38 如何选择适合自己的阻塞队列?

本课时我们主要讲解如何选择适合自己的阻塞队列。

他山之石，可以攻玉。对于如何选择最合适的阻塞队列这个问题，实际上线程池已经率先给我们做了表率。线程池有很多种，不同种类的线程池会根据自己的特点，来选择适合自己的阻塞队列。

所以我们就首先来复习一下这些非常经典的线程池是如何挑选阻塞队列的，借鉴它们的经验之后，我们再去总结一套规则，来归纳出自己在选取阻塞队列时可以对哪些点进行考虑。

线程池对于阻塞队列的选择

FixedThreadPool	LinkedBlockingQueue
SingleThreadExecutor	LinkedBlockingQueue
CachedThreadPool	SynchronousQueue
ScheduledThreadPool	DelayedWorkQueue
SingleThreadScheduledExecutor	DelayedWorkQueue

下面我们来看线程池的选择要诀。上面表格左侧是线程池，右侧为它们对应的阻塞队列，你可以看到 5 种线程池只对应了 3 种阻塞队列，下面我们对它们进行逐一的介绍。

- **FixedThreadPool（SingleThreadExecutor 同理）选取的是 LinkedBlockingQueue**

因为 LinkedBlockingQueue 不同于 ArrayBlockingQueue，ArrayBlockingQueue 的容量是有限的，而 LinkedBlockingQueue 是链表长度默认是可以无限延长的。

由于 FixedThreadPool 的线程数是固定的，在任务激增的时候，它无法增加更多的线程来帮忙处理 Task，所以需要像 LinkedBlockingQueue 这样没有容量上限的 Queue 来存储那些还没处理的 Task。

如果所有的 corePoolSize 线程都正在忙，那么新任务将会进入阻塞队列等待，由于队列是没有容量上限的，队列永远不会被填满，这样就保证了对于线程池 FixedThreadPool 和 SingleThreadExecutor 而言，不会拒绝新任务的提交，也不会丢失数据。

- **CachedThreadPool 选取的是 SynchronousQueue**

对于 CachedThreadPool 而言，为了避免新提交的任务被拒绝，它选择了无限制的 maximumPoolSize（在专栏中，maxPoolSize 等同于 maximumPoolSize），所以既然它的线程的最大数量是无限的，也就意味着它的线程数不会受到限制，那么它就不需要一个额外的空间来存储那些 Task，因为每个任务都可以通过新建线程来处理。

SynchronousQueue 会直接把任务交给线程，而不需要另外保存它们，效率更高，所以 CachedThreadPool 使用的 Queue 是 SynchronousQueue。

- **ScheduledThreadPool（SingleThreadScheduledExecutor同理）选取的是延迟队列**

对于 ScheduledThreadPool 而言，它使用的是 DelayedWorkQueue。延迟队列的特点是：不是先进先出，而是会按照延迟时间的长短来排序，下一个即将执行的任务会排到队列的最前面。

我们来举个例子：例如我们往这个队列中，放一个延迟 10 分钟执行的任务，然后再放一个延迟 10 秒钟执行的任务。通常而言，如果不是延迟队列，那么按照先进先出的排列规则，也就是延迟 10 分钟执行的那个任务是第一个放置的，会放在最前面。但是由于我们此时使用的是阻塞队列，阻塞队列在排放各个任务的位置的时候，会根据延迟时间的长短来排放。所以，我们第二个放置的延迟 10 秒钟执行的那个任务，反而会排在延迟 10 分钟的任务的前面，因为它的执行时间更早。

我们选择使用延迟队列的原因是，ScheduledThreadPool 处理的是基于时间而执行的 Task，而延迟队列有能力把 Task 按照执行时间的先后进行排序，这正是我们所需要的功能。

ArrayBlockingQueue

除了线程池选择的 3 种阻塞队列外，还有一种常用的阻塞队列叫作 ArrayBlockingQueue，它也经常被用于我们手动创建的线程池中。

这种阻塞队列内部是用数组实现的，在新建对象的时候要求传入容量值，且后期不能扩容，

所以 ArrayBlockingQueue的最大特点就是容量是有限且固定的。这样一来，使用 ArrayBlockingQueue 且设置了合理大小的最大线程数的线程池，在任务队列放满了以后，如果线程数也已经达到了最大值，那么线程池根据规则就会拒绝新提交的任务，而不会无限增加任务或者线程数导致内存不足，可以非常有效地防止资源耗尽的情况发生。

归纳

下面让我们总结一下经验，通常我们可以从以下 5 个角度考虑，来选择合适的阻塞队列：

- 功能

第 1 个需要考虑的就是功能层面，比如是否需要阻塞队列帮我们排序，如优先级排序、延迟执行等。如果有这个需要，我们就必须选择类似于 PriorityBlockingQueue 之类的有排序能力的阻塞队列。

- 容量

第 2 个需要考虑的是容量，或者说是否有存储的要求，还是只需要“直接传递”。在考虑这一点的时候，我们知道前面介绍的那几种阻塞队列，有的是容量固定的，如 ArrayBlockingQueue；有的默认是容量无限的，如 LinkedBlockingQueue；而有的里面没有任何容量，如 SynchronousQueue；而对于 DelayQueue 而言，它的容量固定就是 Integer.MAX_VALUE。

所以不同阻塞队列的容量是千差万别的，我们需要根据任务数量来推算出合适的容量，从而去选取合适的 BlockingQueue。

- 能否扩容

第 3 个需要考虑的是能否扩容。因为有时我们并不能在初始的时候很好的准确估计队列的大小，因为业务可能有高峰期、低谷期。

如果一开始就固定一个容量，可能无法应对所有的情况，也是不合适的，有可能需要动态扩容。如果我们需要动态扩容的话，那么就不能选择 ArrayBlockingQueue ，因为它的容量在创建时就确定了，无法扩容。相反，PriorityBlockingQueue 即使在指定了初始容量之后，后续如果有需要，也可以自动扩容。

所以我们可以根据是否需要扩容来选取合适的队列。

- 内存结构

第 4 个需要考虑的点就是内存结构。在上一课时我们分析过 ArrayBlockingQueue 的源码，

看到了它的内部结构是“数组”的形式。

和它不同的是，LinkedBlockingQueue 的内部是用链表实现的，所以这里就需要我们考虑到，ArrayBlockingQueue 没有链表所需要的“节点”，空间利用率更高。所以如果我们对性能有要求可以从内存的结构角度去考虑这个问题。

- 性能

第 5 点就是从性能的角度去考虑。比如 LinkedBlockingQueue 由于拥有两把锁，它的操作粒度更细，在并发程度高的时候，相对于只有一把锁的 ArrayBlockingQueue 性能会更好。

另外，SynchronousQueue 性能往往优于其他实现，因为它只需要“直接传递”，而不需要存储的过程。如果我们的场景需要直接传递的话，可以优先考虑 SynchronousQueue。

在本课时，我们首先回顾了线程池对于阻塞队列的选取规则，然后又看到了 ArrayBlockingQueue 的特点，接下来我们总结归纳了通常情况下，可以从功能、容量、能否扩容、内存结构和性能这 5 个角度考虑问题，结合业务选取最适合我们的阻塞队列。

上一页 下一页