# 扩展java.lang.Thread类

## 注意

1、start()方法调用后不会立即执行多线程代码，而是使得该线程变为可运行态。什么时候运行由操作系统决定。

2、start()方法重复调用会出现java.lang. IllegalThreadStateException异常

3、main也是一个线程。至少有两个线程：垃圾回收和main

# 实现java.lang.Runnable接口

## 优势

1、适合多个相同的程序代码的线程去处理同一个资源

2、可以避免单继承

3、线程池只能放实现Runnable或Callable接口的线程

# 进程和线程

## 进程

1、每个进程都有独立的代码和数据空间（进程上下文）

2、进程间的切换会有较大的开销

3、进程是资源分配的最小单位

4、一个线程包含1-n个线程

## 线程

1、同一类线程共享代码和数据空间，

2、每个线程有独立的运行栈和程序计数器PC

3、进程切换开销小

4、线程是CPU调度的最小单位

# 线程

## 状态转换



1、新建状态：new

2、就绪状态：start()

3、运行态：获取CPU，执行代码

4、阻塞状态：放弃CPU，暂时停止运行

（一）等待同步：wait（放入等待池、释放锁）

（二）同步阻塞：同步锁被占用（放入锁池）

（三）其他阻塞sleep()、join()、I/O请求。（不释放持有锁）

## 线程调度

1、线程优先级

2、线程睡眠：Thread.sleep(1000)。允许较低优先级的线程获得运行机会

3、线程等待：Object类中的wait()（会释放占有的锁）。直到其他线程调用notify和notifyAll（相当于wait(0)，释放锁），且synchronized(){}块结束。

4、线程让步：Thread.yield()。让当前线程回到可运行状态，以允许具有相同优先级的其他线程获得运行机会（可能又是当前线程抢到）

5、线程加入：join()。等待其他线程终止。在当前线程中调用另一个线程的join()方法，则当前线程转入阻塞状态，直到另一个进程运行结束，当前线程再由阻塞转为就绪状态。主线程需要等待子线程的终止

6、线程唤醒：notify()。唤醒在此对象监视器上等待的单个线程（任意）。线程通过调用wait方法，在对象监视器上等待。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争。

## 各种线程概念

1、主线程：main

2、当前线程：Thread.currentThread()

3、后台线程：也称守护线程，是否随主线程的结束而结束（GC）

4、前台线程：接受后天线程服务的线程

# 线程池

使得线程可以复用

## 核心类：ThreadPoolExecutor

### 一、重要方法

#### 1、构造方法

public ThreadPoolExecutor**(**int corePoolSize**,**

int maximumPoolSize**,**

long keepAliveTime**,**

TimeUnit unit**,**

BlockingQueue**<**Runnable**>** workQueue**,**

ThreadFactory threadFactory**,**

RejectedExecutionHandler handler**)** **{**

**if** **(**corePoolSize **<** 0 **||**

maximumPoolSize **<=** 0 **||**

maximumPoolSize **<** corePoolSize **||**

keepAliveTime **<** 0**)**

**throw** **new** IllegalArgumentException**();**

**if** **(**workQueue **==** **null** **||** threadFactory **==** **null** **||** handler **==** **null)**

**throw** **new** NullPointerException**();**

**this.**acc **=** System**.**getSecurityManager**()** **==** **null** **?**

**null** **:**

AccessController**.**getContext**();**

**this.**corePoolSize **=** corePoolSize**;**

**this.**maximumPoolSize **=** maximumPoolSize**;**

**this.**workQueue **=** workQueue**;**

**this.**keepAliveTime **=** unit**.**toNanos**(**keepAliveTime**);**

**this.**threadFactory **=** threadFactory**;**

**this.**handler **=** handler**;**

**}**

1、corePoolSize：核心池大小。默认情况下线程池中并没有任何线程，而等待任务的到来才去创建一个线程。prestartAllCoreThreads()：预创建corePoolSize个线程，prestartCoreThread()：预创建一个线程。当线程池中的线程数达到corePoolSize后，就会把到达的任务放到缓存队列当中

2、maximumPoolSize：线程池最大线程数

3、keepAliveTime：线程中没有任务执行时最多保持多久时间会终止。大于corePoolSize：如果一个线程空闲时间达到keepAliveTime则会被终止。直到线程池中的线程数不超过corePoolSize。调用allowCoreThreadTimeOut(boolean)方法：直到线程池中的线程数为0。

4、unit：参数keepAliveTime的时间单位。

TimeUnit**.**DAYS**;** //天

TimeUnit**.**HOURS**;** //小时

TimeUnit**.**MINUTES**;** //分钟

TimeUnit**.**SECONDS**;** //秒

TimeUnit**.**MILLISECONDS**;** //毫秒

TimeUnit**.**MICROSECONDS**;** //微妙

TimeUnit**.**NANOSECONDS**;** //纳秒

5、workQueue：一个阻塞队列，用来存储等待执行的任务。可指定排队策略（ArrayBlockingQueue、PriorityBlockingQueue、LinkedBlockingQueue、Synchronous）。

6、threadFactory：线程工厂，主要用来创建线程。

7、handler：拒绝处理任务时的策略

ThreadPoolExecutor**.**AbortPolicy//丢弃任务并抛出RejectedExecutionException异常。

ThreadPoolExecutor**.**DiscardPolicy//也是丢弃任务，但是不抛出异常。

ThreadPoolExecutor**.**DiscardOldestPolicy//丢弃队列最前面的任务，然后重新尝试执行任务（重复此过程）

ThreadPoolExecutor**.**CallerRunsPolicy//由调用线程处理该任务

### 二、四种线程池

newCachedThreadPool：创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

newFixedThreadPool：创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

newScheduledThreadPool：创建一个定长线程池，支持定时及周期性任务执行。

newSingleThreadExecutor：创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

### 三、实现原理

#### 1、线程池状态

// 保证线程之间的可见性

volatile int runState**;**

// 线程池处于运行状态

static final int RUNNING **=** 0**;**

// 调用shutdown()方法，线程池处于SHUTDOWN状态，此时线程池不能够接受新的任务，

// 他会等待所有任务执行完毕

static final int SHUTDOWN **=** 1**;**

// 调用shutdownNow()方法，线程池处于STOP状态，此时线程池不能接受新的任务，

// 并且会去尝试终止正在执行的任务

static final int STOP **=** 2**;**

// 当线程池处于SHUTDOWN或STOP状态，并且所有工作线程已经销毁，

// 任务缓存队列已经清空或执行结束后，线程池被置为TERMINATED状态

static final int TERMINATED **=** 3**;**