**Java 多线程编程**

Java给多线程编程提供了内置的支持。一个多线程程序包含两个或多个能并发运行的部分。程序的每一部分都称作一个线程，并且每个线程定义了一个独立的执行路径。

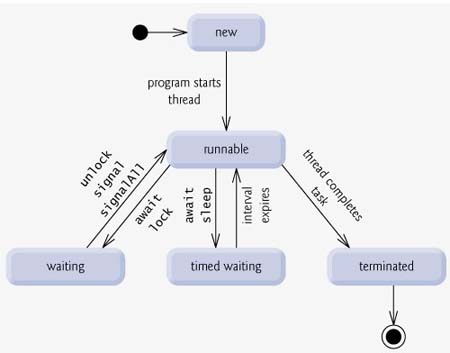
多线程是多任务的一种特别的形式。多线程比多任务需要更小的开销。

这里定义和线程相关的另一个术语：进程：一个进程包括由操作系统分配的内存空间，包含一个或多个线程。一个线程不能独立的存在，它必须是进程的一部分。一个进程一直运行，直到所有的非守候线程都结束运行后才能结束。

多线程能满足程序员编写非常有效率的程序来达到充分利用CPU的目的，因为CPU的空闲时间能够保持在最低限度。

**一个线程的生命周期**

线程经过其生命周期的各个阶段。下图显示了一个线程完整的生命周期。



* **新状态:** 一个新产生的线程从新状态开始了它的生命周期。它保持这个状态知道程序start这个线程。
* **运行状态:**当一个新状态的线程被start以后，线程就变成可运行状态，一个线程在此状态下被认为是开始执行其任务
* **就绪状态:**当一个线程等待另外一个线程执行一个任务的时候，该线程就进入就绪状态。当另一个线程给就绪状态的线程发送信号时，该线程才重新切换到运行状态。
* **休眠状态:** 由于一个线程的时间片用完了，该线程从运行状态进入休眠状态。当时间间隔到期或者等待的时间发生了，该状态的线程切换到运行状态。
* **终止状态:**一个运行状态的线程完成任务或者其他终止条件发生，该线程就切换到终止状态。

## Thread 方法

下表列出了Thread类的一些重要方法：

|  |  |
| --- | --- |
| **序号** | **方法描述** |
| 1 | **public void start()** 使该线程开始执行；**Java** 虚拟机调用该线程的 run 方法。 |
| 2 | **public void run()** 如果该线程是使用独立的 Runnable 运行对象构造的，则调用该 Runnable 对象的 run 方法；否则，该方法不执行任何操作并返回。 |
| 3 | **public final void setName(String name)** 改变线程名称，使之与参数 name 相同。 |
| 4 | **public final void setPriority(int priority)**  更改线程的优先级。 |
| 5 | **public final void setDaemon(boolean on)** 将该线程标记为守护线程或用户线程。 |
| 6 | **public final void join(long millisec)** 等待该线程终止的时间最长为 millis 毫秒。 |
| 7 | **public void interrupt()** 中断线程。 |
| 8 | **public final boolean isAlive()** 测试线程是否处于活动状态。 |

测试线程是否处于活动状态。 上述方法是被Thread对象调用的。下面的方法是Thread类的静态方法。

|  |  |
| --- | --- |
| **序号** | **方法描述** |
| 1 | **public static void yield()** 暂停当前正在执行的线程对象，并执行其他线程。 |
| 2 | **public static void sleep(long millisec)** 在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），此操作受到系统计时器和调度程序精度和准确性的影响。 |
| 3 | **public static boolean holdsLock(Object x)** 当且仅当当前线程在指定的对象上保持监视器锁时，才返回 true。 |
| 4 | **public static Thread currentThread()** 返回对当前正在执行的线程对象的引用。 |
| 5 | **public static void dumpStack()** 将当前线程的堆栈跟踪打印至标准错误流。 |

**线程的优先级**

每一个Java线程都有一个优先级，这样有助于操作系统确定线程的调度顺序。Java优先级在MIN\_PRIORITY（1）和MAX\_PRIORITY（10）之间的范围内。默认情况下，每一个线程都会分配一个优先级NORM\_PRIORITY（5）。

具有较高优先级的线程对程序更重要，并且应该在低优先级的线程之前分配处理器时间。然而，线程优先级不能保证线程执行的顺序，而且非常依赖于平台。

**线程的几个主要概念**

在多线程编程时，你需要了解以下几个概念：

* 线程同步
* 线程间通信
* 线程死锁
* 线程控制：挂起、停止和恢复

**多线程的使用**

有效利用多线程的关键是理解程序是并发执行而不是串行执行的。例如：程序中有两个子系统需要并发执行，这时候就需要利用多线程编程。

通过对多线程的使用，可以编写出非常高效的程序。不过请注意，如果你创建太多的线程，程序执行的效率实际上是降低了，而不是提升了。

请记住，上下文的切换开销也很重要，如果你创建了太多的线程，CPU花费在上下文的切换的时间将多于执行程序的时间！

原文参考：http://www.runoob.com/java/java-multithreading.html