系统底层如何实现数据一致性

1.MESI如果能解决，就使用MESI

2.如果不能，就锁总线

系统地城如何保证有序性

1.内存屏障 sfence mfence Ifence等系统原语

2.锁总线

指令乱序执行

volatile如何解决指令重排序

设计模式：singleTon

DCL:double check lock

类对象在内存中只有一个

应用：

第二段代码，不能保证多个线程下只创建一个对象

解决：双重检查单例

为什么要加volatile?

new对象有哪些操作？

0 new #2<T> 成员变量m为默认值，0

3 dup

4 invokespecial #3 <T.<init>> 调用构造方法

7 astore_1

new对象的时候有中间态，半初始化状态

加问：DCL单例到底需不需要volatile?

0 new #2<T> 成员变量m为默认值，0

4 invokespecial #3 <T.<init>> 调用构造方法

7 astore_1

new的时候，判断t是否为空，设置为成员变量为默认值

这时，发生了指令重排序

t指向了半初始化的m

执行到构造方法，m不为空，使用了半初始化状态

订单

==========

JVM的内存屏障

屏障两边的指令不能重排，保证有序性！

sFence mFence lFence

字节码：private static volatile

会加标记：ACC_VOLATILE

JVM规范：

JSR:java规范 java规范委员会

JSR内存屏障

LoadLoad屏障

StoreStore屏障

LoadStore屏障

StoreLoad屏障

答案：

1.java代码：volatile

2.字节码：ACC_VOLATILE

3.JVM层面：内存屏障

JVM规范要求，屏障两边的指令不能重排，保证有序

StoreStoreBarrier

volatile

4.hotspot实现

bytecodeinterpreter.cpp c++代码

orderAccess::fence

os::is_mp　如果是多个cpu，
lock:.....锁总线

虽然有sFence mFence lFence等系统原语，但是有的系统不支持

AQS

ThreadLock

强软弱虚：java中的引用类型

代码1：强引用
NormalReference
M m =new M();

面试题：GC调优，OOM怎么排查

代码2：软引用
内存不够用的时候，会释放掉
应用：缓存上，兼顾了时间和空间
特别大的图片，放到内存中；如果强引用，释放掉后，还需要重新加载

SoftReference

m为强引用
new byte[1024*1024*10]为软引用

堆内存最大为20M

强、软都是GC来帮助实现的

代码3：弱

特点：垃圾回收器看到就会直接回收，一次性

WeakReference

ThreadLocal线程本地变量

线程独有，线程存在，ThreadLocal就存在

代码4：虚引用

面试题：虚引用的作用是什么？管理堆外内存

PhantomReference

特点：怎么get都get不到

作用：管理堆外内存

NIO

读：网卡----操作系统内存---JVM

写出：JVM--操作系统内存---

zeroCopy

DirectByteBuffer

JVM，专门的GC线程，专门监听堆外内存

虚引用是由JVM创建

BIO、NIO

JDK老的IO是BIO

新的IO是NIO