01 Java并发编程基础

C-2 创建: 张林伟, 最后修改: 张林伟 2018-10-31 15:45

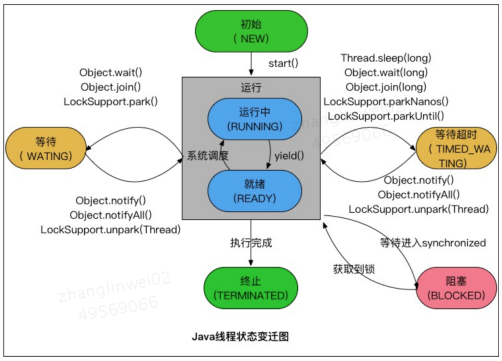
线程简介

现代操作系统在运行一个程序时，会为其创建一个进程。

现代操作系统调度的最小单元是线程，也叫轻量级进程。一个进程里可以创建多个线程，每个线程有自己的计数器、堆栈和局部变量等属性，并且能够访问共享的内存变量。

Java 线程状态

状态名称	说　明
NEW	初始状态，线程被构建，但还没有调用 start() 方法
RUNNABLE	运行状态，Java 线程将操作系统中的就绪和运行两种状态笼统称为“运行中”
BLOCKED	阻塞状态，表示线程阻塞于锁
WAITING	等待状态，表示线程进入等待状态，需要等待其他线程做出一些特定动作（通知或中断）
TIMED_WAITING	超时等待状态，可以在指定时间自行返回
TERMINATED	终止状态，表示当前线程已经执行完毕



Java线程状态变迁图

Daemon 线程

Daemon 线程是一种支持型线程，主要用作程序中后台调度以及支持性工作。

这意味着当 Java 虚拟机中不存在非 Daemon 线程时，Java 虚拟机将会退出。

代码块 Java

```java
// 需要在线程启动前设置
thread.setDaemon(true);
```

启动和终止线程

一个新构造的线程对象是有其 parent 线程来进行空间分配的，而 child 继承了 parent 是否为 Daemon、优先级和加载资源的 contextClassLoader 以及可继承的 ThreadLocal，同时会分配一个唯一的 ID 来标识这个 child 线程。

点击展开内容

代码块 Java

```java
    /**
     * Initializes a Thread.
     *
     * @param g the Thread group
     * @param target the object whose run() method gets called
     * @param name the name of the new Thread
     * @param stackSize the desired stack size for the new thread, or
     *        zero to indicate that this parameter is to be ignored.
```

```
     * @param acc the AccessControlContext to inherit, or
     *            AccessController.getContext() if null
     * @param inheritThreadLocals if {@code true}, inherit initial values for
     *            inheritable thread-locals from the constructing thread
     */
    private void init(ThreadGroup g, Runnable target, String name,
                      long stackSize, AccessControlContext acc,
                      boolean inheritThreadLocals) {
        if (name == null) {
            throw new NullPointerException("name cannot be null");
        }

        this.name = name;

        Thread parent = currentThread();  // 父线程
        SecurityManager security = System.getSecurityManager();
        if (g == null) {
            /* Determine if it's an applet or not */

            /* If there is a security manager, ask the security manager
               what to do. */
            if (security != null) {
                g = security.getThreadGroup();
            }

            /* If the security doesn't have a strong opinion of the matter
               use the parent thread group. */
            if (g == null) {
                g = parent.getThreadGroup();
            }
        }

        /* checkAccess regardless of whether or not threadgroup is
           explicitly passed in. */
        g.checkAccess();

        /*
         * Do we have the required permissions?
         */
        if (security != null) {
            if (isCCLOverridden(getClass())) {
                security.checkPermission(SUBCLASS_IMPLEMENTATION_PERMISSION);
            }
        }

        g.addUnstarted();

        this.group = g;
        this.daemon = parent.isDaemon();  // 继承父线程
        this.priority = parent.getPriority();  // 继承父线程
        if (security == null || isCCLOverridden(parent.getClass()))  // 继承父线程
            this.contextClassLoader = parent.getContextClassLoader();
        else
            this.contextClassLoader = parent.contextClassLoader;
        this.inheritedAccessControlContext =
                acc != null ? acc : AccessController.getContext();
        this.target = target;
        setPriority(priority);
        if (inheritThreadLocals && parent.inheritableThreadLocals != null)  // 继承父线程
            this.inheritableThreadLocals =
                ThreadLocal.createInheritedMap(parent.inheritableThreadLocals);
        /* Stash the specified stack size in case the VM cares */
        this.stackSize = stackSize;

        /* Set thread ID */
```

```
74            tid = nextThreadID();  //分配一个唯一的 ID
75        }
```

线程完成初始化后，调用 start() 方法即可启动线程。

调用 start() 方法含义：当前线程（即 parent 线程）同步告知 Java 虚拟机，只要线程规划器空闲，应立即启动调用 start() 方法的线程。

理解中断

中断可以理解为线程的一个标识位属性，它表示一个运行中的线程是否被其他线程进行了中断操作。
其他线程通过调用该线程的 Interrupt() 方法对其进行中断操作。

安全终止线程

- 中断操作
- 标识位

线程间通信

Java支持多个线程同时访问同一个对象或者对象的成员变量，由于每个线程可以拥有这个变量的拷贝（虽然对象以及成员变量分配的内存是在共享内存中，但是每个执行的线程还是可以拥有一份拷贝，这样做的目的是加速程序执行，这是现代多核处理器的一个显著特性），所以程序在执行过程中，一个线程看到的变量不一定是最新的。

volatile

- 可见性：告知程序任何对该变量的访问均需要从共享内存中获取，而对它的改变必须同步刷新回共享内存，它能保证所有线程对变量访问的可见性。
- 禁止指令重排序

synchronized

- 修饰方法：依靠方法修饰符上的 ACC_SYNCHRONIZED

  修饰普通方法：锁当前实例对象

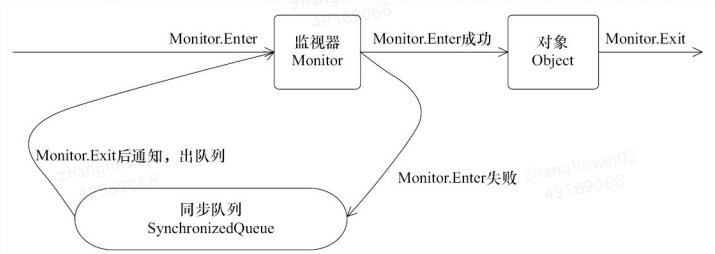
  修饰静态方法：锁当前类的 Class 对象

- 同步块：利用 monitorenter 和 monitorexit 指令

本质都是对一个对象的监视器（monitor）进行获取，这个获取过程是排他的，因此同一时刻只有一个线程获取到。

任意一个对象都有自己的监视器，当这个对象由同步块或者这个对象的同步方法调用时，执行方法的线程必须先获取该对象的监视器才能进入同步块或者同步方法，而没有获取到监视器（执行该方法）的线程

将会被阻塞在同步块或者同步方法的入口处，进入 BLOCKED 状态。



等待 / 通知机制

一个线程修改了一个对象的值，而另一个线程感知到了变化，然后进行相应操作。

- 循环检测变量是否修改（难以确保及时性、难以降低开销）
- Java 内置的 等待 / 通知机制

方法	描述
notify()	通知一个在对象上等待的线程，使其从 wait() 方法返回，而返回的前提是该线程获取到了对象的锁
notifyAll()	通知所有在对象上等待的线程。所有线程全部移到同步队里，WAITING -> BLOCKED
wait()	调用后进入 WAITING 状态，只有等待其他线程通知或者被中断才返回，调用后会释放锁
wait(long)	超时等待，毫秒。等待 n 毫秒，没有通知则超时返回

wait(long, int)	超时等待，纳秒

管道输入/输出流

管道输入/输出流主要用于线程之间的数据传输，而传输媒介是内存。

- PipedOutputStream、PipedInputStream 面向字节
- PipedReader、PipedWriter 面向字符

Thread.join()的使用

线程 A 执行 thread.join()，含义是：当前线程 A 等待 thread 线程终止后才从 thread.join 返回。

当线程终止时，会调用线程自身的 notifyAll() 方法，会通知所有等待在该线程对象上的线程。

ThreadLocal的使用

ThreadLocal，即线程变量，是一个以ThreadLocal对象为键、任意对象为值的存储结构。这个结构被附带在线程上，也就是说一个线程可以根据一个ThreadLocal对象查询到绑定在这个线程上的一个值。

可以通过set(T)方法来设定值，在当前线程通过get()方法拿到值。