

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

23 队列在源码方面的面试题

更新时间：2019-10-22 10:29:12



“ 人要有毅力，否则将一事无成。 ”  
——居里夫人

果断更，请联系QQ/微信64260066

队列在源码方面的面试题，一般面试官会从锁，线程池等知识点作为问题入口，慢慢的问到队列，由于锁、线程池咱们还没有学习到，所以本章就直奔主题，从队列入手，看看队列都有哪些面试题（队列种类很多，本文在说队列的通用特征时，都是在说其大部分队列的通用特征，如有某种队列特征不符，不在一一说明）。

1 面试题

1.1 说说你对队列的理解，队列和集合的区别。

答：对队列的理解：

1. 首先队列本身也是个容器，底层也会有不同的数据结构，比如 `LinkedBlockingQueue` 是底层是链表结构，所以可以维持先入先出的顺序，比如 `DelayQueue` 底层可以是队列或堆栈，所以可以保证先入先出，或者先入后出的顺序等等，底层的数据结构不同，也造成了操作实现不同；
2. 部分队列（比如 `LinkedBlockingQueue`）提供了暂时存储的功能，我们可以往队列里面放数据，同时也可以从队列里面拿数据，两者可以同时进行；
3. 队列把生产数据的一方和消费数据的一方进行解耦，生产者只管生产，消费者只管消费，两者之间没有必然联系，队列就像生产者和消费者之间的数据通道一样，如 `LinkedBlockingQueue`；
4. 队列还可以对消费者和生产者进行管理，比如队列满了，有生产者还在不停投递数据时，队列可以使生产者阻塞住，让其不再能投递，比如队列空时，有消费者过来拿数据时，队列可

|    |   |
|----|---|
| 目录 | 5. 队列还提供阻塞的功能，比如我们从队列拿数据，但队列中没有数据时，线程会一直阻塞到队列有数据可拿时才返回。 |
|----|---|

队列和集合的区别：

1. 和集合的相同点，队列（部分例外）和集合都提供了数据存储的功能，底层的存储数据结构是有些相似的，比如说 `LinkedBlockingQueue` 和 `LinkedHashMap` 底层都使用的是链表，`ArrayBlockingQueue` 和 `ArrayList` 底层使用的都是数组。

2. 和集合的区别：

2.1 部分队列和部分集合底层的存储结构很相似的，但两者为了完成不同的事情，提供的 API 和其底层的操作实现是不同的。

2.2 队列提供了阻塞的功能，能对消费者和生产者进行简单的管理，队列空时，会阻塞消费者，有其他线程进行 `put` 操作后，会唤醒阻塞的消费者，让消费者拿数据进行消费，队列满时亦然。

2.3 解耦了生产者和消费者，队列就像是生产者和消费者之间的管道一样，生产者只管往里面丢，消费者只管不断消费，两者之间互不关心。

1.2 哪些队列具有阻塞的功能，大概是如何阻塞的？

答：队列主要提供了两种阻塞功能，如下：

1. `LinkedBlockingQueue` 链表阻塞队列和 `ArrayBlockingQueue` 数组阻塞队列是一类，前者容量是 `Integer` 的最大值，后者数组大小固定，两个阻塞队列都可以指定容量大小，当队列满时，如果有线程 `put` 数据，线程会阻塞住，直到有其他线程进行消费数据后，才会唤醒阻塞线程继续 `put`，当队列空时，如果有线程 `take` 数据，线程会阻塞到队列不空时，继续 `take`。
2. `SynchronousQueue` 同步队列，当线程 `put` 时，必须有对应线程把数据消费掉，`put` 线程才能返回，当线程 `take` 时，需要有对应线程进行 `put` 数据时，`take` 才能返回，反之则阻塞，举个例子，线程 A `put` 数据 A1 到队列中了，此时并没有任何的消费者，线程 A 就无法返回，会阻塞住，直到有线程消费掉数据 A1 时，线程 A 才能返回。

1.3 底层是如何实现阻塞的？

答：队列本身并没有实现阻塞的功能，而是利用 `Condition` 的等待唤醒机制，阻塞底层实现就是更改线程的状态为沉睡，细节我们在锁小节会说到。

1.4 `LinkedBlockingQueue` 和 `ArrayBlockingQueue` 有啥区别。

答：相同点：

1. 两者的阻塞机制大体相同，比如在队列满、空时，线程都会阻塞住。

不同点：

1. `LinkedBlockingQueue` 底层是链表结构，容量默认是 `Integer` 的最大值，`ArrayBlockingQueue` 底层是数组，容量必须在初始化时指定。
2. 两者的底层结构不同，所以 `take`、`put`、`remove` 的底层实现也就不同。

答：是线程安全的，在 put 之前，队列会自动加锁，put 完成之后，锁会自动释放，保证了同一时刻只会有一条线程能操作队列的数据，以 LinkedBlockingQueue 为例子，put 时，会加 put 锁，并只对队尾 tail 进行操作，take 时，会加 take 锁，并只对队头 head 进行操作，remove 时，会同时加 put 和 take 锁，所以各种操作都是线程安全的，我们工作中可以放心使用。

### 1.6 take 的时候也会加锁么？既然 put 和 take 都会加锁，是不是同一时间只能运行其中一个方法。

答：1：是的，take 时也会加锁的，像 LinkedBlockingQueue 在执行 take 方法时，在拿数据的同时，会把当前数据删除掉，就改变了链表的数据结构，所以需要加锁来保证线程安全。

2：这个要看情况而言，对于 LinkedBlockingQueue 来说，队列的 put 和 take 都会加锁，但两者的锁是不一样的，所以两者互不影响，可以同时进行的，对于 ArrayBlockingQueue 而言，put 和 take 是同一个锁，所以同一时刻只能运行一个方法。

### 1.7 工作中经常使用队列的 put、take 方法有什么危害，如何避免。

答：当队列满时，使用 put 方法，会一直阻塞到队列不满为止。

当队列空时，使用 take 方法，会一直阻塞到队列有数据为止。

两个方法都是无限（永远、没有超时时间的意思）阻塞的方法，容易使得线程全部都阻塞住，大流量时，导致机器无线程可用，所以建议在流量大时，使用 offer 和 poll 方法来代替两者，我们只需要设置好超时阻塞时间，这两个方法如果在超时时间外，还没有得到数据的话，就会返回默认值（LinkedBlockingQueue 为例），这样就不会导致流量大时，所有的线程都阻塞住了。

这个也是生产事故常常发生的原因之一，尝试用 put 和 take 方法，在平时自测中根本无法发现，对源码不熟悉的同学也不会意识到会有问题，当线上大流量打进来时，很有可能会发生故障，所以我们平时工作中使用队列时，需要谨慎再谨慎。

### 1.8 把数据放入队列中后，有木有办法让队列过一会儿再执行？

答：可以的，DelayQueue 提供了这种机制，可以设置一段时间之后再执行，该队列有个唯一的缺点，就是数据保存在内存中，在重启和断电的时候，数据容易丢失，所以定时的时间我们都不会设置很久，一般都是几秒内，如果定时的时间需要设置很久的话，可以考虑采取延迟队列中间件（这种中间件对数据会进行持久化，不怕断电的发生）进行实现。

### 1.9 DelayQueue 对元素有什么要求么，我把 String 放到队列中去可以么？

答：DelayQueue 要求元素必须实现 Delayed 接口，Delayed 本身又实现了 Comparable 接口，Delayed 接口的作用是定义还剩下多久就会超时，给使用者定制超时时间的，Comparable 接口主要用于对元素之间的超时时间进行排序的，两者结合，就可以让越快过期的元素能够排在前面。

所以把 String 放到 DelayQueue 中是不行的，编译都无法通过，DelayQueue 类在定义的时候，是有泛型定义的，泛型类型必须是 Delayed 接口的子类才行。

### 1.10 DelayQueue 如何让快过期的元素先执行的？

|    |  |
|----|--|
| 目录 | 实现过期时间和当前时间的差，这样越快过期的元素，计算出来的差值就会越小，就会越先被执行。 |
|----|--|

1.11 如何查看 SynchronousQueue 队列的大小？

答：此题是个陷阱题，题目首先设定了 SynchronousQueue 是可以查看大小的，实际上 SynchronousQueue 本身是没有容量的，所以也无法查看其容量的大小，其内部的 size 方法都是写死的返回 0。

1.12 SynchronousQueue 底层有几种数据结构，两者有何不同？

答：底层有两种数据结构，分别是队列和堆栈。

两者不同点：

1. 队列维护了先入先出的顺序，所以最先进去队列的元素会最先被消费，我们称为公平的，而堆栈则是先入后出的顺序，最先进入堆栈中的数据可能会最后才会被消费，我们称为不公平的。
2. 两者的数据结构不同，导致其 take 和 put 方法有所差别，具体的可以看《SynchronousQueue 源码解析》章节。

1.13 假设 SynchronousQueue 底层使用的是堆栈，线程 1 执行 take 操作阻塞住了，然后有线程 2 执行 put 操作，问此时线程 2 是如何把 put 的数据传递给 take 的？

答：这是一个好问题，也是理解 SynchronousQueue 的核心问题。

首先线程 1 被阻塞住，此时堆栈头就是线程 1 了，此时线程 2 执行 put 操作，会把 put 的数据赋值给堆栈头的 match 属性，并唤醒线程 1，线程 1 被唤醒后，拿到堆栈头中的 match 属性，就能够拿到 put 的数据了。

严格上说并不是 put 操作直接把数据传递给了 take，而是 put 操作改变了堆栈头的数据，从而 take 可以从堆栈头上直接拿到数据，堆栈头是 take 和 put 操作之间的沟通媒介。

1.14 如果想使用固定大小的队列，有几种队列可以选择，有何不同？

答：可以使用 LinkedBlockingQueue 和 ArrayBlockingQueue 两种队列。

前者是链表，后者是数组，链表新增时，只要建立起新增数据和链尾数据之间的关联即可，数组新增时，需要考虑到索引的位置（takeIndex 和 putIndex 分别记录着下次拿数据、放数据的索引位置），如果增加到了数组最后一个位置，下次就要重头开始新增。

1.15 ArrayBlockingQueue 可以动态扩容么？用到数组最后一个位置时怎么办？

答：不可以的，虽然 ArrayBlockingQueue 底层是数组，但不能够动态扩容的。

假设 put 操作用到了数组的最后一个位置，那么下次 put 就需要从数组 0 的位置重新开始了。

假设 take 操作用到数组的最后一个位置，那么下次 take 的时候也会从数组 0 的位置重新开始。

目录

答：ArrayBlockingQueue 有两个属性，为 takeIndex 和 putIndex，分别标识下次 take 和 put 的位置，每次 take 和 put 完成之后，都会往后加一，虽然底层是数组，但和 HashMap 不同，并不是通过 hash 算法计算得到的。

2 总结

队列是锁、线程池等复杂 API 的基础，很多面试官都会在问这些 API 时冷不防的问你队列的知识，如果你回答不好，面试官可能会认为你仅仅是用过锁和线程池，但却对其底层的原理和实现了解的不够全面，所以说队列还是蛮重要的，但队列的源码比较复杂，建议大家可以尝试 debug 的方式来理解源码。

← 22 ArrayBlockingQueue 源码解析

24 举一反三：队列在 Java 其它源码中的应用 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

慕码人6169125

老师，1.1的第一条是不是打错了哦？DelayQueue内部的q是PriorityQueue对象，PriorityQueue的源码好像底层是数组哦。SynchronousQueue底层是两种数据结构

👍 0 回复

2019-10-24

文贺 回复 慕码人6169125

谢谢，你是对的，正在修改中。

回复

2019-10-24 20:42:06

千学不如一看，千看不如一练