# 1、线程基础、线程之间的共享和协作

## 基础概念

**CPU核心数和线程数的关系**

核心数:线程数=1:1 ;使用了超线程技术后---> 1:2

**CPU时间片轮转机制**

又称RR调度，会导致上下文切换

**什么是进程和线程**

进程：程序运行资源分配的最小单位，进程内部有多个线程，会共享这个进程的资源

线程：CPU调度的最小单位，必须依赖进程而存在。

**澄清并行和并发**

并行：同一时刻，可以同时处理事情的能力

并发：与单位时间相关，在单位时间内可以处理事情的能力

**高并发编程的意义、好处和注意事项**

好处：充分利用cpu的资源、加快用户响应的时间，程序模块化，异步化

问题：

线程共享资源，存在冲突；

容易导致死锁；

启用太多的线程，就有搞垮机器的可能

## 认识Java里的线程

**新启线程的方式**

三种

**怎么样才能让Java里的线程安全停止工作呢**

线程自然终止：自然执行完或抛出未处理异常

stop()，resume(),suspend()已不建议使用，stop()会导致线程不会正确释放资源，suspend()容易导致死锁。

java线程是协作式，而非抢占式

调用一个线程的interrupt() 方法中断一个线程，并不是强行关闭这个线程，只是跟这个线程打个招呼，将线程的中断标志位置为true，线程是否中断，由线程本身决定。

isInterrupted() 判定当前线程是否处于中断状态。

static方法interrupted() 判定当前线程是否处于中断状态，同时中断标志位改为false。

方法里如果抛出InterruptedException，线程的中断标志位会被复位成false，如果确实是需要中断线程，要求我们自己在catch语句块里再次调用interrupt()。

## 对Java里的线程再多一点点认识

**线程常用方法和线程的状态**

线程只有5种状态。整个生命周期就是这几种状态的切换。

run()和start() ：run方法就是普通对象的普通方法，只有调用了start()后，Java才会将线程对象和操作系统中实际的线程进行映射，再来执行run方法。

yield() ：让出cpu的执行权，将线程从运行转到可运行状态，但是下个时间片，该线程依然有可能被再次选中运行。

**线程的优先级**

取值为1~10，缺省为5，但线程的优先级不可靠，不建议作为线程开发时候的手段

**守护线程**

和主线程共死，finally不能保证一定执行

## 线程间的共享

#### synchronized内置锁

对象锁，锁的是类的对象实例。

类锁 ，锁的是每个类的的Class对象，每个类的的Class对象在一个虚拟机中只有一个，所以类锁也只有一个。

#### volatile关键字

适合于只有一个线程写，多个线程读的场景，因为它只能确保可见性。

#### ThreadLocal

线程变量。可以理解为是个map，类型 Map<Thread,Integer>

## 线程间协作

轮询：难以保证及时性，资源开销很大，

### 等待和通知

wait() 对象上的方法

notify/notifyAll 对象上的方法

**等待和通知的标准范式**

等待方：

1. 获取对象的锁；
2. 循环里判断条件是否满足，不满足调用wait方法，
3. 条件满足执行业务逻辑

通知方来说

* 1. 获取对象的锁；
  2. 改变条件
  3. 通知所有等待在对象的线程

notify和notifyAll应该用谁？

应该尽量使用notifyAll，使用notify因为有可能发生信号丢失的的情况

**等待超时模式实现一个连接池**

假设 等待时间时长为T，当前时间now+T以后超时

long overtime = now+T;

long remain = T;//等待的持续时间

while(result不满足条件&& remain>0){

wait(remain);

remain = overtime – now;//等待剩下的持续时间

}

return result;

### join()方法

面试点

线程A，执行了线程B的join方法，线程A必须要等待B执行完成了以后，线程A才能继续自己的工作

### 调用yield() 、sleep()、wait()、notify()等方法对锁有何影响？

面试点

线程在执行yield()以后，持有的锁是不释放的

sleep()方法被调用以后，持有的锁是不释放的

调动方法之前，必须要持有锁。调用了wait()方法以后，锁就会被释放，当wait方法返回的时候，线程会重新持有锁

调动方法之前，必须要持有锁，调用notify()方法本身不会释放锁的