## 等待/通知机制（wait/notify）

线程与线程之间不一定是独立的个体，他们之间可以相互通信和协作。

等待通知机制的应用案例非常广泛，比如常见的消息发布订阅就是一种等待通知的实现，一个线程订阅某个消息/事件，然后就开始等待，然后另一个线程发布这个消息，然后通知第一个线程，第一个线程接收这个消息并处理。

Java多线程中等待通知机制的实现离不开下面这两个方法：

1）wait() ：这个方法的作用是使当前执行代码的线程进行等待，wait()方法是Object类的方法，该方法用来将当前线程置入“预执行队列”中，并且在wait()中所在的代码出停止执行，直到接到通知或被中断为止。在调用wait()之前，线程必须获得该对象的对象级别锁，即只能在同步方法或同步代码块中调用wait()方法。在执行wait()方法后，当前线程释放锁。在从wait()返回前，线程与其他线程竞争重新获得锁。如果调用wait()时没有持有适当的锁，则抛出一个非检查异常（IllegalMonitorStateException）。

（Java Object对象还有一个wait(long) 方法，这个方法接收一个长整型参数（毫秒数），调用这个方法后会先等待参数指定的时间长度，超时自动唤醒）

2）notify() ：这个方法也要在同步方法或同步代码块中调用，即在调用之前线程也必须取得该对象的锁，否则也会抛出一个同样的非检查异常IllegalMonitorStateException，该方法用来通知那些可能等待该对象的对象锁的其他线程，如果有多个线程等待，则由线程规划器随机挑选出一个呈wait态的线程，对其发出notify通知，并使它获得该对象的锁。需要说明一点，在执行notify()方法之后，当前线程不会马上释放该对象锁，呈wait态的线程也不能马上获得该对象的锁，需要等执行notify方法的线程将剩余的程序执行完，也就是退出synchronized代码块之后，当前线程才会释放锁，同样的wait态的线程也才能获取到锁。调用一次notify()方法只能唤醒一个wait态的线程，且是随机唤醒的，并不能指定唤醒哪一个线程，除非调用notifyAll方法。

一句话总结这两个方法：

wait是线程停止运行（退出运行态，进入等待队列）

notify使某个停止的线程（等待队列的线程）继续运行。

（注：这里的“继续运行”是指线程进入运行态，而Java的线程的运行态实际相当于操作系统的就绪态和运行态的统称，因此有可能线程被notify了也得到了对象锁并不一定马上执行，线程会进入操作系统的就绪态，直到符合操作系统或者说处理器的调度策略，使之真正开始运行。后面我会专门抽一篇博客讲讲Java中的线程状态）

## 方法join的使用

假设在线程X中有z线程调用了start方法（z.start()），并且z线程调用了start方法后还调用了z.join()方法。

那么线程X在z.join()后会被无限期挂起，知道z线程执行完被销毁才开始执行后面的故事。

*X线程中实例化并运行z线程* {

MyThread z = new MyThread();

z.start();

z.join();

System.out.println(“我等z线程执行完被销毁才开始执行”);

……

}

因此---

方法join具有是线程排队的所用，有点类似同步的效果，join与synchronized的区别是：join在内部使用wait方法进行等待，而synchronized关键字使用的是“对象监视器”原理做同步。

在join方法中，如果线程对象出现异常，则当前线程同样也会出现异常。

Object还有个join(long) 方法，与上面的wait(long)有点类似，就是等待参数指定的时间，如果线程对象依旧没有执行完被销毁，那么自动唤醒执行后面的故事。

## 类ThreadLocal的使用

变量值的共享可以使用public static变量的形式，所有线程都会使用同一个public static变量。如果想实现每一个线程都有自己的共享变量该如何解决呢？于是好心的JDK帮我们设计了一个ThreadLocal。

ThreadLocal主要解决的就是每个线程绑定自己的值，可以将之比喻成全局存放数据的盒子，盒子中可以存储每个线程的私有数据。

来段代码认识一下ThreadLocal这位盆友：

（这段代码是用记事本打的哦，如有编译错误纯属意外）

public class TestThreadLocal() {

public static ThreadLocal tl = new ThreadLocal();

public static void main(String[] args) {

if(tl.get == null) {

System.out.println(“从未存放过值”);

tl.set(“我的值”);

}

System.out.println(tl.get());

}

}

运行结果（应该是这样的）：

从未存放过值

我的值

通过上面的一段代码可以看到ThreadLocal有两个很重要的方法set()和get()，我们可以将需要在线程上下文访问的对象放入其中，然后必要的时候去get。