Java内存模型

原子性

```java
public class SelfIncreasement {
    private long counter = 0;

    public void add() {
        counter++;
    }

    public static void main(String[] argvs) throws Exception {
        SelfIncreasement si = new SelfIncreasement();
        Thread[] threads = new Thread[10000];
        for(int i=0; i<threads.length; i++){
            threads[i] = new Thread(() -> {
                for(int j=0; j<10000; j++){
                    si.add();
                }
            }, "T-" + i);
            threads[i].start();
        }
        for(int i=0; i<threads.length; i++){
            threads[i].join();
        }
        //期望是 10000 （1万个线程） * 10000 （每个线程自加1万次） = 100000000 （1亿）
        System.out.println(si.counter);
    }
}
```

原子性

一个操作是不可中断的。即使是在多个线程一起执行的时候，一个操作一旦开始，就不会被其它线程干扰

原子性

- 基本类型变量(long和double除外）的读取和赋值操作具有原子性
- long, double大部分JVM（尤其是64位的JVM）也实现了他们的原子性，但不是JMM要求必须的
- 自加操作有读、加、赋值三步，不具有原子性
- 如果程序要实现原子性，需要靠synchronized / Lock等来实现

```
int x;
int y;
Xyz xyz;
```

```
1.  x = 1;  ✓
2.  y ++;  ✗
3.  x = y;  ✗
```

```
4.  x = x + 1;  ✗
5.  x = 2 * x;  ✗
6.  xyz = new Xyz();  ✗
```

Java并发编程

- 原子性
  - 一个操作是不可中断的。即使是在多个线程一起执行的时候，一个操作一旦开始，就不会被其它线程干扰
- 可见性
  - 当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值
- 有序性
  - 程序执行的顺序按照代码的先后顺序执行

	volatile	sychnorized	lock
原子性	✘	✔	✔
可见性	✔	✔	✔
有序性	✔	✔	✔

代码复杂度上升

CAS

- Compare And Swap 比较并交换
- sun.misc.Unsafe 提供 compareAndSwap系列方法
-

java.util.concurrent.atomic

- AtomicBoolean
- AtomicInteger
- AtomicInterArray
- AtomicIntegerFieldUpdate<T>
- AtomicLong
- AtomicLongArray
- AtomicLongFieldUpdater<T>
- AtomicMarkableReference<V>
- AtomicReference<V>
- AtomicReferenceArray<E>
- AtomicReferenceFieldUpdater<T,V>
- AtomicStampedReference<V>
- DoubleAccumulator
- DoubleAdder
- LongAccumulator
- LongAdder

```java
public class LongAtomTest {
    private static long value = 0;

    public void test(final long v){
        for(long i=0; i<1000000000; i++) {
            value = v;
            long t = value;
            if (t != -1l && t != 1l) {
                System.out.println("错误的值: " + t);
                System.exit(0);
            }
        }
        System.out.println("all done");
    }
    public static void main(String[] argvs) throws Exception{
        LongAtomTest lat = new LongAtomTest();
        Thread t1 = new Thread(() -> {
            lat.test(-1l);
        }, "T1");
        Thread t2 = new Thread(() -> {
            lat.test(1l);
        }, "T2");

        t1.start();
        t2.start();
    }
}
```

long, double赋值不具有原子性

按照JSR 133规范，long和double的赋值不要求具有原子性

一般的64位JDK把这2个也做成了原子性操作

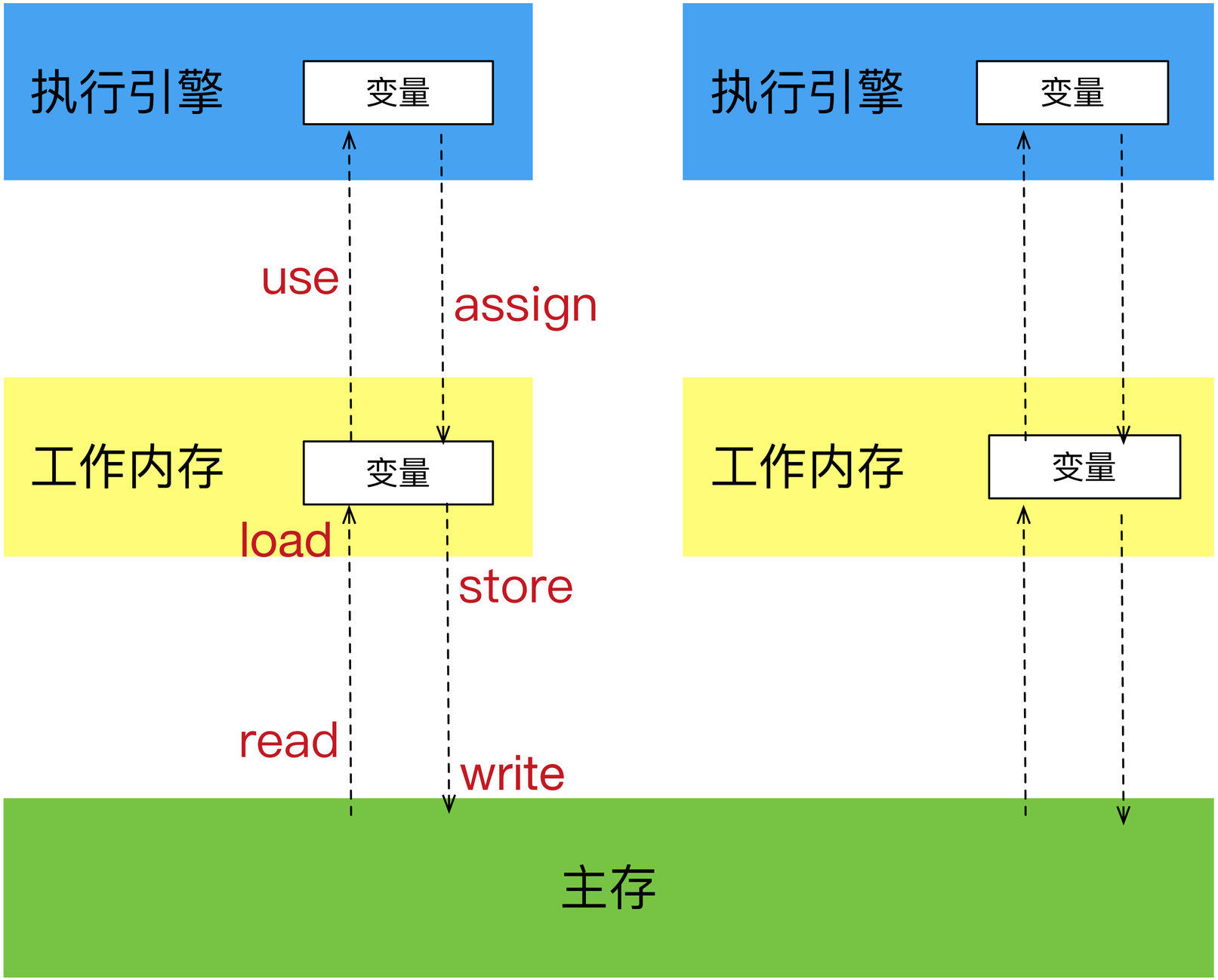
32位的JVM下运行，

有一定几率会执行到出错的状况

输出：错误的值：-4294967295

8个底层内存操作

lock	主存变量	锁定	use	工作内存变量	使用
unlock	主存变量	解锁	assign	工作内存变量	赋值
read	主存变量	读取	store	工作内存变量	存储
load	工作内存变量	载入	write	主存变量	写入


执行引擎
变量
执行引擎
变量
use
assign
工作内存
变量
工作内存
变量
load
store
read
write
主存