在并发编程中，多个线程之间采取什么机制进行通信,什么机制进行数据的同步？

在java语言中，采用的时**共享内存模型**来实现多线程之间的信息交换和数据同步。

线程之间通过共享程序公共的状态，通过读—写内存中公共状态的方式来进行隐式的通信。同步指的是程序在控制多个线程之间执行程序的相对顺序的机制。

Java在运行数据时候数据区分为下面几个内存区域：

1. Pc寄存器/程序计数器

每个线程有一个独立的程序计数器，为了线程切换后能恢复到正确的执行位置，是线程安全的。

1. Java stack

Java栈总是与线程关联在一起的，每创建一个线程，jvm就会为该线程创建对应的java栈，在这个java栈中又包含多个栈帧，这些栈帧都是与每个方法关联起来的，执行一个方法就类似于入栈和出栈的过程。

1. 堆

堆是java线程共享的，不是线程安全的，在jvm启动的时候创建。堆是存储java对象的地方。

1. 方法区

方法区存放了要加载的类的信息，类中的静态变量，类中final变量，类中方法信息。被线程共享

1. 常量池

常量池本身是方法区中的一个数据结构。其中存储了字符串，final变量值，类名和方法名常量。常量池在编译期间就确定，并保存在已经编译的.class文件中。

1. 本地方法栈

**Jmm规定了所有的变量都存储在主内存中，每个线程还有自己的工作内存，线程的工作内存中保存了该线程使用到的变量的主内存的副本拷贝，线程对变量的所有操作都是必须在工作内存中进行。**

Java内存模型三个特征：原子性，可见性，有序性。

1. 原子性：一个操作不能被打断，要么全部执行，要么全部不执行。
2. 可见性一个线程对共享变量做了修改之后，其他的线程立即能够看到该变量的这种变化。除了volatile关键字能实现可见性之外，还有synchronized,Lock，final也是可以的。
3. 对于一个线程的代码而言，我们总是以为代码的执行是从前往后的，依次执行的。这么说不能说完全不对，在单线程程序里，确实会这样执行；但是在多线程并发时，程序的执行就有可能出现乱序。

Java提供了两个关键字volatile和synchronized来保证多线程之间操作的有序性,volatile关键字本身通过加入内存屏障来禁止指令的重排序，而synchronized关键字通过一个变量在同一时间只允许有一个线程对其进行加锁的规则来实现，

线程对变量的所有操作都必须在工作内存中进行，而不能直接读写主内存中的变量。JMM从工作内存到主内存实现主要由以下八种操作来完成。

**Lock**（锁定）：作用于主内存变量，把**一个变量标识为一条线程独占的状态**。

**unlock**（解锁）：作用于主内存变量，把一个处于锁定状态的**变量释放**出来，释放后的变量才可以被其他线程锁定。

**read**（读取）：作用于主内存变量，把一个变量值从主内存传输到线程的工作内存中，以便随后的load动作使用

**load**（载入）：作用于工作内存的变量，它把read操作从主内存中得到的变量值放入**工作内存的变量副本**中。

**use**（使用）：作用于工作内存的变量，把工作内存中的一个变量值传递给执行引擎，每当虚拟机遇到一个需要使用变量的值的字节码指令时将会执行这个操作。

**assign**（赋值）：作用于工作内存的变量，它把一个从执行引擎接收到的值赋值给工作内存的变量，每当虚拟机遇到一个给变量赋值的字节码指令时执行这个操作。

**store**（存储）：作用于工作内存的变量，把工作内存中的一个变量的值传送到主内存中，以便随后的write的操作。

**write**（写入）：作用于主内存的变量，它把store操作从工作内存中一个变量的值传送到主内存的变量中。

Java happen-before原则：

一个操作”时间上的先发生“不代表这个操作会是”先行发生“

1. 程序次序规则：一个线程内，按照代码的顺序，书写正在前面的操作先于书写在后面的操作。

b.管程锁定规则(Monitor Lock Rule)：一个unlock操作先行发生于后面对同一个锁的lock操作。这里必须强调的是同一个锁，而”后面“是指时间上的先后顺序。

　　c.volatile变量规则(Volatile Variable Rule)：对一个volatile变量的写操作先行发生于后面对这个变量的读取操作，这里的”后面“同样指时间上的先后顺序。

　　d.线程启动规则(Thread Start Rule)：Thread对象的start()方法先行发生于此线程的每一个动作。

　　e.线程终于规则(Thread Termination Rule)：线程中的所有操作都先行发生于对此线程的终止检测，我们可以通过Thread.join()方法结束，Thread.isAlive()的返回值等作段检测到线程已经终止执行。

　f.线程中断规则(Thread Interruption Rule)：对线程interrupt()方法的调用先行发生于被中断线程的代码检测到中断事件的发生，可以通过Thread.interrupted()方法检测是否有中断发生。

　g.对象终结规则(Finalizer Rule)：一个对象初始化完成(构造方法执行完成)先行发生于它的finalize()方法的开始。

　　g.传递性(Transitivity)：如果操作A先行发生于操作B，操作B先行发生于操作C，那就可以得出操作A先行发生于操作C的结论。