**1、并发与并行的区别**

并发指的是单个CPU利用时间片轮转的方式去处理多个线程。

并行指的是进程产生的线程数少于CPU的核数，不同进程产生的线程可以分配给不同的CPU运行。

**2、守护线程与非守护线程**

守护线程：后台运行的线程，进程结束守护线程自然就结束，比如java的垃圾回收线程，将线程设置为守护线程一定要在线程调用前设置。

jre判断程序结束的标准为前台线程是否是否执行完毕，不管后台线程。

**3、线程组与线程池的区别**

线程组是统一管理线程，后者是管理线程的生命周期，复用线程，减少创建于销毁线程的开销。

线程池主要是减少线程的创建于销毁，减少系统的开销。

**4.JVM分为主内存与工作内存，**主内存存放的是程序中所有的类实例、静态数据等变量，工作内存是该线程从主内存拷过来的变量以及访问方法所取得的局部变量。

**可见性问题：**可见性是指当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。

volatiles、snchronized、Lock关键字来保证可见性

**原子性问题：**即一个操作或者多个操作 要么全部执行并且执行的过程不会被任何因素打断，要么就都不执行。

synchronized和Lock保证原子性

**顺序性：**即程序执行的顺序按照代码的先后顺序执行。主要在多线程中会发生顺序性问题。volatiles保证顺序性。

**5、synchronized与lock的区别（隐式锁和显示锁的区别）**

synchronized获得锁后其他线程只能等待

lock必须手动释放与开启锁，synchronized不需要手动释放和开启

Lock可以知道线程有没有成功获取到锁

**死锁**

概念：多个线程循环等待其他线程占有的资源而无线等待下去的局面。

四个条件

互斥

不可抢占

占有且申请

循环等待

**自旋锁**

自旋锁可以使线程在没有取得锁的时候，不被挂起（阻塞），而转去执行一个空循环，（即所谓的自旋，就是自己执行空循环），若在若干个空循环后，线程如果可以获得锁，则继续执行。若线程依然不能获得锁，才会被挂起。

好处是减少线程切换导致的时间浪费、过多占据CPU时间：如果锁的当前持有者长时间不释放该锁，那么等待者将长时间的占据cpu时间片，导致CPU资源的浪费，因此可以设定一个时间，当锁持有者超过这个时间不释放锁时，等待者会放弃CPU时间片阻塞；

**锁的种类**

公平锁与非公平锁

可重入锁：synchronized,reentrantLock

共享锁与独享锁

分段锁：conCurrentHashMap

自旋锁

**6、终止线程**

使用标志位

使用stop

使用interrupt

class MyThread implements Runnable {

  private boolean flag = true;

  @Override

  public void run() {

    // TODO Auto-generated method stub

    int i = 0;

    while (this.flag) {

      System.out.println(Thread.currentThread().getName() + "线程运行，i = "

          + (i++));

    }

  }

  public void stop() {

    this.flag = false;

  }

}

参考资料：<http://blog.csdn.net/zhanjichun_2008/article/details/6612980>

**7、多线程同步的方法**

wait方法：

        该方法属于Object的方法，wait方法的作用是使得当前调用wait方法所在部分（代码块）的线程停止执行，并释放当前获得的调用wait所在的代码块的锁，并在其他线程调用notify或者notifyAll方法时恢复到竞争锁状态（一旦获得锁就恢复执行）。

        调用wait方法需要注意几点：

        第一点：wait被调用的时候必须在拥有锁（即synchronized修饰的）的代码块中。

        第二点：恢复执行后，从wait的下一条语句开始执行，因而wait方法总是应当在while循环中调用，以免出现恢复执行后继续执行的条件不满足却继续执行的情况。

        第三点：若wait方法参数中带时间，则除了notify和notifyAll被调用能激活处于wait状态（等待状态）的线程进入锁竞争外，在其他线程中interrupt它或者参数时间到了之后，该线程也将被激活到竞争状态。

        第四点：wait方法被调用的线程必须获得之前执行到wait时释放掉的锁重新获得才能够恢复执行。

notify方法和notifyAll方法：

        notify方法通知调用了wait方法，但是尚未激活的一个线程进入线程调度队列（即进入锁竞争），注意不是立即执行。并且具体是哪一个线程不能保证。另外一点就是被唤醒的这个线程一定是在等待wait所释放的锁。

        notifyAll方法则唤醒所有调用了wait方法，尚未激活的进程进入竞争队列。

synchronized关键字：

        第一点：synchronized用来标识一个普通方法时，表示一个线程要执行该方法，必须取得该方法所在的对象的锁。

        第二点：synchronized用来标识一个静态方法时，表示一个线程要执行该方法，必须获得该方法所在的类的类锁。

        第三点：synchronized修饰一个代码块。类似这样：synchronized(obj) { //code.... }。表示一个线程要执行该代码块，必须获得obj的锁。这样做的目的是减小锁的粒度，保证当不同块所需的锁不冲突时不用对整个对象加锁。利用零长度的byte数组对象做obj非常经济。

volatile关键字

重入锁  ReentrantLock类

使用局部变量实现线程同步 ：ThreadLocal，则每一个使用该变量的线程都获得该变量的副本，  副本之间相互独立，这样每一个线程都可以随意修改自己的变量副本，而不会对其他线程产生影响。比如线程私有的looper

**8、线程池的作用**

好处

重用线程池中的线程：减少线程创建于销毁带来的开销

有效控制线程的最大并发数，避免大量的线程之间因相互抢占系统资源而导致的阻塞现象。

能够对线程进行简单的管理，并提供定时执行以及指定间隔循环执行的功能。

2.具体分析

减少在创建和销毁线程上所花的时间以及系统资源的开销

如不使用线程池，有可能造成系统创建大量线程而导致消耗完系统内存以及”过度切换”。

线程池，就是在调用线程的时候初使化一定数量的线程，有线程过来的时候，先检测初使化的线程还有空的没有，没有就再看当前运行中的线程数是不是已经达到了最大数，如果没有，就新分配一个线程去处理；但如果已经达到了最大数，另外的线程就只有等了，直到有新的“服务员”为止。线程池的优点就是可以管理线程，有一个高度中枢，这样程序才不会乱，保证系统不会因为大量的并发而因为资源不足挂掉。

3.实现类：ThreadPoolExecutor

corePoolSize:核心线程数：一直存活，超时停止。

maximumPollSize：最大线程数

keepAliveTime：超时时长，非核心线程等待的时间超过就会被回收。也可以作用于核心线程。

workQueue：任务队列，通过execute方法提交的runnable会存在在这里面。

threadFactory：为线程池创建新线程的功能。

4.四中常见的线程池

FixedThreadPool：线程数量固定的线程，只有核心线程，任务队列没有限制，没有超时机制。更快的响应外界的请求。

CachedThreadPool：线程数量不固定，只有非核心线程，所有的线程都有超时机制。适合执行大量的耗时较少的任务

ScheduledThreadPool：核心线程数量固定，非核心线程不固定。非核心线程闲置会被立刻回收。适合于定时任务和具有固定周期的任务。

SingleThreadPool：只有一个核心线程，所有的任务按同一个线程中顺序执行。统一外界的所有任务到一个线程中。

**9、多线程的实现方式**

继承Thread，重写run方法，start开启线程，无返回值

实现Runnable接口，start开启，无返回值

实现Callable接口，有返回值

使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。

使用FutureTask对象作为Thread对象的target创建并启动新线程。

调用FutureTask对象的get()方法来获得子线程执行结束后的返回值

**经典题：**

1、有一个变量int a=0；两个线程同时进行+1操作，每个线程加100次，不加锁，最后a的值是2-200；100-200；

<http://zhan.renren.com/h5/entry/3602888498047361672>

**参考链接：**

**Java并发编程**

[**http://www.importnew.com/18126.html**](http://www.importnew.com/18126.html)

**synchronized与lock的区别**

[**http://www.cnblogs.com/handsomeye/p/5999362.html**](http://www.cnblogs.com/handsomeye/p/5999362.html)