Synchronized

# 一、多线程基础

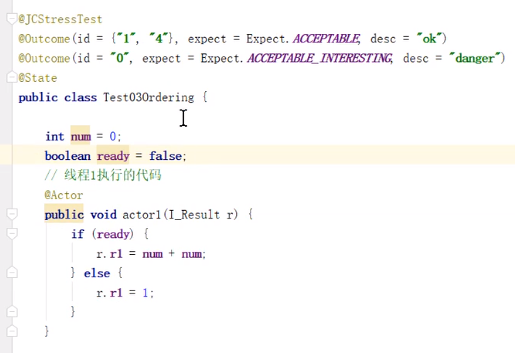
多个线程同时操作一个共享数据，这个数据的值很可能会出现错乱，这就叫线程安全问题。

导致这个数据异常的三个原因是：可见性、原子性、有序性。

1. 可见性问题：一个线程对数据修改之后，其他线程不知道这个值已经被修改。
2. 原子性问题：当一个线程对共享数据操作到一半的时候，其他线程也来操作共享变量，干扰了前一个线程的操作。
3. Java在编译和运行的时候会对指令进行重新排序。

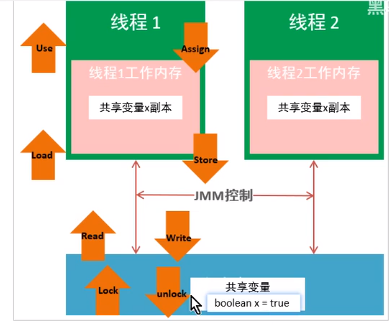
压力测试工具：

org.openjdk.jcstress



# Java内存模型

Java内存模型是一套规范，描述了Java程序中各种变量，线程共享变量的访问规则，以及在JVM中将变量存储到内存和从内存中读取变量的细节。主内存和各自线程的工作内存。



Lock -> Read -> Load -> Use -> Assign ->Store -> Write ->Unlock

# 三、sinchronized

能够保证在同一时刻最多只有一个线程执行一段代码。

保证原子性：synchronized能够保证只有一个线程那到锁进入同步代码块。

可见性保证：执行synchronized的时候，会对应Lock原子操作，会刷新工作内存中共享变量的值。

重排序：模拟在单线程情况下，改变代码顺序不影响计算结果的时候，CPU会重排序。

两大特性： 可重入，不可中断。

1、可重入： syncgronized的锁对象中有一个计数器（recursions变量）会记录线程获得几次锁。

可以避免死锁。可以更好的封装代码。

1. 不可中断特性

当一个线程获得锁之后，另一个线程想要获得锁，就必须处于阻塞或者等待状态，如果第一个线程不释放锁，第二个线程会一直阻塞或者等待，并且后一个线程不可以被中断。

synchronized属于不可被中断 线程的状态是BLOCK。

Lock对象，如果使用lock()方法，是不可中断的，线程状态是WAITING。

Lock对象，使用的是tryLock()方法，是可以中断的，线程去执行其他操作了。

# 四、synchronized原理

## monitorenter

对象的关联对象monitor对象才是真正的锁，它又两个重要的变量，一个叫owner，记录拥有锁的线程是哪一个，

另一个是recursions,记录了获取锁对象的次数。

在同步代码块开始的地方。

每一个对象都会和一个监视器monitor关联。监视器被占用时会被锁住，其他线程无法来获取monitor。当JVM执行某个线程的某个方法内部的monitorenter的时候，它会尝试去获取当前对象对应的monitor的所有权。其过程如下：

1. 当mointor的进入数为0，线程可以进入monitor，并将monitor的进入数变为1，当前线程称为mointor的所有者。
2. 若线程已经拥有monitor的所有权，允许它重入monitor，则进入monitor的进入数加一。
3. 若其他线程已经占有monitor的所有权，那么当前尝试获取monitor的所有权的线程会被阻塞，直到monitor的进入数量变为0，才能重新尝试获取monitor的所有权。

小结：

synchronized的锁对象会关联一个monitor，这个monitor不是主动创建的，是JVM的线程执行到这个听不代码块，发现锁对象没有monitor就会创建monitor，它内部的两个重要成员变量，owner拥有这把锁的线程，recursions会记录线程拥有锁的次数。当一个线程拥有monitor 其他线程只能等待。

## monitorexit

能执行monitorexit指令的线程一定时当前对象的monitor的所有者的线程。

执行monitorexit的时候会将monitor的进入数减一，当减到0的时候，当前线程退出monitor，不再拥有monitor的所有权，此时其他被这个monitor阻塞的线程可以尝试获取这个monitor的所有权。

JVM在方法的结束处和异常处都有monitorexit。保证在出现异常的时候也能释放锁。

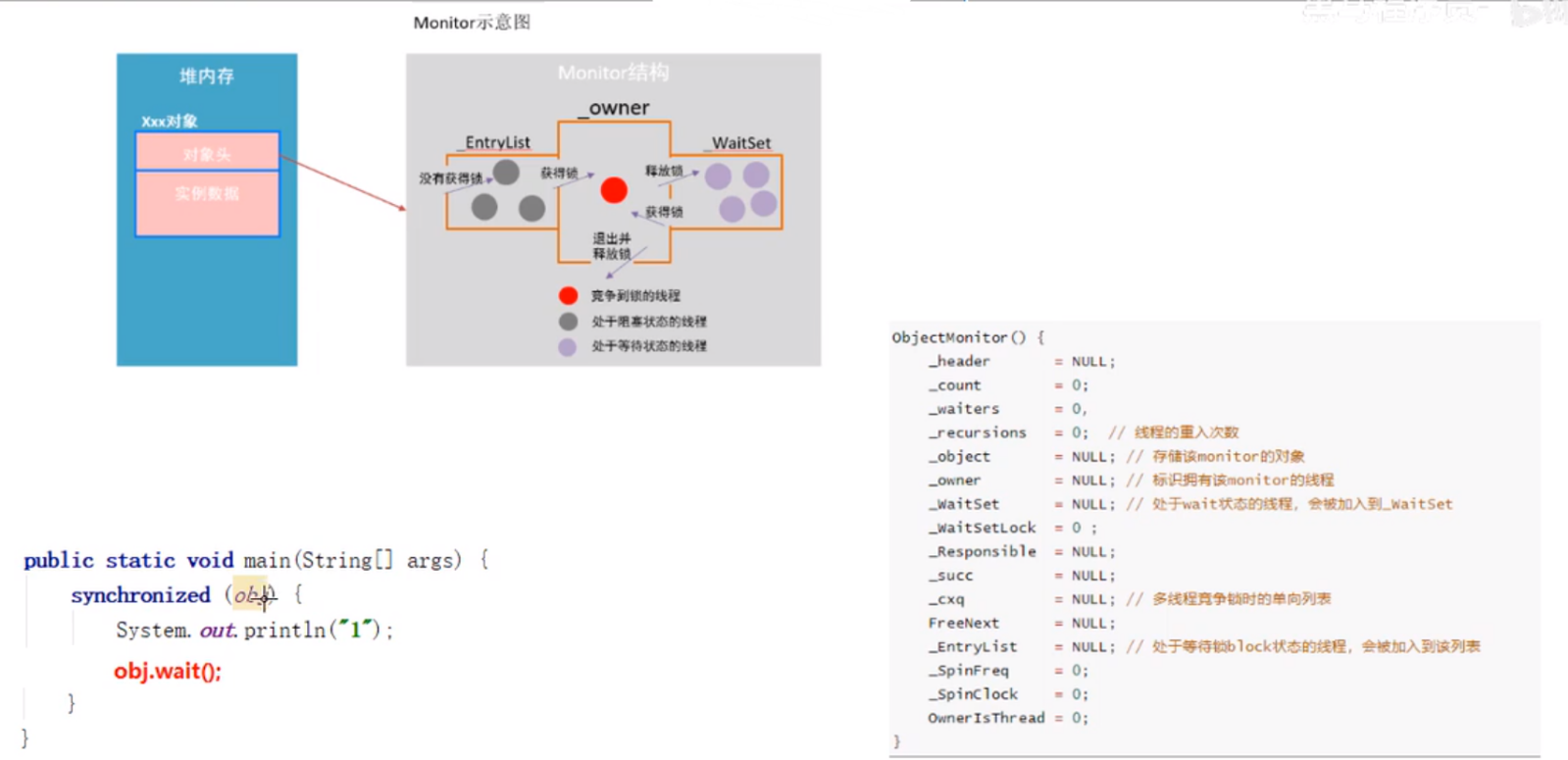
字节码指令的异常表。

同步方法在编译的时候不会生产monitorenter等指令，但是这个方法体会被ACC\_SYNCHRONIZED修饰，会隐式的调用monitorenter和monitorexit，在执行同步方法之前会调用monitroenter，执行同步方法之后会调用monitorexit。

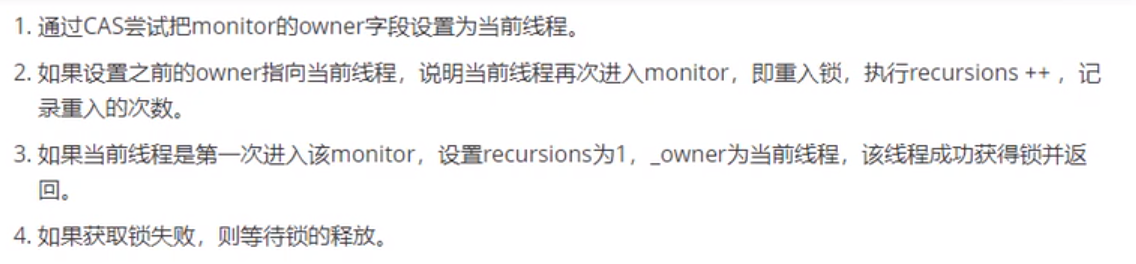
# synchronized和Lock的区别

1. Synchronized是关键字，JVM直接支持。Lock是一个接口。
2. synchronized会自动释放锁，即时抛出异常也会释放，而Lock必须手动释放锁。
3. synchronized是不可中断的，而Lock使用lock方法是不可中断的，使用trylock是可以中断的。
4. Lock可以知道当前线程有木有拿点锁，synchronized不可以。
5. synchronized可以锁住方法和代码块，而Lock只能锁住代码块。
6. Lock可以使用读锁提高多线程读取数据的效率。
7. synchronized是非公平锁，使用ReentrantLock可以控制是否公平。

# 六、synchronized底层原理

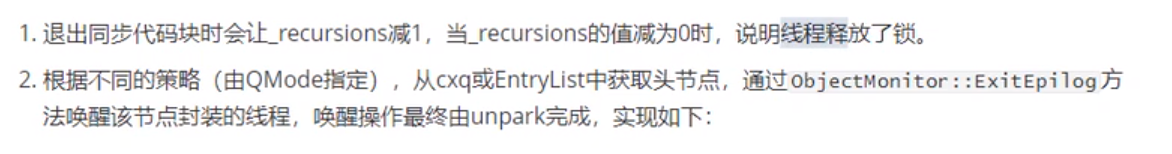


## 锁竞争过程



## 2、锁等待过程

## 3、锁释放过程



# 七、synchronized优化

CAS能保证内存共享数据，变量赋值时候的原子操作。

内存值，预估值，最新值。

CAS是一种乐观锁，适合在竞争不激烈，多核CPU的情况下。

synchronized和reentrantlock都是悲观锁

synchronized升级的过程

无锁 偏向锁 轻量级锁 重量级锁

Java对象由3部分组成，对象头，实例数据，对齐数据

对象头分为两部分

和 Klass pointer

对齐数据吧对象补齐称为8的倍数。





## 偏向锁

大多数情况下，锁不仅不存在多线程竞争，而且总是由同一个线程多次获得，为了让线程获得锁的代价更低，就有了偏向锁。

偏心偏袒的意思。这个锁会偏向于第一个获得它的线程，会在对象头存储偏向的线程ID，以后该线程进入和退出同步块的时候只需要检查是否是偏向锁、锁标志位以及ThreadID即可。

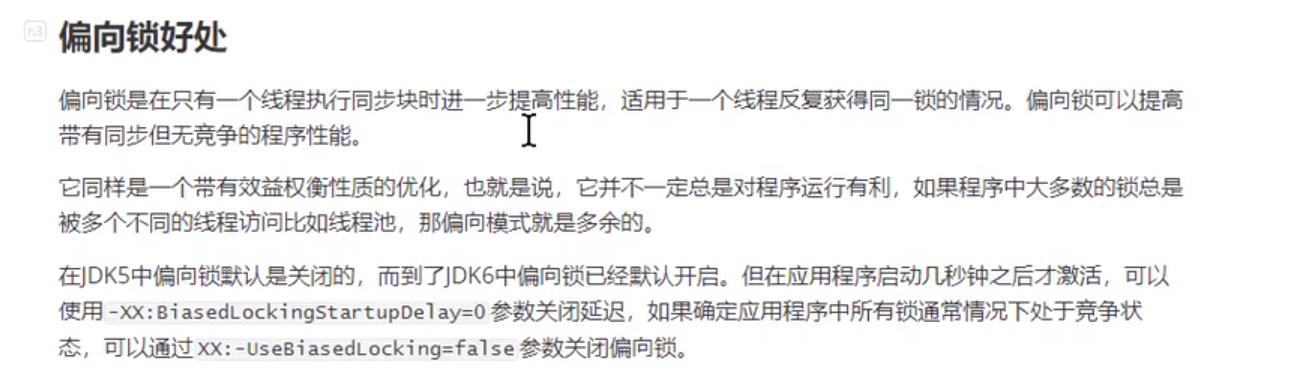
当线程第一次访问同步代码块并获取锁的时候，偏向锁的处理过程：

1. 虚拟机将会把锁住对象的对象头中的标志位设置为01，也就是偏向模式。
2. 同时使用CAS操作把获取到的这个锁的线程ID记录在对象的Mark word中，如果CAS操作成功，持有偏向锁的线程以后进入这个锁相关的同步块的时候，细腻及都可以不再进行任何同步操作，使得效率提高。

## 偏向锁的撤销

1. 偏向锁的撤销动作必须等待一个全局安全点。也就是让所有的线程停下来。
2. 暂停拥有偏向锁的线程，判断锁对象是否处于被锁定的状态。
3. 撤销偏向锁，恢复到无锁（标志位01）或者是轻量级锁（标志位00）的状态。

JVM可以使用参数关闭偏向锁。开启的时候是有延迟的，也可以使用参数关闭延迟。



小总结：

锁对象第一次被线程获取的时候，虚拟机会把对象头中的标志位设置为01，也就是偏向模式。

同时还使用CAS操作把获取到这个锁的线程ID记录在对象头的Mark word中，如果CAS操作成功，持有偏向锁的线程以后每次进入这个锁相关的听不代码块时，虚拟机都不可以再次进行任何操作，偏向锁的效率比较高。

偏向锁的好处？

偏向锁是在一个线程执行同步代码块的时候进一步提升性能，适用于一个线程反复获得同一把锁的情况。偏向锁可以提高带**有同步但无竞争**的程序的性能。

## 轻量级的锁

多个线程交替执行，没有争抢。

一个方法进过调用就会压入栈之后就叫栈帧。

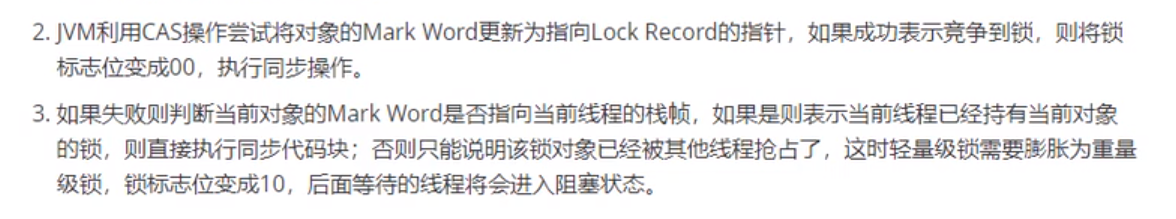
升级为轻量级锁的过程：

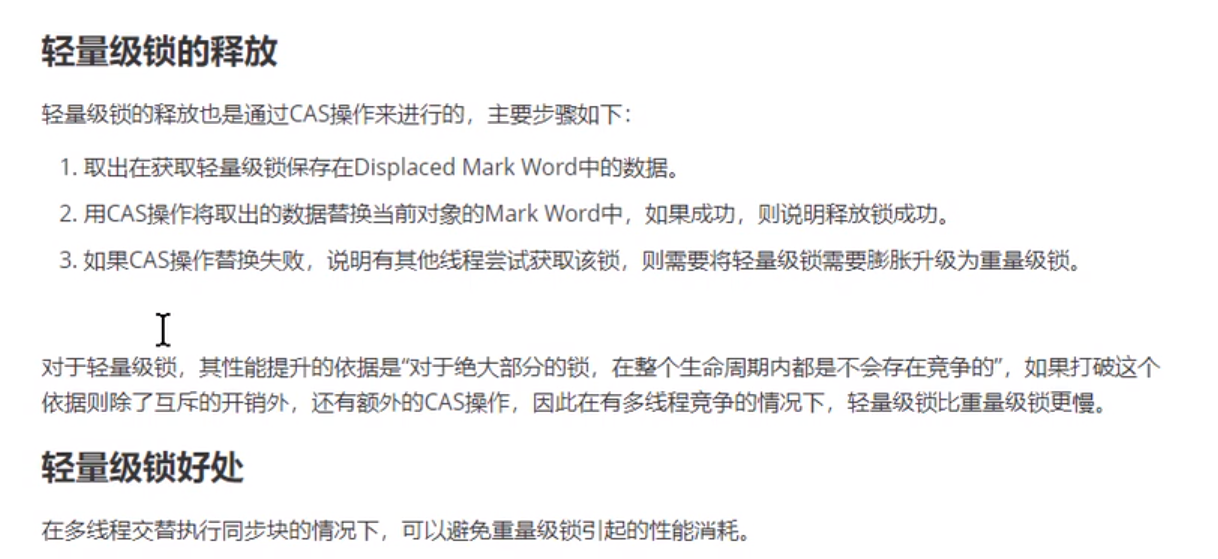
现实在栈帧中创建一个Lock record。

关闭偏向锁或者多个线程竞争偏向锁的时候都会导致偏向锁升级为轻量级锁。

1. 判断当前对象是否处于无锁的状态（hashcode 0 01）,如果是，则JVM首先会在当前线程的栈帧中建立一个名为锁记录的空间，用于存储锁对象目前的MarkWord的拷贝叫做Displaced Mark Word 将对象的MarkWord复制到栈帧中的Lock Record中，将Lock record的owner指向当前对象。

hashcode 分代年龄 锁标志拷贝到Lock record中。





## 锁消除

即时编译器，判断同步代码是否存在竞争。在方法中的对象没有逃逸，就不存在竞争，调用的同步方法就是没有必要的。



## 锁粗化

JVM探测到一连串的重复小操作都使用了同一个对象加锁（多次使用moitorenter），这时JVM就会把同步代码的返回扩大，放到这串重复小操作的外面统一加一次锁即可。

# 平时写代码应该注意什么

1. 减少synchronized的范围 代码减少，减少执行时间，减少锁的竞争。

可能用轻量级锁或者自旋锁就搞定了，避免升级到重量级锁。

1. 降低synchronized锁的粒度 不太无关功能使用不同的锁对象

linkedBlockQueue在入队和出队的时候使用的就是两把锁。

1. 读写分离 读的时候不加锁 写的时候加锁

AQS

AbstractQueueSynchronizer 抽象队列同步器。是其他阻塞式锁和相关的同步器工具的框架。

阻塞式的锁：像synchronized

非阻塞式的锁：CAS

AQS重要属性：state 表示资源的状态（独占和共享）。独占代表只有一个线程能够访问这个共享资源。共享代表运行说个线程来访问资源。

等待队列，先进先出的队列。

等待变量。

从使用CPU到不使用CPU就叫一次山下文切换。

线程常用方法：

start() 创建一个线程，进入就绪状态，等待CPU

run() 线程启动的时候就后就会调用run方法

jon()等待某个线程运行结束，两个线程之间通讯。

interrputed()中断线程

sleep() 暂时放弃对CPU的使用权

yield()提升线程调度器，让出当前线程对CPU的使用

start()和run()都是线程中的方法

直接调用run方法没有达到异步的效果。

sleep()和yield()

sleep调用之后线程会变成Timed Waiting状态

yield()调用之后会变成Runnable就绪状态

join()方法 谁调用join就等待那个线程结束。

比如main线程要等待t1线程运行结束，就使用 t1.join()就行。

LockSupport.pack(); 打断线程