# 线程

## 定义

进程：cpu分配资源的最小单位

线程：cpu调度的最小单位

在java中要想实现多线程，有两种手段，一种是继续Thread类，另外一种是实现Runable接口.(其实准确来讲，应该有三种，还有一种是实现Callable接口，继承Thread类也是通过实现runable接口实现的)

线程的5种状态：创建 就绪 运行 阻塞 终止

注意：

①start()方法的调用后并不是立即执行多线程代码，而是使得该线程变为可运行态（Runnable），什么时候运行是由操作系统决定的。

② start方法重复调用，会出现java.lang.IllegalThreadStateException异常

## **Thread 和Runable**

一个类继承Thread，不适合资源共享，实现Runable，容易资源共享，java特性--单继承，多实现。

实现Runable比继承Thread的优势：

1. 适合多个相同程序代码的线程处理同一个资源
2. 可以避免java中单继承限制
3. 增加程序健壮性，代码可以被多个线程共享，代码和数据独立
4. 线程池中只能放入实现Runable或callable类线程，不能直接放入继承Thread的类

Runable和callable区别

Runable 返回类型是void callable返回类型是泛型和Future、futureTask配合可以获取到异步执行的结果，在一些执行时间过长的任务中可以中断取消任务。

线程安全：

如果你的代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么你的代码就是线程安全的。

## **线程运行状态图**



1、新建状态（New）：新创建了一个线程对象。

2、就绪状态（Runnable）：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态（Running）：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态（Blocked）：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

（一）、等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。(wait会释放持有的锁)

（二）、同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

（三）、其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。（注意,sleep是不会释放持有的锁）

5、死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

①sleep(long millis): 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行）  
②join():指等待t线程终止。

③yield():暂停当前正在执行的线程对象，并执行其他线程。

④setPriority(): 更改线程的优先级。

⑤interrupt():不要以为它是中断某个线程！它只是线线程发送一个中断信号，让线程在无限等待时（如死锁时）能抛出抛出，从而结束线程，但是如果你吃掉了这个异常，那么这个线程还是不会中断的！

⑥wait()

Obj.wait()，与Obj.notify()必须要与synchronized(Obj)一起使用，也就是wait,与notify是针对已经获取了Obj锁进行操作，从语法角度来说就是Obj.wait(),Obj.notify必须在synchronized(Obj){...}语句块内。从功能上来说wait就是说线程在获取对象锁后，主动释放对象锁，同时本线程休眠。直到有其它线程调用对象的notify()唤醒该线程，才能继续获取对象锁，并继续执行。相应的notify()就是对对象锁的唤醒操作。但有一点需要注意的是notify()调用后，并不是马上就释放对象锁的，而是在相应的synchronized(){}语句块执行结束，自动释放锁后，JVM会在wait()对象锁的线程中随机选取一线程，赋予其对象锁，唤醒线程，继续执行。这样就提供了在线程间同步、唤醒的操作。Thread.sleep()与Object.wait()二者都可以暂停当前线程，释放CPU控制权，主要的区别在于Object.wait()在释放CPU同时，释放了对象锁的控制。

Sleep() wait() 区别

①如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

**sleep()和yield()的区别**  
        sleep()和yield()的区别):sleep()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；yield()只是使当前线程重新回到可执行状态，所以执行yield()的线程有可能在进入到可执行状态后马上又被执行。  
        sleep 方法使当前运行中的线程睡眼一段时间，进入不可运行状态，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。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  的占有权交给此线程，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优先级的其他线程  
       另外，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些时获得 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，又没有受到 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，才有机会运行。

[关于假如有Thread1、Thread2、Thread3、Thread4四条线程分别统计C、D、E、F四个盘的大小，所有线程都统计完毕交给Thread5线程去做汇总，应当如何实现？](http://blog.csdn.net/wenwen360360/article/details/62104612)

第一种方法：

直接用join把线程5加入进去即可

第二种方法：

用java.util.concurrent下的方法解决

用CountDownLatch : 一个线程(或者多个)， 等待另外N个线程完成某个事情之后才能执行

CountDownLatch 是计数器, 线程完成一个就记一个, 就像 报数一样, 只不过是递减的.

解析很全：<https://www.cnblogs.com/GarfieldEr007/p/5746362.html>

## **线程池**

1.Executor接口是Executor框架中最基础的部分，定义了一个用于执行Runnable的execute方法，它没有实现类只有另一个重要的子接口ExecutorService

2.ExecutorService接口继承自Executor接口，定义了终止、提交,执行任务、跟踪任务返回结果等方法

2.1 execute（Runnable command）：履行Ruannable类型的任务,

2.2 submit（task）：可用来提交Callable或Runnable任务，并返回代表此任务的Future对象  
 2.3 shutdown（）：在完成已提交的任务后封闭办事，不再接管新任务,

2.4 shutdownNow（）：停止所有正在履行的任务并封闭办事。  
 2.5 isTerminated（）：测试是否所有任务都履行完毕了。,

2.6 isShutdown（）：测试是否该ExecutorService已被关闭

3.Executors的静态方法：负责生成各种类型的ExecutorService线程池实例。

+newFixedThreadPool(numberOfThreads:int):（固定线程池）ExecutorService 创建一个固定线程数量的线程池，并行执行的线程数量不变，线程当前任务完成后，可以被重用执行另一个任务  
+newCachedThreadPool():（可缓存线程池）ExecutorService 创建一个线程池，按需创建新线程，就是有任务时才创建，空闲线程保存60s，当前面创建的线程可用时，则重用它们

+new SingleThreadExecutor();（单线程执行器）线程池中只有一个线程，依次执行任务

+new ScheduledThreadPool()：线程池按时间计划来执行任务，允许用户设定执行任务的时间

+new SingleThreadScheduledExcutor();线程池中只有一个线程，它按规定时间来执行任务

1. threalPoolExecutor 线程池的实现类

**public** ThreadPoolExecutor(**int** corePoolSize,线程池基本大小

**int** maximumPoolSize,线程池最大大小

**long** keepAliveTime,线程存活时间

TimeUnit unit,线程存货时间单位

BlockingQueue<Runnable> workQueue) {

//任务队列，保存阻塞的线程

**this**(corePoolSize, maximumPoolSize, keepAliveTime, unit, workQueue,

Executors.*defaultThreadFactory*(), ***defaultHandler***);

}

向线程池提交任务的两种方式：

1）通过execute()方法

[java] [view plain](http://blog.csdn.net/tuke_tuke/article/details/51353925" \o "view plain) [copy](http://blog.csdn.net/tuke_tuke/article/details/51353925" \o "copy)

ExecutorService threadpool= Executors.newFixedThreadPool(10);

threadpool.execute(new Runnable(){...});

这种方式提交没有返回值，也就不能判断任务是否被线程池执行成功。

2）通过submit()方法

[java] [view plain](http://blog.csdn.net/tuke_tuke/article/details/51353925" \o "view plain) [copy](http://blog.csdn.net/tuke_tuke/article/details/51353925" \o "copy)

Future<?> future = threadpool.submit(new Runnable(){...});

    try {

            Object res = future.get();//获取任务执行结果

        } catch (InterruptedException e) {

            // 处理中断异常

            e.printStackTrace();

        } catch (ExecutionException e) {

            // 处理无法执行任务异常

            e.printStackTrace();

        }finally{

            // 关闭线程池

            executor.shutdown();

        }

使用submit 方法来提交任务，它会返回一个Future对象，通过future的get方法来获取返回值，get方法会阻塞住直到任务完成，而使用get(long timeout, TimeUnit unit)方法则会阻塞一段时间后立即返回，这时有可能任务没有执行完。

线程池的关闭：

• shutdown()：不会立即终止线程池，而是再也不会接受新的任务，要等所有任务缓存队列中的任务都执行完后才终止  
• shutdownNow()：立即终止线程池，再也不会接受新的任务，并尝试打断正在执行的任务，并且清空任务缓存队列，返回尚未执行的任务