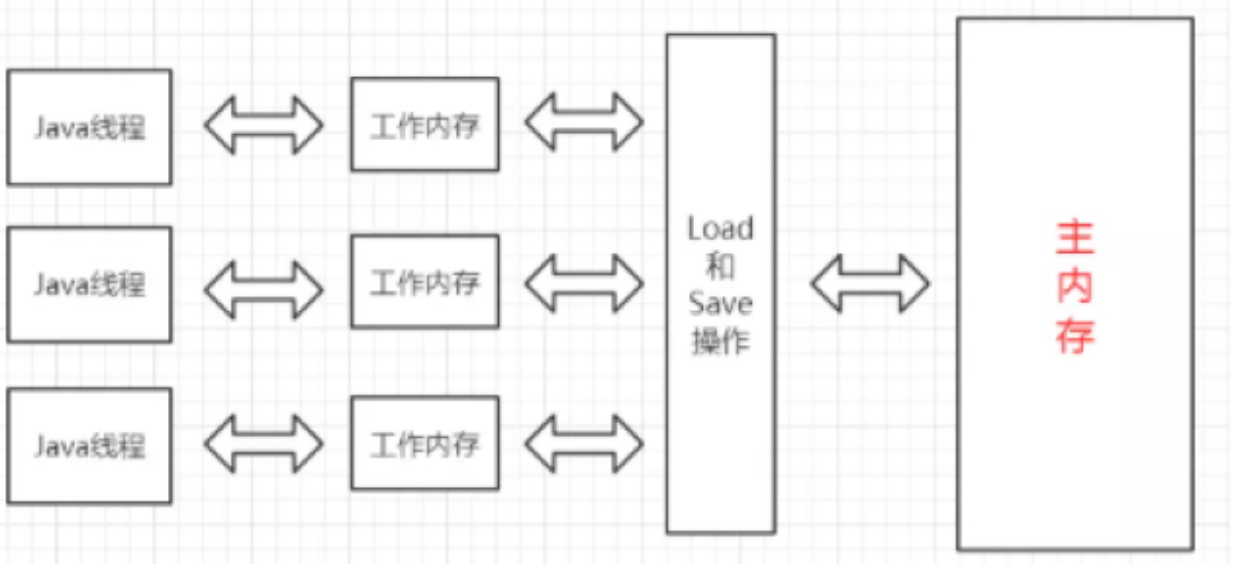
1、volatile 能保证内存可见性

volatile 修饰的变量, 能够保证 "内存可见性".



代码在写入 volatile 修饰的变量的时候,

- 改变线程工作内存中volatile变量副本的值
- 将改变后的副本的值从工作内存刷新到主内存

代码在读取 volatile 修饰的变量的时候,

- 从主内存中读取volatile变量的最新值到线程的工作内存中
- 从工作内存中读取volatile变量的副本

前面我们讨论内存可见性时说了, 直接访问工作内存(实际是 CPU 的寄存器或者 CPU 的缓存), 速度非常快, 但是可能出现数据不一致的情况.

加上 volatile , 强制读写内存. 速度是慢了, 但是数据变的更准确了.

代码示例:

在这个代码中

- 创建两个线程 t1 和 t2

- t1 中包含一个循环, 这个循环以 flag == 0 为循环条件.
- t2 中从键盘读入一个整数, 并把这个整数赋值给 flag.
- 预期当用户输入非 0 的值的时候, t1 线程结束.

```java
public class ThreadDemo13 {
 static class Counter{
 // 一旦给这个 flag 加上 volatile 之后,此时针对 flag 的读写操作,就能保证一定是操作内存了
 public int flag = 0 ;
 }
 public static void main(String[] args) {
 Counter c = new Counter();
 Thread t1 = new Thread(){
 @Override
  public void run() {
  while(c.flag == 0){
 // 假设执行一些操作
 }
 System.out.println("循环结束!");
 }
 };
 t1.start();

 Thread t2 = new Thread(){
 @Override
 public void run() {
 // 让用户输入一个整数,输入的值替换 c.flag 的值
 Scanner scan = new Scanner(System.in);
 System.out.println("请输入一个整数:");
 c.flag = scan.nextInt();
```

```
 }
 };
 t2.start();
 }
}
// 执行效果
// 当用户输入非0值时，t1 线程循环不会结束.（这显然是一个 bug）
```

t1 读的是自己工作内存中的内容.

当 t2 对 flag 变量进行修改, 此时 t1 感知不到 flag 的变化.

如果给 flag 加上 volatile

```
static class Counter {
 public volatile int flag = 0;
}
// 执行效果
// 当用户输入非0值时，t1 线程循环能够立即结束
```

2、volatile 不保证原子性

volatile 和 synchronized 有着本质的区别. synchronized 能够保证原子性, volatile 保证的是内存可见性.

代码示例:

这个是最初的演示线程安全的代码.

i. 给 increase 方法去掉 synchronized

ii. 给 count 加上 volatile 关键字.

```
static class Counter {
 volatile public int count = 0;
 void increase() {
 count++;
 }
}
public static void main(String[] args) throws InterruptedExcepti
on {
 final Counter counter = new Counter();
 Thread t1 = new Thread(() -> {
 for (int i = 0; i < 50000; i++) {
```

```
11   counter.increase();
12   }
13   });
14   Thread t2 = new Thread(() -> {
15   for (int i = 0; i < 50000; i++) {
16   counter.increase();
17   }
18   });
19
20   t1.start();
21   t2.start();
22   t1.join();
23   t2.join();
24   System.out.println(counter.count);
25  }
```

此时可以看到, 最终 count 的值仍然无法保证是 100000.

3、synchronized 也能保证内存可见性

synchronized 既能保证原子性, 也能保证内存可见性.

对上面的代码进行调整:

a. 去掉 flag 的 volatile

b. 给 t1 的循环内部加上 synchronized, 并借助 counter 对象加锁.

```
1  static class Counter {
2   public int flag = 0;
3  }
4
5  public static void main(String[] args) {
6   Counter counter = new Counter();
7   Thread t1 = new Thread(() -> {
8   while (true) {
9   synchronized (counter) {
10   if (counter.flag != 0) {
11   break;
12   }
```

```
    }
    // do nothing
    }
    System.out.println("循环结束!");
    });
    Thread t2 = new Thread(() -> {
    Scanner scanner = new Scanner(System.in);
    System.out.println("输入一个整数:");
    counter.flag = scanner.nextInt();
    });
    t1.start();
    t2.start();
}
```

4、经典面试题：synchronized 和 volatile 的区别