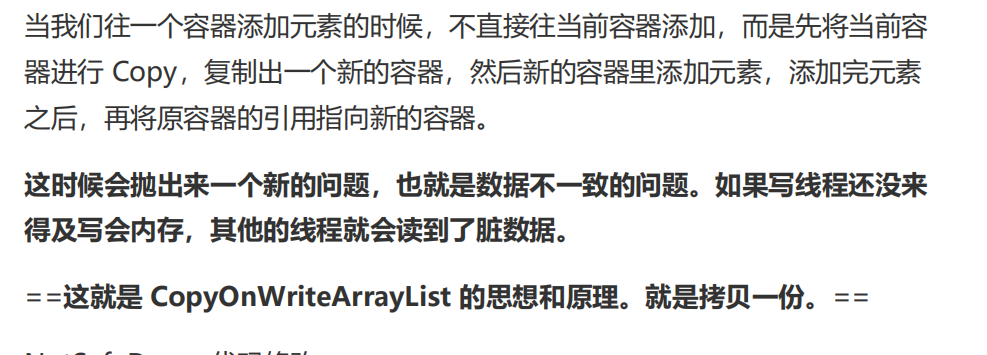
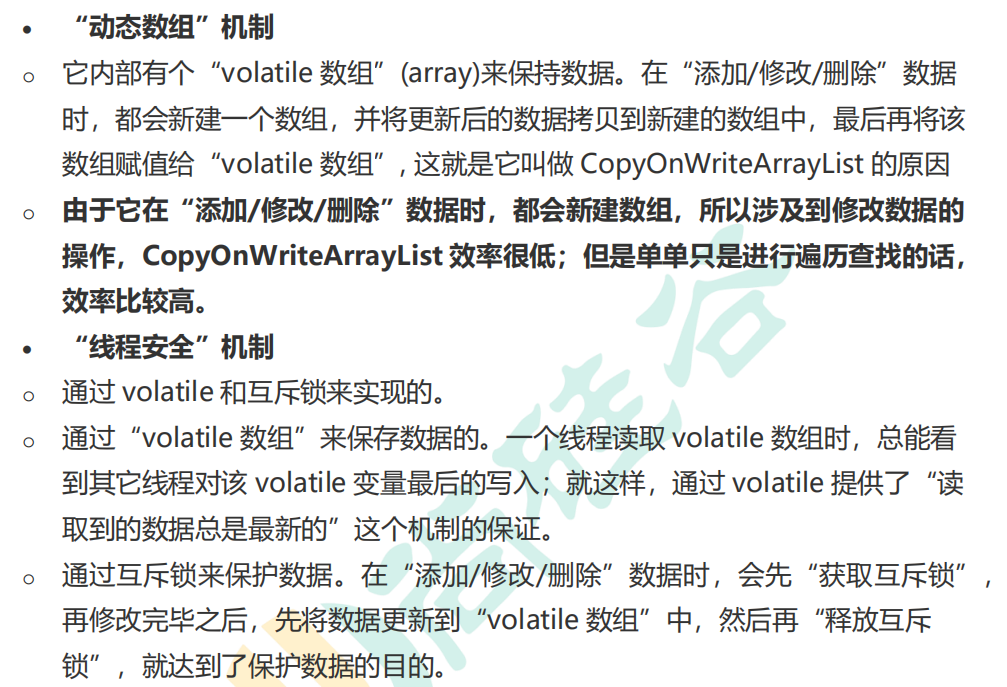
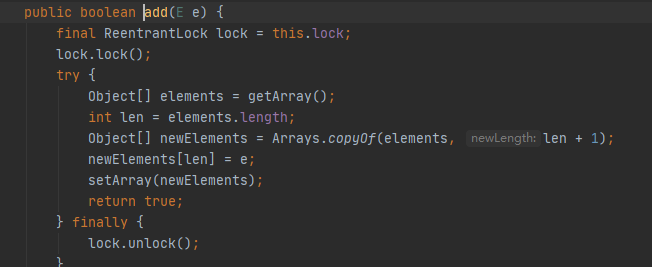
**CopyOnWriteArrayList原理：**



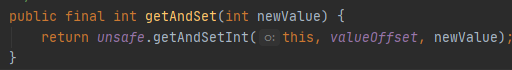


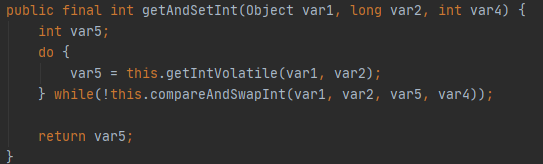
互斥锁使用reentrantlock,部分源码:



**CAS+volatile 解决AtomicInteger原子性：**







**volatile 解决可见性和**防止指令重排**的原理：**

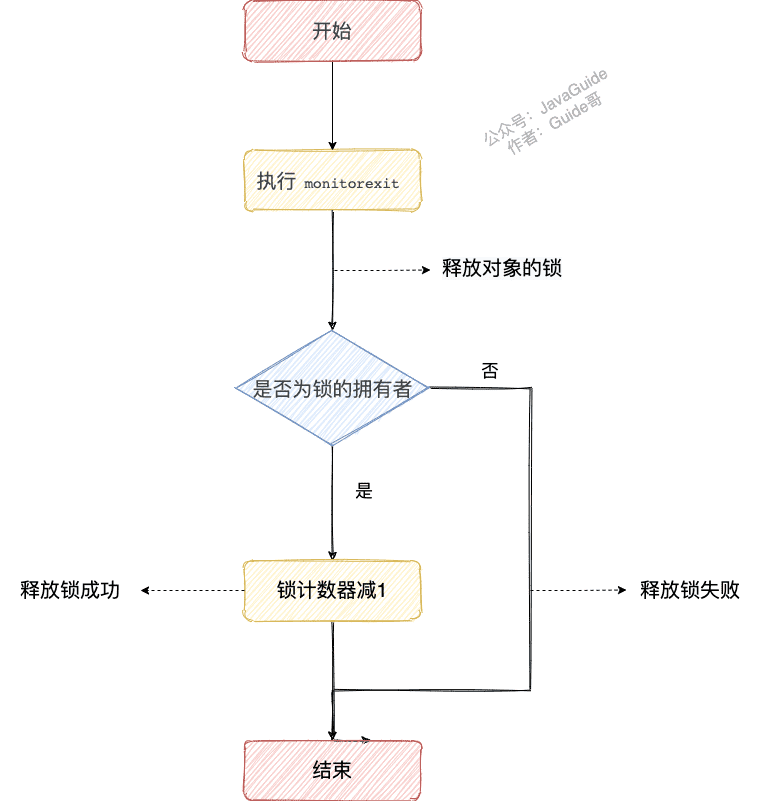
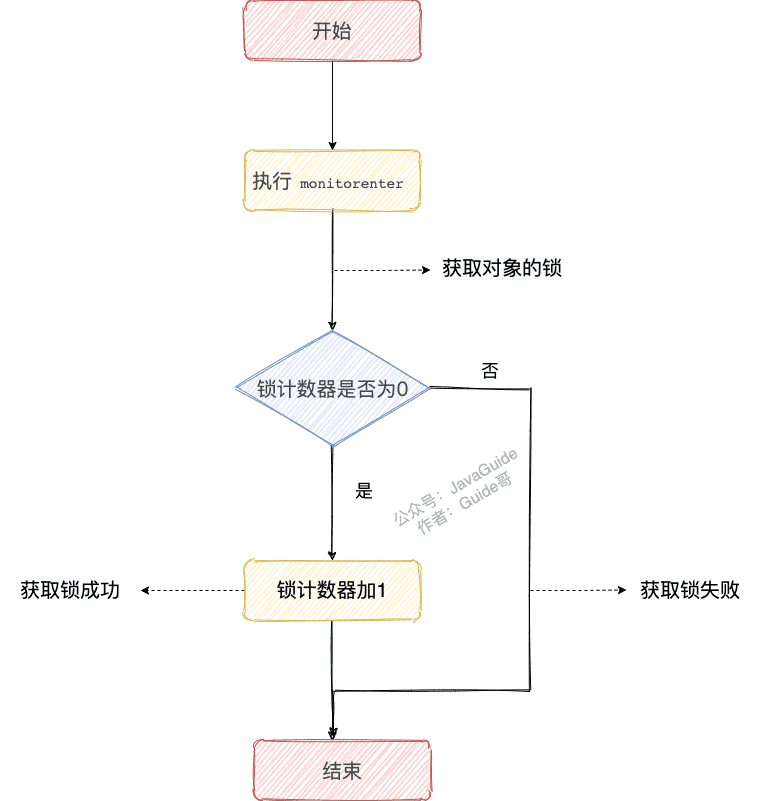
编译之后会多出两行汇编代码，lock为前缀的代码会做两件事

1：将cpu缓存写到内存

2：这个写回内存的操作会使在其他CPU里缓存了该内存地址的数据无效。

**Synchronized原理：**

通过mornitorenter和mornitorexit来实现。锁对象的情况。



**公平锁 非公平锁**

**Lock和synchronized的区别：**

1. lock是接口，synchronized是关键字
2. synchronized自动释放锁，lock需要手动释放

\*细节：lock.unlock()放在finally{}执行，避免无法释放

|  |
| --- |
| public interface Lock {  void lock();  void lockInterruptibly() throws InterruptedException;  boolean tryLock();  boolean tryLock(long time, TimeUnit unit) throws InterruptedException;  void unlock();  Condition newCondition(); //实现await()和signal()  } |



**Synchronized有什么问题？**

如果一个代码块被 synchronized 修饰了，当一个线程获取了对应的锁，并执 行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁，而这里 获取锁的线程释放锁只会有两种情况：

1）获取锁的线程执行完了该代码块，然后线程释放对锁的占有；

2）线程执行发生异常，此时 JVM 会让线程自动释放锁。

那么如果这个获取锁的线程由于要等待 IO 或者其他原因（比如调用 sleep 方法）被阻塞了，但是又没有释放锁，其他线程便只能干巴巴地等待，试想一下，这多么影响程序执行效率。

因此就需要有一种机制可以不让等待的线程一直无期限地等待下去（比如只等待一定的时间或者能够响应中断），通过 Lock 就可以办到。

CountDownLatch:

|  |
| --- |
| package com.atguigu.test;  import java.util.concurrent.CountDownLatch;  public class CountDownLatchDemo {  /\*\*  \* 6 个同学陆续离开教室后值班同学才可以关门  \*/  public static void main(String[] args) throws Exception{  //定义一个数值为 6 的计数器  CountDownLatch countDownLatch = new CountDownLatch(6);  //创建 6 个同学  for (int i = 1; i <= 6; i++) {  new Thread(() ->{  try{  if(Thread.currentThread().getName().equals("同学 6")){  Thread.sleep(2000);  }  System.out.println(Thread.currentThread().getName() + "离开了");  //计数器减一,不会阻塞 countDownLatch.countDown();  }catch (Exception e){  e.printStackTrace();  }  }, "同学" + i).start();  }  //主线程 await 休息  System.out.println("主线程睡觉");  countDownLatch.await();  //全部离开后自动唤醒主线程  System.out.println("全部离开了,现在的计数器为" +  countDownLatch.getCount());  }  } |

**为什么要设计可重入锁：**

比如a()和b()方法 都是需要对对象o加锁才可以执行，那么a调用b就会死锁。所以必须可重入，检查这个锁的线程是不是自己。

**AQS**

[**https://blog.csdn.net/u011863024/article/details/115270840**](https://blog.csdn.net/u011863024/article/details/115270840)

**ConcurrentHashMap**

**分段锁**

**乐观锁 悲观锁**

**数据库**

**https://baijiahao.baidu.com/s?id=1717095300761675602&wfr=spider&for=pc**

**脏读：**

“脏读脏读又称无效数据的读出，是指在数据库访问中，事务T1将某一值修改，然后事务T2读取该值，此后T1因为某种原因撤销对该值的修改，这就导致了T2所读取到的数据是无效的，值得注意的是，脏读一般是针对于update操作的。

**不可重复读**

**事务 A 多次读取同一数据，但事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。**

**幻读**

一个事务内部，第一次查询某数据是存在的，下一次查，不存在了，仿佛出现了幻觉。

**可重复读 脏读 幻读 解决方式**

**索引**

**快排 归并 冒泡（交换） 选择**

**网络**

**TCP/IP**

**TCP UDP**

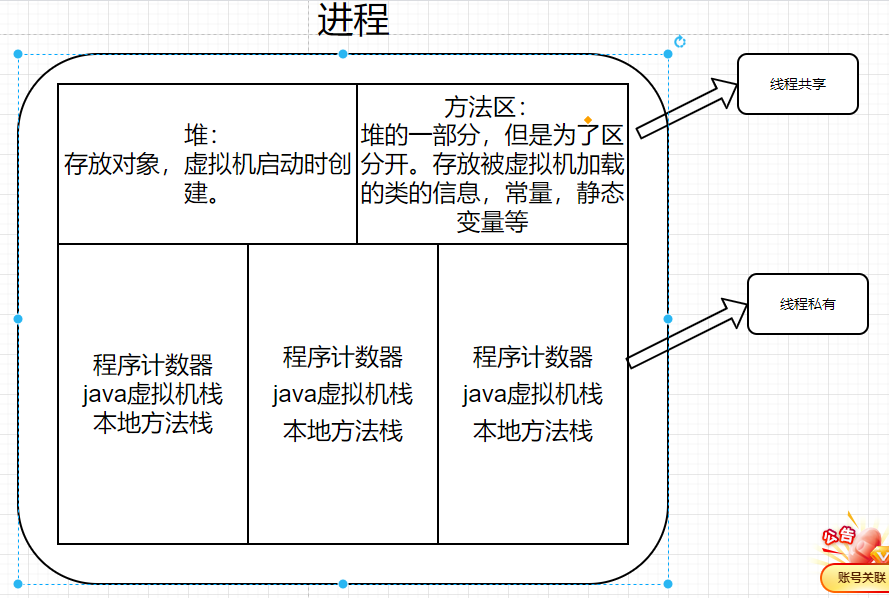
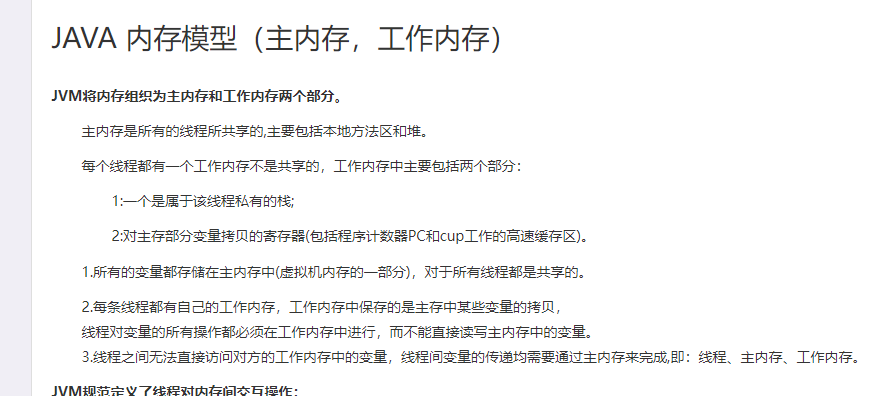
**三次握手**

**jvm**

**算法**

**Bean的生命周期**

**JMM**

**z**