内存屏障是多线程编程的必要装备。它们形式多样，某些是显式的，某些是隐式的。某些是双向的，某些是单向的。JVM利用这些形式在所有平台中有效地支持 Java内存模型。

**内存屏障，又称内存栅栏，是一组处理器指令，用于实现对内存操作的顺序限制。**本文介绍了内存屏障对多线程程序的影响。（在使用synchronzied或者volatile等时，就会产生内存屏障，保证可见性等。）

### 内存屏障为何重要？

对主存的一次访问一般花费硬件的数百次时钟周期。处理器通过缓存（caching）能够从数量级上降低内存延迟的成本这些缓存为了性能重新排列待定内存操 作的顺序。也就是说，程序的读写操作不一定会按照它要求处理器的顺序执行。当数据是不可变的，同时/或者数据限制在线程范围内，这些优化是无害的。如果把 这些优化与对称多处理（symmetric multi-processing）和共享可变状态（shared mutable state）结合，那么就是一场噩梦。当基于共享可变状态的内存操作被重新排序时，程序可能行为不定。一个线程写入的数据可能被其他线程可见，原因是数据 写入的顺序不一致。**适当的放置内存屏障通过强制处理器顺序执行待定的内存操作来避免这个问题。**

**内存屏障**

      内存屏障（Memory Barrier，或有时叫做内存栅栏，Memory Fence）是一种CPU指令，用于控制特定条件下的重排序和内存可见性问题。Java编译器也会根据内存屏障的规则禁止重排序。  
      **内存屏障可以被分为以下几种类型**  
**LoadLoad屏障：**对于这样的语句Load1; LoadLoad; Load2，在Load2及后续读取操作要读取的数据被访问前，保证Load1要读取的数据被读取完毕。  
**StoreStore屏障：**对于这样的语句Store1; StoreStore; Store2，在Store2及后续写入操作执行前，保证Store1的写入操作对其它处理器可见。  
**LoadStore屏障：**对于这样的语句Load1; LoadStore; Store2，在Store2及后续写入操作被刷出前，保证Load1要读取的数据被读取完毕。  
**StoreLoad屏障：**对于这样的语句Store1; StoreLoad; Load2，在Load2及后续所有读取操作执行前，保证Store1的写入对所有处理器可见。它的开销是四种屏障中最大的。        在大多数处理器的实现中，这个屏障是个万能屏障，兼具其它三种内存屏障的功能。

       有的处理器的重排序规则较严，无需内存屏障也能很好的工作，Java编译器会在这种情况下不放置内存屏障。



     为了保证final字段的特殊语义，也会在下面的语句加入内存屏障。  
     x.finalField = v; StoreStore; sharedRef = x;