同步（synchronous）和异步（asynchronous）

概念

同步和异步通常用来形容一次方法调用。同步方法调用一旦开始，调用者必须等到方法调用完成并返回后，才能继续后续的工作。异步方法调用更像一个消息传递，被调用之后，调用者马上开始后续工作。而异步方法通常在另外一个线程

中“真实”地执行。整个过程不会阻碍调用者的工作，如果异步调用需要返回结果，那么当这个异步调用真实完成时，则会通知调用者

举例说明

同步： 去超市买食物，通常是选购食物，结账，拿回家，到了这里才算完成了购物。在购物期间你无法再家里看电视，吹空调。
异步： 打开电脑，通过网络超市订购食物，当你网上支付完成。这个时候购物就已经完成，后面只需要等待快递员送货上门即可。在收到货的期间，可以看电视，吹空调

并发与并行

概念

并行：真实的多个任务同时进行
并发：一会执行A任务，一会儿执行B任务，来回切换的速度非常快，导致错觉认为任务是并行执行

临界区

概念

临界区用来表示一种公共的资源或者共享数据，可以被多个线程使用。但是每一次，只有一个线程使用它，一旦临界区被占用，那么其他线程想要使用这个资源，就必须等待

举例说明

办公司有一台打印机，打印机一次只能执行一个任务，如果A和B两个人同时需要打印文件，很显然，如果A先下发了打印任务，那么B就只能等待A打印完成。如果一会打印A，一会打印B，最终打印出来的文件既不是A所需要的也不是B所需要的

阻塞（Blocking）和非阻塞（Non-Blocking）

概念

阻塞：当多个线程同时访问一个临界区资源的时候，只能排队，一个线程处理完

毕之后，后面的线程就必须等待正在访问的线程执行完毕。这种情况就叫做阻塞
非阻塞：当多个线程访问同一个资源的时候，不设置任何限制，多个线程可以同时处理

举例说明

阻塞：在跑步机上跑步，一台跑步机上面同时只能有一个人进行跑步，如果后面的人要进行跑步，必须等待正在跑步的人下来
非阻塞：看展览，在一个展览上所展示的物品，可以同时供多个围观群众欣赏

死锁、活锁、饥饿

概念

死锁

当两个线程同时都需要资源A和B 才能进行下一步的工作，但是线程a拿到了A，线程b拿到了资源B，两个线程互不释放，最后造成线程a无法正常工作，线程b也无法正常工作

活锁

当两个线程同时都需要资源A和B 才能进行下一步的工作,但是线程a拿到了A，线程b拿到了资源B。 这时候a发现无法完成任务，就释放了手里的自由A，线程b也发现无法完成任务，也释放了手里的资源B，这时候a线程拿到了资源B，b线程拿到了资源A，又无法满足条件进行下一步。又继续各自释放手里的资源，如此循环

饥饿

当一个线程的优先级非常的低，每次都无法执行，一直被高优先级的线程执行，迟迟不能执行

举例说明

死锁

开一把锁需要同时需要两把钥匙，两个人各自拿了其中一把，都不放开手里的钥匙，这样永远无法完成打开锁

活锁

电梯出来一个人，一个人进电梯。 出来的人 往走，进去的也往左。这时候出来的往右，进去的也往右，不断循环。。。。最终出去的无法离开，进去的无法进去

饥饿

鸟妈妈喂食物给小鸟吃，有一个小鸟弱小，每次鸟妈妈抓回来的虫子都被其他的小鸟吃掉，这个弱小的鸟每次都吃不到虫子

并发级别

由于临近区的存在，多线程之间的并发必须受到控制，根据并发的策略，可以把并发级别进行分类，大致上可以分为阻塞、无饥饿、无障碍，无等待等几种.

阻塞（blocking）

一个线程是阻塞的，那么在其他线程释放资源之前，当前线程无法继续执行。当使用synchronize关键字，或者重入锁，用的就是阻塞线程. 无论是synchronize或者重入锁，都会试图在执行后续代码前，得到临近区的锁，如果得不到，线程会被挂起等待，直到占了所需要的资源为止.

无饥饿

如果线程之间存在优先级，那么线程调度的时候总是倾向于满足高优先级的线程，也就是说，对于同一个资源的分配，这是不公平的，就像排队一样，不断有人插队，那么后面的人永远无法排到前面去

无障碍

无障碍是一种最弱的非阻塞调度。两个线程如果无障碍的执行，那么他们不会因为临界区的问题一方线程被挂起。 但是当两个线程都进入了临界区，一起把数据改坏了，那么无障碍线程一旦检查到了这种状况，它就会立即对自己所做的修改进行回滚，确保数据的安全。但是如果没有数据资源的竞争发生，那么线程就会顺利完成工作，然后走出临界区。

无锁（非阻塞同步）

无锁的并行都是无障碍的，在无锁的情况下，所有的线程都尝试对临界区进行访问，但是不同的是，无锁的并发保证必然有一个线程能够在有限的次数内完成操作，离开临界区。

无等待

无锁只需要有一个线程可以在有限的次数内完成操作，而无等待则是在无锁的基础上更进一步进行扩展。它要求所有的线程都必须在有限次数内完成，这样就不会产生饥饿问题。如果限制这个步骤上限，还可以进一步分解为有界无等待和线程无关的无等待几种，它们之间的区别只是对循环次数的限制不同。

基本思想是，对数据的读取可以不加以任何控制，因此所有的读取线程都是无等待的，它们既不会被锁定等待，也不会引起任何的冲突。但是在写操作的时候，

先取得原始数据的副本，接着只修改副本数据，修改完成后，在合适的时机回写数据

Java内存模型（JMM）

上面所描述的概念，是使用任何一门语言编写并发程序都会涉及到的问题，在Java内存模型中主要围绕着原子性、有序性、可见性来建立的

在java中每个线程都有自己独立的工作内存，里面保存该线程使用到的变量的副本（主内存中该变量的一份拷贝）

程对共享变量的所有操作都必须在自己的工作内存，不能直接从相互内存中读写也不能从主内存中操作

原子性

原子性是指一个操作是不可间断，即多个线程开始执行的时候，一旦操作开始执行，就不会被其他线程干扰.

cas算法

在cas算法中包含三个参数 V，E，N 。 V表示要更新的变量，E表示预期值，N表示新值。仅当V等于E时，才会将V设置为N，如果V不等于E，说明其他线程做了更新，则当前线程什么都不做。最后cas返回当前V的真实值。CAS操作是抱着乐观的态度进行的，它总是认为自己的操作可以成功。当多个线程同时操作一个变量时，只会有一个会操作成功，其他的都会失败。 而原子操作则是在cas的基础上，写入一个while（true) 在循环中，操作成功的线程就会跳出，而失败的线程在循环里面继续执行，直到成功操作为止

可见性

可见性是指当一个线程修改了某一个共享变量的值，其他线程是否能够立即知道这个修改

有序性

有序性是指多个线程的操作在并发时，执行先后问题

有关并行的2个重要定律

AmdahI定律（阿姆达尔定律）

Gustafson定律（古斯塔夫森）