# 基本概念

## 线程和进程的概念、

进程就是一个程序运行起来的状态，线程是一个进程中的不同的执行路径。专业：进程是OS分配资源的基本单位，线程是执行调度的基本单位。分配资源最重要的是：独立的内存空间，线程调度执行（线程共享进程的内存空间，没有自己独立的内存空间）

纤程：用户态的线程，线程中的线程，切换和调度不需要经过OS

优势：1：占有资源很少 OS : 线程1M Fiber：4K 2：切换比较简单 3：启动很多个10W+

目前2020 3 22支持内置纤程的语言：Kotlin Scala Go Python(lib)... Java? （open jdk : loom）

Java中对于纤程的支持：没有内置，盼望内置

纤程 vs 线程池：很短的计算任务，不需要和内核打交道，并发量高！

## 并行和并发的概念

1. 说说线程安全问题，什么是线程安全，如何保证线程安全
2. 线程的生命周期，状态是如何转移的
3. 线程状态以及API怎么操作会发生这种转换；
4. 创建线程的方式及实现
5. 进程间通信的方式
6. 一个线程连着调用start两次会出现什么情况？
7. （由于状态只有就绪、阻塞、执行，状态是无法由执行转化为执行的，所以会报不合法的状态！）
8. 如何调试多线程的程序；

# 线程池

1. 讲讲线程池的实现原理
2. 多线程是解决什么问题的？线程池解决什么问题？
3. 线程池的四种实现方式，重要参数及原理？任务拒接策略有哪几种？
4. 线程池，如何设计的，里面的参数有多少种，里面的工作队列和线程队列是怎样的结构，如果给你，怎样设计线程池？
5. Java线程池的核心属性以及处理流程；
6. Java线程池的几个参数的意义和实现机制；
7. Java线程池使用无界任务队列和有界任务队列的优劣对比；
8. 线程池，如何根据CPU的核数来设计线程大小，如果是计算机密集型的呢，如果是IO密集型的呢？
9. 线程池内的线程如果全部忙，提交⼀个新的任务，会发⽣什么？队列全部塞满了之后，还是忙，再提交会发⽣什么？
10. ExecutorService你一般是怎么⽤的？是每个Service放一个还是个项目放一个？有什么好处？
11. 从源码详细说下 Java 里面的线程池吧，使用线程池有什么要注意的地方？你们公司有规范吗？

# 三、Synchronized

1. synchronized 实现原理（对象监视器）
2. synchronized 与 lock 的区别
3. synchronized与ReentraLock哪个是公平锁；
4. synchronized关键字锁住的是什么东西？在字节码中是怎么表示的？在内存中的对象上表现为什么？
5. sync原理详细，sync内抛异常会怎样，死锁吗？还是释放掉？怎么排查死锁？死锁会怎样？有没有什么更好的替代方案？
6. 讲下 ReentrantLock 可重入锁？ 什么是可重入锁？为什么要设计可重入锁？Synchronized 和 RecentLock 有什么区别？这两个有没有深入了解源码
7. 讲下 Synchronized？
8. wait/notify/notifyAll⽅法需不需要被包含在synchronized块中？这是为什么？

# 四 Volatile

volatile 轻量级同步机制 ：保证可见性，不保证原子性，禁止指令重排

volatile关键字作用，他是怎样做到同步的？

volatile 实现原理

（禁止指令重排、刷新内存）

Volatile 为什么不支持原子性？举个例子

# 四 常见的原子操作类

1. AtomicInteger底层实现原理；
2. 说说 CountDownLatch、CyclicBarrier 原理和区别

Atomic 怎么设计的？有用过里面的类吗？

# 五、用过并发包下边的哪些类

ConcurrentHashMap 和 HashTable 中线程安全的区别？为啥建议用 ConcurrentHashMap ？能把 ConcurrentHashMap 里面的实现详细的讲下吗？

把所有认识熟用的JUC( java.util.concurrent(简称JUC)包)下的类写出来，讲下使用，然后讲下原生的线程操作;

ConcurrentHashMap jdk7使用分段锁，1.8以后使用cas

# 五 一些高级概念

重入锁的概念，重入锁为什么可以防止死锁

解释一下自旋;

说说 Semaphore 原理

说说 Exchanger 原理

ThreadLocal 原理分析，ThreadLocal为什么会出现OOM，出现的深层次原理

产生死锁的四个条件（互斥、请求与保持、不剥夺、循环等待）

如何检查死锁（通过jConsole检查死锁）

AQS同步队列

CAS无锁的概念、乐观锁和悲观锁

CAS机制会出现什么问题；

什么是ABA问题，出现ABA问题JDK是如何解决的

Java 8并法包下常见的并发类

偏向锁、轻量级锁、重量级锁、自旋锁的概念

wait方法能不能被重写，wait能不能被中断；

常用的避免死锁方法；

写出一个必然会产生死锁的伪代码；

Java中有哪些同步方案（重量级锁、显式锁、并发容器、并发同步器、CAS、volatile、AQS等）

多个线程同时读写，读线程的数量远远⼤于写线程，你认为应该如何解决并发的问题？你会选择加什么样的锁？

乐观悲观锁的设计，如何保证原子性，解决的问题；

乐观锁的业务场景及实现方式

AQS原理，ReentranLock源码，设计原理，整体过程。

非公平锁/公平锁 Lock lock = new ReentrantLock({true：公平锁；空或false：非公平锁})，默认非公平锁; 公平锁：先来后到，顺序执行，按照申请锁的顺序；非公平锁：可加塞，会造成优先级翻转或饥饿现象，吞吐量大，synchronized也是一种非公平锁

可重入锁（递归锁）ReentrantLock和Synchronized 线程能够进入任何它已经拥有的锁的所同步的代码块，（获取外层方法锁，可直接获取内层方法的锁）可避免死锁，lock与unlock必须配对

自旋锁：尝试获取锁的线程不会立即阻塞，而是采用循环的方式尝试获取锁，可以减少线程上下文的切换的消耗，不过会增加cpu的消耗

怎么认为一个类是线程安全？线程安全的定义是什么？Java有多少个关键字进行同步？为什么这样设计？（聊了一大堆，一堆为什么）；

两个线程设计题。记得一个是：t1,t2,t3，让t1，t2执行完才执行t3，原生实现。

说下多线程，我们什么时候需要分析线程数，怎么分析，分析什么因素;

解释一下信号量

保证线程安全的还有其他的方式吗？

线程安全类和线程不安全的类，项目使用的时候你会怎么选择？怎么判断项目代码哪里会有线程不安全问题？

cas 比较交换 自旋锁 unsafe

缺点:do while 循环开销大；只能保证一个共享数据；引发出ABA问题（版本号）

读写锁/互斥锁（独占锁：写锁；共享锁：读锁） reentrantLock和synchronized都是独占锁，只能同时被一个线程共享；reentrantReadWriteLock读写锁

countDownLatch 计数器 countdownlatch.countdown() 计数器减一;countdownlatch.await()主线程阻塞，等待计数器减为零。

CyclicBarrier new CyclicBarrier(上限屏障数，Runable接口) cyclicBarrier.await（）阻塞当前线程，待所有线程完成后，执行runable接口

Semaphore 信号量 semaphore.acquire()获得锁；semaphore.replace()释放锁；tryacquire（time，时间单位）尝试在时间内获取锁，若获取不到抛异常

|  |
| --- |
|  |

阻塞队列

ArrayBlockingQueue由有限数组组成的阻塞队列

LinkedBlockingQueue由有限（默认integer max）链表组成的阻塞队列

synchronousQueue：单元素队列

抛出异常 特殊值 阻塞 超时

插入 add（e） offer(e)true/false put(e) offer(e,time,unit)

移除 remove()删除队首 poll()队首 take() poll(time,unit)

检查 element()返回队首 peek()null 无 无

futureTask实现了runable接口，构造方法中需要传入callAble接口实现类，多个线程共用一个futureTask，只有一个线程执行runable

synchronized与lock的区别

1、原始构成

synchronized是关键字属于jvm层次，底层通过monitor对象来完成，wait和notify等方法也依赖monotor对象在同步块或方法中才能被调用monitorenter，monitorexist；

lock属于api层次的调用

2、使用方法

synchronized不需要手动释放锁，不会造成死锁

lock需手动释放，lock和unclock需配对使用

3、等待是否可中断

synchronized锁不可中断，只能正常结束或异常退出

renntranLock可设置超时退出tryLock(time,unit);lockInterruptibly()方法代码块中调用interrup()方法可中断

4、加锁是否公平

synchronized是非公平锁

lock可通过构造方法，true公平锁，false非公平锁，默认非公平锁

5、绑定多个条件condition

synchronized不支持，只能随机唤醒一个或所有

reentrantLock可分组唤醒，精确唤醒某个线程

非静态方法的锁默认是this，静态方法的锁是对应的Class实例

某一个时刻内，只能有一个线程持有锁，无论几个方法