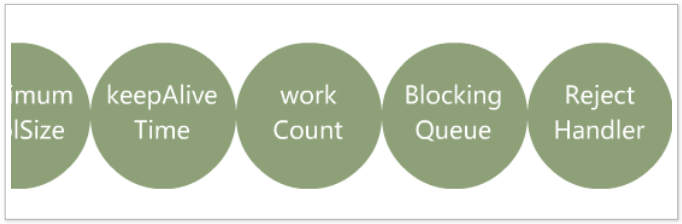
小伙伴被线程池的参数绕晕了，看完这 10 个动态图，终于懂了

> 从问题出发，走一遍线程池的思想之旅，你会发现它很简单

小宇：闪客，我最近看到线程池，被里边乱七八糟的参数给搞晕了，你能不能给我讲讲呀？



闪客：没问题，这个我擅长，咱们从一个最简单的情况开始，假设有一段代码，你希望异步执行它，是不是要写出这样的代码？

```
new Thread(r).start();
```

小宇：嗯嗯，最简单的写法似乎就是这样呢。
闪客：这种写法当然可以完成功能，可是你这样写，老王这样写，老张也这样写，程序中到处都是这样创建线程的方法，能不能写一个统一的工具类让大家调用呢？

小宇：可以的，感觉有一个统一的工具类，更优雅一些。

闪客：那如果让你来设计这个工具类，你会怎么写呢？我先给你定一个接口，你来实现。

```
new Thread(r).start();
```

小宇：emmm，我可能先定义几个成员变量，比如核心线程数、最大线程数 ... 反正就是那些乱七八糟的参数。
闪客：STOP！小宇呀，你现在深受面试手册的毒害，你先把这些全部的概念忘掉，就说让你写一个最简单的工具类，第一反应，你会怎么写？

小宇：那我可能会这样

```
public interface Executor {    public void execute(Runnable r);}
```

闪客：嗯嗯很好，你的思路非常棒。
小宇：啊，我这个会不会太 low 了呀，我还以为你会骂我呢。

怎么会， Doug Lea 大神在 JDK 源码注释中给出的就是这样的例子，这是最根本的功能。你在这个基础上，尝试着优化一下？

小宇：还能怎么优化呢？这不已经用一个工具类实现了异步执行了嘛！

闪客：我问你一个问题，假如有 10000 个人都调用这个工具类提交任务，那就会创建 10000 个线程来执行，这肯定不合适吧！能不能控制一下线程的数量呢？

小宇：这不难，我可以把这个任务 r 丢到一个 tasks 队列中，然后只启动一个线程，就叫它 **Worker** 线程吧，不断从 tasks 队列中取任务，执行任务。这样无论调用者调用多少次，永远就只有一个 Worker 线程在运行，像这样。

闪客：太棒了，这个设计有了三个重大的意义：

1. 控制了线程数量。

2. 队列不但起到了缓冲的作用，还将任务的提交与执行解耦了。

3. 最重要的一点是，解决了每次重复创建和销毁线程带来的系统开销。

小宇：哇真的么，这么小的改动有这么多意义呀。

闪客：那当然，不过只有一个后台的工作线程 Worker 会不会少了点？还有如果这个 tasks 队列满了怎么办呢？

小宇：哦，的确，只有一个线程在某些场景下是很吃力的，那我把 Worker 线程的数量增加？

闪客：没错，Worker 线程的数量要增加，但是具体数量要让使用者决定，调用时传入，就叫核心线程数 **corePoolSize** 吧。

小宇：好的，那我这样设计。

1. 初始化线程池时，直接启动 corePoolSize 个工作线程 Worker 先跑着。

2. 这些 Worker 就是死循环从队列里取任务然后执行。

3. execute 方法仍然是直接把任务放到队列，但队列满了之后直接抛弃

闪客：太完美了，奖励你一块费列罗吧。

小宇：哈哈谢谢，那我先吃一会儿哈。

闪客：好，你边吃我边说。现在我们已经实现了一个至少不那么丑陋的线程池了，但还有几处小瑕疵，比如初始化的时候，就创建了一堆 Worker 线程在那空跑着，假如此时并没有异步任务提交过来执行，这就有点浪费了。

小宇：哦好像是诶！

闪客：还有，你这队列一满，就直接把新任务丢弃了，这样有些粗暴，能不能让调用者自己决定该怎么处理呢？

小宇：哎呀，想不到我这么温柔的妹纸居然写出了这么粗暴的代码。

闪客：额，你先把费列罗咽下去吧。

小宇：我吃完了，现在脑子有点不够用了，得先消化消化食物，要不你帮我分析分析吧。

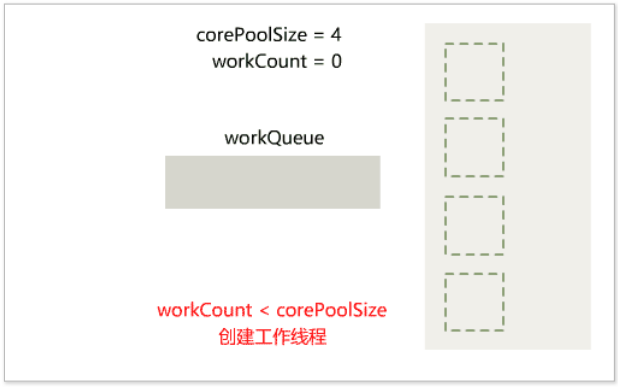
闪客：好的，现在我们做出如下改进。

1. 按需创建 Worker：刚初始化线程池时，不再立刻创建 corePoolSize 个工作线程，而是等待调用者不断提交任务的过程中，逐渐把工作线程 Worker 创建出来，等数量达到 corePoolSize 时就停止，把任务直接丢到队列里。那就必然要用一个属性记录已经创建出来的工作线程数量，就叫 **workCount** 吧。

2. 加拒绝策略：实现上就是增加一个入参，类型是一个接口 **RejectedExecutionHandler**，由调用者决定实现类，以便在任务提交失败后执行 rejectedExecution 方法。

3. 增加线程工厂：实现上就是增加一个入参，类型是一个接口 **ThreadFactory**，增加工作线程时不再直接 new 线程，而是调用这个由调用者传入的 ThreadFactory 实现类的 newThread 方法。

就像下面这样。



小宇：哇，还是你厉害，这一版应该很完美了吧？

闪客：不不不，离完美还差得很远，接下来的改进，由你来想吧，我这里可以给你一个提示

弹性思维

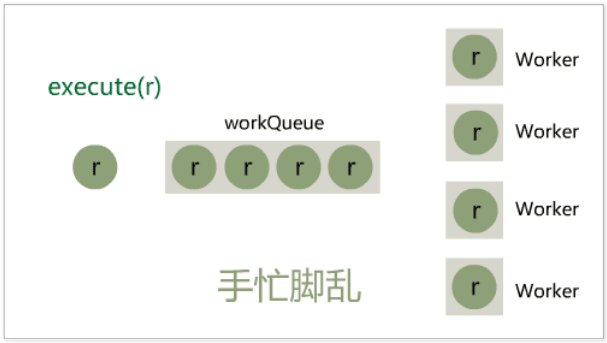
小宇：弹性思维？哈哈闪客你这术语说的真是越来越不像人话了

闪客：咳咳

小宇：哦，我是说你肯定是指我这个代码写的没有弹性，对吧？可是弹性是指什么呢？

闪客：简单说，在这个场景里，弹性就是在任务提交比较频繁，和任务提交非常不频繁这两种情况下，你这个代码是否有问题？

小宇：emmm 让我想想，我这个线程池，当提交任务的量突增时，工作线程和队列都被占满了，就只能走拒绝策略，其实就是被丢弃掉



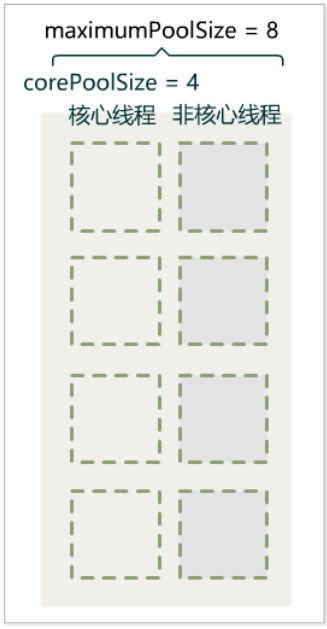
闪客：是的

小宇：这样的确是太硬了，诶不过我想了下，调用方可以通过设置很大的核心线程数 corePoolSize 来解决这个问题呀。

闪客：的确是可以，但一般场景下 QPS 高峰期都很短，而为了这个很短暂的高峰，设置很大的核心线程数，简直太浪费资源了，你看上面的图不觉得眼晕么？

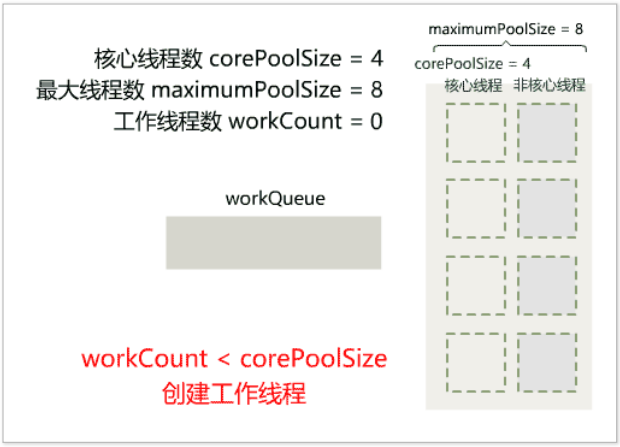
小宇：是呀，那怎么办呢，太大了也不行，太小了也不行。

闪客：我们可以发明一个新的属性，叫最大线程数 **maximumPoolSize**。当核心线程数和队列都满了时，新提交的任务仍然可以通过创建新的工作线程（叫它**非核心线程**），直到工作线程数达到 maximumPoolSize 为止，这样就可以缓解一时的高峰期了，而用户也不用设置过大的核心线程数。



小宇：哦好像有点感觉了，可是具体怎么操作呢？

闪客：想象力不行呀小宇，那你看下面的演示。



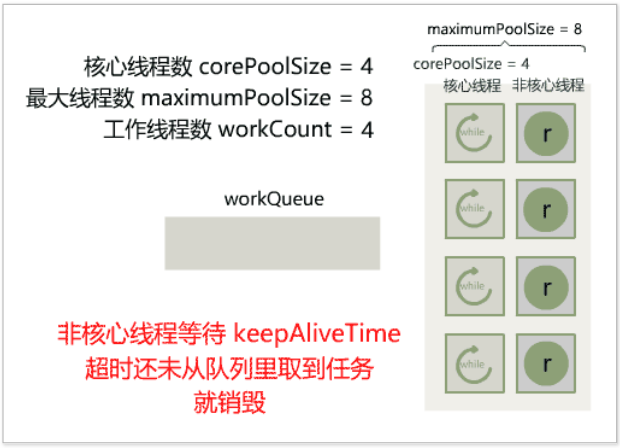
1. 开始的时候和上一版一样，当 workCount < corePoolSize 时，通过创建新的 Worker 来执行任务。

2. 当 workCount >= corePoolSize 就停止创建新线程，把任务直接丢到队列里。

3. 直当队列已满时仍然 workCount < maximumPoolSize 时，不再直接走拒绝策

略，而是创建非核心线程，直到 workCount = maximumPoolSize，再走拒绝策略。

小宇：哎呀，我怎么没想到，这样 corePoolSize 就负责平时大多数情况所需要的工作线程数，而 maximumPoolSize 就负责在高峰期临时扩充工作线程数。
闪客：没错，高峰时期的弹性搞定了，那自然就还要考虑低谷时期。当长时间没有任务提交时，核心线程与非核心线程都一直空跑着，浪费资源。我们可以给**非核心线程**设定一个超时时间 **keepAliveTime**，当这么长时间没能从队列里获取任务时，就不再等了，销毁线程。



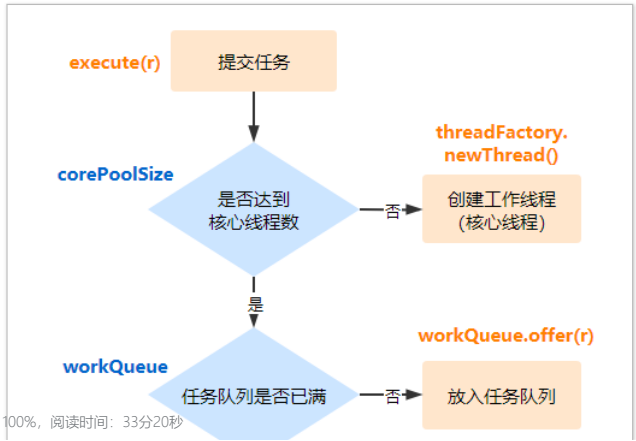
小宇：嗯，这回咱们的线程池在 QPS 高峰时可以临时扩容，QPS 低谷时又可以及时回收线程（非核心线程）而不至于浪费资源，真的显得十分 Q 弹呢。

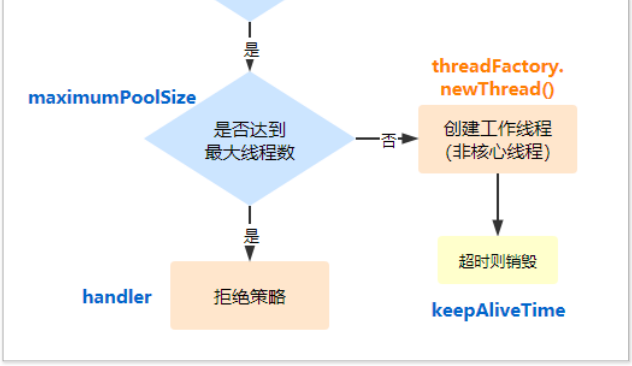
闪客：是呀是呀。诶不对，怎么又变成我说了，不是说这一版你来思考么？
小宇：我也想啊，但你这一讲技术就自说自话的毛病老是不改，我有啥办法。
闪客：额抱歉抱歉，那接下来你总结一下我们的线程池吧
小宇：嗯好的，首先它的构造方法是这个样子滴

```
public interface Executor {    public void execute(Runnable r);}
```

这些参数分别是

int corePoolSize：核心线程数
int maximumPoolSize：最大线程数
long keepAliveTime：非核心线程的空闲时间
TimeUnit unit：空闲时间的单位
BlockingQueue<Runnable> workQueue：任务队列（线程安全的阻塞队列）
ThreadFactory threadFactory：线程工厂
RejectedExecutionHandler handler：拒绝策略
整个任务的提交流程是





闪客：不错不错，这可是你自己总结的哟，现在还用我给你讲什么是线程池了么？

小宇：啊天呢，我才发现这似乎就是我一直弄不清楚的线程池的参数和原理呢！

闪客：没错，而且最后一版代码的构造方法，就是 Java 面试常考的 **ThreadPoolExecutor** 最长的那个构造方法，参数名都没变。

小宇：哇，太赞了！我都忘了一开始我想干嘛了，嘻嘻。

闪客：哈哈，不知不觉学到了技术才爽呢，对吧？晚饭时间快到了，要不要一块去吃山西面馆呀？

小宇：哦，那家店餐桌的颜色我不太喜欢，下次吧。

闪客：哦好吧。

后记

线程池是面试常考的知识点，网上很多文章都是直接从它那有 7 个参数的构造方法讲起，强行把各个参数的含义说给你听，让人云里雾里。

希望读者读完本篇文章后，线程池的这些参数不再是死记硬背，而是像本文中这些动图一样在你的脑中活灵活现，这样就能永远记住他们啦~

全文完