# 线程池

## 什么是线程池

java.util.concurrent.Executors提供了一个 java.util.concurrent.Executor接口的实现用于创建线程池.它是用来利用”容器”管理线程的一种机制.传统情况下,线程的是使用一般会经历三个过程,即线程的创建,服务,销毁.所以在高并发情况下,这三个环节(创建,服务,销毁)会反复执行,费时费力,那么利用线程池,可以大大减少系统的开销,创建/销毁次数显著减少.

**补充:查看CPU信息**

**Cmd**

**wmic**

**cpu get \***

## 为什么要使用线程池

因为频繁的开启线程或者停止，线程需要重新被cpu从就绪状态到调度，效率非常低，所以使用线程池可以**实现复用**，从而提高效率。

在开发过程中，合理地使用线程池可以带来三个好处:

第一：降低资源消耗,通过重复利用已创建的线程,降低线程创建和销毁造成的消耗

第二：提高响应速度,当任务到达时，任务可以不需要等到线程创建就能立即执行

第三：提高线程的可管理性。线程是稀缺资源,如果无限制地创建，不仅会消耗系统资源，

还会降低系统的稳定性，使用线程池可以进行统一分配,调优和监控

**但是，要做到合理利用线程池，必须对其实现原理了了如指掌。**

## 线程池的作用

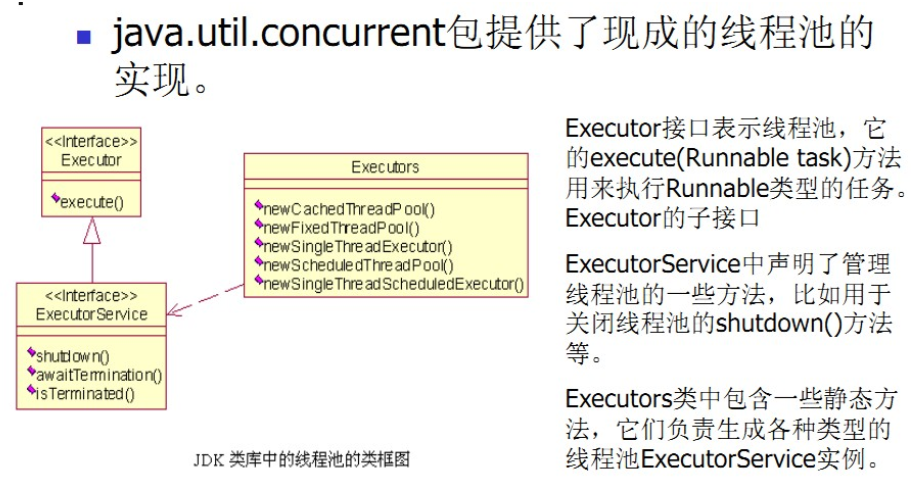
1.降低资源消耗：通过池化技术重复利用已创建的线程，降低线程创建和销毁造成的损耗。

2.提高响应速度：任务到达时，无需等待线程创建即可立即执行。

3.提高线程的可管理性：线程是稀缺资源，如果无限制创建，不仅会消耗系统资源，还会因为线程的不合理分布导致资源调度失衡，降低系统的稳定性。使用线程池可以进行统一的分配、调优和监控。

4.提供更多更强大的功能：线程池具备可拓展性，允许开发人员向其中增加更多的功能。比如延时定时线程池ScheduledThreadPoolExecutor，就允许任务延期执行或定期执行。

## 线程池的体系结构



负责线程的使用和调度的根接口:

**java.util.concurrent.Executor**

子接口:线程池的主要接口

**ExecutorService**

线程池实现类:

**ThreadPoolExecutor**

子接口:负责线程的调度

**ScheduledExecutorService**

继承ThreadPoolExecutor,实现了ScheduledExecutorService

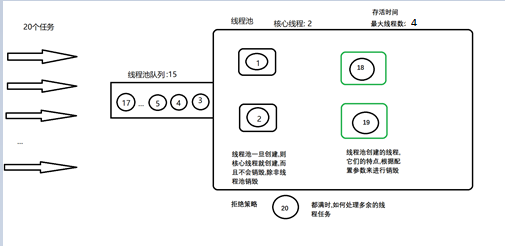
**ScheduledThreadPoolExecutor**

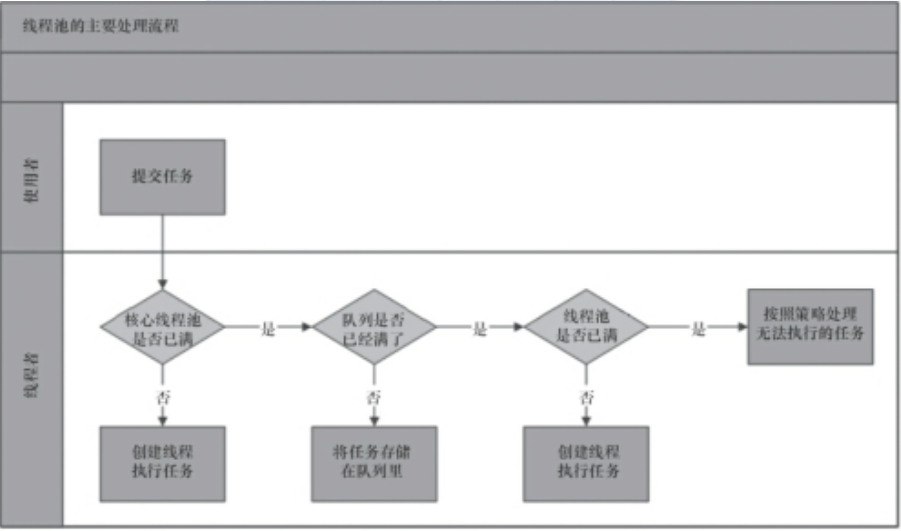
工具类:

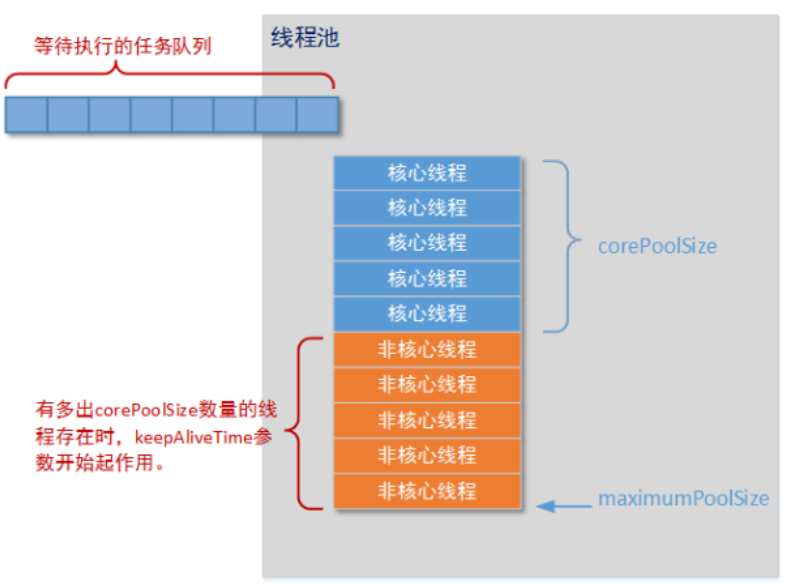
**Executors**

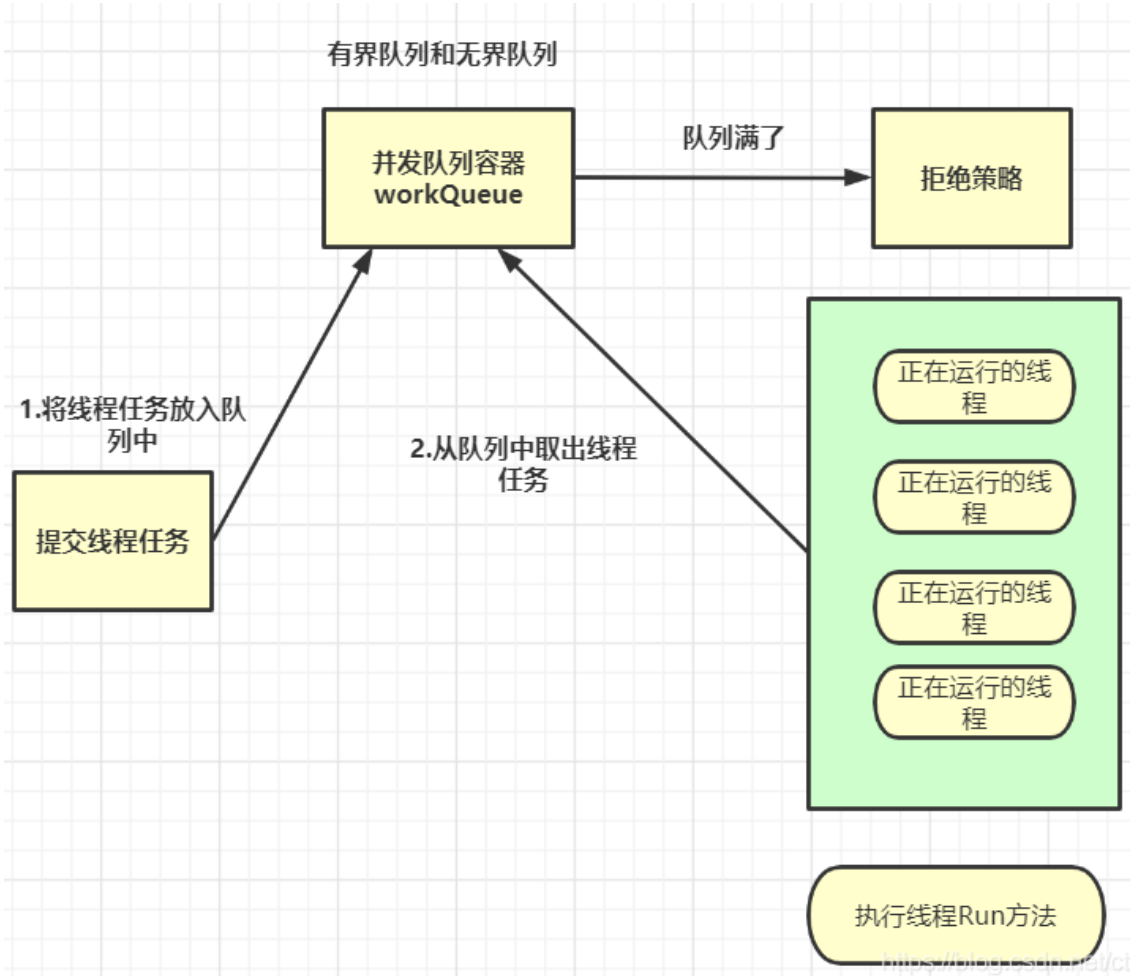
## 线程池核心原理

1. 提交任务的时候比较**核心线程数**，如果当前任务数量小于核心线程数的情况下，则直接复用线程执行
2. 如果任务量大于核心线程数，则**缓存到队列**中
3. 如果缓存队列满了，且任务小于**最大线程数**的情况下，则创建线程执行
4. 如果队列和最大线程数都满的情况下，则走**拒绝策略**









## 线程池的创建

方式一: 传统创建方法,优点:线程池所有参数都可以自行定制,缺点:门槛较高

**ThreadPoolExecutor**的完整构造方法的签名是：

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

方式二:使用工具类来快捷创建线程池,优先:简便,缺点:无法个性定制线程池参数

**ThreadPoolExecutor是Executors类的底层实现**

**总结:**

**学习阶段或者小型项目可以使用方式二快速构建,但是中大型项目,推荐使用方式一来根据实际业务情况来调整线程池各参数!!!**

## ThreadPoolExecutor核心参数

corePoolSize：核心线程数量，一直正在保持运行的线程,当线程池创建时,核心线程就创建,

当线程池销毁时,核心线程销毁

maximumPoolSize：最大线程数，是当队列满的情况下，核心线程数+非核心线程数=最大线程数

keepAliveTime：超时时间，在一定时间内没有线程使用非核心线程数的情况下，则销毁，节约cpu资源

unit：keepAliveTime的时间单位

workQueue：缓存线程队列,有很多不同的类型可供选择

threadFactory: 执行程序创建新线程时使用的工厂

handler: 由于超出线程范围和队列容量而使执行被阻塞时所使用的处理程序

## 阻塞队列

**阻塞队列**：添加元素的时候，当队列满的情况下，可以等待一定的时间，看看有没有空的位置，如果有的话就添加进去，没有的话再返回插入失败

**非阻塞队列**：添加元素的时候，当队列满的情况下，不等待，直接返回插入失败

1. **ArrayBlockingQueue：**

它是一个有界缓存等待队列,可以指定缓存队列的大小,当正在执行的线程数等于corePoolSize时,多余的元素缓存在ArrayBlockingQueue队列中等待有空闲的线程时继续执行,当ArrayBlockingQueue已满时,加入ArrayBlockingQueue失败,会开启新的线程去执行,当线程数已经达到最大的maximumPoolSizes时,再有新的元素尝试加入ArrayBlockingQueue时会报错.

1. **LinkedBlockingQueue:**

它是一个无界缓存等待队列.当前执行的线程数量达到corePoolSize的数量时,剩余的元素会在阻塞队列里等待.(所以在使用此阻塞队列时maximumPoolSizes就相当于无效了),每个线程完全独立于其他线程.生产者和消费者使用独立的锁来控制数据的同步,即在高并发的情况下可以并行操作队列中的数据.

1. **SynchronousQueue：**

它没有容量,是无缓冲等待队列,是一个不存储元素的阻塞队列,会直接将任务交给消费者,必须等队列中的添加元素被消费后才能继续添加新元素.

使用它阻塞队列一般要求maximumPoolSizes为无界,避免线程拒绝执行操作.；

1. **PriorityBlockingQueue：**

优先级队列，具有优先级的无限阻塞队列。

## 线程池队列拒绝策略(饱和策略)

线程池拒绝线程任务的条件是任务：>最大线程数+队列缓存容量

两种情况会拒绝处理任务：

1.当线程数已经达到maxPoolSize，且队列已满，会拒绝新任务

2.当线程池被调用shutdown()后，会等待线程池里的任务执行完毕，再shutdown。如果在调用shutdown()和线程池真正shutdown之间提交任务，会拒绝新任务。线程池会调用rejectedExecutionHandler来处理这个任务。如果没有设置，默认是AbortPolicy，会抛出异常。

ThreadPoolExecutor类有几个内部实现类来处理拒绝任务：

1. AbortPolicy 丢弃任务，抛运行时异常rejectedExecutionException
2. CallerRunsPolicy 对于拒绝的请求交给调用方(caller)去执行,此测试类的调用方式main函数
3. DiscardPolicy 抛弃新的请求,不抛出异常
4. DiscardOldestPolicy 抛弃旧的未执行的请求,不抛出异常

**默认是AbortPolicy拒绝策略**

## 使用工具类创建线程池

ExecutorService Executors.newCachedThreadPool(); 可缓存的线程池，该线程池中没有核心线程，非核心线程的数量为Integer.max\_value，就是无限大，当有需要时创建线程来执行任务，没有需要时回收线程，适用于耗时少，任务量大的情况。

ExecutorService Executors.newFixedThreadPool()；定长的线程池，有核心线程，核心线程的即为最大的线程数量，没有非核心线程

ExecutorService Executors.newSingleThreadExecutor(); 只有一条线程来执行任务，适用于有顺序的任务的应用场景

ScheduledExecutorService Executors.newScheduledThreadPool() ；周期性执行任务的线程池，按照某种特定的计划执行线程中的任务，有核心线程，但也有非核心线程，非核心线程的大小也为无限大。适用于执行周期性的任务

**底层都是基于ThreadPoolExecutor构造函数封装**

## 线程池的状态

1.RUNNING：线程池能够接受新任务，以及对新添加的任务进行处理。

2.SHUTDOWN：线程池不可以接受新任务，但是可以对已添加的任务进行处理。

3.STOP：线程池不接收新任务，不处理已添加的任务，并且会中断正在处理的任务。

4.TIDYING：当所有的任务已终止，ctl记录的"任务数量"为0，线程池会变为TIDYING状态。当线程池变为TIDYING状态时，会执行构造函terminated()。terminated()在ThreadPoolExecutor类中是空的，若用户想在线程池变为TIDYING时，进行相应的处理；可以通过重载terminated()函数来实现。

5.TERMINATED：线程池彻底终止的状态。

