853769620

1、面试题

常用的线程池有哪些？不同线程池的使用场景是什么？说一下线程池配置的几个参数？线程池队列满的时候怎么处理（FixedThreadPool满了之后会怎么办）？线程池启动线程的原理（说说线程池的底层原理）？线程池被关闭的方式有哪几种？

2、面试官心里分析

如果是对一个20多k的工程师，一般并发这块第一个问题就是聊一下线程池，因为非常常用，上一讲的简单问题基本都是默认你会的，但是如果是20k以内的同学，上一讲的问题应该也会抽几个问问。

而且线程池这块，就是会跟上面那样来几个连环炮问问的，从一般怎么用，到参数设置，再到工作原理，实际上这块很多人都是模模糊糊知道一点儿，但是能说清楚的人很少

3、面试题剖析

**3.1 线程池的最最基本的原理**

线程池，说白了就是为了避免频繁创建和销毁线程带来的巨大开销，就维护一定数量的线程池，创建以后别销毁，用完了扔回去再个下一个任务来使用，这样就可以提高线程的使用效率。一般常见于后台处理类的系统，或者是复杂接口中的多数据并发获取。

池子，里面比如说固定就是100个线程，后面的人过来了就是从线程池里获取线程来工作，线程干完活儿了，别销毁，直接还到线程池里去，避免频繁的创建和销毁线程，会导致系统的运行效率很低

Java线程池包含4个部分

（1）线程池管理器（ThreadPool）：就是负责创建和销毁线程池的

（2）工作线程（PoolWorker）：就是线程池中的一个线程

（3）工作任务（Task）：这个就是线程池里的某个线程需要执行的业务代码，这个是你自己编写的业务逻辑

（4）任务队列（TaskQueue）：这个是扔到线程池里的任务需要进行排队，要进任务队列

**3.2 常用的几种线程池和API**

常用的线程池有：

（1）SingleThreadExecutor（单线程线程池，很少用 -> 自己做一个内存队列 -> 启动后台线程去消费）

（2）FixedThreadExecutor（固定数量线程池）：比如说，线程池里面固定就100个线程，超过这个线程数就到队列里面去排队等待

（3）CachedThreadExecutor（自动回收空闲线程，根据需要自动新增线程，传说中的无界线程池）：无论有多少任务，根据你的需要，无限制的创建任意多的线程，在最短的时间内来满足你，但是高峰过去之后，如果有大量的线程处于空闲状态，没有活儿可以干，等待60s之后空闲的线程就被销毁了

（4）ScheduledThreadExecutor（线程数量无限制，支持定时调度执行某个线程）：提交一个任务，对于这个任务不是立马执行的，是可以设定一个定时调度的逻辑，比如说每隔60s执行一次，这个一般不用，一般来说就用spring schedule的支持

一般其实最常用的是FixedThreadExecutor和CachedThreadExecutor

ScheduleThreadExecutor，也可能会使用，但是就是除非你的那个线程任务要定时调度，才会用这个线程池，不过说实话，简单的定时调度一般就是走spring的schedule支持就行了，当然如果你要用这个也行

Java的线程池比较重要的几个API

（1）Executor：代表线程池的接口，有个execute()方法，扔进去一个Runnable类型对象，就可以分配一个线程给你执行

（2）ExecutorService：这是Executor的子接口，相当于是一个线程池的接口，有销毁线程池等方法 -> ExecutorService就代表了一个线程池管理器，会负责管理线程池 -> 线程的创建和销毁 -> 队列排队

（3）Executors：线程池的辅助工具类，辅助入口类，可以通过Executors来快捷的创建你需要的线程池。创建线程池的入口类，包含newSingleThreadExecutor()、newCachedThreadPool()、newScheduleThreadPool()、newFixedThreadPool()，这些方法，就是可以让你创建不同的线程池出来

（4）ThreadPoolExecutor：这是ExecutorService的实现类，这才是正儿八经代表一个线程池的类，一般在Executors里创建线程池的时候，内部都是直接创建一个ThreadPoolExecutor的实例对象返回的，然后同时给设置了各种默认参数。

如果我们要创建一个线程池，两种方式，要么就是Executors.newXX()方法，快捷的创建一个线程池出来，线程池的所有参数设置都采取默认的方式；要么是自己手动构建一个THreadPoolExecutor的一个对象，所有的线程池的参数，都可以自己手动来调整和设置

public class Executors {

public static ExecutorService newFixedThreadPool(int nThreads) {

return new ThreadPoolExecutor(nThreads, nThreads,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

}

public static ExecutorService newCachedThreadPool() {

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

}

如果我们要用线程池，大部分同学，我估计如果之前使用过线程池的话，一遍都是怎么用的呢？

public class MyTask implements Runnable {

public void run() {

// 实现你的业务逻辑

// 这段业务逻辑，就会交给线程池里的某个线程去执行

}

}

// 创建一个固定线程数量为100的FixedThreadPool线程池

ExecutorService myThreadPool = Executors.newFixedThreadPool(100);

// 要往线程池里提交一个任务

Runnable myTask = new MyTask();

myThreadPool.execute(myTask);

// 上面的代码就是平时大家最最常用的线程池的使用代码

// 执行了线程池的execute()方法，就相当等于是给这个线程提交了一个任务

// 线程池会优先用有已有的线程来处理这个任务

// 但是如果所有的线程池里的线程都处于一个繁忙的状态，此时就会将这个任务扔到队列里去排队，等待某个线程空闲之后来处理这个任务

到此为止，都是讲的很easy的，但是你会发现只是知道这么点是完全不够用的

**3.3 线程池的构造参数和真正的工作原理**

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

ThreadPoolExecutor就是线程池，那么这个类的构造函数的所有入参，就是你可以设置的参数，我们来解释一下这些参数吧

corePoolSize：线程池里的核心线程数量

maximumPoolSize：线程池里允许有的最大线程数量

keepAliveTime：如果线程数量大于corePoolSize的时候，多出来的线程会等待指定的时间之后就被释放掉，这个就是用来设置空闲线程等待时间的

unit：这个是上面那个keepAliveTime的单位

workQueue：这个是说，通过ThreadPoolExecutor.execute()方法扔进来的Runnable工作任务，会进入一个队列里面去排队，这就是那个队列

threadFactory：如果需要创建新的线程放入线程池的时候，就是通过这个线程工厂来创建的

handler：假如说上面那个workQueue是有固定大小的，如果往队列里扔的任务数量超过了队列大小，咋办？就用这个handler来处理，AbortPolicy、DiscardPolicy、DiscardOldestPolicy，如果说线程都繁忙，队列还满了，此时就会报错，RejectException

这些参数的含义先解释一下：

假设我们自己手动创建一个ThreadPoolExecutor线程池，设置了以下的一些参数

corePoolSize：2个

mamximumPoolSize：4个

keepAliveTime：60s

workQueue：ArrayBlockingQueue，有界阻塞队列，队列大小是4

handler：默认的策略，抛出来一个ThreadPoolRejectException

（1）一开始线程池里的线程是空的，一个都没有。有一个变量维护的是当前线程数量，这个变量是poolSize，poolSize = 0，如果当前线程的数量小于corePoolSize（2），poolSize < corePoolSize，那么来了一个任务优先创建线程，直到线程池里的线程数量跟corePoolSize一样；poolSize = 1，poolSize < corePoolSize（2），又创建一个线程来处理这个任务；poolSize = 2

（2）如果当前线程池的线程数量（poolSize = 2）大于等于corePoolSize（2）的时候，而且任务队列没满（最大大小是4，但是当前元素数量是0），那么就扔到任务队列里去

（3）如果当前线程池的线程数量大于等于corePoolSize的时候，而且任务队列满了（最大大小是4，当前已经放了4个元素了，已经满了），那么如果当前线程数量小于最大线程数（poolSize = 2，maimumPoolSize = 4，poolSize < maximumPoolSize），就继续创建线程；poolSize = 3，提交了一个任务，poolSize >= corePoolSize，任务队列满，poolSize < maximumPoolSize，再次创建一个任务

（4）如果此时poolSize >= corePoolSize，任务队列满，poolSize == maximumPoolSize，此时再次提交一个任务，当前线程数已经达到了最大线程数了，那么就使用handler来处理，默认是抛出异常，ThreadPoolRejectExeception

（5）此时线程池里有4个线程，都处于空闲状态，corePoolSize指定的是就2个线程就可以了，但是此时超过了corePoolSize 2个线程，所以如果那超出的2个线程空闲时间超过了60s，然后线程池就会将超出的2个线程给回收掉

如何设置池的这些参数？先来看看创建线程池的默认代码

其实上面都说过了，啥时候会创建新线程？其实就是线程数没到corePoolSize的时候，会创建线程；接着就是任务队列满了，但是线程数小于maximumPoolSize的时候，也会创建线程；创建的时候通过threadFactory来创建即可

**3.4 常用线程池的工作原理**

FixedThreadPool

public static ExecutorService newFixedThreadPool(int nThreads) {

return new ThreadPoolExecutor(nThreads, nThreads,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

}

（1）corePoolSize = 100，maximumPoolSize = 100，keepAliveTime = 0，workQueue = 无界队列

（2）刚开始的时候，比如说假设一开始线程池里没有线程，你就不断的提交任务，瞬间提交了100个任务，一下子创建100个线程出来，其实poolSize == corePoolSize，再提交任务，直接就会发现LinkedBlockQueue根本就没有大小的限制，所以说根本就不会满，所以此时后续的所有任务直接扔到LinkedBlockingQueue里面去排队

（3）100个线程，只要出现了空闲，就会从队列里面去获取任务来处理，以此类推，就100个线程，不停的处理任务

1. LinkedBlockingQueue根本就不会满，直到扔任务扔的内存溢出，扔了几百万个任务，几千万个任务，队列实在是太大太大了，导致内存溢出，满了，就死了
2. maximumPoolSize根本就没用，而且其实也是直接跟corePoolSize设置成一样
3. keepAliveTime = 0，也就是，创建出来的线程，根本就不会去判断是否超过了指定的空闲时间，不会去回收空闲线程数量，后面就维护这么固定数量的一个线程，有任务就往队列里面怼，固定数量的比如100个线程不停的处理任务，如果100个线程都处理不过来，那么就无限制的往LinkedBlockingQueue里面去排队，直到内存溢出

public static ExecutorService newCachedThreadPool() {

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

1. corePoolSize = 0，maximumPoolSize = 最大值（无限大），keepAliveTime = 60s，workQueue=SynchronousQueue
2. SynchronousQueue（实际上没有存储数据的空闲，是用来做多线程通信之间的协调作用的），一开始提交一个任务过来，要求线程池里必须有一个线程对应可以处理这个任务，但是此时一个线程都没有，poolSize >= corePoolSize , workQueue已经满了，poolSize < maximumPoolSize（最大值），直接就会创建一个新的线程来处理这个任务
3. 如果短期内有大量的任务都涌进来，实际上是走一个直接提交的思路，对每个任务，如果没法找到一个空闲的线程来处理它，那么就会立即创建一个新的线程出来，来处理这个新提交的任务
4. 短时间内，如果大量的任务涌入，可能会导致瞬间创建出来几百个线程，几千个线程，是不固定的
5. 但是当这些线程工作完一段时间之后，就会处于空闲状态，就会看超过60s的空闲，就会直接将空闲的线程给释放掉

总结一下：

（1）FixedThreadPool：线程数量是固定的，如果所有线程都繁忙，后续任务全部在一个无界队列里面排队，无限任务来排队，直到内存溢出

（2）CachcedThreadPool：线程数量是不固定的，如果一下子涌入大量任务，没有空闲线程，那么就创建新的线程来处理；如果有空闲线程，就是一个任务配对一个空闲线程来处理；如果线程空闲时间超过60s，就给回收掉空闲线程

**3.5 各种线程池在什么样的场景下使用**

FixedThreadPool：比较适用于什么场景呢？负载比较重，而且负载比较稳定的这么一个场景，我给大家来举个例子，我们之前线上有一套系统，负载比较重，后台系统，每分钟要执行几百个复杂的大SQL，就是用FixedThreadPool是最合适的。

因为负载稳定，所以一般来说，不会出现说突然瞬间涌入大量的请求，100个线程处理不过来，然后就直接无限制的排队，然后oom内存溢出，死了

CachedThreadPool：负载很稳定的场景，用CachedThreadPool就浪费了；每天大部分时候可能就是负载很低的，CachedThreadPool，用少量的线程就可以满足低负载，不会给系统引入太大的压力；但是每天如果有少量的高峰期，比如说中午或者是晚上，高峰期可能需要一下子搞几百个线程出来，那么CachedThreadPool就可以满足这个场景；高峰期应付过去之后，线程如果处于空闲状态超过60s，自动回收空闲线程，避免给系统带来过大的负载

**3.6 如何设置线程池的参数**

这是Executors自己创建的，其实无非就是实例化一个ThreadPoolExecutor对象，你要是不想用Executors创建，就自己构造一个ThreadPoolExecutor对象也行，那么参数你就自己设置不就得了

通常来说，建议大家就用Executors提供的默认的线程池就可以了，我觉得还是挺合适的，因为他们的默认的参数设置可以满足大部分的场景了，但是如果你在学习了这一课之后，确实需要自己去动手定制一下线程池的策略，那么就自己动手构建ThreadPoolExecutor实例就可以了，所有的参数自己设置

先来看看默认的参数设置，其实固定大小的线程池，说白了，就是corePoolSize和maximumPooSize是一样的，那么只要达到了你传入的那个线程数量，而且任务队列满了，就报错；cached线程池，说白了，就是可以无限的创建线程，因为maximumPooSize是无限大的，但是超过60秒空闲就给你回收

如果你要自己创建个线程池，一般自己设置参数好了：

corePoolSize：这个其实就是说你算一下每个任务要耗费多少时间，比如一个任务大概100ms，那么每个线程每秒可以处理10个任务，然后你算算你每秒总共要处理多少个任务啊，比如说200个任务，那么你就需要20个线程，每个线程每秒处理10个任务，不就可以处理200个任务。但是一般都会多设置一些，比如你可以设置个30个线程。

坦白来讲，如果你面试被问到这个问题，体现你水平的地方，其实在于不同场景的理解：

你希望用类似于FixedThreadPool的这个线程池，corePoolSize和maximumPoolSize按照上面说的策略设置成一样的就可以了

如果用的是FixedPool的话，一般在于workQueue和handler的理解，因为你看下默认的实现，其实线程数量达到corePoolSize的时候，就会放入workQueue排队，但是默认使用的是无界队列，LinkedBlockingQueue，所以会无限制往里面排队，然后就是你corePooSize指定数量的线程不断的处理，队列里的任务可能会无限制的增加

这个其实就是适合处理那种长期不断有大量任务进来，长期负载都很重，所以你不能用CachedPool，否则长期让机器运行大量线程，可能导致机器死掉，cpu耗尽。所以你就只能控制线程的数量，用有限的线程不断的处理源源不断进入的任务，有时高峰时任务较多，就做一下排队即可。

所以FixedPool的参数里，对于workQueue，你要考虑一点，默认的是无界队列，可能会有问题，就是要是无限排队，别把机器给搞死了，那么这个时候你可以换成ArrayBlockingQueue，就是有界队列，自己设置一个最大长度，一旦超出了最大长度，就通过handler去处理，你可以自己对handler接口实现自己的逻辑，我给你举个例子，此时你可以把数据放到比如数据库里去，做离线存储或者是什么的

需要去实现CachedThreadPool的这么一个策略

corePoolSize可以设置为0，但是maximumPoolSize考虑不用设置成无限大，有一个风险，假设突然进来的流量高峰，导致你的线程池一下子出来了几万个线程，瞬间会打满cpu负载，直接机器会死掉

maximumPoolSize可以设置成一个，你的机器cpu能负载的最大的线程数，一个经验值，4核8G的虚拟机，你线程池启动的线程数量达到100个就差不多了，如果同时有100个线程，而且做很频繁的操作，cpu可能就快到70%，80%，90%

corePoolSize = 0，maximumPoolSize = 150 -> handler报错 -> 实现一个handler，将多余的线程给离线存储起来，后续高峰过了，再重新扫描出来重新提交

你看下CachedPool，他那里用的是SynchronousQueue，这个queue的意思是如果要插入一个任务，必须有一个任务已经被消费掉了，所以很可能出现说，线程数量达到corePoolSize之后，大量的任务进来，此时SynchronousQueue里的任务如果还没被拿走，那么就会认为队列满了，此时就会创建新的线程，但是maximumPoolSize是无限大的，所以会无限制的创建新的线程。但是如果后续有线程空闲了，那么就会被回收掉。

所以如果你用CachedPool，相当于是在高峰期，无限制的创建线程来拼命耗尽你的机器资源来处理并发涌入的大量的任务

所以CachedPool，可以用在那种瞬时并发任务较高，但是每个任务耗时都较短的场景，就是短时间内突然来个小高峰，那么就快速启动大量线程来处理，但是每个线程处理都很快，而且高峰很快就过去了

**3.7 线程池关闭原理**

shutdown()：调用之后不允许提交新的任务了，所有调用之前提交的任务都会执行，等所有任务执行完了，才会真正关闭线程池，这就是优雅的关闭方式

shutdownNow()：返回还没执行的task列表，然后不让等待的task执行，尝试停止正在执行的task，非优雅关闭，强制关闭