目录

39 经验总结：不同场景，如何使用线程池

更新时间：2019-11-21 10:55:44

> 人的影响短暂而微弱，书的影响则广泛而深远。
>
> ——普希金

引导语

ThreadPoolExecutor 初始化时，主要有如下几个参数：

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler) {
```

大家对这几个参数应该都很熟悉了，虽然参数很少，但实际工作中却有很多门道，大多数的问题主要集中在线程大小的设置，队列大小的设置两方面上，接下来我们一起看看工作中，如何初始化 ThreadPoolExecutor。

1 coreSize == maxSize

我相信很多人都看过，或自己写过这样的代码：

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(10, 10, 600000L, TimeUnit.DAYS,
                                new LinkedBlockingQueue());
```

这行代码主要展示了在初始化 ThreadPoolExecutor 的时候，coreSize 和 maxSize 是相等的，这样设置的话，随着请求的不断增加，会是这样的现象：

3. 队列满时，此时因为 coreSize 和 maxSize 相等，任务会被直接拒绝。

这么写的最大目的：是想让线程一下子增加到 maxSize，并且不要回收线程，防止线程回收，避免不断增加回收的损耗，一般来说业务流量都有波峰低谷，在流量低谷时，线程不会被回收；流量波峰时，maxSize 的线程可以应对波峰，不需要慢慢初始化到 maxSize 的过程。

这样设置有两个前提条件：

1. allowCoreThreadTimeOut 我们采取默认 false，而不会主动设置成 true，allowCoreThreadTimeOut 是 false 的话，当线程空闲时，就不会回收核心线程；
2. keepAliveTime 和 TimeUnit 我们都会设置很大，这样线程空闲的时间就很长，线程就不会轻易的被回收。

我们现在机器的资源都是很充足的，我们不用去担心线程空闲会浪费机器的资源，所以这种写法目前是很常见的。

2 maxSize 无界 + SynchronousQueue

在线程池选择队列时，我们也会看到有同学选择 SynchronousQueue，SynchronousQueue 我们在 《SynchronousQueue 源码解析》章节有说过，其内部有堆栈和队列两种形式，默认是堆栈的形式，其内部是没有存储的容器的，放元素和拿元素是一一对应的，比如我使用 put 方法放元素，如果此时没有对应的 take 操作的话，put 操作就会阻塞，需要有线程过来执行 take 操作后，put 操作才会返回。

基于此特点，如果要使用 SynchronousQueue 的话，我们需要尽量将 maxSize 设置大一点，这样就可以接受更多的请求。

假设我们设置 maxSize 是 10 的话，选择 SynchronousQueue 队列，假设所有请求都执行 put 操作，没有请求执行 take 操作，前 10 个 put 请求会消耗 10 个线程，都阻塞在 put 操作上，第 11 个请求过来后，请求就会被拒绝，所以我们才说尽量把 maxSize 设置大一点，防止请求被拒绝。

maxSize 无界 + SynchronousQueue 这样的组合方式优缺点都很明显：

优点：当任务被消费时，才会返回，这样请求就能够知道当前请求是已经在被消费了，如果是其他的队列的话，我们只知道任务已经被提交成功了，但无法知道当前任务是在被消费中，还是正在队列中堆积。

缺点：

1. 比较消耗资源，大量请求到来时，我们会新建大量的线程来处理请求；
2. 如果请求的量难以预估的话，maxSize 的大小也很难设置。

3 maxSize 有界 + Queue 无界

在一些对实时性要求不大，但流量忽高忽低的场景下，可以使用 maxSize 有界 + Queue 无界的组合方式。

比如我们设置 maxSize 为 20，Queue 选择默认构造器的 LinkedBlockingQueue，这样做的优缺点如下：

1. 电脑 cpu 固定的情况下，每秒能同时工作的线程数是有限的，此时开很多的线程其实也是浪费，还不如把这些请求放到队列中去等待，这样可以减少线程之间的 CPU 的竞争；
2. LinkedBlockingQueue 默认构造器构造出来的链表的最大容量是 Integer 的最大值，非常适合流量忽高忽低的场景，当流量高峰时，大量的请求被阻塞在队列中，让有限的线程可以慢慢消费。

缺点：流量高峰时，大量的请求被阻塞在队列中，对于请求的实时性难以保证，所以当对请求的实时性要求较高的场景，不能使用该组合。

4 maxSize 有界 + Queue 有界

这种组合是对 3 缺点的补充，我们把队列从无界修改成有界，只要排队的任务在要求的时间内，能够完成任务即可。

这种组合需要我们把线程和队列的大小进行配合计算，保证大多数请求都可以在要求的时间内，有响应返回。

5 keepAliveTime 设置无穷大

有些场景下我们不想让空闲的线程被回收，于是就把 keepAliveTime 设置成 0，实际上这种设置是错误的，当我们把 keepAliveTime 设置成 0 时，线程使用 poll 方法在队列上进行超时阻塞时，会立马返回 null，也就是空闲线程会立马被回收。

所以如果我们想要空闲的线程不被回收，我们可以设置 keepAliveTime 为无穷大值，并且设置 TimeUnit 为时间的大单位，比如我们设置 keepAliveTime 为 365，TimeUnit 为 TimeUnit.DAYS，意思是线程空闲 1 年内都不会被回收。

在实际的工作中，机器的内存一般都够大，我们合理设置 maxSize 后，即使线程空闲，我们也不希望线程被回收，我们常常也会设置 keepAliveTime 为无穷大。

6 线程池的公用和独立

在实际工作中，某一个业务下的所有场景，我们都不会公用一个线程池，一般有以下几个原则：

1. 查询和写入不公用线程池，互联网应用一般来说，查询量远远大于写入的量，如果查询和写入都要走线程池的话，我们一定不要公用线程池，也就是说查询走查询的线程池，写入走写入的线程池，如果公用的话，当查询量很大时，写入的请求可能会到队列中去排队，无法及时被处理；
2. 多个写入业务场景看情况是否需要公用线程池，原则上来说，每个业务场景都独自使用自己的线程池，绝不共用，这样在业务治理、限流、熔断方面都比较容易，一旦多个业务场景公用线程池，可能就会造成业务场景之间的互相影响，现在的机器内存都很大，每个写入业务场景独立使用自己的线程池也是比较合理的；
3. 多个查询业务场景是可以公用线程池的，查询的请求一般来说有几个特点：查询的场景多、rt 时间短、查询的量比较大，如果给每个查询场景都弄一个单独的线程池的话，第一个比较耗资源，第二个很难定义线程池中线程和队列的大小，比较复杂，所以多个相似的查询业务场景是可以公用线程池的。

7 如何算线程大小和队列大小

目录

1. 根据业务进行考虑，初始化线程池时，我们需要考虑所有业务涉及的线程池，如果目前所有的业务同时都有很大流量，那么在对于当前业务设置线程池时，我们尽量把线程大小、队列大小都设置小，如果所有业务基本上都不会同时有流量，那么就可以稍微设置大一点；
2. 根据业务的实时性要求，如果实时性要求高的话，我们把队列设置小一点，coreSize == maxSize，并且设置 maxSize 大一点，如果实时性要求低的话，就可以把队列设置大一点。

假设现在机器上某一时间段只会运行一种业务，业务的实时性要求较高，每个请求的平均 rt 是 200ms，请求超时时间是 2000ms，机器是 4 核 CPU，内存 16G，一台机器的 qps 是 100，这时候我们可以模拟一下如何设置：

1. 4 核 CPU，假设 CPU 能够跑满，每个请求的 rt 是 200ms，就是 200 ms 能执行 4 条请求，2000ms 内能执行 2000/200 * 4 = 40 条请求；
2. 200 ms 能执行 4 条请求，实际上 4 核 CPU 的性能远远高于这个，我们可以拍脑袋加 10 条，也就是说 2000ms 内预估能够执行 50 条；
3. 一台机器的 qps 是 100，此时我们计算一台机器 2 秒内最多处理 50 条请求，所以此时如果不进行 rt 优化的话，我们需要加至少一台机器。

线程池可以大概这么设置：

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(15, 15, 365L, TimeUnit.DAYS,
                                    new LinkedBlockingQueue(35));
```


线程数最大为 15，队列最大为 35，这样机器差不多可以在 2000ms 内处理最大的请求 50 条，当然根据你机器的性能和实时性要求，你可以调整线程数和队列的大小占比，只要总和小于 50 即可。

以上只是很粗糙的设置，在实际的工作中，还需要根据实际情况不断的观察和调整。

总结

线程池设置非常重要，我们尽量少用 Executors 类提供的各种初始化线程池的方法，多根据业务的量，实时性要求来计算机器的预估承载能力，设置预估的线程和队列大小，并且根据实时请求不断的调整线程池的大小值。

← 38 线程池源码面试题

40 打动面试官：线程池流程编排中的运用实战 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

八卦一百二十八掌

开发中一般是自定义线程池呢？还是使用JDK提供的几种线程池呢？

0 回复 2019-11-21

目录

回复 2019-11-23 16:37:55

千学不如一看，千看不如一练