java,.(,)

.

CPU.CPUCPU.CPU...

.

(https://blog.csdn.net/qq_20610631/article/details/81534485)
. .

CAS

- IDHash
- CASJavaAtomicCAS
-
-

-
-
- lock.tryLocktimeout
-

JavaJava,JVM,JVM.CPU.,JavaJVMCPU.

volatile

..

,

volatile,Java.

volatile?

.volatile,Lock.2.

-
- CPU

L1L2volatileJVMLock

synchronized

synchronized,.1.6synchronized,.

Java,

- ..
- . class
- . Synchonized

https://blog.csdn.net/luoweifu/article/details/46613015

,,,.?

Java

synchronizedJava,

长　　度	内　　容	说　　明
32/64bit	Mark Word	存储对象的 hashCode 或锁信息等
32/64bit	Class Metadata Address	存储到对象类型数据的指针
32/32bit	Array length	数组的长度（如果当前对象是数组）

JavaMark WordHashCode32JVMMark Word

锁状态	25bit	4bit	1bit 是否是偏向锁	2bit 锁标志位
无锁状态	对象的 hashCode	对象分代年龄	0	01

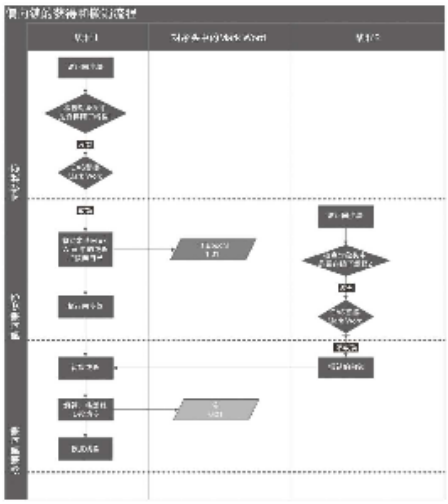
Mark WordMark Word4

锁状态	25bit		4bit	1bit	2bit
	23bit	2bit		是否是偏向锁	锁标志位
轻量级锁	指向栈中锁记录的指针				00
重量级锁	指向互斥量（重量级锁）的指针				10
GC 标记	空				11
偏向锁	线程 ID	Epoch	对象分代年龄	1	01

Java SE 1.6“”“”Java SE 1.64

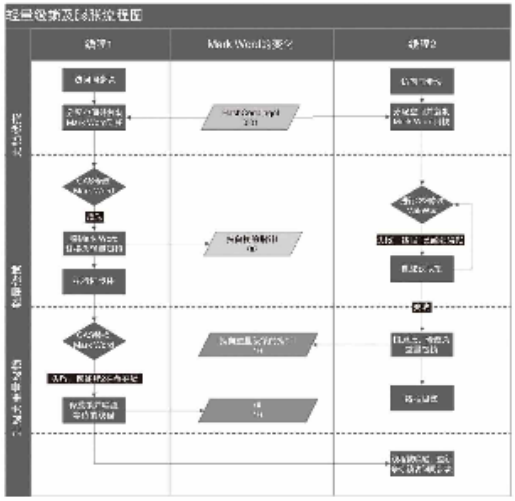
IDCASMark WordMark Word1CASCAS

Mark Word12



JVMMark WordDisplaced Mark WordCASMark Word

CASDisplaced Mark Word



CPU

atomic“”atomic operation“”

术语名称	英　　文	解　　释
缓存行	Cache line	缓存的最小操作单位
比较并交换	Compare and Swap	CAS 操作需要输入两个数值，一个旧值（期望操作前的值）和一个新值，在操作期间先比较旧值有没有发生变化，如果没有发生变化，才交换成新值，发生了变化则不交换
CPU 流水线	CPU pipeline	CPU 流水线的工作方式就像工业生产上的装配流水线，在 CPU 中由 5 ~ 6 个不同功能的电路单元组成一条指令处理流水线，然后将一条 X86 指令分成 5 ~ 6 步后再由这些电路单元分别执行，这样就能实现在一个 CPU 时钟周期完成一条指令，因此提高 CPU 的运算速度
内存顺序冲突	Memory order violation	内存顺序冲突一般是由假共享引起的，假共享是指多个 CPU 同时修改同一个缓存行的不同部分而引起其中一个 CPU 的操作无效，当出现这个内存顺序冲突时，CPU 必须清空流水线

1.
   i++i=1i++32.
   i1CPU1CPU2
   LOCK
2.
   CPU
   L1L2L3Pentium 6“”“”LockLOCK2-3CPU1iCPU2i

cache line
Intel 486Pentium

1. Java
   a. CAS
   b. CAS
      JavaCASLinkedTransferQueueXferCASCASABA
      1ABACASABACASABA1ABA1A2B3AJava 1.5JDKAtomicAtomicStampedReferenceABAcompareAndSet
      1ABACASABACASABA1ABA1A2B3AJava 1.5JDKAtomicAtomicStampedReferenceABAcompareAndSet
      public boolean compareAndSet(
      V　　expectedReference,　　//
      V　　newReference,　　//
      int　　expectedStamp,　　//
      int　　newStamp　　//
)
      2CASCPUJVMpausepausede-pipelineCPUMemory Order ViolationCPUCPU Pipeline FlushCPU
      3CASCASi2j=aij=2aCASijJava 1.5JDKAtomicReferenceCAS
   c.
      JVMJVMCASCASCAS

Java

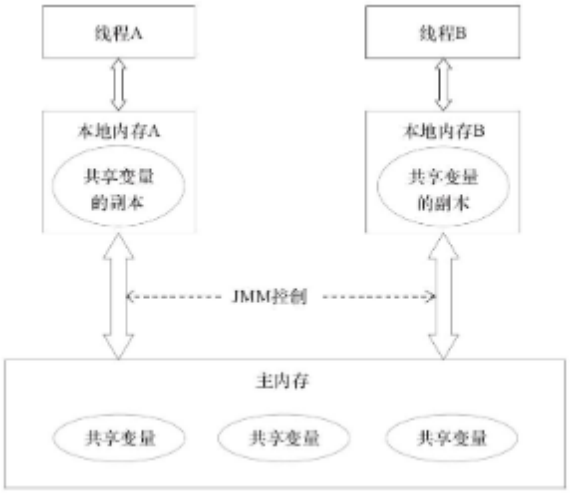
Java

-

JavaJavaJava

Java

Java
JavaJavaJMMJMMJMMMain MemoryLocal Memory/JMMJava

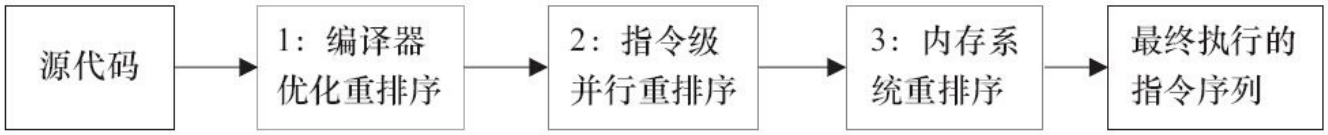


ABJMMJava

3

1.
2. Instruction-Level ParallelismILP
3. /

Java3



123JMMJMMJavaMemory BarriersIntelMemory Fence
JMM

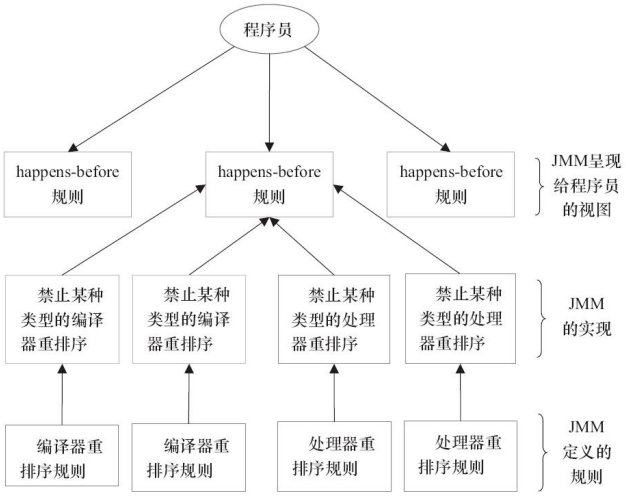
JavaJMM4

屏障类型	指令示例	说　　明
LoadLoad Barriers	Load1; LoadLoad; Load2	确保 Load1 数据的装载先于 Load2 及所有后续装载指令的装载
StoreStore Barriers	Store1; StoreStore; Store2	确保 Store1 数据对其他处理器可见（刷新到内存）先于 Store2 及所有后续存储指令的存储
LoadStore Barriers	Load1; LoadStore; Store2	确保 Load1 数据装载先于 Store2 及所有后续的存储指令刷新到内存
StoreLoad Barriers	Store1; StoreLoad; Load2	确保 Store1 数据对其他处理器变得可见（指刷新到内存）先于 Load2 及所有后续装载指令的装载。StoreLoad Barriers 会使该屏障之前的所有内存访问指令（存储和装载指令）完成之后，才执行该屏障之后的内存访问指令

StoreLoad Barriers“”3Buffer Fully Flush

happens-before

JDK 5JavaJSR-133JSR-133happens-beforeJMMhappens-before
happens-before
. happens-before
. happens-before
. volatilevolatilehappens-beforevolatile
. A happens-before BB happens-before CA happens-before C



3

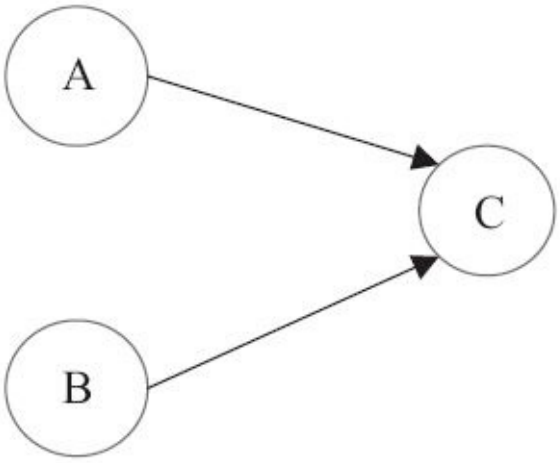
名　称	代码示例	说　　明
写后读	a = 1; b = a;	写一个变量之后，再读这个位置
写后写	a = 1; a = 2;	写一个变量之后，再写这个变量
读后写	a = b; b = 1;	读一个变量之后，再写这个变量

3

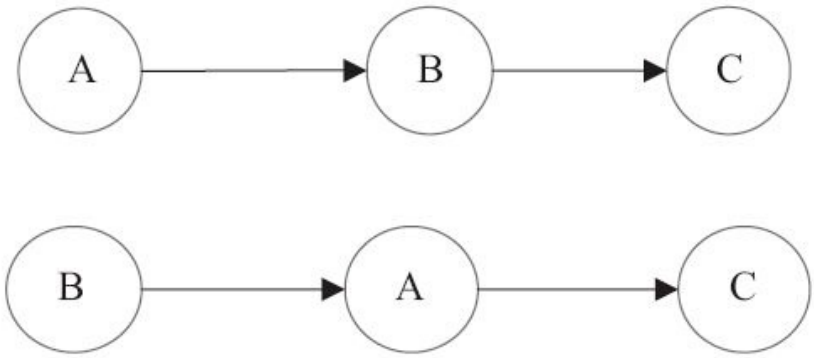
as-if-serial

as-if-serialruntimeas-if-serial

```
double pi  = 3.14;          // A
double r   = 1.0;           // B
double area = pi * r * r;   // C
```



ACBC,.CAB. .



按程序顺序的执行结果：
area = 3.14

重排序后的执行结果：
area = 3.14

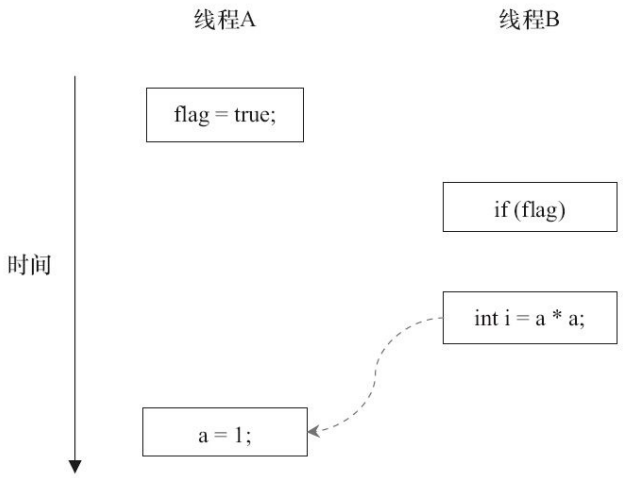
as-if-serialas-if-serialruntimeas-if-serial

happens-before A-B B-C so A-C

happens-beforeJMM

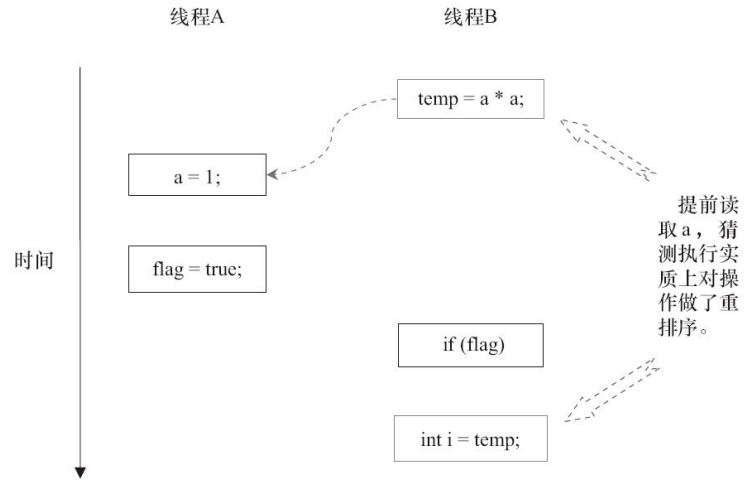
ReorderExample.
flagaABAwriter()Breader()B4A1a

123412



12AflagBBaaA

3434



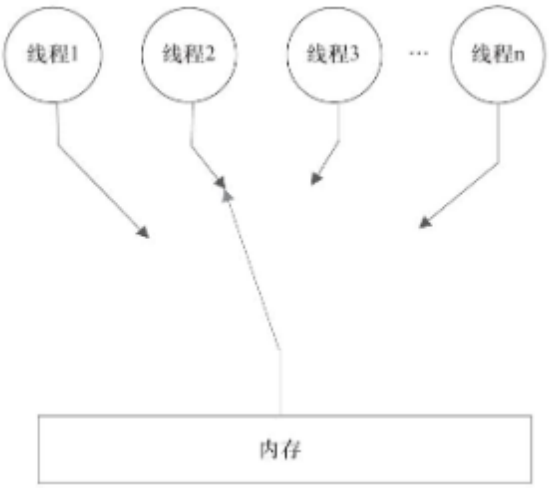
34SpeculationBa*aReorder BufferROB3i
34
as-if-serial

Java

JMM
Sequentially Consistent——synchronizedvolatilefinal

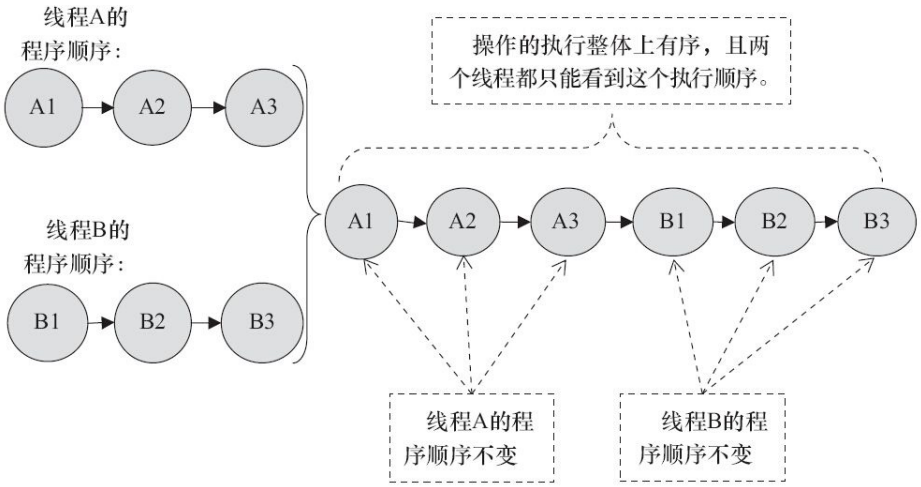
1

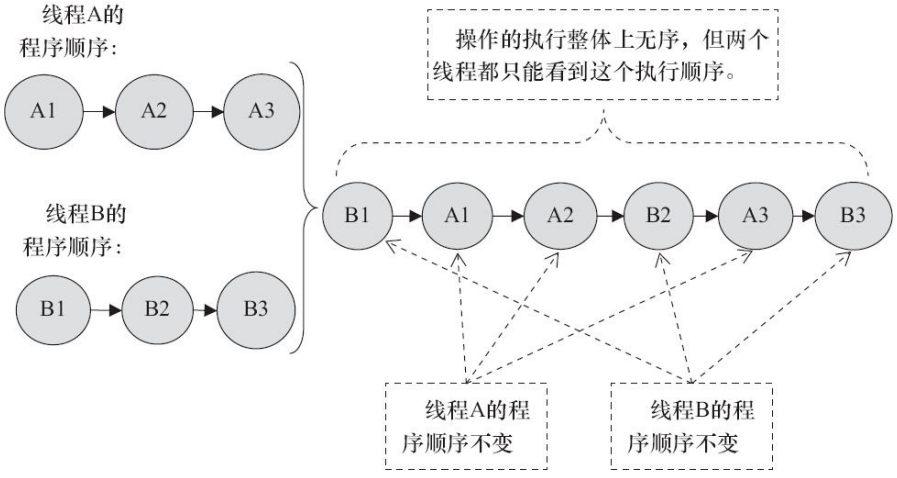
2



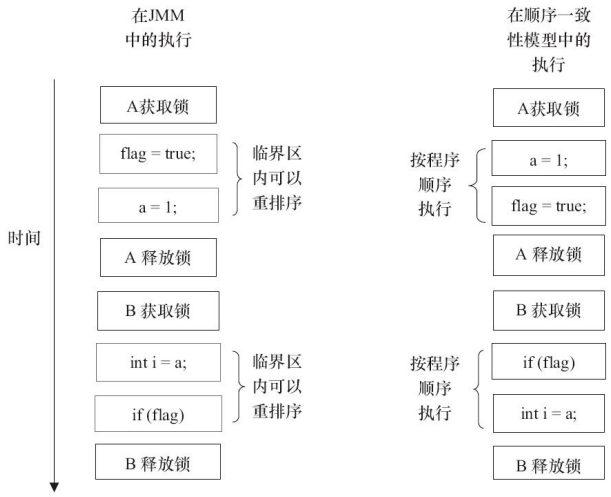
//

ABA3A1A2A3B3B1B2B3
A3B





JMMJMM



JMM0NullFalseJMMOut Of Thin AirJVMJVMPre-zeroed Memory

volatile

volatile

volatilevolatile//
VolatileExampleVolatileFeaturesExample

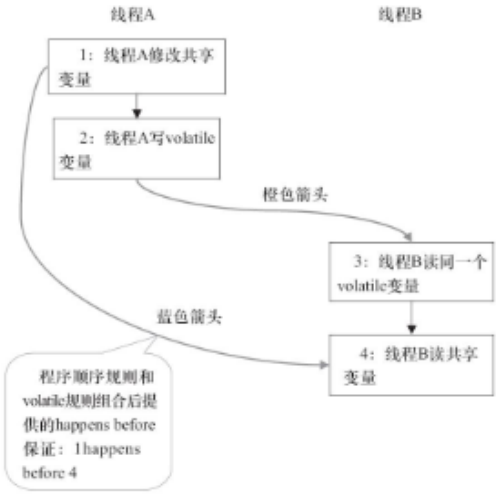
volatile
·volatilevolatile
·volatile/volatile++

volatile-happens-before

VolatileExampleHe
Awriter().Breader().happens-before.happens-before3:

1. 1 hb 2; 3 hb 4
2. volatile 2 hb 3
3. . 1 hb 4

4.



volatile-

volatileJMM
volatileJMM

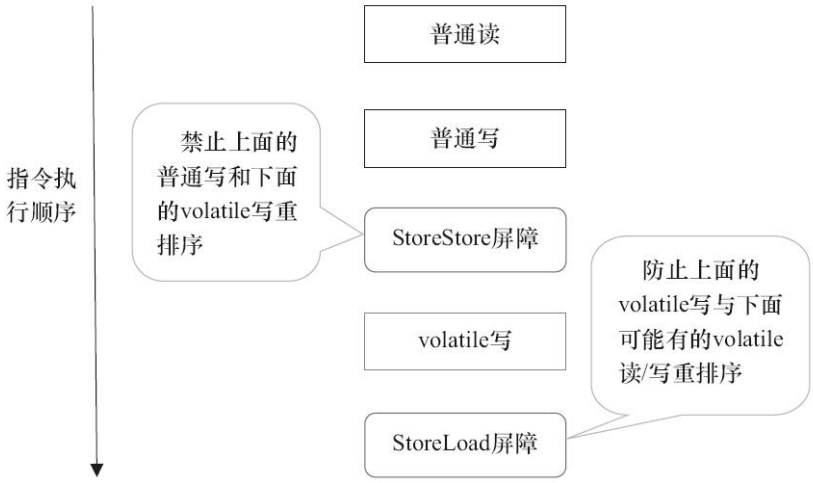
volatilevolatile
·AvolatileAvolatile
·BvolatileBvolatile
·AvolatileBvolatileAB

volatile

volatileJMMJMM

·volatileStoreStore
·volatileStoreLoad
·volatileLoadLoad
·volatileLoadStore

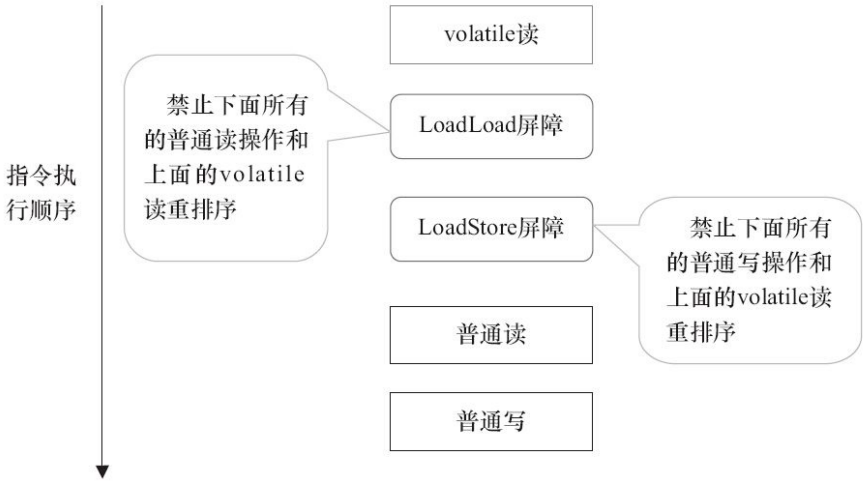
volatile



StoreStorevolatileStoreStorevolatile

volatileStoreLoadvolatilevolatile/volatileStoreLoadvolatilereturnvolatileJMMvolatilevolatileStoreLoadJMMvolatileStoreLoadvolatile-volatilevolatilevolatileStoreLoadJMM

volatile

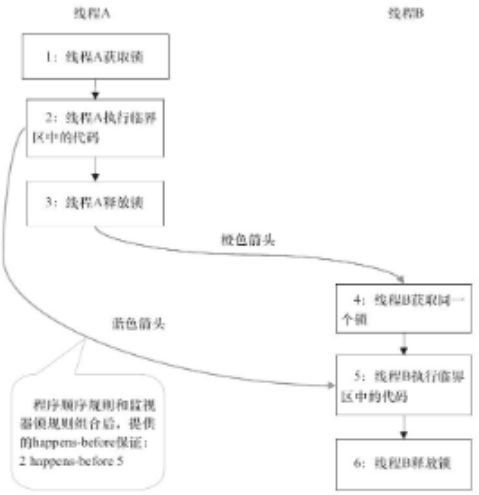


LoadLoadvolatileLoadStorevolatile

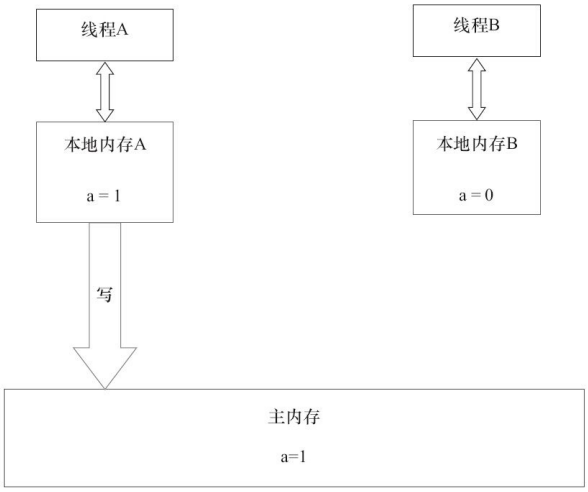
volatilevolatile/volatilevolatilevolatileBrian GoetzJavaVolatile

-happens-before

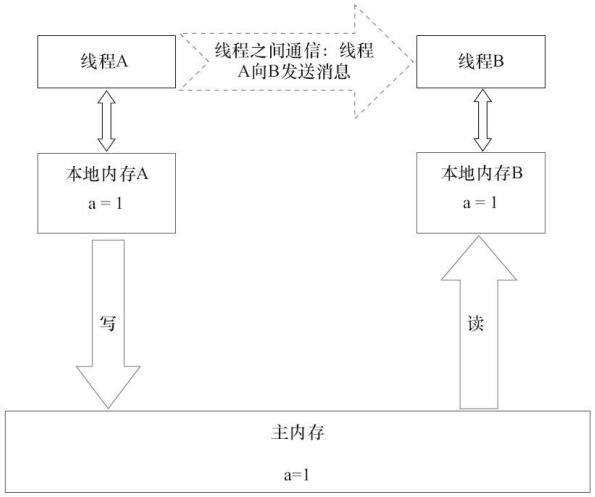
:MonitorExample
Awriter()Breader()happens-beforehappens-before3
11 happens-before 2,2 happens-before 3;4 happens-before 5,5 happens-before 6
23 happens-before 4
3happens-before2 happens-before 5
happens-before



JMMMonitorExampleA



JMM



-volatile-volatilevolatile

·AAA
·BB
·ABAB

final

volatilefinalfinal

final

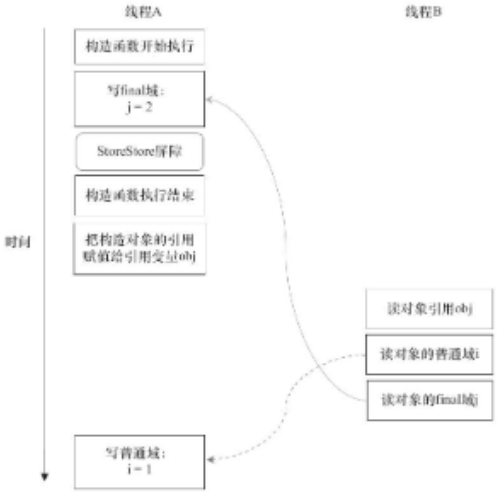
final
1final
2finalfinal
FinalExample

Awriter()Breader()

final

finalfinal2
1JMMfinal
2finalreturnStoreStorefinal

writer()writer()finalExample=new FinalExample()
1FinalExample
2obj
B

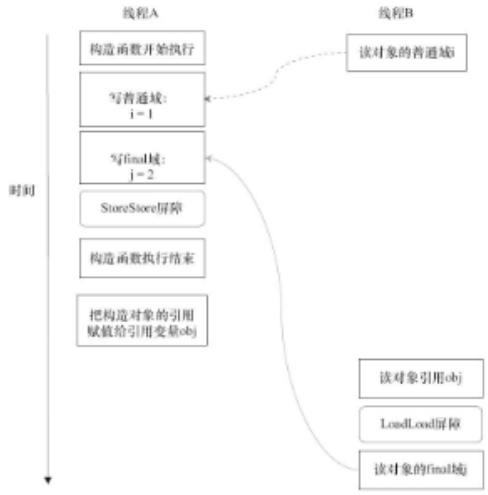


Bifinalfinal“”Bfinal

finalfinalB“”objobji1i

final

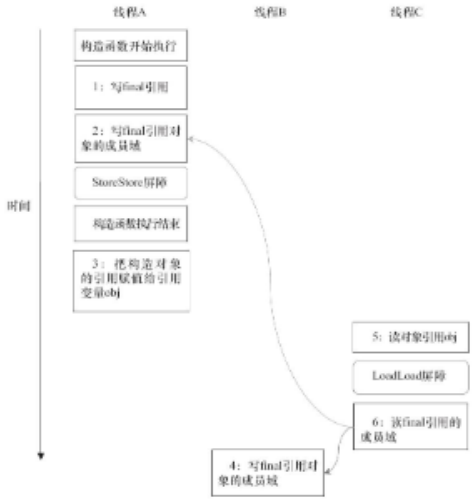
finalfinalJMMfinalLoadLoad
finalalpha
reader()3
·obj
·objj
·objfinali
A



Afinalfinal“”finalA
finalfinalfinalnullfinalA

final

FinalReferenceExample
finalfinal
AwriterOne()BwriterTwo()Creader()



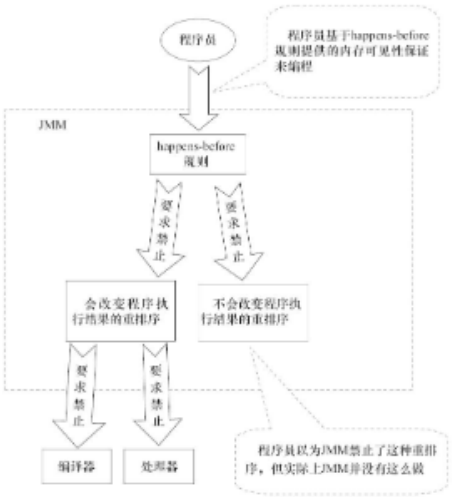
1final2final31323

JMMCAfinalC01BCJMMBCBC
CBBClockvolatile

happens-before

JMM

JMMJMM
.
.
JSR-133JMMJSR-133
double pi = 3.14;   // A
double r   = 1.0;   // B
double area = pi * r * r; // C
3happens-before
·A happens-before B
·B happens-before C
·A happens-before C
3happens-before231JMMhappens-before
.
.
JMM
·JMM
·JMMJMM
JMM



·JMMhappens-beforeJMMhappens-beforeA happens-before B
·JMMJMMvolatilevolatile

happens-beforeLeslie LamportTimeClocks and the Ordering of Events in a Distributed SystemLeslie Lamporthappens-beforepartial orderingLeslie Lamport
JSR-133happens-beforeJMMhappens-beforeAaBbhappens-beforeabJMMab
JSR-133:Java Memory Model and Thread Specificationhappens-before
1happens-before
2happens-beforeJavahappens-before
happens-beforeJMM
1JMMhappens-beforeA happens-before BJava——ABABJava
2JMMJMMJMMhappens-beforeas-if-serial
. as-if-serialhappens-before
. as-if-serialhappens-beforehappens-before
as-if-serialhappens-before

happens-before

JSR-133:Java Memory Model and Thread Specificationhappens-before
1happens-before
2happens-before
3volatilevolatilehappens-beforevolatile
4A happens-before BB happens-before CA happens-before C
5start()AThreadB.start()BAThreadB.start()happens-beforeB
6join()AThreadB.join()Bhappens-beforeAThreadB.join()

:

```
public class Test {
    public static void main(String[] args) {
        Account account = new Account( name: "zhang san",  amount: 10000.0f);
        AccountOperator accountOperator = new AccountOperator(account);

        final int THREAD_NUM = 5;
        Thread threads[] = new Thread[THREAD_NUM];
        for (int i = 0; i < THREAD_NUM; i ++) {
            threads[i] = new Thread(accountOperator,  name: "Thread" + i);
            threads[i].start();
        }
    }
}
```

```java
*/
public class MonitorExample {
    int a = 0;
    public synchronized  void writer() { //1
        a++;//2
    }//3

    public synchronized void reader() {//4
        int i =a;//5
    }//6
}
```

```java
class ReorderExample {
    int a = 0;
    boolean flag = false;

    public void writer() {
        a = 1; //1
        flag = true; //2
    }

    public void reader() {
        if (flag) { //3
            int i = a * a; //4

        }
    }
}
```

```java
public class VolatileExample {
    volatile long vl = 1L; // 使用volatile声明64位的long型变量

    public void set(long l) { vl = l; //单个volatile变量的写 }

    public long get() { return vl; //单个volatile变量的读 }

    public void getAndIncrement() { vl++; // 复合(多个)volatile变量的读/写 }
}
```

```java
*/
public class VolatileExampleHe {
    int a = 0;
    volatile boolean flag = false;

    public void writer() {
        a = 1; //1
        flag = true; //2
    }

    public void reader() {
        if (flag) { //3
            int i = a; //4

        }
    }


}
```

```java
public class VolatileFeaturesExample {
    long vl = 0L;

    public synchronized void set(long l) { //对单个的普通变量的写用同一个锁同步
        vl = l;
    }

    public synchronized long get() { //对单个的普通变量的读用同一个锁同步
        return vl;
    }

    public void getAndIncrement() { //普通方法调通
        long temp = get(); //调用已同步的读方法
        temp += 1L;// 普通写操作
        set(temp); // 调用已同步的写方法
    }
}
```

```java
public class FinalExample {
    int i; //普通变量
    final int j; //final变量
    static FinalExample obj;

    public FinalExample() { //构造函数
        i = 1; //写普通域
        j = 2;//写final域
    }

    public static void writer() {// 写线程A执行
        obj = new FinalExample();
    }

    public static void reader() {// 读线程B执行
        FinalExample object = obj; // 读对象引用
        int a = object.i;// 读普通域
        int b = object.j;// 读final域
    }

}
```

```java
public class FinalReferenceExample {
    final int[] intArray;// final是引用类型
    static FinalReferenceExample obj;

    public FinalReferenceExample() {// 构造函数
        intArray = new int[1]; // 1
        intArray[0] = 1; // 2
    }

    public static void writerOne() { obj = new FinalReferenceExample(); // 3 }


    public static void writerTwo() { obj.intArray[0] = 2; // 4 }

    public static void reader() { // 读线程C执行
        if (obj != null) { // 5
            int temp1 = obj.intArray[0]; // 6
        }
    }
}
```