激凌，

线程3：吃一个冰激凌。

1. 线程的死锁：

参考面试宝典。

1. Lock

Lock是接口。

优点：Lock实现提供比使用synchronized方法和语句可以获得的更广泛的锁定操作。

通常使用 ReentrantLock 类。

方法成员：

ReentrantLock()：创建锁对象。

lock()：上锁操做。如果处于上锁状态则阻塞。

unlock()：解锁操做。

tryLock()：尝试上锁。如果处于上锁状态则返回false，不会阻塞。

tryLock(long time, TimeUnit unit)：尝试在指定时间内上锁。如果处于上锁状态，进入阻塞状态，在指定时间内，如果解锁，则上锁返回true；在指定时间结束后还是上锁状态，则结束阻塞，返回false。

多个线程上锁的对象要相同。

try{

lock();

操做共享资源；

}catch(){

}finally{

unlock();

}

同步：

Condition接口

使用Condition类对象等待和唤醒。

await()：等待。对wait方法进行了封装。解锁，阻塞。

signal()：唤醒。对notify方法进行了封装。唤醒等待的线程。

通过Lock对象获取Condition对象：

Condition newCondition()

注意：两个线程的Lock对象要相同。相同的Condition对象才能实现等待和唤醒。

1. 线程状态：

参考图片。

1. 定时器：

Timer类

线程调度任务以供将来在后台线程中执行的功能。 任务可以安排一次执行，或定期重复执行。

设置延迟一定时间创建线程执行一次操做；

设置延迟一定时间创建线程周期性的执行操做。

Timer(String name, boolean isDaemon)

schedule(TimerTask task, long delay) //毫秒

schedule(TimerTask task, long delay, long period)

cancel()

1. 线程池：
2. 好处：

线程池就是事先将线程对象放到一个容器中，当使用的时候不用new线程，而是直接在线程池中取线程即可，这样可以创建线程的时间，提高代码执行效率。

1. 线程池的种类：

单线程线程池、固定大小线程池、可缓存的线程池、支持周期执行的固定大小线程池

Executors类中的静态方法创建线程池。

单线程线程池：

所有任务的执行顺序按照任务的提交顺序执行。

newSingleThreadExecutor();

固定大小线程池：

每次提交一个任务就是一个线程，直到线程达到线程池的最大大小。

newFixedThreadExecutor();

可缓存的线程池：

线程池不会对线程池中的线程个数限制。

newCachedThreadExecutor();

支持周期执行的固定大小线程池：

线程池中线程个数固定，可以周期性的执行操做。

newScheduledThreadExecutor();

1. 线程池的使用

创建线程池；

执行execute方法将线程添加到线程池；

执行shutdown关闭线程池；

1. volatile
2. 并发编程中的三个概念：
3. 原子性：

一个操作或者多个操作要么全部执行，执行过程中不会被任何的因素打断；要么就全部不执行。

如：银行账户转帐。

通过同步块、同步方法实现原子性。

1. 可见性：

多个线程访问同一个变量时，一个线程修改了这个变量，其他的线程能够立即看到修改后的值。

int a=0;

线程1：

a = 2;

线程2：

b = a; //从a中取的值可能还是0

1. 有序性：

程序执行的顺序按照代码的先后顺序执行。

Int a;

Float b;

a = 1;

b = 0.4f;

指令重排：处理器为了提高程序的运行效率，可能会对输入的代码进行优化，它不保证各个语句的执行先后顺序同代码中的顺序一致，但是它会保证程序最终执行结果和代码顺序执行的结果保持一致。

指令重排不会影响单线程的执行结果，但可能会影响多线程的执行结果。

如：

线程一：

b=false;

c = f();

b = true; //指令重排：该语句放在c=f()；前面

线程二：

While(!b){

sleep();

}

g(c);

要想程序正确执行，必须保证原子性、可见性、有序性。只有有一个没有保证，就有可能导致程序执行不正确。

1. volatile的作用

volatile修饰全局变量。

1. 保证了不同线程对该变量进行操作时的可见性。

线程一：

boolean a=false;

While(!a){

f();

}

线程二：

a=true;

1. 禁止指令重排。不能将 在 对volatile变量访问的前面的语句放在其后执行，也不能将 在 对volatile变量访问的后面的语句放在前面执行。

如：

Int x, y;

Volatile Int flag;

X = 1;

Y = 2;

Flag = 3;

X = 4;

Y = 5;

C、不保证原子性。

需要保证操作是原子性操作，才能保证使用volatile关键字的程序在并发时能够正确执行。