L05-线程安全根因详解

JAVA并发编程

学习目标

- 掌握变量可见性问题
- 掌握JMM内存模型
- 掌握Volatile关键字
- 理解可见性、原子性、有序性

并发中的变量可见性问题

什么是并发中的变量可见性问题

- 问题1、变量分为哪几类?

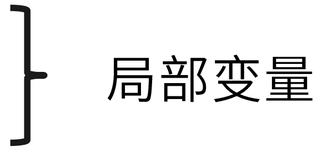
全局变量

局部变量

属性（静态的、非静态的）

本地变量

参数



什么是并发中的变量可见性问题

- 问题2、如何在多个线程间共享数据？

  用 全局变量： 静态变量，或共享对象

- 问题3、一个变量在线程1中被改变值了，在线程2中能看到该变量的最新值吗?

什么是并发中的变量可见性问题

- 示例代码

代码逻辑：通过共享变量，在一个线程中控制另一个线程的执行流程。

请问：线程会停止循环，打印出i的值吗？

```
import java.util.concurrent.TimeUnit;

public class VisibilityDemo {
        // 状态标识
        private static boolean is = true;

        public static void main(String[] args) {
                new Thread(new Runnable() {
                        public void run() {
                                int i = 0;
                                while (VisibilityDemo.is) {
                                        i++;
                                }
                                System.out.println(i);
                        }
                }).start();

                try {
                        TimeUnit.SECONDS.sleep(2);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                // 设置is为false，使上面的线程结束while循环
                VisibilityDemo.is = false;
                System.out.println("被置为false了.");
        }
}
```

什么是并发中的变量可见性问题

并发的线程能不能看到变量的最新值，这就是并发中的变量可见性问题

- 为什么会不可见?
- 怎样才能可见?

怎样才能可见

方式一： synchronized

```
import java.util.concurrent.TimeUnit;

public class VisibilityDemo {
        // 状态标识
        private static  boolean is = true;

        public static void main(String[] args) {
                new Thread(new Runnable() {
                        public void run() {
                                int i = 0;
                                while (VisibilityDemo.is) {
                                        synchronized (this) {
                                           i++;
                                         }
                                }
                                System.out.println(i);
                        }
                }).start();

                try {
                        TimeUnit.SECONDS.sleep(2);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                // 设置is为false，使上面的线程结束while循环
                VisibilityDemo.is = false;
                System.out.println("被置为false了.");
        }
}
```

怎样才能可见

方式一： synchronized

方式二： volatile

用它们，为什么就可见了？

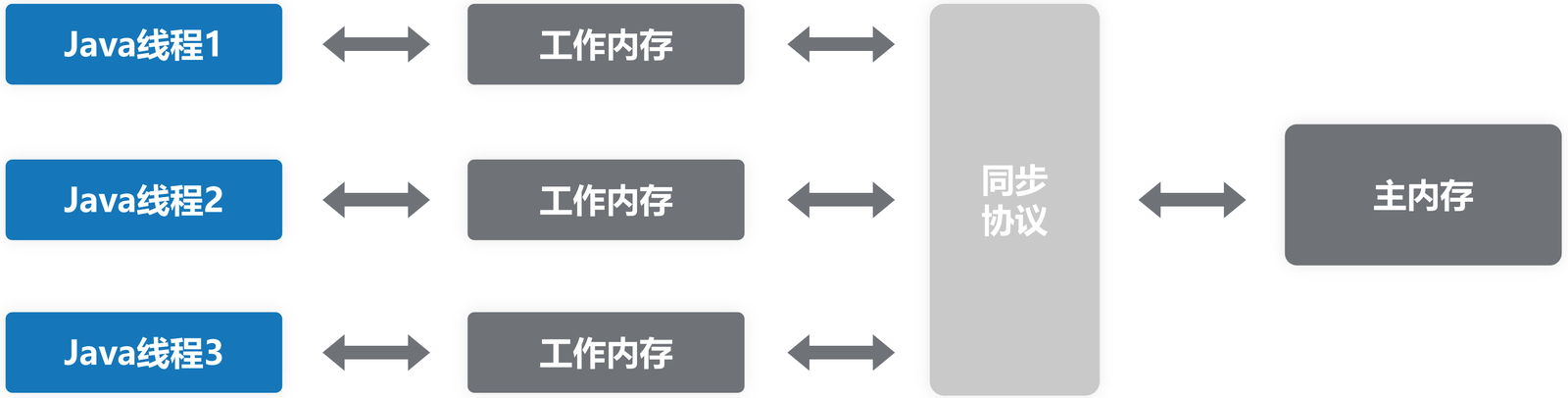
```
import java.util.concurrent.TimeUnit;

public class VisibilityDemo {
        // 状态标识
        private static volatile boolean is = true;

        public static void main(String[] args) {
                new Thread(new Runnable() {
                        public void run() {
                                int i = 0;
                                while (VisibilityDemo.is) {
                                        i++;
                                }
                                System.out.println(i);
                        }
                }).start();

                try {
                        TimeUnit.SECONDS.sleep(2);
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                // 设置is为false，使上面的线程结束while循环
                VisibilityDemo.is = false;
                System.out.println("被置为false了.");
        }
}
```

变量可见性、线程安全问题原因

JAVA内存模型

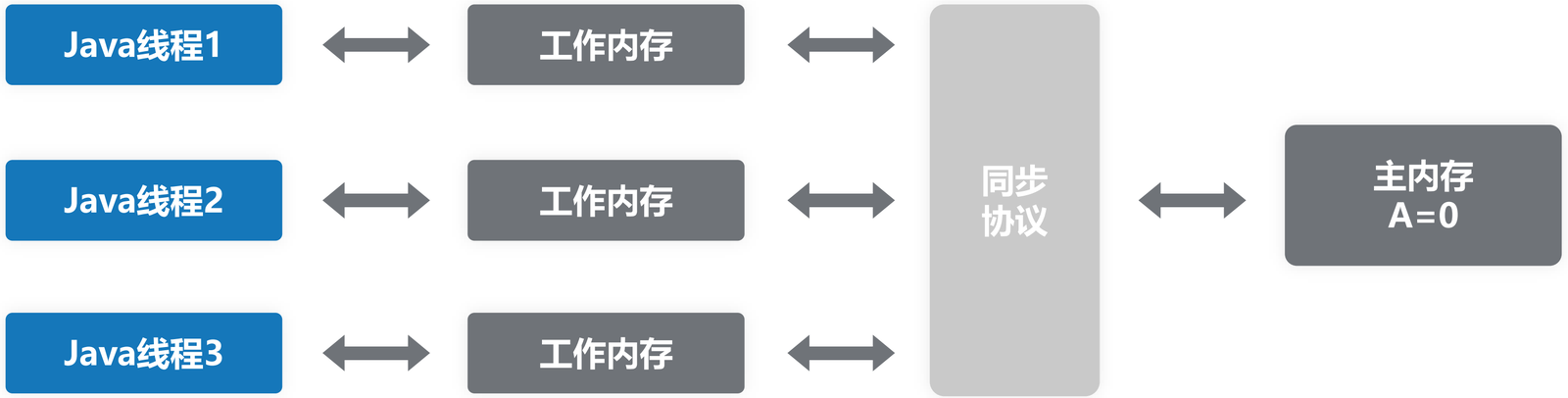
JAVA内存模型及操作规范

- 1、共享变量必须存放在主内存中；
- 2、线程有自己的工作内存，线程只可操作自己的工作内存；
- 3、线程要操作共享变量，需从主内存中读取到工作内存，改变值后需从工作内存同步到主内存中。



JAVA内存模型-带来的问题

- JAVA内存模型会带来什么问题?



- 请思考，有变量A，多线程并发对其累加会有什么问题？如三个线程并发操作变量A，大家读取A时都读到A=0，都对A+1，再将值同步回主内存。结果是多少？

1

这就是线程安全问题。你是否明白线程安全问题的原因了？

如何解决线程安全问题?

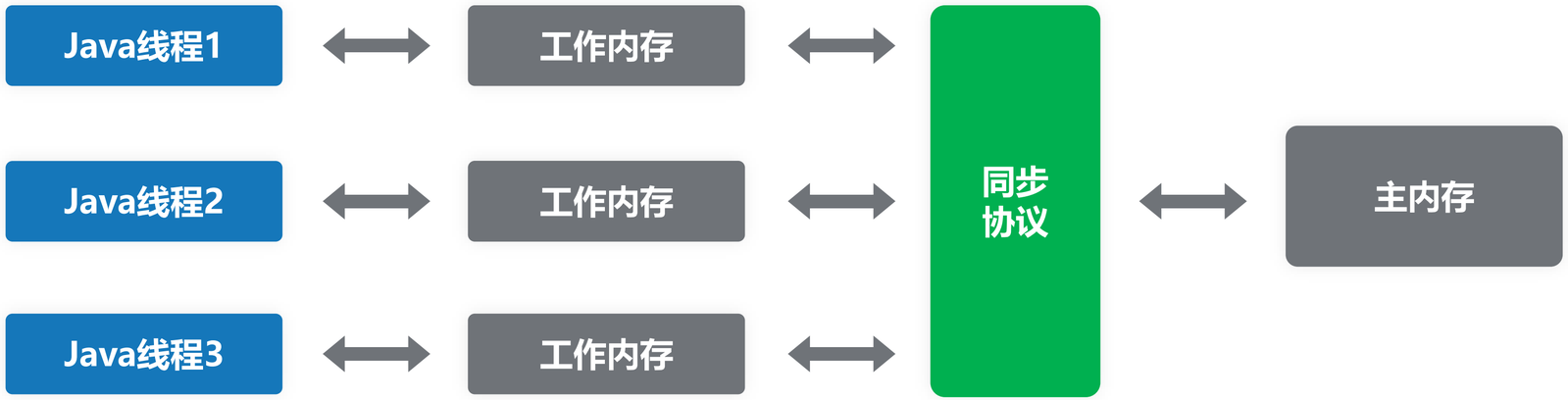
内存模型也产生了变量可见性问题。

JAVA内存模型-带来的问题

- 如何让线程2使用A时看到最新值?
  - 1、线程1修改A后必须立马同步回主内存;
  - 2、线程2使用A前需重新从主内存读取到工作内存。
- 疑问1：使用前不会重新从主内存读取到工作内存吗?
- 疑问2：修改后不会立马同步回主内存吗?

JAVA内存模型-同步协议

- JAVA内存模型



JAVA内存模型-同步协议

- JAVA内存模型—同步交互协议，规定了8种原子操作：

1.lock（锁定）：将主内存中的变量锁定，为一个线程所独占

2.unclock（解锁）：将lock加的锁定解除，此时其它的线程可以有机会访问此变量

3.read（读取）：作用于主内存变量，将主内存中的变量值读到工作内存当中

4.load（载入）：作用于工作内存变量，将read读取的值保存到工作内存中的变量副本中

5.use（使用）：作用于工作内存变量，将值传递给线程的代码执行引擎

6.assign（赋值）：作用于工作内存变量，将执行引擎处理返回的值重新赋值给变量副本

7.store（存储）：作用于工作内存变量，将变量副本的值传送到主内存中。

8.write（写入）：作用于主内存变量，将store传送过来的值写入到主内存的共享变量中

JAVA内存模型-同步协议

- **JAVA内存模型—同步交互协议，执行上述8种原子操作时必须满足如下规则**

1.不允许read和load、store和write操作之一单独出现，即不允许加载或同步工作到一半。

2.不允许一个线程丢弃它最近的assign操作，即变量在工作内存中改变了之后，必须把该变化同步回主内存。

3.不允许一个线程无原因地（无assign操作）把数据从工作内存同步到主内存中。

4.一个新的变量只能在主内存中诞生。

5.一个变量在同一时刻只允许一条线程对其进行lock操作，但lock操作可以被同一条线程重复执行多次，，多次lock之后必须要执行相同次数的unlock操作，变量才会解锁。

6.如果对一个对象进行lock操作，那会清空工作内存变量中的值，在执行引擎使用这个变量前，需要重新执行load或assign操作初始化变量的值。

7.如果一个变量事先没有被lock，就不允许对它进行unlock操作，也不允许去unlock一个被其他线程锁住的变量。

8.对一个变量执行unlock操作之前，必须将此变量同步回主内存中（执行store、write）。

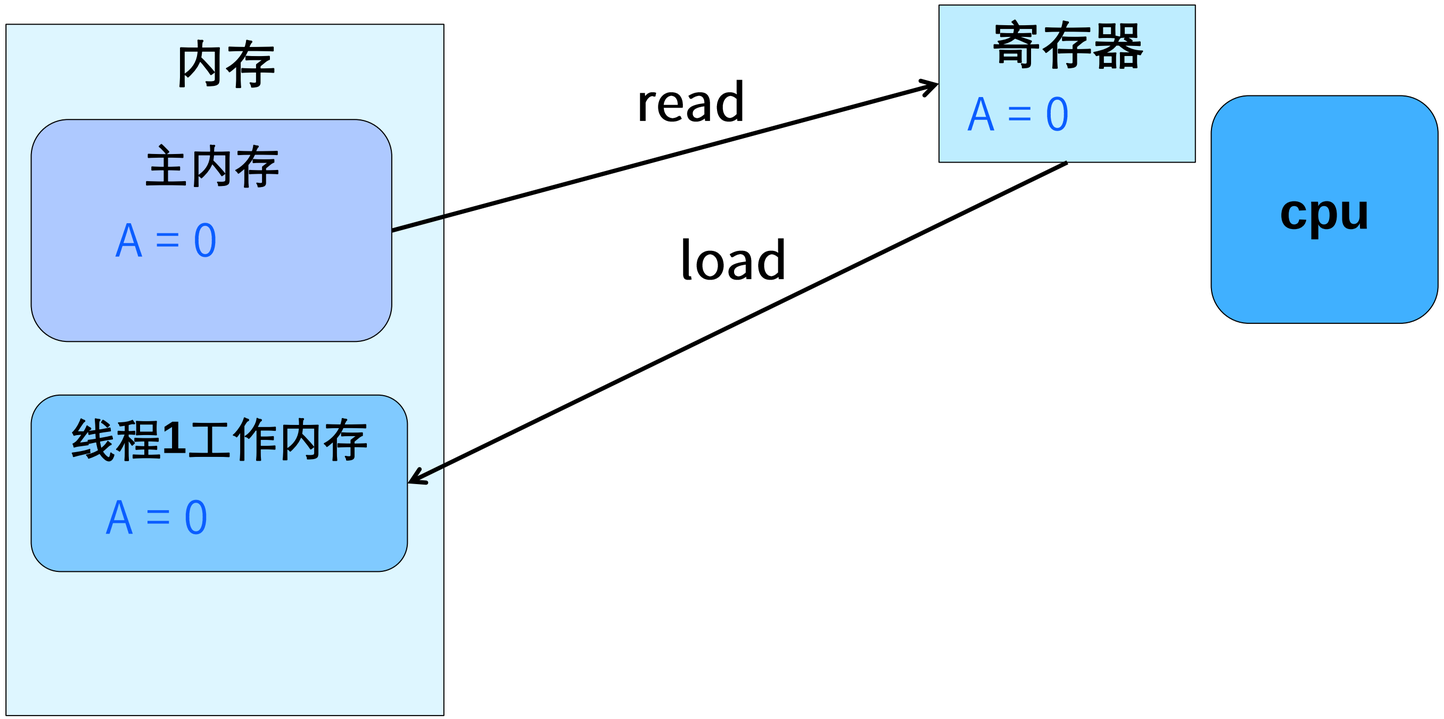
JAVA内存模型-同步协议

JAVA内存模型—同步交互协议，操作规范

1. 将一个变量从主内存复制到工作内存要顺序执行 read、load操作；要将变量从工作内存同步回主内存要顺序执行store、write操作。只要求顺序执行，不一定是连续执行。

JAVA内存模型-同步协议

- read、load 操作示例



保证变量可见性的方式

- 并发中保证变量可见性的方式：

1. final 变量

```
// final 不可变变量
private final int  var1 = 1;
```

2. Synchronized

```
while (VisibilityDemo.is) {
        synchronized (this) {
            i++;
        }
}
```

3. 用volatile修饰

```
// 状态标识
private static volatile boolean is = true;
```

Synchronized怎么做到可见性

Synchronized语义规范：

1.进入同步块前，先清空工作内存中的共享变量，从主内存中重新加载。

2.解锁前必须把修改的共享变量同步回主内存

Synchronized是如何做到线程安全的?

1.锁机制保护共享资源，只有获得锁的线程才可操作共享资源；

2.Synchronized语义规范保证了修改共享资源后，会同步回主内存，就做到了线程安全。

volatile关键字解密

volatile怎么做到可见性

volatile语义规范：

1.使用volatile变量时，必须重新从主内存加载，并且read、load是连续的。

2.修改volatile变量后，必须立马同步回主内存，并且store、write是连续的。

volatile能做到线程安全吗?

1.不能，因为它没有锁机制，线程可并发操作共享资源。

为什么使用 volatile

- **Synchronized可以保证可见性，为什么要用volatile?**

1. 主要原因：使用volatile比synchronized简单
2. volatile性能比synchronized稍好。

volatile还有什么用途

volatile可用于限制局部代码指令重排序

1. 线程A和线程B的部分代码：

```
线程A：
content = initContent(); //(1)
isInit = true; //(2)
```

```
线程B：
if (isInit) { //(3)
        content.operation(); //(4)
}
```

2. jvm优化指令重排序后，代码的执行顺序可能如下：

```
线程A：
isInit = true; //(2)
content = initContent(); //(1)
```

3.当两个线程并发执行时，就可能出现线程B中抛空指针异常。

4.当我们在变量上加volatile修饰时，则用到该变量的块中就不会进行指令重排优化。

volatile的使用场景

volatile的使用范围

1. volatile只可修饰成员变量（静态的、非静态的）。 Why?
2. 多线程并发下，才需要使用它。

volatile典型的应用场景

1.只有一个修改者，多个使用者，要求保证可见性的场景

1.状态标识，如示例中的介绍标识。

2.数据定期发布，多个获取者。

原子性、可见性、有序性

原子性-并发编程时需考虑代码操作的原子性

- **原子操作：不可被中断的一个或一系列操作**

1. JMM定义的变量原子操作：read、load、assign、use、store、write 可认为保证了基本数据类型的访问读写每步操作是原子性（long、double例外）
2. JMM 定义的原子操作对 long、double数据类型允许例外，也规定加了volatile修饰则JVM实现需保证这些操作每个的原子性。不管有无volatile修饰，商用jvm实现都保证了原子性。
3. 要保证一系列操作的原子性，可用synchronized同步快（不可被中断)
4. 对于不支持byte操作的系统，在代码中操作Byte[]时，会出 word tearing（字分裂）问题，代码中需对此进行处理

- **Volatile 只提供单个操作的原子性，不能保证一系列操作，不能保证线程安全，因此说：不保证原子性**

可见性-并发编程时需考虑共享变量的可见性

- Volatile/synchronized/final保证可见性

有序性-并发编程时需考虑代码执行的有序性（指令重排）

Java程序的有序性情况

1. 一个线程内，所有操作都是有序的，即 “线程内表现为串行的语义 ”(Within Thread As-if-Serial Semantics)
2. 在一个线程中观察另一个线程，所有的操作都是无序的：指令重排、工作内存与主内存同步延迟

并发编程时，我们如何保证有序性?

1. JMM规定了 happens-before （先行发生）原则，来保证很多操作的有序性；
2. 当我们的代码操作不满足先行发生原则时，则需在编码时使用volatile、synchronized来保证有序性。

JMM的HB法则

happens-before 关系用于描述两个有冲突的动作之间的顺序，如果一个action happends before 另一个action，则第一个操作被第二个操作可见，JVM需要实现如下happens-before规则：

- 程序顺序规则：每个线程中的每个操作都 happens-before 该线程中任意的后续操作。
- 监视器锁规则：一个锁的解除，happens-before与随后对这个锁的加锁。
- volatile变量规则：对volatile域的写，happens-before于任意后续对这个volatile域的读
- 线程启动规则：在某个线程对象上调用 start()方法 happens-before 被启动线程中的任意动作
- 线程终止规则：线程中的所有操作都先行发生于对此线程的终止检测，如在线程t1中成功执行了t2.join()，则t2中的所有操作对t1可见。
- 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。
- 对象终结规则：一个对象的初始化完成（构造函数执行结束）先行发生于它的finalize方法的开始。
- 传递性：如果A happens-before于B，且B happens-before于C，那么A happens-before于C

动手实践，下节课见