Java并发

## 线程

Java对并发编程提供了语言级别的支持。Java通过线程来实现并发编程。一个线程通常完成某个特定的任务，一个进程可以拥有多个线程，当这些线程一起执行的时候，就实现了并发。与操作系统中的进程相似，每个线程看起来好像拥有自己的CPU，但是其底层是通过切分CPU时间来实现的。与进程不同的是，线程并不是相互独立的，它们通常要相互合作来完成一些任务。

## 2. 创建线程

  Java中实现多线程有两种方法：继承Thread类、实现Runnable接口，在程序开发中只要是多线程，肯定永远以实现Runnable接口为主，因为实现Runnable接口相比继承Thread类有如下优势：

    1、可以避免由于Java的单继承特性而带来的局限；

    2、增强程序的健壮性，代码能够被多个线程共享，代码与数据是独立的；

3、适合多个相同程序代码的线程区处理同一资源的情况。

**休眠（sleep）**

Thread类有一个sleep**静态**方法，你可以将一个long类型的数据做参数传进去，单位是毫秒，表示线程将会休眠的时间。

**让步（yield）**

Thread类还有一个名为yield()的静态方法。这个方法的作用是为了**建议**当前正在运行的线程做个让步，让出CPU时间给别的线程来运行。

**串入（join）**

当一个线程t1在另一个线程t2上调用**t1.join()**方法的时候，线程t2将等待线程t1运行结束之后再开始运行。

**守护线程（deamon thread）**

通常，程序中有一些线程的工作并不是不可或缺的，它只是用来协助其他线程来工作。这样的线程叫做守护线程或者后台线程。当进程中的所有非守护线程结束时，守护线程也就终止了，就算它还没有完全完成自己的任务。我们可以在一个线程启动之前调用**setDaemon()**方法来将这个线程设定成守护线程。

一个线程在休眠，不能保证不被中断和一个人睡觉，不能保证不被打搅是一个道理。这就是为什么Thread的sleep()方法会可能抛出一个InterruptedException的原因。同样地，Thread类的join()方法，Object类的wait()方法，都可能抛出InterruptedException。

## 3.并发编程模型

并发模型指定了系统中的线程如何通过协作来完成分配给它们的作业。不同的并发模型采用不同的方式拆分作业，同时线程间的协作和交互方式也不相同。

**1.并行工作者**

**2. 流水线模式**

**3. 函数式并行**

## Synchronized

**竞态条件 & 临界区**

当两个线程竞争同一资源时，如果对资源的访问顺序敏感，就称存在竞态条件。导致竞态条件发生的代码区称作临界区。允许被多个线程同时执行的代码称作线程安全的代码。线程安全的代码不包含竞态条件。当多个线程同时更新共享资源时会引发竞态条件。

如果一个资源的创建，使用，销毁都在同一个线程内完成，

且永远不会脱离该线程的控制，则该资源的使用就是线程安全的。

每一个对象本身都隐含着一个**锁对象**，这个锁对象就是用来解决并发问题的互斥量（mutex）。要调用一个对象的synchronized方法的线程，必须持有这个对象的锁对象，在执行完毕之后，必须释放这个锁对象，以让别的线程得到这个锁对象。因为一个对象仅有一个锁对象，这就保证了在同一时刻，最多只有一个线程能够调用并执行这个对象的synchronized方法。其他想调用这个对象的synchronized方法的线程必须等待当前线程释放锁。

synchronized除了能修饰方法之外，还能创建同步块。

public void doSomething() {

//一些操作

synchronized(this) {

//一些需要被同步的操作

}

//另外一些操作

}

原子性（atomicity）

具有原子性的操作被称为原子操作。原子操作在操作完毕之前不会线程调度器中断。在Java中，对除了long和double之外的基本类型的简单操作都具有原子性。简单操作就是赋值或者return。比如”a = 1;“和 “return a;”这样的操作都具有原子性。

## Volatile

在多线程并发编程中synchronized和Volatile都扮演着重要的角色，Volatile是**轻量级的synchronized**，它在多处理器开发中保证了共享变量的“可见性”。可见性的意思是当一个线程修改一个共享变量时，另外一个线程能读到这个修改的值。如果给一个变量加上volatile修饰符，就相当于：每一个线程中一旦这个值发生了变化就马上刷新回主存，使得各个线程取出的值相同。编译器不要对这个变量的读、写操作做优化。但是volatile并不能提供原子性。