volatile的用途

1.线程可见性

```
package com.mashibing.testvolatile;

public class T01_ThreadVisibility {
    private static volatile boolean flag = true;

    public static void main(String[] args) throws InterruptedException {
        new Thread(()-> {
            while (flag) {
                //do sth
            }
            System.out.println("end");
        }, "server").start();


        Thread.sleep(1000);

        flag = false;
    }
}
```

2.防止指令重排序

问题：DCL单例需不需要加volatile?

CPU的基础知识

- 缓存行对齐
  缓存行64个字节是CPU同步的基本单位，缓存行隔离会比伪共享效率要高
  Disruptor

```
package com.mashibing.juc.c_028_FalseSharing;

public class T02_CacheLinePadding {
    private static class Padding {
        public volatile long p1, p2, p3, p4, p5, p6, p7; //
    }

    private static class T extends Padding {
        public volatile long x = 0L;
    }

    public static T[] arr = new T[2];

    static {
        arr[0] = new T();
        arr[1] = new T();
    }
```

```
18
19      public static void main(String[] args) throws Exception {
20          Thread t1 = new Thread(()->{
21              for (long i = 0; i < 1000_0000L; i++) {
22                  arr[0].x = i;
23              }
24          });
25
26          Thread t2 = new Thread(()->{
27              for (long i = 0; i < 1000_0000L; i++) {
28                  arr[1].x = i;
29              }
30          });
31
32          final long start = System.nanoTime();
33          t1.start();
34          t2.start();
35          t1.join();
36          t2.join();
37          System.out.println((System.nanoTime() - start)/100_0000);
38      }
39  }
40
```

MESI

- 伪共享
- 合并写
  CPU内部的4个字节的Buffer

```
1   package com.mashibing.juc.c_029_WriteCombining;
2
3   public final class WriteCombining {
4
5       private static final int ITERATIONS = Integer.MAX_VALUE;
6       private static final int ITEMS = 1 << 24;
7       private static final int MASK = ITEMS - 1;
8
9       private static final byte[] arrayA = new byte[ITEMS];
10      private static final byte[] arrayB = new byte[ITEMS];
11      private static final byte[] arrayC = new byte[ITEMS];
12      private static final byte[] arrayD = new byte[ITEMS];
13      private static final byte[] arrayE = new byte[ITEMS];
14      private static final byte[] arrayF = new byte[ITEMS];
15
16      public static void main(final String[] args) {
17
18          for (int i = 1; i <= 3; i++) {
19              System.out.println(i + " SingleLoop duration (ns) = " +
    runCaseOne());
20              System.out.println(i + " SplitLoop  duration (ns) = " +
    runCaseTwo());
21          }
22      }
23
24      public static long runCaseOne() {
25          long start = System.nanoTime();
```

```
        int i = ITERATIONS;

        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
        }
        return System.nanoTime() - start;
    }

    public static long runCaseTwo() {
        long start = System.nanoTime();
        int i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayA[slot] = b;
            arrayB[slot] = b;
            arrayC[slot] = b;
        }
        i = ITERATIONS;
        while (--i != 0) {
            int slot = i & MASK;
            byte b = (byte) i;
            arrayD[slot] = b;
            arrayE[slot] = b;
            arrayF[slot] = b;
        }
        return System.nanoTime() - start;
    }
}

```

- 指令重排序

```
package com.mashibing.jvm.c3_jmm;

public class T04_Disorder {
    private static int x = 0, y = 0;
    private static int a = 0, b =0;

    public static void main(String[] args) throws InterruptedException
{
        int i = 0;
        for(;;) {
            i++;
            x = 0; y = 0;
            a = 0; b = 0;
            Thread one = new Thread(new Runnable() {
                public void run() {

```

```
                    //由于线程one先启动，下面这句话让它等一等线程two. 读着可根据自己电脑的实际性能适当调整等待时间.
                    //shortWait(100000);
                    a = 1;
                    x = b;
                }
            });

            Thread other = new Thread(new Runnable() {
                public void run() {
                    b = 1;
                    y = a;
                }
            });
            one.start();other.start();
            one.join();other.join();
            String result = "第" + i + "次 (" + x + "," + y + "）";
            if(x == 0 && y == 0) {
                System.err.println(result);
                break;
            } else {
                //System.out.println(result);
            }
        }
    }


    public static void shortWait(long interval){
        long start = System.nanoTime();
        long end;
        do{
            end = System.nanoTime();
        }while(start + interval >= end);
    }
}
```

volatile如何解决指令重排序

1: volatile i

2: ACC_VOLATILE

3: JVM的内存屏障

4：hotspot实现

bytecodeinterpreter.cpp

```
int field_offset = cache->f2_as_index();
          if (cache->is_volatile()) {
            if (support_IRIW_for_not_multiple_copy_atomic_cpu) {
              OrderAccess::fence();
            }
```

orderaccess_linux_x86.inline.hpp

```
inline void OrderAccess::fence() {
  if (os::is_MP()) {
    // always use locked addl since mfence is sometimes expensive
#ifdef AMD64
    __asm__ volatile ("lock; addl $0,0(%%rsp)" : : : "cc", "memory");
#else
    __asm__ volatile ("lock; addl $0,0(%%esp)" : : : "cc", "memory");
#endif
  }
}
```