32 | 时间轮：Kafka是如何实现定时任务的？

2022-03-08 黄清昊

《业务开发算法50讲》 课程介绍

讲述：黄清昊

时长 12:06 大小 11.10M

你好，我是微扰君。

今天我们来聊一聊日常开发中非常常见的技术需求：延时队列。

之前在学 Kafka 二分搜索的时候，我们已经学过了消息队列，它是一个用于传递消息的组件，大部分场景下，我们都希望消息尽快送达，并且消息之间要严格遵循先进先出的约束。但在有一些时候，我们也会希望消息不要立刻送达，而是在一段时间之后才会被接收方收到，也就是延后消息被处理的时间，像这样的场景就是“延时队列”。

常见业务场景

延时队列的应用非常多。我们回想一下有哪些业务场景，比如一个网上售卖电影票的平台，用户在买票的时候肯定要先选好位置，也就是说用户下单一张电影票有两个动作：选位置、付费。

我们不希望用户在付费的时候发现，自己选好的位置被别人买了，所以往往会在用户选定座位之后，就把这个位置锁定；但这个时候用户还没有付费，我们肯定不能让锁定一直持续下去，所以也会想要有一种定时机制，在用户超过一定时间没有付费时，在系统中自动取消这个订单，将对应的座位释放。

类似的场景还有许多，对应需要的定时周期跨度也很大。比如在云平台上如果用户资源过期，一般不会立刻清理所有数据，而会在超过一段时间之后再进行资源回收；再比如外卖平台上订单，如果快超过送餐时限了，就需要提醒外卖小哥加紧配送等等。

在这些业务场景下，一个好用的延时队列应该具备什么样的功能呢：**我们只需要把任务和期望的执行时间存储到队列中，等到指定的时候，任务消息就会通过队列被发送给需要执行任务的主体，比如某个订单服务，让主体执行**。

当然，我们也可以把队列直接实现在业务里，但是延时特性和具体业务无关，其实是一个完全通用的技术方案，所以一般会用通用的中间件来处理这样的问题。

Kafka 就是一个非常好用的选择，作为一款高性能的消息队列，Kafka 天然支持了延时消息的能力，可以帮助我们处理所有的延时场景下的问题。其实上一讲我们介绍的 Redis 中的 ZSET，也是一种实现延时队列的常见手段。

不过在学习基于 ZSET 的实现之前，我们先从更简单的实现学起，边学边思考这些常见实现的场景和原理差异是什么。

JDK 中的 DelayedQueue

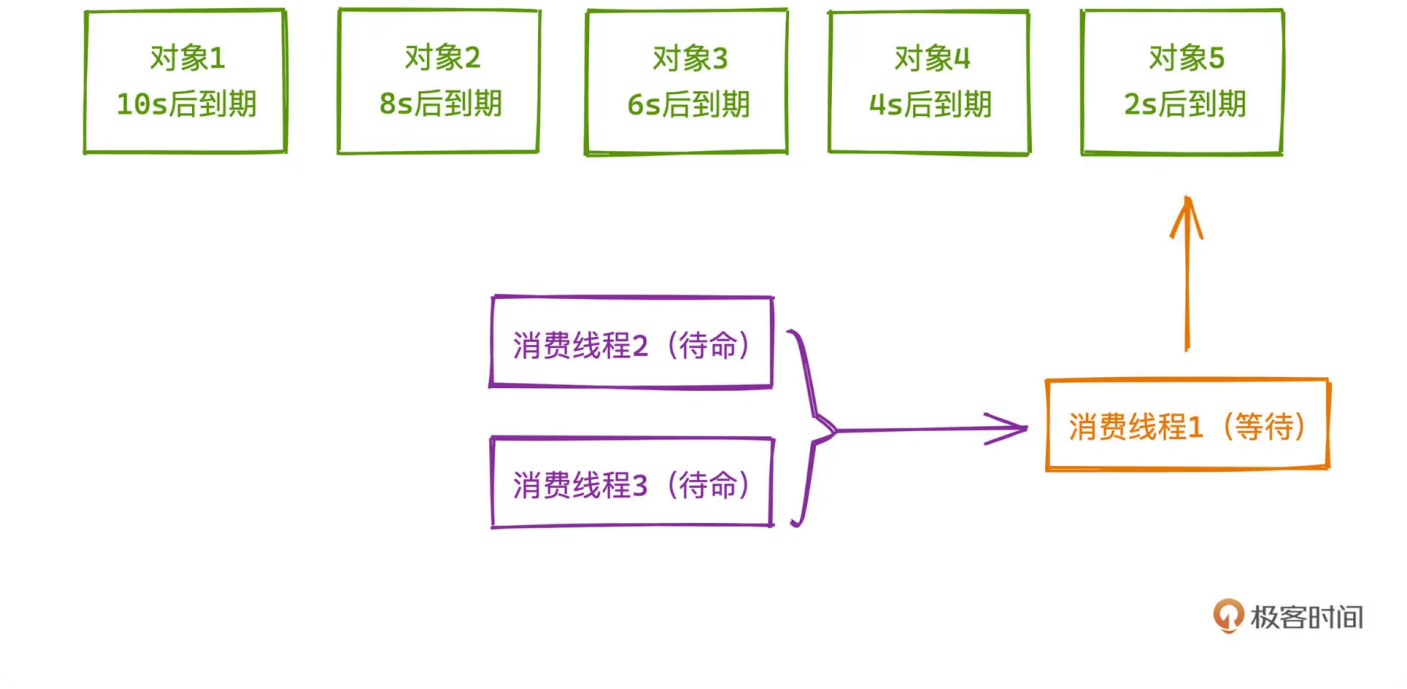
除了上面说的业务场景，在一些纯技术的领域，定时任务的需求也非常普遍，比如 Linux 下就支持了定时任务调度的功能。如果你熟悉 Java，估计会想到 JDK 也默认支持了 DelayedQueue 的数据结构。

DelayedQueue，作为 JDK 原生支持的数据结构，能非常方便地帮助我们支持单机、数据规模不大的延时队列的场景。它的实现思路也是一种非常典型的延时队列实现思路，事实上也经常是面试官常考的八股文之一，值得我们好好掌握。

DelayedQueue 实现延时队列的本质，是在内存中维护一个有序的数据结构，按任务应该被执行的时间来排序。对外提供了 offer 和 take 两个主要的接口，分别用于从队列中插入元素和

请求元素。

- 在插入元素时，既然是所谓的延时队列，我们会插入一个带执行时间的任务，底层会对这些任务进行排序，保证队列最前的任务是最快到期的；
- 调用 take 接口后，会有一个线程检查头部元素，如果队列头的任务没有到期，我们就阻塞这个线程，直到任务到期，再唤醒这个线程；如果检查头部的时候任务已经到期，我们就会让这个消费进程真的从队列取出该元素，并执行。



为了提高效率，DelayedQueue 底层还采用了一种 leader-follower 的线程模型，也非常常用，你可以理解成任务的执行会有多个线程进行，参考示意图，这样任务的具体执行和到期时间的检查就不会产生冲突，可以并行地进行。

分析清楚了设计思路，那 DelayedQueue 底层是如何对任务做有序排列的，用的是什么数据结构呢？你可以先猜想一下。

链表？数组？事实上，这里的有序排列并不会像你想的那样从头到尾维护一个线性的序列。我们之前也讲过，如果维护一个线性的序列，不管是链表还是数组，排序的时候都需要 O(n*logn) 的时间复杂度；而在这里**我们所需要的其实只是判断整个队列中，最接近到期的那**

个任务的执行时间，是否已经被当前系统时间所超过。也就是说并不需要整个队列有序，只需要最值。

这不正是堆这个数据结构的长处嘛？所以 DelayedQueue 底层的存储结构正是堆。

由于整个数据结构都维护在内存上，也没有线性扩展性，空间上会受一定的制约，但从时间效率上来说，DelayedQueue 还是一个非常不错的延时队列实现，特别适合在业务层面上直接解决一些规模不大、比较简单的延时队列场景。具体的代码可以直接在 JDK 中找到，感兴趣你可以自己研究。

学完 DelayedQueue， 我们再来看 Redis 中的 ZSET 是如何实现延时队列的，对比理解。

Redis 中的 ZSET

底层基于跳表（上节课讲过），Redis 的有序集合性能非常不错，而且 Redis 本身是一个稳定、性能良好且能支持大量数据的 KV 存储引擎，用来实现延时队列自然比基于 DelayedQueue 的本地实现适用场景更大。

借助 ZSET 来实现延时队列，本质思想和 DelayedQueue 是类似的，主要就是我们会用 ZSET 来维护按任务执行时间排列的数据结构。

在使用 ZSET 做延时队列的时候，一般会用任务 ID 作为 key，任务详情作为 value，任务执行时间作为 score，这样所有的待执行任务，在 ZSET 中，就会按任务执行时间 score 有序排列。

极客时间

在需要被调度的延时任务执行主体上，我们可以开启一个线程定时轮询 `ZRANGEBYSCORE KEY -inf +inf limit 0 1 WITHSCORES` 查询 ZSET 中最近可执行的任务：

- 如果发现任务时间戳仍然大于当前时间戳，说明没有任务过期，什么都不执行；
- 如果发现任务时间戳已经小于当前时间戳，说明任务已经可以执行，我们按照约定的协议执行就可以了。

当然，这里 Redis 里存储的任务详情其实就是个值，我们需要按照自己的场景序列化和反序列化。写成 Java 代码大概如下：

复制代码

```
    public void poll() {
        while (true) {
            Set<Tuple> set = jedis.zrangeWithScores(DELAY_QUEUE, 0, 0);
            String value = ((Tuple) set.toArray()[0]).getElement();
            int score = (int) ((Tuple) set.toArray()[0]).getScore();

            Calendar cal = Calendar.getInstance();
            int nowSecond = (int) (cal.getTimeInMillis() / 1000);
            // 任务已经过期；可以执行
            if (nowSecond >= score) {
                jedis.zrem(DELAY_QUEUE, value);
                // TODO：执行任务
            }
```

```
            // 队列为空
            if (jedis.zcard(DELAY_QUEUE) <= 0) {
                return;
            }
            Thread.sleep(1000);
        }
    }
```

不过在这种方案中我们需要主动轮询，这会带来一定的开销，也有一定的精度问题，毕竟最小的粒度就是轮训的时间间隔。

既然引入了精度的问题，那我们有没有什么更好的方式呢，尤其是在有大量超时任务的场景下，有什么办法可以进一步优化超时任务的调度呢？

时间轮

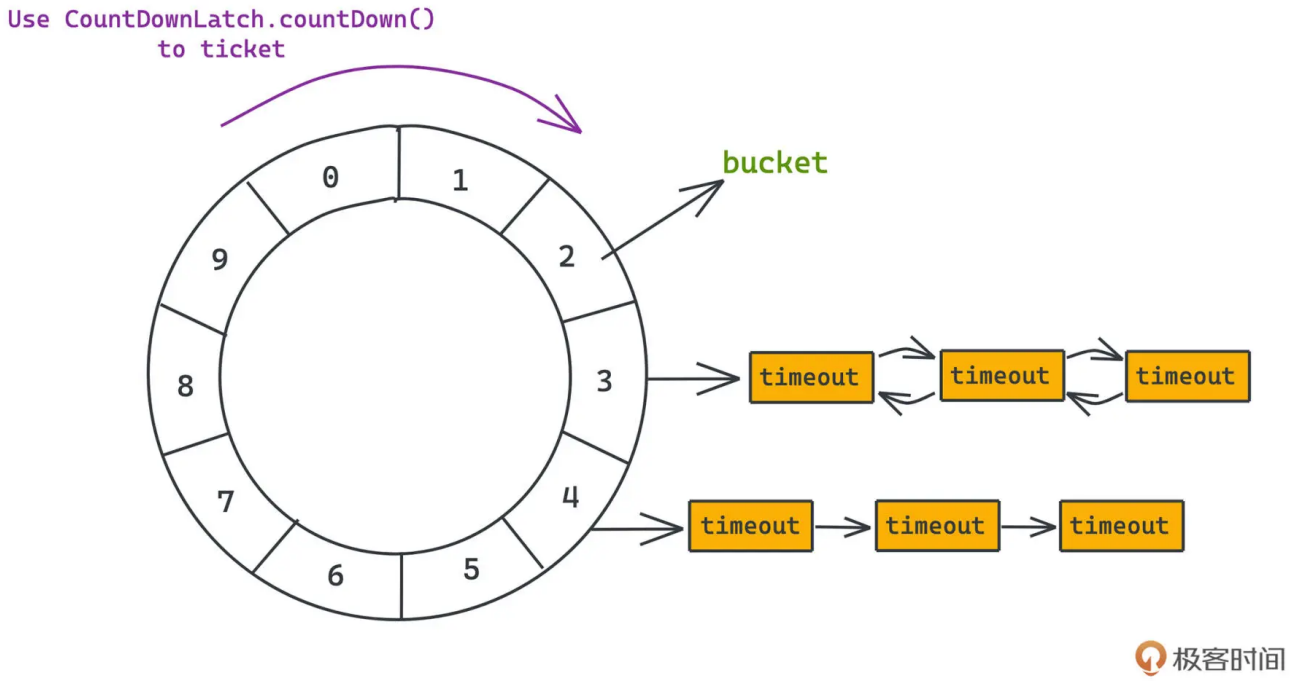
这就是时间轮算法的用武之地了。在 Kafka、Netty、ZooKeeper 等知名组件中都有用到时间轮算法，可以说是久经考验。

思路其实很简单，如果用排序来类比的话，刚才 JDK 中基于堆的实现当然就是堆排序，永远可以拿到最快要过期的任务；那为了维护有序性，我们是不是也可以用类似桶排序的思想呢？

这正是时间轮的本质。

- 把任务按时间分成不同的槽（bucket），每个槽位里放着任务的列表，通常采用一个双链表来实现；
- 把槽位加在一起，构成一个循环队列，底层用数组实现；
- 一个槽代表一个时间跨度，每个槽内队列中存储的任务就都是在这个跨度内应该被执行的任务。

这样整个时间轮看起来就像一个时钟。



我们还会有一个类似于秒针的指针，以槽位时间跨度为周期固定地转动，就像秒针一样，永远指向当前时间所应该对应的槽位，当然在这里精度不是秒，而是槽位的时间跨度。比如我们期待任务调度的精度是一分钟，在图中就可以让每个槽位代表的时间为 60s，**这样我们就可以很确定的在时间轮里表示 600s 也就是 10 分钟内的任务，每隔 1 分钟，就将当前的槽位指针 +1，指向下一个槽位，并判断槽位中是否有任务需要执行**。

槽位编号更小的任务，自然就会得到更先的执行，从而就实现了在某个精度下定时任务或者延时任务的需求。

这里的精度也可以调整，时间轮的整个时间周期除以刻度数量，就是我们最小的任务调度的精度，在不同的场景下，可以设计不同的时间轮刻度。比如以 24 小时、以秒，甚至毫秒为刻度都是可以的，当然精度越高，我们所需要的成本也就更高。

对比思考之前的实现，**时间轮方案很大的提升就在于，我们大大减少了任务插入和取出时的锁竞争**。相比于只维护一个堆，让所有的线程并发修改，在时间轮中，我们可以将锁的粒度减少到以刻度为单位，大大减少了锁冲突的可能性，取出任务时也只要从槽位中直接遍历，避免了从堆或者其他有序结构中取出元素和调整的开销。

当然这里还有个问题需要处理。比如在 600s 的时间轮中，我们不难发现 601s 的任务和 9s 的任务在同一个时间轮的槽位里，因为 601s 已经超过 600s 了，由于循环队列的特性，它会又

一次被加入到第一个槽中。

不过这个问题也很好处理，只需要多判断一次当前时间和槽位中时间的关系就行，如果发现是 601s 或者更后期的任务，直接跳过即可。我们可以类比 Hashmap 冲突的情况，相信你很容易想明白其中的思想，无非就是遍历链表进行判断。

总结

今天我们一起学习了延时队列的底层实现方式和应用场景。

JDK 中的 DelayedQueue，以及借助 Redis 中 ZSET 的实现方式，两者总体思路比较相似，都是通过某种数据结构，来维护按任务执行时间排列的任务集合，然后定时或者轮询地去判断最接近过期的任务是否已经过期，选择执行或者继续等待。

当然单机的 JDK 可以更好地利用系统内置的定时机制，避免轮询的成本，不过也因为单机本身的限制，不能很好的扩展来支持海量的数据场景。

第三种实现方式，时间轮，是一个巧妙又高效的设计。牺牲了一定精度，但通过在内存中以循环队列的方式维护任务，降低了任务并行插入的锁竞争，也减少了取出任务的时间复杂度，特别适用于大量定时任务存在的场景，也因此成为 Kafka 实现延时队列的一种常用方式。

总体来说，这几种方式各有利弊，你可以好好体会一下其中的差异，结合自己的业务场景做一些选型的思考。

课后作业

今天的课后作业是时间轮的实现，整体思路不难，不考虑并发的场景下 100 行左右的代码就可以完成了，这也是面试官常考的题目之一，值得好好练习。

欢迎你在评论区留下你的代码作业，一起讨论。如果觉得这篇文章对你有帮助的话，也欢迎转发给你的朋友一起学习，我们下节课见~

生成海报并分享

赞 2　提建议

上一篇　31 | 跳表：Redis是如何存储有序集合的？

下一篇　33 | 限流算法：如何防止系统过载？

精选留言 (1)

写留言

peter
2022-03-08

请教老师几个问题：

Q1：延时队列和定时任务的区别是什么？

延时队列可以看作是一种特殊的定时任务吗？

Q2：时间轮算法：

A JDK是否实现了时间轮？

B SpringBoot应用是否可以直接利用时间轮？比如引入一个包就可以用。

Q3：时间轮的图中，槽位“3”的队列是双向箭头，槽位“4”对应的队列用了单向箭头，为什么？

Q4：ZSET部分，redis是<K:V>对，是两个值，怎么处理id:value:score三个值的情况呢？

Q5：ZSET只存储任务ID，不存储具体任务，即redis中并没有具体任务信息。消费端需要维护“任务id --- 具体任务”映射关系，根据ID找到具体任务执行，是这样吗？