

Java内存模型

有序性

有序性

- 程序按照代码的先后顺序执行
- 编译器和处理器都可能对指令重新排序
- 重排不会影响单线程的运行结果

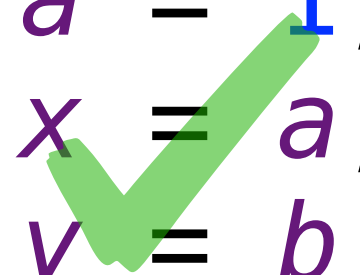
有序性

```
int a;  
int b;  
int x;  
int y;
```

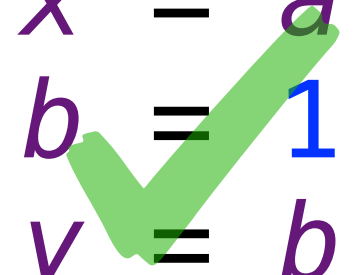
```
1.  a = 1;  
2.  b = 1;  
3.  x = a;  
4.  y = b;
```

as if serial

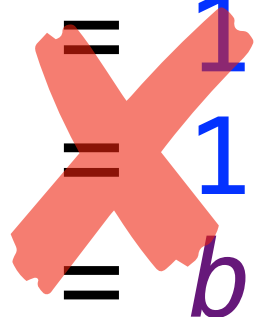
```
2.  b = 1;  
1.  a = 1;  
3.  x = a;  
4.  y = b;
```



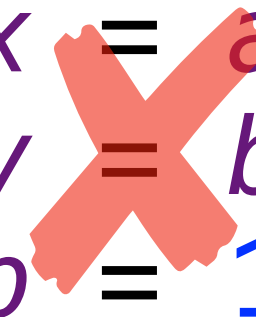
```
1.  a = 1;  
3.  x = a;  
2.  b = 1;  
4.  y = b;
```



```
3.  x = a;  
1.  a = 1;  
2.  b = 1;  
4.  y = b;
```



```
1.  a = 1;  
3.  x = a;  
4.  y = b;  
2.  b = 1;
```



```
private int a = 2;  
private boolean flag = false;
```

```
public void method1() {  
    a = 1;  
    flag = true;  
}
```

线程A

```
public void method2() {  
    if (flag) {  
        // 由于指令重排, 此时可能 a == 2  
    }  
}
```

线程B

happens-before的8条原则

- 程序次序规则：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作；
- 锁定规则：一个 `unlock` 操作先行发生于后面对同一个锁的 `lock` 操作；
- `volatile` 变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作；
- 传递规则：如果操作A先行发生于操作B，而操作B又先行发生于操作C，则可以得出操作A先行发生于操作C；
- 线程启动规则：Thread对象的`start()`方法先行发生于此线程的每一个动作；
- 线程中断规则：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；
- 线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过`Thread.join()`方法结束、`Thread.isAlive()`的返回值手段检测到线程已经终止执行；
- 对象终结规则：一个对象的初始化完成先行发生于他的 `finalize()` 方法的开始

volatile 和有序性

- 当程序执行到volatile变量的读操作或者写操作时，在其前面的操作的更改肯定全部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行；
- Java 在进行指令优化时，不能将在对volatile变量访问的语句放在其后面执行，也不能把volatile变量后面的语句放到其前面执行。

```
int a;  
int b;  
volatile int x;  
int y;
```

1.	<i>a</i>	=	1;
2.	<i>b</i>	=	1;
3.	<i>x</i>	=	<i>a</i> ;
4.	<i>y</i>	=	<i>b</i> ;
5.	<i>a</i>	=	2;

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

    public static void main(String[] args) throws Exception {
        long i = 0;
        while (true) {
            i++;
            x = 0; y = 0; a = 0; b = 0;
            Thread t1 = new Thread(() -> {
                a = 1;
                x = b;
            });
            Thread t2 = new Thread(() -> {
                b = 1;
                y = a;
            });
            t1.start(); t2.start();
            t1.join(); t2.join();
            if (x == 0 && y == 0) {
                System.out.println("在第" + i + "次时遇到了(0,0)");
                break;
            }
        }
    }
}
```

1	2	3	4	x, y
T1 a = 1	T1 x = b	T2 b = 1	T2 y = a	0, 1
T1 a = 1	T2 b = 1	T2 y = a	T1 x = b	1, 1
T1 a = 1	T2 b = 1	T1 x = b	T2 y = a	1, 1
T2 b = 1	T2 y = a	T1 a = 1	T1 x = b	1, 0
T2 b = 1	T1 a = 1	T1 x = b	T2 y = a	1, 1
T2 b = 1	T1 a = 1	T2 y = a	T1 x = b	1, 1

有序性

- 程序按照代码的先后顺序执行
- 编译器和处理器都可能对指令重新排序
- 重排不会影响单线程的运行结果
- 多线程可能会受到影响
- `as-if-serials` `happens-before`

