> 作者：花Gie
>
> 微信公众号：Java开发零到壹

前言

多线程系列我们前面已经更新过七个章节了，强烈建议小伙伴按照顺序学习：

前面几章我们学习了线程的使用方法以及原理，但是认真的小伙伴会思考了，线程是不是可以无限创建呢，那我们处理并发不就很轻松了，瞬间感觉撑住淘宝双十一的并发都不是问题了呢。

其实并不是这样的，如果并发的线程数量很多，并且每个线程都是执行一个时间很短的任务就结束了，这样频繁创建线程就会大大降低系统的效率，因为频繁创建线程和销毁线程需要时间。本章就会和小伙伴们介绍如何解决这种问题，**即线程池的用法。**

正文

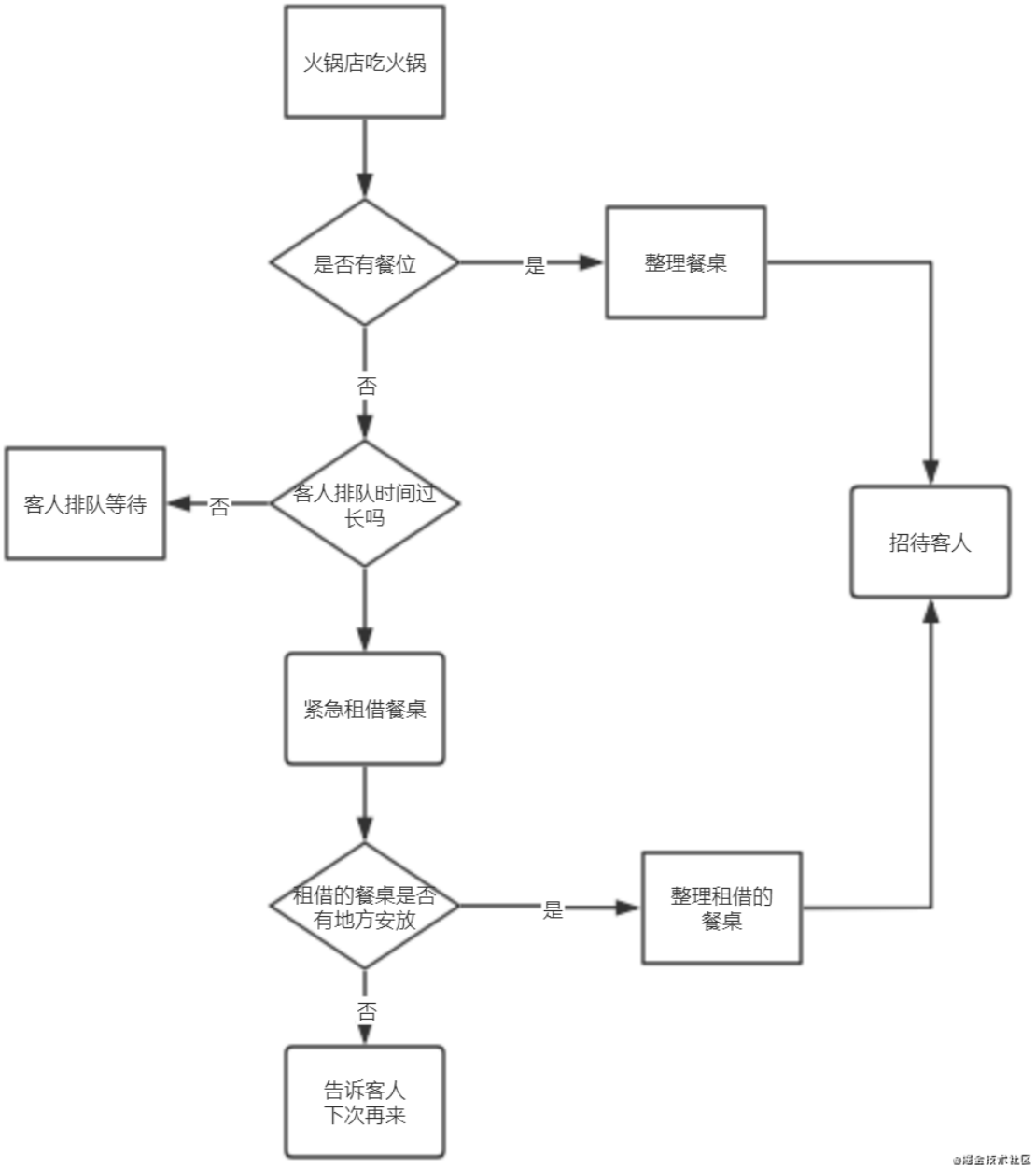
> **我：狗哥，你用过线程池吗？我看大家都在用这个，直接创建线程不香吗？**

我先举个**栗子**，我们都去火锅店吃过饭，假如一家火锅店有固定餐位10桌，当有10桌及10桌以内的客人来消费时，火锅店是可以同时招待的，而如果客人数量大于10桌时，我们就会搬个凳子，打开王者，进入漫长的等待。直到其他客人吃饱喝足之后，才会按照顺序轮到下一位客人进场。

但是到了周五晚上（不知道大家什么时候），反正我们公司小伙伴就非常喜欢周五晚上聚餐，这个时候也是人特别多的晚上，队伍可能都会排几十米，这个时候精明的老板想呀，这么多客人，等那么久，那这客户体验也太差了，于是老板赶忙从隔壁冷清的不知名火锅店租来5张桌子放入店内，这样店铺内有15个餐桌，大大减少了客户等待时间。

但是有可能即使租来了5张餐桌，等待的队伍还是非常长，那这时候老板只能告诉后来的客人，说今天爆满了，对不起了。等到工作日的时候，大家又开始划水了，因为没人吃饭，老板把借来的餐桌又还了回去。

让我们通过一张流程图来解读这个过程：



> **我：狗子，你是来凑字数的吧，说这么多?**

你这是狗咬吕洞宾，说这么多不还是为了你和小伙伴们好理解嘛。

> **我：好吧好吧，我错怪你了，给你点个赞。那用线程池有啥具体的好处呢，我只关心面试的时候该怎样回答？不能把这个例子说给面试官听吧....**

傻狗...这个例子只是为了理解。线程池的优点我总结了有以下三点：

- **降低资源消耗。**反复创建线程开销大，线程池通过重复利用已创建的线程降低消耗。
- **提高响应速度。**线程池维护部分核心线程，执行完任务不会被销毁，当下个任务进来时无需创建就能立即执行。
- **提高线程可控性。**使用线程池可以d对线程统一分配，调优和监控。

> **我：那我应该怎样使用线程池呢?**

线程池最核心的一个类是java.uitl.concurrent.ThreadPoolExecutor，它的构造方法有4种：

```
public class ThreadPoolExecutor extends AbstractExecutorService {
    .....
    public ThreadPoolExecutor(int corePoolSize,int maximumPoolSize,long
keepAliveTime,TimeUnit unit,
```

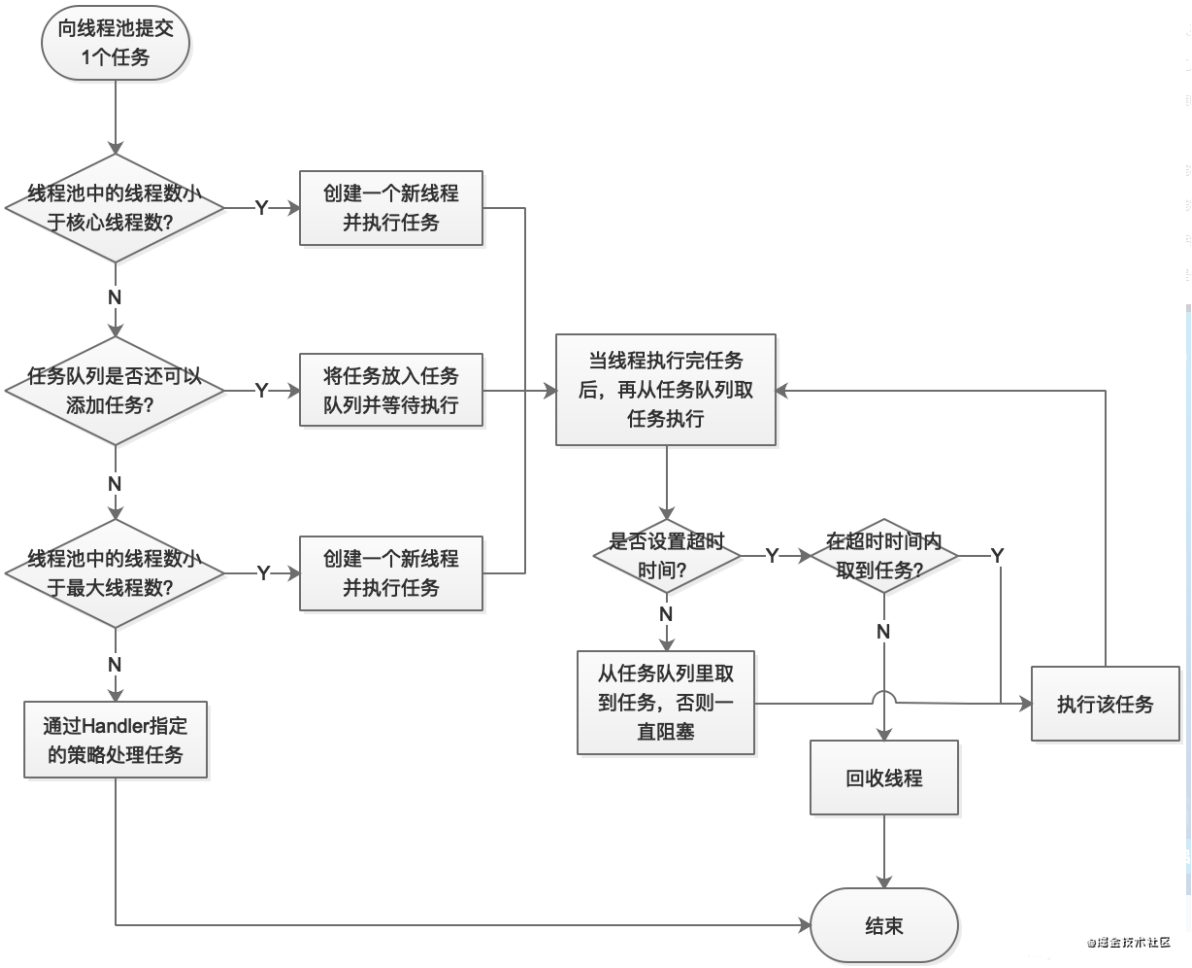
```
            BlockingQueue<Runnable> workQueue);

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

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

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

综合上面的4种创建方式，可以整理成以下几种参数：

- **corePoolSize**：核心线程数的大小，默认情况下，在创建了线程池后，线程池中的线程数为0，当有任务来之后，就会创建一个线程去执行任务，当线程池中的线程数目达到`corePoolSize`后，就会把到达的任务放到缓存队列当中；
- **maximumPoolSize**：线程池允许创建的最大线程数，当`活跃线程数`达到该数值后，后续的新任务将会阻塞。
- **keepAliveTime**：线程闲置超时时长。默认情况下，当线程数大于`核心线程数`时，keepAliveTime才会起作用，即一个线程空闲的时间达到keepAliveTime，就会被回收，直到线程数不超过`核心线程数`。但是如果调用了**allowCoreThreadTimeOut(boolean)**方法，核心线程数也会被回收，直到线程池中的线程数为0；
- **unit**：指定 keepAliveTime 参数的时间单位，常用值有TimeUnit.SECONDS（秒）；
- **workQueue**：阻塞队列，用于存储等待执行的任务，常用队列下方具体介绍；
- **threadFactory**：线程工厂。用于指定为线程池创建新线程的方式。
- **handler**：拒绝策略。当达到最大线程数时需要执行的饱和策略。

> **我：这一波介绍我头皮发麻，完全懵逼了。**

不要急，大胸弟，这里看下流程图，一图胜千言：



通过上图，再结合各个参数解释，大家已经对所有参数有了大致的了解。核心线程数、最大线程数及超时时间都非常容易理解，下面再对任务队列、拒绝策略做详细的说明。

> **我：那先说说我们有哪些常用的队列吧?**

从流程图中可以看到，当核心线程数耗尽，来不及处理的任务就会进入阻塞队列，等待被执行。最常用的有三种队列，如下：

- **ArrayBlockingQueue**：一个由数组结构组成的有界阻塞队列（数组结构可配合指针实现一个环形队列）。
- **SynchronousQueue**： 一个不存储元素的阻塞队列，消费者线程调用 take() 方法的时候就会发生阻塞，直到有一个生产者线程生产了一个元素，消费者线程就可以拿到这个元素并返回；生产者线程调用 put() 方法的时候也会发生阻塞，直到有一个消费者线程消费了一个元素，生产者才会返回。
- **LinkedBlockingDeque**： 使用双向队列实现的有界双端阻塞队列。双端意味着可以像普通队列一样 FIFO（先进先出），也可以像栈一样 FILO（先进后出）。

> **我：原来是这样（疯狂暗示自己已经掌握了），那还要一个拒绝策略是什么呢?**

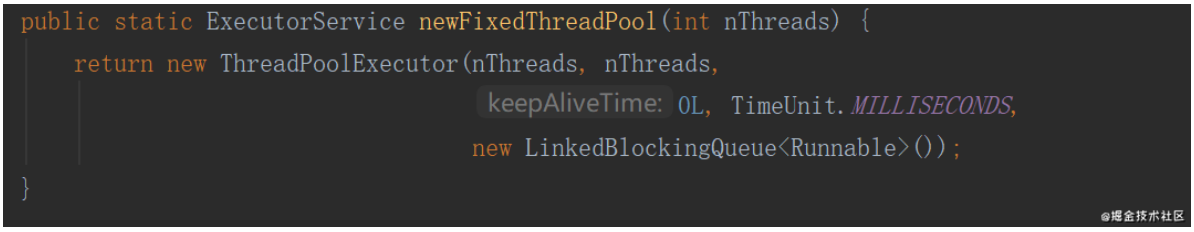
当线程池中新建的线程数到达最大线程数时，就会执行拒绝策略，Executors为我们提供4种拒绝策略：

- **AbortPolicy**：默认拒绝策略。丢弃任务并抛出RejectedExecutionException异常。
- **DiscardPolicy**：丢弃任务，但是不抛出异常。
- **DiscardOldestPolicy**：丢弃队列最前面的任务，然后重新尝试执行任务
- **CallerRunsPolicy**：由调用线程处理该任务

> **我：我的天哪，创建一个线程池怎么要考虑这么多，不玩了。**

别急，稳住，如果我们要求不高，又嫌上面使用线程池的方法太麻烦？其实Executors已经为我们封装好了 4 种常见的功能线程池，可以直接使用，如下：

- **newFixedThreadPool**：最大线程数等于核心线程数，使用的队列为LinkedBlockingQueue，是没有容量上线的，所以当请求越来越多，并且无法及时处理完毕的情况下，也不会新增线程，只是将等待执行的线程放入队列中，这样会造成内存大量被占用，可能会导致OOM。

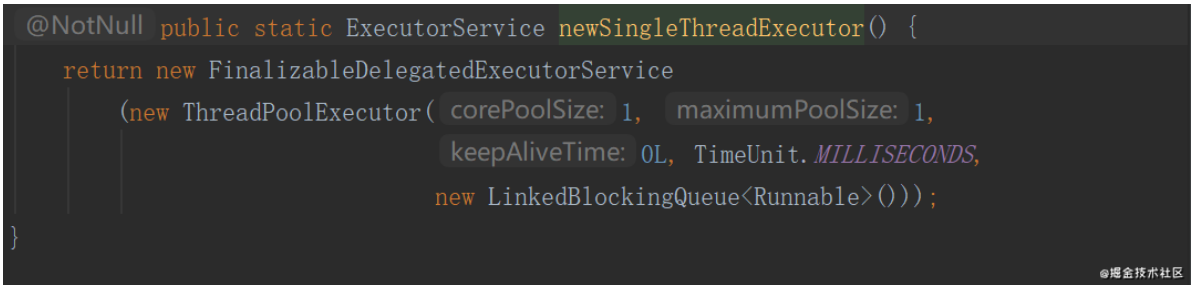


使用场景：通过限制最大线程数，控制并发数。

代码demo：

```
ExecutorService service = Executors.newFixedThreadPool(10);
for (int i = 0; i < 1000; i++) {
    service.execute(new Runnable() {
        @Override
        public void run() {
            System.out.println("线程名: " + Thread.currentThread().getName());
        }
    });
}
```

- **newSingleThreadExecutor**：和newFixedThreadPool原理基本一样，只是把线程数直接设置成1，因此也会导致同样的问题。

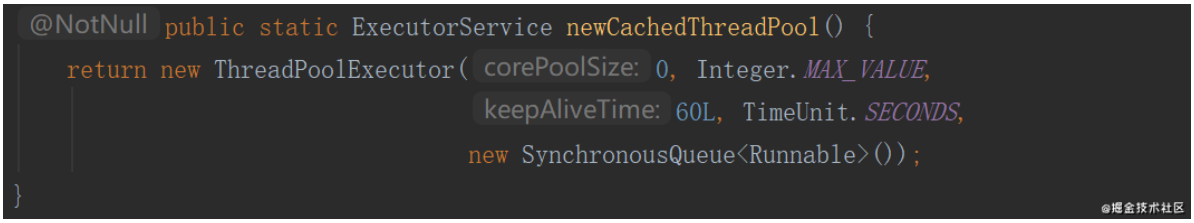


使用场景：不适用并发场景。

代码demo：

```
ExecutorService service = Executors.newSingleThreadExecutor();
for (int i = 0; i < 1000; i++) {
    service.execute(new Runnable() {
        @Override
        public void run() {
            System.out.println("线程名: " + Thread.currentThread().getName());
        }
    });
}
```

- **CachedThreadPool**：可以无限制的创建线程，使用SynchronousQueue作为队列（不存储元素），具有自动回收多余线程的功能，其弊端在于maximumPoolSize(最大线程数)设置成Integer.MAX_VALUE，这样有可能导致创建非常多的线程，造成OOM。

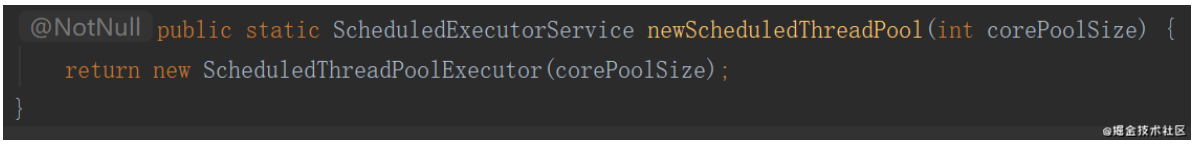


使用场景：适合处理大量并发请求，且每个请求耗时短的场景。

代码demo：

```
ExecutorService service = Executors.newCachedThreadPool();
for (int i = 0; i < 1000; i++) {
    service.execute(new Runnable() {
        @Override
        public void run() {
            System.out.println("线程名: " + Thread.currentThread().getName());
        }
    });
}
```

- **newScheduledThreadPool**：支持定时及周期性执行任务的线程池。



使用场景：执行定时或周期性的任务。

代码demo：

```
public class ThreadPoolDemo {
 public static void main(String[] args) {
    //用法一：3秒后执行
    ScheduledExecutorService service = Executors.newScheduledThreadPool(10);
    service.schedule(new ThreadTask(),3,TimeUnit.SECONDS);

    //用法二：第一次3秒后执行，以后每个5秒执行
    service.scheduleAtFixedRate(new ThreadTask(),3,5,TimeUnit.SECONDS);
 }
}
class ThreadTask implements Runnable{
 @Override
 public void run() {
     System.out.println("大家好，我是花Gie");
 }
}
```

类型	池内 线程类型	池内 线程数量	处理特点	应用场景
定长线程池 (FixedThreadPool)	核心线程	固定	• 核心线程处于空闲状态时也不会被回收，除非线程被关闭 • 当所有线程都处于活动状态时，新的任务都会处于等待状态，直到有线程空闲出来 • 任务队列无大小限制 （超出的线程任务会在队列中等待）	控制线程最大并发数
定时线程池 (ScheduledThreadPool)	核心 & 非核心线程	• 核心线程 固定 • 非核心线程 无限制	当非核心线程闲置时，则会被立即回收	执行定时 / 周期性 任务
可缓存线程池 (CachedThreadPool)	非核心线程	不固定 （可无限大）	• 优先利用闲置线程处理新任务 （即 会重用线程） • 无线程可用时，即 新建线程 （即 任何线程任务到来都会立刻执行，不需要等待） • 灵活回收空闲线程 （具备超时机制=60s，即空闲超过60s才回收，全部回收时几乎不占系统资源）	执行数量多、耗时少的线程任务
单线程化线程池 (SingleThreadExecutor)	核心线程	1个	• 保证所有任务按照指定顺序在一个线程中执行 （相当于顺序执行任务） • 不需要处理线程同步的问题	单线程 （不适合并发但可能引起IO阻塞性及影响UI线程响应的操作，如数据库操作等）

> **我：感觉奇怪的知识又增加了，这么多线程池，我应该如何选择呢？**

我们需要根据自己的业务场景，来选择符合自己的线程池，比如我们想自定义线程名称、任务被拒绝后记录日志以及结合自身内存大小等因素进行选择。

如果想要深入的了解线程池，源码还是非常有必要看一下的，我现在和你说一下....

> **我：打住....我现在都快炸裂了，等我消化一波，明天再说**

总结

以上就是线程池的基本用法，以及常用的参数介绍，里面涉及了一部分队列知识，后续也会给大家详细介绍。学会了用法，下一章我们将会对线程池的源码进行解析，知其然还要知其所以然，这样才能彻底掌握其精髓。

点关注，防走丢

以上就是本期全部内容，**如有纰漏之处，请留言指教，非常感谢。**我是花GieGie ，有问题大家随时留言讨论 ，我们下期见。

> 文章持续更新，可以微信搜一搜 **Java开发零到壹** 第一时间阅读，并且可以获取**面试资料学习视频**等，有兴趣的小伙伴欢迎关注，一起学习，一起哈。

原创不易，你怎忍心白嫖，如果你觉得这篇文章对你有点用的话，感谢老铁为本文`点个赞、评论或转发一下`，因为这将是我输出更多优质文章的动力，感谢！