# 多线程基于源码的总结:

1.synchronized用在线程安全问题上,只是保证了线程安全,但此时的程序是串行的.

2.synchronized放在方法上,修饰普通方法,内置锁就是当前的实例对象,修饰静态方法,内置锁就是当前的class字节码文件.

3.synchronized一般用于修饰代码块

4.synchronized的两个主要指令,moniterenter和moniterexit;

5.任何对象都可以作为锁,锁信息存储在对象头中,

对象头中的信息,MarkWord的字段,Class MetaData Address,Array Length.

6.jdk1.6以后,偏向锁,轻量级锁,重量级锁

7.对象头的marwork字段中,记录了线程的id,epoch,对象的分代年龄信息,是否是偏向锁,锁标志位.

8.单线程情况下,偏向锁是不释放锁的,多线程情况下,偏向锁采用了一种等待竞争出现释放锁的机制.

9.轻量级锁,可以同时让多个线程进入同步代码块中.

10.轻量级锁自旋形成重量级锁.

11.单例模式不允许直接通过new创建对象,name对象就是属于类的.

12.双检查锁不能保证线程安全,在指令执行的过程中,含有指令重排序,指令重排序,a.申请一块内存空间,b.在空间里实例化对象,c.instance的引用指向这块空间地址,但是不能保证指令是按照abc顺序执行的,

避免指令重排序,在对象前添加volatile字段

13.synchronized属于重入锁,重入锁的解释,对于一个线程,使用同一个锁对象,线程中的一个方法获取到锁并且没有释放,同一个线程中的另一个方法同样能够获取锁.

14.自旋锁,旋转的是CPU的时间片,使用while(true)一直等待锁获取

15.volatile相当于一个轻量级锁,被volatile修饰的变量在线程中是可见的,在一个线程中修改对象的值,在另一个线程中可以读取到,修改后的值,synchronized除了互斥以外,还保证变量的可见性.

16.volatile使用的场景,死循环中不执行,死循环后执行属于自旋.

17.对象添加了volatile等于添加了LOCK指令,LOCK指令,多处理器中将当前处理器缓存行中的内容写回到系统内存,针对多操作系统,协会到内存中的操作会使其他cpu中缓存该内存地址的数据失效

18.synchronize的实现锁,是一直上下文切换,导致资源浪费

19.实现一个普通锁,就是利用Object对象的wait和notify方法.

20.普通锁的实现方式,分为lock方法和unlock方法,

lock方法,判断当前线程的状态,如果没有被锁,那么使用wait方法锁住,修改标识,变为锁住,

unlock方法,唤醒当前线程,继续执行修改标识,

21.重入锁的实现方式,

在普通锁的基础上,判断是否是当前线程, 如果是当前线程,直接放行,每放行一个方法,就是当前的放行方法个数+1,在释放锁时,需要这些放行的方法全部放行,才能释放锁

22.如果想要使用object对象的wait和notify方法,则调用的方法必须加synchronized 关键字,wait和notify一般和condition协同使用,condition是由其他线程设置的,为了协同,必须使用内置锁.往wait和notify必须释放锁,必须有一个内置锁,

23.使用aqs重写字节的锁

将其作为一个非公共的内部帮助类,重写Tryacquire方法和tryrelease方法,

tryacquire的父类方法只会抛出一个异常,重写acquire方法:

如果是第一个线程,则可以获取锁,如果不是第一个线程,则不能获取锁,判断是否是第一个线程,需要通过当前的state判断,如果是0,则是第一个线程,然后设置当前的state,直接setState的时候会出现线程安全性问题,此处需要通过cas比较设置状态,然后设置独占线程为当前线程,锁的获取和释放时一一对应的,如果不是当前线程就跑出异常.

24.synchronized和reetranlock是排他锁,

读写锁是排他锁和共享锁,读锁是共享锁,写锁是排他锁,

排他锁属于,在同一时刻,只有一个线程能够获取锁.

25.readLock共享锁,调用acquireshare方法,

writeLock 排他锁,调用acquire(1)方法

读写锁需要保存的状态,int值表示重入的次数,

写锁重入的次数,读锁的个数,每个读锁重入的个数(读锁释放,写锁才能进入).

写锁使用低位,读锁使用高位,

26.写锁支持重入

27.读写锁的降级,如果读的过程中含有更新:

读取过程中,需要先加读锁,加完读锁后,有更新,此时读锁与写锁互斥,需要释放读锁,然后加写锁,释放写锁

写锁降级为读锁,在写锁没有释放之前,获取到读锁,然后释放写锁

reentrantlock不支持升级



28.读锁的重入是允许多个读操作的线程的,写锁同时只允许单个线程占有,该线程的写操作可以重入.

29.如果一个线程占有了写锁,在不释放写锁的情况下,他还能占有读锁,即锁降级为读锁.

30.对于同时占有写锁和读锁的线程,如果完全释放了写锁,他就完全转换成读锁,以后的写操作无法重入,在写锁未完全释放时,写操作是可以重入的.

31.公平模式下,无论读锁还是写锁的申请都必须按照AQS锁等待队列先进先出的顺序,公平模式下,读操作插队的条件是锁等待队列head节点后的下一个节点是shared型节点,写锁则无条件插队

32.读锁不允许newCondition获取Condition接口,而写锁的newCondition接口实现方法同ReentrantLock.

33.信号量的判断,使用自旋,自旋是十分消耗cpu资源的.

34.某个线程调用wait方法后,调用wait之前的代码也并不会执行.

35.调用wait方法时,会释放,synchronized的锁

36.condition,使用Condition的相关方法实现wait和notify

生成condition对象的方法,

ReetrantLock.newCondition();

37.同步队列

等待队列

38.简单的数据连接池

39.线程间通信join,一个线程在执行过程中,调用另一个线程

join是thread的方法.加塞线程

40.synchronized是当先线程的锁,isalive是调用方法的对象的方法

41.countDownlatch维护了一个锁存器的计数,当计数为0时才会执行await以下的方法

42.cyclicBayier 所有线程都达到某个状态,后续动作才会执行,

每个线程需要等待,都需要执行cb的await方法,在创建cb对象时,两个参数,一个需要指定需要等待的线程个数,另一个需要定义,等待条件成立后,要执行的结果.

43.semaphore控制容量,限制通过的量,上公交车,只有人下去了,才能有人上去.



44.exchanger多个线程达到某个状态,进行数据交互

45.futuretask的构造方法有两个,一个(callAble),另一个(RunnAble,result),其实是属于一种,result是作为callable的泛型传入的,

Runnable线程是异步执行的,是在线程中执行的,callable是在futuretask中执行的

46.fork/join