## 线程池之 ThreadPoolExecutor + 面试题

### 线程池介绍

线程池（Thread Pool）：把一个或多个线程通过统一的方式进行调度和重复使用的技术，避免了因为线程过多而带来使用上的开销。

#### 为什么要使用线程池？

* 可重复使用已有线程，避免对象创建、消亡和过度切换的性能开销。
* 避免创建大量同类线程所导致的资源过度竞争和内存溢出的问题。
* 支持更多功能，比如延迟任务线程池（newScheduledThreadPool）和缓存线程池（newCachedThreadPool）等。

### 线程池使用

创建线程池有两种方式：ThreadPoolExecutor 和 Executors，其中 Executors 又可以创建 6 种不同的线程池类型，会在下节讲，本节重点来看看 ThreadPoolExecutor 的使用。

#### ThreadPoolExecutor 的使用

线程池使用代码如下：

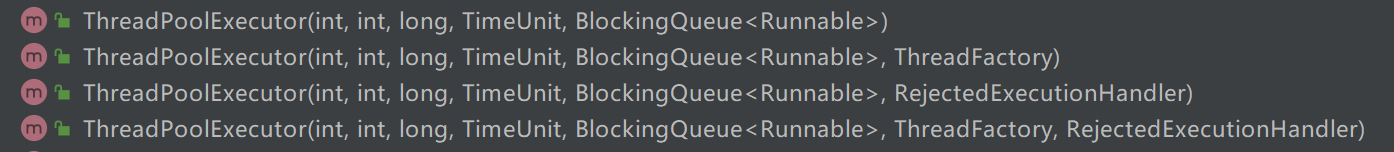
ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(2, 10, 10L, TimeUnit.SECONDS, new LinkedBlockingQueue(100));  
threadPoolExecutor.execute(new Runnable() {  
 @Override  
 public void run() {  
 // 执行线程池  
 System.out.println("Hello, Java.");  
 }  
});

以上程序执行结果如下：

Hello, Java.

#### ThreadPoolExecutor 参数说明

ThreadPoolExecutor 构造方法有以下四个，如下图所示：



enter image description here

其中最后一个构造方法有 7 个构造参数，包含了前三个方法的构造参数，这 7 个参数名称如下所示：

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

其代表的含义如下：

##### ① corePoolSize

线程池中的核心线程数，默认情况下核心线程一直存活在线程池中，如果将 ThreadPoolExecutor 的 allowCoreThreadTimeOut 属性设为 true，如果线程池一直闲置并超过了 keepAliveTime 所指定的时间，核心线程就会被终止。

##### ② maximumPoolSize

线程池中最大线程数，如果活动的线程达到这个数值以后，后续的新任务将会被阻塞（放入任务队列）。

##### ③ keepAliveTime

线程池的闲置超时时间，默认情况下对非核心线程生效，如果闲置时间超过这个时间，非核心线程就会被回收。如果 ThreadPoolExecutor 的 allowCoreThreadTimeOut 设为 true 的时候，核心线程如果超过闲置时长也会被回收。

##### ④ unit

配合 keepAliveTime 使用，用来标识 keepAliveTime 的时间单位。

##### ⑤ workQueue

线程池中的任务队列，使用 execute() 或 submit() 方法提交的任务都会存储在此队列中。

##### ⑥ threadFactory

为线程池提供创建新线程的线程工厂。

##### ⑦ rejectedExecutionHandler

线程池任务队列超过最大值之后的拒绝策略，RejectedExecutionHandler 是一个接口，里面只有一个 rejectedExecution 方法，可在此方法内添加任务超出最大值的事件处理。ThreadPoolExecutor 也提供了 4 种默认的拒绝策略：

* new ThreadPoolExecutor.DiscardPolicy()：丢弃掉该任务，不进行处理
* new ThreadPoolExecutor.DiscardOldestPolicy()：丢弃队列里最近的一个任务，并执行当前任务
* new ThreadPoolExecutor.AbortPolicy()：直接抛出 RejectedExecutionException 异常
* new ThreadPoolExecutor.CallerRunsPolicy()：既不抛弃任务也不抛出异常，直接使用主线程来执行此任务

包含所有参数的 ThreadPoolExecutor 使用代码：

public class ThreadPoolExecutorTest {  
 public static void main(String[] args) throws InterruptedException, ExecutionException {  
 ThreadPoolExecutor threadPool = new ThreadPoolExecutor(1, 1,  
 10L, TimeUnit.SECONDS, new LinkedBlockingQueue<Runnable>(2),  
 new MyThreadFactory(), new ThreadPoolExecutor.CallerRunsPolicy());  
 threadPool.allowCoreThreadTimeOut(true);  
 for (int i = 0; i < 10; i++) {  
 threadPool.execute(new Runnable() {  
 @Override  
 public void run() {  
 System.out.println(Thread.currentThread().getName());  
 try {  
 Thread.sleep(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 }  
 }  
}  
class MyThreadFactory implements ThreadFactory {  
 private AtomicInteger count = new AtomicInteger(0);  
 @Override  
 public Thread newThread(Runnable r) {  
 Thread t = new Thread(r);  
 String threadName = "MyThread" + count.addAndGet(1);  
 t.setName(threadName);  
 return t;  
 }  
}

#### 线程池执行方法 execute() VS submit()

execute() 和 submit() 都是用来执行线程池的，区别在于 submit() 方法可以接收线程池执行的返回值。

下面分别来看两个方法的具体使用和区别：

// 创建线程池  
ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(2, 10, 10L, TimeUnit.SECONDS, new LinkedBlockingQueue(100));  
// execute 使用  
threadPoolExecutor.execute(new Runnable() {  
 @Override  
 public void run() {  
 System.out.println("Hello, Java.");  
 }  
});  
// submit 使用  
Future<String> future = threadPoolExecutor.submit(new Callable<String>() {  
 @Override  
 public String call() throws Exception {  
 System.out.println("Hello, 老王.");  
 return "Success";  
 }  
});  
System.out.println(future.get());

以上程序执行结果如下：

Hello, Java.

Hello, 老王.

Success

#### 线程池关闭

线程池关闭，可以使用 shutdown() 或 shutdownNow() 方法，它们的区别是：

* shutdown()：不会立即终止线程池，而是要等所有任务队列中的任务都执行完后才会终止。执行完 shutdown 方法之后，线程池就不会再接受新任务了。
* shutdownNow()：执行该方法，线程池的状态立刻变成 STOP 状态，并试图停止所有正在执行的线程，不再处理还在池队列中等待的任务，执行此方法会返回未执行的任务。

下面用代码来模拟 shutdown() 之后，给线程池添加任务，代码如下：

threadPoolExecutor.execute(() -> {  
 for (int i = 0; i < 2; i++) {  
 System.out.println("I'm " + i);  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 System.out.println(e.getMessage());  
 }  
 }  
});  
threadPoolExecutor.shutdown();  
threadPoolExecutor.execute(() -> {  
 System.out.println("I'm Java.");  
});

以上程序执行结果如下：

I’m 0

Exception in thread “main” java.util.concurrent.RejectedExecutionException: Task com.interview.chapter5.Section2$$Lambda$2/1828972342@568db2f2 rejected from java.util.concurrent.ThreadPoolExecutor@378bf509[Shutting down, pool size = 1, active threads = 1, queued tasks = 0, completed tasks = 0]

I’m 1

可以看出，shutdown() 之后就不会再接受新的任务了，不过之前的任务会被执行完成。

### 相关面试题

#### 1.ThreadPoolExecutor 有哪些常用的方法？

答：常用方法如下所示：

* submit()/execute()：执行线程池
* shutdown()/shutdownNow()：终止线程池
* isShutdown()：判断线程是否终止
* getActiveCount()：正在运行的线程数
* getCorePoolSize()：获取核心线程数
* getMaximumPoolSize()：获取最大线程数
* getQueue()：获取线程池中的任务队列
* allowCoreThreadTimeOut(boolean)：设置空闲时是否回收核心线程

#### 2.以下程序执行的结果是什么？

ThreadPoolExecutor threadPoolExecutor = new ThreadPoolExecutor(2, 10, 10L, TimeUnit.SECONDS, new LinkedBlockingQueue());  
threadPoolExecutor.execute(new Runnable() {  
 @Override  
 public void run() {  
 for (int i = 0; i < 2; i++) {  
 System.out.println("I：" + i);  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
});  
threadPoolExecutor.shutdownNow();  
System.out.println("Java");

答：程序执行的结果是：

I：0

Java

java.lang.InterruptedException: sleep interrupted（报错信息）

I：1

题目解析：因为程序中使用了 shutdownNow() 会导致程序执行一次之后报错，抛出 sleep interrupted 异常，又因为本身有 try/catch，所以程序会继续执行打印 I：1 。

#### 3.在 ThreadPool 中 submit() 和 execute() 有什么区别？

答：submit() 和 execute() 都是用来执行线程池的，只不过使用 execute() 执行线程池不能有返回方法，而使用 submit() 可以使用 Future 接收线程池执行的返回值。

submit() 方法源码（JDK 8）如下：

public <T> Future<T> submit(Callable<T> task) {  
 if (task == null) throw new NullPointerException();  
 RunnableFuture<T> ftask = newTaskFor(task);  
 execute(ftask);  
 return ftask;  
}

execute() 源码（JDK 8）如下：

public void execute(Runnable command) {  
 if (command == null)  
 throw new NullPointerException();  
 //..... 其他  
}

#### 4.说一下 ThreadPoolExecutor 都需要哪些参数？

答：ThreadPoolExecutor 最多包含以下七个参数：

* corePoolSize：线程池中的核心线程数
* maximumPoolSize：线程池中最大线程数
* keepAliveTime：闲置超时时间
* unit：keepAliveTime 超时时间的单位（时/分/秒等）
* workQueue：线程池中的任务队列
* threadFactory：为线程池提供创建新线程的线程工厂
* rejectedExecutionHandler：线程池任务队列超过最大值之后的拒绝策略

更多详细介绍，请见正文。

#### 5.在线程池中 shutdownNow() 和 shutdown() 有什么区别？

答：shutdownNow() 和 shutdown() 都是用来终止线程池的，它们的区别是，使用 shutdown() 程序不会报错，也不会立即终止线程，它会等待线程池中的缓存任务执行完之后再退出，执行了 shutdown() 之后就不能给线程池添加新任务了；shutdownNow() 会试图立马停止任务，如果线程池中还有缓存任务正在执行，则会抛出 java.lang.InterruptedException: sleep interrupted 异常。

#### 6.说一说线程池的工作原理？

答：当线程池中有任务需要执行时，线程池会判断如果线程数量没有超过核心数量就会新建线程池进行任务执行，如果线程池中的线程数量已经超过核心线程数，这时候任务就会被放入任务队列中排队等待执行；如果任务队列超过最大队列数，并且线程池没有达到最大线程数，就会新建线程来执行任务；如果超过了最大线程数，就会执行拒绝执行策略。

#### 7.以下线程名称被打印了几次？

ThreadPoolExecutor threadPool = new ThreadPoolExecutor(1, 1,  
 10L, TimeUnit.SECONDS, new LinkedBlockingQueue<Runnable>(2),  
 new ThreadPoolExecutor.DiscardPolicy());  
threadPool.allowCoreThreadTimeOut(true);  
for (int i = 0; i < 10; i++) {  
 threadPool.execute(new Runnable() {  
 @Override  
 public void run() {  
 // 打印线程名称  
 System.out.println(Thread.currentThread().getName());  
 try {  
 Thread.sleep(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });

答：线程名被打印了 3 次。  
题目解析：线程池第 1 次执行任务时，会新创建任务并执行；第 2 次执行任务时，因为没有空闲线程所以会把任务放入队列；第 3 次同样把任务放入队列，因为队列最多可以放两条数据，所以第 4 次之后的执行都会被舍弃（没有定义拒绝策略），于是就打印了 3 次线程名称。

### 总结

ThreadPoolExecutor 是创建线程池最传统和最推荐使用的方式，创建时要设置线程池的核心线程数和最大线程数还有任务队列集合，如果任务量大于队列的最大长度，线程池会先判断当前线程数量是否已经到达最大线程数，如果没有达到最大线程数就新建线程来执行任务，如果已经达到最大线程数，就会执行拒绝策略（拒绝策略可自行定义）。线程池可通过 submit() 来调用执行，从而获得线程执行的结果，也可以通过 shutdown() 来终止线程池。

[点击此处下载本文源码](https://github.com/vipstone/java-%20interview/tree/master/interview-code/src/main/java/com/interview)

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持