

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

34 只求问倒：连环相扣系列锁面试题

更新时间：2019-11-13 11:49:45



“

自信和希望是青年的特权。

——大仲马

果断更，请联系QQ/微信6426006

面试中，问锁主要是两方面：锁的日常使用场景 + 锁原理，锁的日常使用场景主要考察对锁 API 的使用熟练度，看看你是否真的使用过这些 API，而不是纸上谈兵，锁原理主要就是问 AQS 底层的源码原理了，如果问得更加深入的话，可能会现场让你实现一个简单的锁，简单要求的会让你直接使用 AQS API，复杂要求的可能需要重新实现 AQS。

接下来我们一起看一看关于锁的常见源码面试题。

1 AQS 相关面试题

1.1 说说自己对 AQS 的理解？

答：回答这样的问题的时候，面试官主要考察的是你对 AQS 的知识有没有系统的整理，建议回答的方向是由大到小，由全到细，由使用到原理。

如果和面试官面对面的话，可以边说边画出我们在 AQS 源码解析上中画出的整体架构图，并且可以这么说：

1. AQS 是一个锁框架，它定义了锁的实现机制，并开放出扩展的地方，让子类去实现，比如我们在 lock 的时候，AQS 开放出 state 字段，让子类可以根据 state 字段来决定是否能够获得锁，对于获取不到锁的线程 AQS 会自动进行管理，无需子类锁关心，这就是 lock 时锁的内部机制，封装的很好，又暴露出子类锁需要扩展的地方；
2. AQS 底层是由同步队列 + 条件队列联手组成，同步队列管理着获取不到锁的线程的排队和释放，条件队列是在一定场景下，对同步队列的补充，比如获得锁的线程从空队列中拿数据，

目录

列的唤醒，分别对应着 AQS 架构图中的四种颜色的线的走向。

以上三点都是 AQS 全局方面的描述，接着你可以问问面试官要不要说细一点，可以的话，按照 AQS 源码解析上下两篇，把四大场景都说一下就好了。

这样说的好处是很多的：

1. 面试的主动权把握在自己手里，而且都是自己掌握的知识点；
2. 由全到细的把 AQS 全部说完，会给面试官一种你对 AQS 了如指掌的感觉，再加上全部说完耗时会很久，面试时间又很有限，面试官就不会再问关于 AQS 一些刁钻的问题了，这样 AQS 就可以轻松过关。

当然如果你对 AQS 了解的不是很深，那么就大概回答下 AQS 的大体架构就好了，就不要说的特别细，免得给自己挖坑。

1.2 多个线程通过锁请求共享资源，获取不到锁的线程怎么办？

答：加锁(排它锁)主要分为以下四步：

1. 尝试获得锁，获得锁了直接返回，获取不到锁的走到 2；
2. 用 Node 封装当前线程，追加到同步队列的队尾，追加到队尾时，又有两步，如 3 和 4；
3. 自旋 + CAS 保证前一个节点的状态置为 signal；
4. 阻塞自己，使当前线程进入等待状态。

获取不到锁的线程会进行 2、3、4 步，最终会陷入等待状态，这个描述的是排它锁。

1.3 问题 1.2 中，排它锁和共享锁的处理机制是一样的么？

答：排它锁和共享锁在问题 1.2 中的 2、3、4 步骤都是一样的，不同的是在于第一步，线程获得排它锁的时候，仅仅把自己设置为同步队列的头节点即可，但如果是共享锁的话，还会去唤醒自己的后续节点，一起来获得该锁。

1.4 共享锁和排它锁的区别？

答：排它锁的意思是同一时刻，只能有一个线程可以获得锁，也只能有一个线程可以释放锁。

共享锁可以允许多个线程获得同一个锁，并且可以设置获取锁的线程数量，共享锁之所以能够做到这些，是因为线程一旦获得共享锁，把自己设置成同步队列的头节点后，会自动的去释放头节点后等待获取共享锁的节点，让这些等待节点也一起来获得共享锁，而排它锁就不会这么干。

1.5 排它锁和共享锁说的是加锁时的策略，那么锁释放时有排它锁和共享锁的策略么？

答：是的，排它锁和共享锁，主要体现在加锁时，多个线程能否获得同一个锁。

但在锁释放时，是没有排它锁和共享锁的概念和策略的，概念仅仅针对锁获取。

1.6 描述下同步队列？

答：同步队列底层的数据结构就是双向的链表，节点叫做 Node，头节点叫做 head，尾节点叫做 tail，节点和节点间的前后指向分别叫做 prev、next，如果是面对面面试的话，可以画一下

同步队列的作用：阻塞获取不到锁的线程，开在适当时机释放这些线程。

实现的大致过程：当多个线程都来请求锁时，某一时刻有且只有一个线程能够获得锁（排它锁），那么剩余获取不到锁的线程，都会到同步队列中去排队并阻塞自己，当有线程主动释放锁时，就会从同步队列中头节点开始释放一个排队的线程，让线程重新去竞争锁。

1.7 描述下线程入、出同步队列的时机和过程？

答：(排它锁为例)从 AQS 整体架构图中，可以看出同步队列入队和出队都是有二个箭头指向，所以入队和出队的时机各有二个。

同步队列入队时机：

1. 多个线程请求锁，获取不到锁的线程需要到同步队列中排队阻塞；
2. 条件队列中的节点被唤醒，会从条件队列中转移到同步队列中来。

同步队列出队时机：

3. 锁释放时，头节点出队；
4. 获得锁的线程，进入条件队列时，会释放锁，同步队列头节点开始竞争锁。

四个时机的过程可以参考 AQS 源码解析，1 参考 acquire 方法执行过程，2 参考 signal 方法，3 参考 release 方法，4 参考 await 方法。

1.8 为什么 AQS 有了同步队列之后，还需要条件队列？

答：的确，一般情况下，我们只需要有同步队列就好了，但在上锁后，需要操作队列的场景下，一个同步队列就搞不定了，需要条件队列进行功能补充，比如当队列满时，执行 put 操作的线程会进入条件队列等待，当队列空时，执行 take 操作的线程也会进入条件队列中等待，从一定程度上来看，条件队列是对同步队列的场景功能补充。

1.9 描述一下条件队列中的元素入队和出队的时机和过程？

答：入队时机：执行 await 方法时，当前线程会释放锁，并进入到条件队列。

出队时机：执行 signal、signalAll 方法时，节点会从条件队列中转移到同步队列中。

具体的执行过程，可以参考源码解析中 await 和 signal 方法。

1.10 描述一下条件队列中的节点转移到同步队列中去的时机和过程？

答：时机：当有线程执行 signal、signalAll 方法时，从条件队列的头节点开始，转移到同步队列中去。

过程主要是以下几步：

1. 找到条件队列的头节点，头节点 next 属性置为 null，从条件队列中移除了；
2. 头节点追加到同步队列的队尾；
3. 头节点状态（waitStatus）从 CONDITION 修改成 0（初始化状态）；
4. 将节点的前一个节点状态置为 SIGNAL。

1.11 线程入条件队列时，为什么需要释放持有的锁？

果断更，请联系QQ/微信64260066

当前锁。

2 AQS 子类锁面试题

2.1 你在工作中如何使用锁的，写一个看一看？

答：这个照实说就好了，具体 demo 可以参考：demo.sixth.ConditionDemo。

2.1 如果我要自定义锁，大概的实现思路是什么样子的？

答：现在有很多类似的问题，比如让你自定义队列，自定义锁等等，面试官其实并不是想让我们重新造一个轮子，而是想考察一下我们对于队列、锁理解的深度，我们只需要选择自己最熟悉的 API 描述一下就好了，所以这题我们可以选择 ReentrantLock 来描述一下实现思路：

1. 新建内部类继承 AQS，并实现 AQS 的 tryAcquire 和 tryRelease 两个方法，在 tryAcquire 方法里面实现控制能否获取锁，比如当同步器状态 state 是 0 时，即可获得锁，在 tryRelease 方法里面控制能否释放锁，比如将同步器状态递减到 0 时，即可释放锁；
2. 对外提供 lock、release 两个方法，lock 表示获得锁的方法，底层调用 AQS 的 acquire 方法，release 表示释放锁的方法，底层调用 AQS 的 release 方法。

2.2 描述 ReentrantLock 两大特性：可重入性和公平性？底层分别如何实现的？

答：可重入性说的是线程可以对共享资源重复加锁，对应的，释放时也可以重复释放，对于 ReentrantLock 来说，在获得锁的时候，state 会加 1，重复获得锁时，不断的对 state 进行递增即可，比如目前 state 是 4，表示线程已经对共享资源加锁了 4 次，线程每次释放共享资源的锁时，state 就会递减 1，直到递减到 0 时，才算真正释放掉共享资源。

公平性和非公平指的是同步队列中的线程得到锁的机制，如果同步队列中的线程按照阻塞的顺序得到锁，我们称之为公平的，反之是非公平的，公平的底层实现是 ReentrantLock 的 tryAcquire 方法（调用的是 AQS 的 hasQueuedPredecessors 方法）里面实现的，要释放同步队列的节点时（或者获得锁时），判断当前线程节点是不是同步队列的头节点的后一个节点，如果是就释放，不是则不能释放，通过这种机制，保证同步队列中的线程得到锁时，是按照从头到尾的顺序的。

2.3 如果一个线程需要等待一组线程全部执行完之后再继续执行，有什么好的办法么？是如何实现的？

答：CountDownLatch 就提供了这样的机制，比如一组线程有 5 个，只需要在初始化 CountDownLatch 时，给同步器的 state 赋值为 5，主线程执行 CountDownLatch.await，子线程都执行 CountDownLatch.countDown 即可。

2.4 Atomic 原子操作类可以保证线程安全，如果操作的对象是自定义的类的话，要如何做呢？

答：Java 为这种情况提供了一个 API：AtomicReference，AtomicReference 类可操作的对象是个泛型，所以支持自定义类。

3 总结

果断更，请联系QQ/微信642600606

目录

都弄清楚了，回答 AQS 的各种问题都会游刃有余。

← 33 CountDownLatch、Atomic 等其它源码解析

35 经验总结：各种锁在工作中使用场景和细节 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示



目前暂无任何讨论

千学不如一看，千看不如一练

果断更，请联系QQ/微信6426006