Java 多线程系列课程01

------ Java 多线程基础知识

本文档包含以下内容：

1. 学习目标
2. 线程和进程的基本概念
3. 如何实现多线程
4. 什么是守护线程
5. Java多线程的几种状态

**前言：**

**一、线程是什么？进程是什么？二者有什么区别和联系？** （1）线程是CPU独立运行和独立调度的基本单位；  
 （2）进程是资源分配的基本单位；进程是系统中正在运行的一个程序，程序一旦运行就是进程。

**两者的联系：**

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程是操作系统可识别的最小执行和调度单位。

（2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。 同一进程中的多个线程共享代码段(代码和常量)，数据段(全局变量和静态变量)，扩展段(堆存储)。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。

（3）处理机分给线程，即真正在处理机上运行的是线程。

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

**二、进程和线程的区别体现在以下几个方面：**

1.地址空间和其他资源（如打开文件）：进程间相互独立，同一进程的各线程间共享。某进程内的线程在其他进程内不可见。

2.通信：进程间通信IPC（管道，信号量，共享内存，消息队列），线程间可以直接独写进程数据段（如全局变量）来进程通信——需要进程同步和互斥手段的辅助，以保证数据的一致性。

3.调度和切换：线程上下文切换比进程上下文切换快得多。

4.在多线程OS中，进程不是一个可执行的实体。

5.进程具有独立的空间地址，一个进程崩溃后，在保护模式下不会对其它进程产生影响。  
6.线程只是一个进程的不同执行路径，线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉。

三、进程和线程的选择取决以下几点：

1.需要频繁创建销毁的优先使用线程；因为对进程来说创建和销毁一个进程的代价是很大的。

2.线程的切换速度快。所以在需要大量计算，切换频繁时使用线程，还有耗时的操作时用使用线程可提高应用程序的响应。

3.因为对CPU系统的效率使用上线程更占优势，所以可能要发展到多机分布的用进程，多核分布用线程。

4.并行操作时用线程**，**如C/S架构的服务器端并发线程响应用户的请求。

5.需要更稳定安全时，适合选择进程；需要速度时，选择线程更好。

**多线程的优点和缺点：  
多线程的好处：**（1）使用多线程可以把程序中占据时间长的任务放到后台去处理，如图片、视屏的下载  
（2）发挥多核处理器的优势，并发执行让系统运行的更快、更流畅，用户体验更好  
多线程的缺点：  
（1）大量的线程降低代码的可读性；  
（2）更多的线程需要更多的内存空间  
（3）当多个线程对同一个资源出现争夺时候要注意线程安全的问题。

Java多线程的几种实现方式：

1.什么是守护线程

不是说当子线程是守护线程，主线程结束，子线程就跟着结束，这里的前提条件是：当前jvm应用实例中没有用户线程继续执行，如果有其他用户线程继续执行，那么后台线程不会中断

守护线程的使用场景

守护线程的典型代表是垃圾回收，这是很多人说守护进程非常有用的理由，但实际上守护进程在用户开发上的应用场景几乎用处不大，可能的应用场景：

内存资源或者线程的管理，但是非守护线程也可以做

守护线程负责一个可以将当前的JVM退出的功能，即将非damon的线程都退出，然后jvm自动退出，感觉用的也非常少，可以直接通知相关线程退出不就可以了，考虑设计上优雅一些，可能有点好处。

守护线程：

通过 thread.setDaemon(true)将当前的线程设置为守护线程。

守护线程，专门用于服务其他的线程，如果其他的线程（即用户自定义线程）都执行完毕，连main线程也执行完毕，那么jvm就会退出（即停止运行）——此时，连jvm都停止运行了，守护线程当然也就停止执行了。

守护线程（即daemon thread），是个服务线程，准确地来说就是服务其他的线程，这是它的作用——而其他的线程只有一种，那就是用户线程。所以java里线程分2种：

1、守护线程，比如垃圾回收线程，就是最典型的守护线程。

2、用户线程，就是应用程序里的自定义线程。

(1) thread.setDaemon(true)必须在thread.start()之前设置，否则会跑出一IllegalThreadStateException异常。  
你不能把正在运行的常规线程设置为守护线程。  
 (2) 在Daemon线程中产生的新线程也是Daemon的。  
 (3) 不要认为所有的应用都可以分配给Daemon来进行服务，比如读写操作或者计算逻辑。  
 写java多线程程序时，一般比较喜欢用java自带的多线程框架，比如ExecutorService，但是java的线程池会将守护线程转换为用户线程，所以如果要使用后台线程就不能用java的线程池。  
如下，线程池中将daemon线程转换为用户线程的程序片段：  
注意到，这里不仅会将守护线程转变为用户线程，而且会把优先级转变为Thread.NORM\_PRIORITY。

Java多线程的状态：

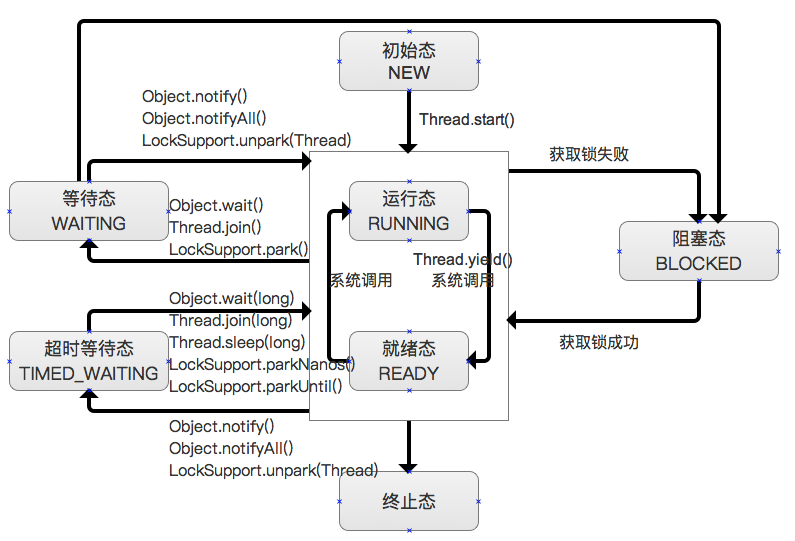
public enum State {  
 NEW,  
 RUNNABLE,  
 BLOCKED, //等待锁  
 WAITING,  
 TIMED\_WAITING，  
 TERMINATED;}  
  
  
（NEW）初始态：

NEW创建一个Thread对象，但还未调用start()启动线程时，线程处于初始态。（RUNNABLE）运行态：RUNNABLE在Java中，运行态包括就绪态 和 运行态。  
 就绪态:该状态下的线程已经获得执行所需的所有资源，只要CPU分配执行权就能运行。所有就绪态的线程存放在就绪队列中。 运行态：获得CPU执行权，正在执行的线程。由于一个CPU同一时刻只能执行一条线程，因此每个CPU每个时刻只有一条运行态的线程。  
(BLOCKED)阻塞态:当一条正在执行的线程请求某一资源失败时，就会进入阻塞态。而在Java中，阻塞态专指请求锁失败时进入的状态。  
由一个阻塞队列存放所有阻塞态的线程。处于阻塞态的线程会不断请求资源，一旦请求成功，就会进入就绪队列，等待执行。  
PS：锁、IO、Socket等都资源。

(WAITING)等待态: 当前线程中调用wait、join、park函数时，当前线程就会进入等待态。也有一个等待队列存放所有等待态的线程。线程处于等待态表示它需要等待其他线程的指示才能继续运行。进入等待态的线程会释放CPU执行权，并释放资源（如：锁）

超时等待态:当运行中的线程调用sleep(time)、wait、join、parkNanos、parkUntil时，就会进入该状态；  
它和等待态一样，并不是因为请求不到资源，而是主动进入，并且进入后需要其他线程唤醒；进入该状态后释放CPU执行权   
和 占有的资源。 (wati\notify()\notifyAll())  
与等待态的区别：到了超时时间后自动进入阻塞队列，开始竞争锁。

(TERMINATED)终止态:线程执行结束后的状态。

****