1. Java内存模型问题
2. 线程通信

并发编程中，有两个关键问题，一个是线程之间如何通信，一个是线程之间如何同步。通信是指线程之间以什么机制来交换信息，在命令式编程中，线程之间通过共享内存和消息传递来进行通信。在共享内存的并发模型中，线程之间共享程序的公共状态，通过写，读内存中的公共区域进行隐式通信，在消息传递的并发模型中，线程通过发送消息来显式地通信。

同步指的是程序中用于控制不同线程间操作发生相对顺序的机制，在共享内存并发模型中，同步是显式进行的，程序员需要指定某段代码需要在线程之间互斥执行。在消息传递的并发模型中，由于消息的发送必须在消息的接受之前，因此同步是隐式进行的。

Java的并发采用的是共享内存模型，线程间的通信是隐式进行的，通信过程对于程序员来说是透明的，因此开发者需要深入了解线程通信过程。

1. Java内存模型结构

在Java中，所有实例域，静态域和数组元素都存储在堆内存中，堆在线程之间是共享的。局部变量，方法定义参数和异常处理器参数不会在线程间共享，不会有内存可见性问题。Java线程间的通信由Java内存模型（JMM）控制，线程之间的共享变量存储在主内存中，每个线程都有一个私有的工作内存，该工作内存中存储了该线程已读/写共享变量的副本，工作内存并不存在，只是一个抽象概念。线程A通过将工作内存中的共享变量刷新到主内存中，线程B读取该变量，这就是线程A与B之间的通信。

1. 重排序问题

执行程序时，为了提高性能，编译器和处理器都会对指令进行重排序。编译器优化的重排序，会重新安排语句的执行顺序；对于机器指令，如果不存在数据依赖性问题，处理器可以改变语句对应机器指令的执行顺序。

这些重排序可能会导致多线程程序出现内存可见性问题，对于编译器，JMM的编译器重排序规则会禁止特定类型的编译器重排序，对于处理器重排序，JMM的处理器重排序规则会要求编译器在生成指令序列时，插入特定类型的内存屏障指令，来禁止特定类型的处理器重排序。

1. 缓冲区与内存屏障

CPU 使用写缓冲区来临时保存向内存写入的数据，保证指令流水线持续运行，避免由于处理器停顿下来等待向内存写入数据而产生的延迟。但是每个CPU上的写缓冲区仅仅对该缓冲区可见，会对内存操作的执行顺序产生影响，有的指令可能先执行，但是出于缓冲区中，下一个指令可能后执行，但是直接写入内存了，这样就会导致执行顺序问题。

为了保证内存可见性，Java编译器在生成指令序列的适当位置会插入内存屏障指令来禁止特定类型的处理器重排序。内存屏障指令分为4类：

1. LoadLoad

Load1; LoadLoad; Load2

确保LOAD1的数据的从内存加载指令先于Load2以及所有后续加载，可以将命令拆为两部分，一个Load对应前面的加载指令，另一个Load对应后面的加载指令。

理解：某些主内存中的数据需要先于其他的数据加载到线程的工作内存中进行处理。

1. StoreStore

Store1; StoreStore; Store2

同理，确保Store2的存储到内存指令先于Store2以及后续的存储到内存指令。

理解： 某些线程中的数据需要先于其他的线程中的数据存储到主内存中。

1. LoadStore

Load1; LoadStore; Store2;

确保Load1数据加载先于Store2以及后续的存储到内存指令。

理解： 某些主内存中的数据的处理需要先于其他线程中的数据刷新到主内存中。

1. StoreLoad

Store1; StoreLoad; Load2

确保Store1数据存储到内存指令先于Load2以及后续的从内存加载指令；

理解：某些线程中的数据的刷新到主内存需要先于一些主内存中数据的处理。

StoreLoad是一个全能型的屏障，它同时具有其他三个屏障的功能，因为该屏障会把写缓冲区中的数据全部刷新到内存中，所有性能消耗很大。

1. happens-before

JDK5开始的JSR-133内存模型，使用happens-before来描述操作之间的内存可见性，如果一个操作执行的结果需要对另一个操作可见，那么这两个操作之间必须要存在happens-before关系。

1. 重排序
2. 概述

重排序是指编译器和处理器为了优化程序性能而对指令序列进行重新排序的一种手段。

1. 数据依赖性

如果两个操作访问同一个变量，且这两个操作中有一个为写操作，那么这两个操作之间就具有数据依赖性。数据依赖性有三种类型，一个是写后读，一个是写后写，一个是读后写，这三种情况下只要重排序了两个操作的执行顺序，执行结果就会被改变。

由于编译器和处理器会对操作做重排序，但是会遵守数据依赖性，不会改变存在数据依赖性的两个操作的执行顺序。这里说的数据依赖性指的是对于的单个处理器中执行的指令序列和单个线程中的执行操作。

1. 顺序一致性
2. 概述

顺序一致性内存模型是一个理论参考模型，CPU的内存模型和编程语言的内存模型都会以顺序一致性模型为参考。

1. 数据竞争

当程序未正确同步时，就可能存在数据竞争，如果程序是正确同步的，那么程序的执行将具有顺序一致性，即程序的执行结果将于在顺序一致性内存模型中结果相同。

1. 顺序一致性内存模型

顺序一致性模型为程序员提供了内存可见性保证，该模型有两大特性：

1. 一个线程中的所有操作必须按照程序的顺序来执行；
2. 所有线程只能看到一个单一的操作执行顺序，每个操作必须原子执行，并且立即对所有线程可见；

而在JMM中，若当前线程没有将写过的数据刷新到主内存中，那么其他线程就无法得知最新的数据情况，那么每个线程看到的操作执行顺序都是不一致的。

1. 64位变量操作

计算机中，数据通过总线在CPU和内存之间传递，每次CPU和内存之间的数据传递都是通过一系列步骤完成的，称为总线事务，包括读事务（读入CPU）和写事务（写入内存），总线会同步试图并发使用总线的事务，在一个CPU执行总线事务期间，其他的CPU和IO设置是被禁止执行内存的读写操作的，确保了内存读写的原子性。

在32位的CPU上，如果要求对64位的数据具有读写原子性，会有很大开销，可能会把64位数据的写操作拆分为两个32位的写操作来执行，分配到不同的总线事务中，此时将不具有原子性。从JDK5开始，任意的读操作都是原子性的，但是64位的变量的写操作依然要拆分为两个32位。