Thread

> Thread 类的属性分析

Thread类中的常见属性包括：
线程名称(Name)：每个线程都有一个唯一的名称，可以通过setName方法设置，通过getName方法获得。
线程状态(State)：表示线程的当前状态，如新建、运行、阻塞
优先级(Priority)：线程执行的优先级，可以通过setPriority方法设置，通过getPriority方法获
是否为守护线程（Daemon：通过setDaemon(true)可以将线程设置为守护线程，守护线程不会阻止进程结束
是否属于线程池（IsThreadPoolThread：通过setDaemon(true)方法设置，表示线程是否属于托管线程
唯一标识符（ManagedThreadId：每个托管线程都有一个唯一的标识符

```
public class Thread implements Runnable {
    private volatile String name;
    private int priority;

    /* Whether or not the thread is a daemon thread. */
    private boolean daemon = false;

    /* Interrupt state of the thread - read/written directly by JVM */
    private volatile boolean interrupted;

    /* Fields reserved for exclusive use by the JVM */
    private boolean stillborn = false;
    private volatile long eetop;

    /* What will be run. */
    private Runnable target;

    /* The group of this thread */
    private ThreadGroup group;

    /* The context ClassLoader for this thread */
    private ClassLoader contextClassLoader;

    /* The inherited AccessControlContext of this thread */
    @SuppressWarnings("removal")
    private AccessControlContext inheritedAccessControlContext;

    /* For autonumbering anonymous threads. */
    private static int threadInitNumber;


    // The minimum priority that a thread can have.

    public static final int MIN_PRIORITY = 1;


    // The default priority that is assigned to a thread.

    public static final int NORM_PRIORITY = 5;
```

```
    // The maximum priority that a thread can have.

    public static final int MAX_PRIORITY = 10;
}
```

> thread 类的构造方法

- public Thread() :分配一个新的线程对象。
- public Thread(String name) :分配一个指定名字的新的线程对象。
- public Thread(Runnable target) :指定创建线程的目标对象，它实现了Runnable接口中的run方法
- public Thread(Runnable target,String name) :分配一个带有指定目标新的线程对象并指定名字。

> thread 类的常用方法

* public void run() :此线程要执行的任务在此处定义代码。
* public void start() :导致此线程开始执行; Java虚拟机调用此线程的run方法。
* public String getName() :获取当前线程名称。
* public void setName(String name)：设置该线程名称。
* public static Thread currentThread() :返回对当前正在执行的线程对象的引用。在Thread子类中就是this，通常用于主线程和Runnable实现类
* public static void sleep(long millis) :使当前正在执行的线程以指定的毫秒数暂停（暂时停止执行）不释放锁
* public static void yield()：yield只是让当前线程暂停一下，让系统的线程调度器重新调度一次，希望优先级与当前线程相同或更高的其他线程能够获得执行机会，但是这个不能保证，完全有可能的情况是，当某个线程调用了yield方法暂停之后，线程调度器又将其调度出来重新执行，释放锁。

    public final boolean isAlive()：测试线程是否处于活动状态。如果线程已经启动且尚未终止，则为活动状态。
    void join() ：等待该线程终止。
    void join(long millis) ：等待该线程终止的时间最长为 millis 毫秒。如果millis时间到，将不再等待。
    void join(long millis, int nanos) ：等待该线程终止的时间最长为 millis 毫秒 + nanos 纳秒。
    public final void stop()：`已过时`，不建议使用。强行结束一个线程的执行，直接进入死亡状态。run()即刻停止，可能会导致一些清理性的工作得不到完成，如文件，数据库等的关闭。同时，会立即释放该线程所持有的所有的锁，导致数据得不到同步的处理，出现数据不一致的问题。
    void suspend() / void resume() ： 这两个操作就好比播放器的暂停和恢复。二者必须成对出现，否则非常容易发生死锁。suspend()调用会导致线程暂停，但不会释放任何锁资源，导致其它线程都无法访问被它占用的锁，直到调用resume()。`已过时`，不建议使用。

    每个线程都有一定的优先级，同优先级线程组成先进先出队列（先到先服务），使用分时调度策略。优先级高的线程采用抢占式策略，获得较多的执行机会。每个线程默认的优先级都与创建它的父线程具有相同的优先级。
- Thread类的三个优先级常量：
  - MAX_PRIORITY（10）：最高优先级
  - MIN _PRIORITY （1）：最低优先级
  - NORM_PRIORITY （5）：普通优先级，默认情况下main线程具有普通优先级。

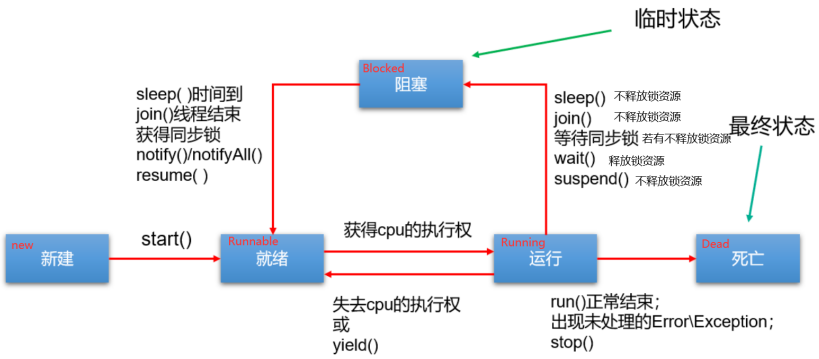
* public final int getPriority() ：返回线程优先级

* public final void setPriority(int newPriority) ：改变线程的优先级，范围在[1,10]之间。

调用setDaemon(true)方法可将指定线程设置为守护线程。必须在线程启动之前设置，否则会报IllegalThreadStateException异常。
调用isDaemon()可以判断线程是否是守护线程。

多线程的生命周期

JDK1.5之前



1.新建
当一个Thread类或其子类的对象被声明并创建时，新生的线程对象处于新建状态。此时它和其他Java对象一样，仅仅由JVM为其分配了内存，并初始化了实例变量的值。此时的线程对象并没有任何线程的动态特征，程序也不会执行它的线程体run()。
2.就绪
但是当线程对象调用了start()方法之后，就不一样了，线程就从新建状态转为就绪状态。JVM会为其创建方法调用栈和程序计数器，当然，处于这个状态中的线程并没有开始运行，只是表示已具备了运行的条件，随时可以被调度。至于什么时候被调度，取决于JVM里线程调度器的调度。
> 注意：
>
> 程序只能对新建状态的线程调用start()，并且只能调用一次，如果对非新建状态的线程，如已启动的线程或已死亡的线程调用start()都会报错IllegalThreadStateException异常。
3.运行
如果处于就绪状态的线程获得了CPU资源时，开始执行run()方法的线程体代码，则该线程处于运行状态。如果计算机只有一个CPU核心，在任何时刻只有一个线程处于运行状态，如果计算机有多个核心，将会有多个线程并行(Parallel)执行。
当然，美好的时光总是短暂的，而且CPU讲究雨露均沾。对于抢占式策略的系统而言，系统会给每个可执行的线程一个小时间段来处理任务，当该时间用完，系统会剥夺该线程所占用的资源，让其回到就绪状态等待下一次被调度。此时其他线程将获得执行机会，而在选择下一个线程时，系统会适当考虑线程的优先级。
4.阻塞
当在运行过程中的线程遇到如下情况时，会让出 CPU 并临时中止自己的执行，进入阻塞状态：
* 线程调用了sleep()方法，主动放弃所占用的CPU资源；
* 线程试图获取一个同步监视器，但该同步监视器正被其他线程持有；
* 线程执行过程中，同步监视器调用了wait()，让它等待某个通知（notify）；
* 线程执行过程中，同步监视器调用了wait(time)
* 线程执行过程中，遇到了其他线程对象的加塞（join）；
* 线程被调用suspend方法被挂起（已过时，因为容易发生死锁）；
当前正在执行的线程被阻塞后，其他线程就有机会执行了。针对如上情况，当发生如下情况时会解除阻塞，让该线程重新进入就绪状态，等待线程调度器再次调度它：
* 线程的sleep()时间到；
* 线程成功获得了同步监视器；
* 线程等到了通知(notify)；
* 线程wait的时间到了
* 加塞的线程结束了；

* 被挂起的线程又被调用了resume恢复方法（已过时，因为容易发生死锁）；

5.死亡

线程会以以下三种方式之一结束，结束后的线程就处于死亡状态：

* run()方法执行完成，线程正常结束
* 线程执行过程中抛出了一个未捕获的异常（Exception）或错误（Error）
* 直接调用该线程的stop()来结束该线程（已过时）

> JDK1.5以及以后

```
public enum State {
    NEW,
    RUNNABLE,
    BLOCKED,
    WAITING,
    TIMED_WAITING,
    TERMINATED;
}
```