线程池——治理线程的最大法宝

线程池——治理线程的法宝

1. 线程池的自我介绍
2. 创建和停止线程池
3. 常见线程池的特点和用法

线程池——治理线程的法宝

4. 任务太多，怎么拒绝？
5. 钩子方法，给线程池加点料
6. 实现原理、源码分析
7. 使用线程池的注意点

线程池的自我介绍

- 线程池的重要性
- 什么是“池”

  -- 软件中的“池”，可以理解为计划经济

线程池的自我介绍

- 如果不使用线程池，每个任务都新开一个线程处理
  - 一个线程
  - for循环创建线程
  - 当任务数量上升到1000

这样开销太大，我们希望有固定数量的线程，来执行这1000个线程，这样就避免了反复创建并销毁线程所带来的开销问题

为什么要使用线程池

- 问题一：反复创建线程开销大
- 问题二：过多的线程会占用太多内存
- 解决以上两个问题的思路
  - 用少量的线程——避免内存占用过多
  - 让这部分线程都保持工作，且可以反复执行任务——避免生命周期的损耗

线程池的好处

- 加快响应速度
- 合理利用CPU和内存
- 统一管理

线程池适合应用的场合

- 服务器接收到大量请求时，使用线程池技术是非常合适的，它可以大大减少线程的创建和销毁次数，提高服务器的工作效率
- 实际上，在开发中，如果需要创建5个以上的线程，那么就可以使用线程池来管理

创建和停止线程池

- 线程池构造方法的参数
- 线程池应该手动创建还是自动创建
- 线程池里的线程数量设定为多少比较合适?
- 停止线程池的正确方法

线程池构造方法的参数

参数名	类型	含义
corePoolSize	int	核心线程数，详解见下文
maxPoolSize	int	最大线程数，详解见下文
keepAliveTime	long	保持存活时间
workQueue	BlockingQueue	任务存储队列
threadFactory	ThreadFactory	当线程池需要新的线程的时候，会使用threadFactory来生成新的线程
Handler	RejectedExecutionHandler	由于线程池无法接受你所提交的任务的拒绝策略

参数中的corePoolSize和maxPoolSize

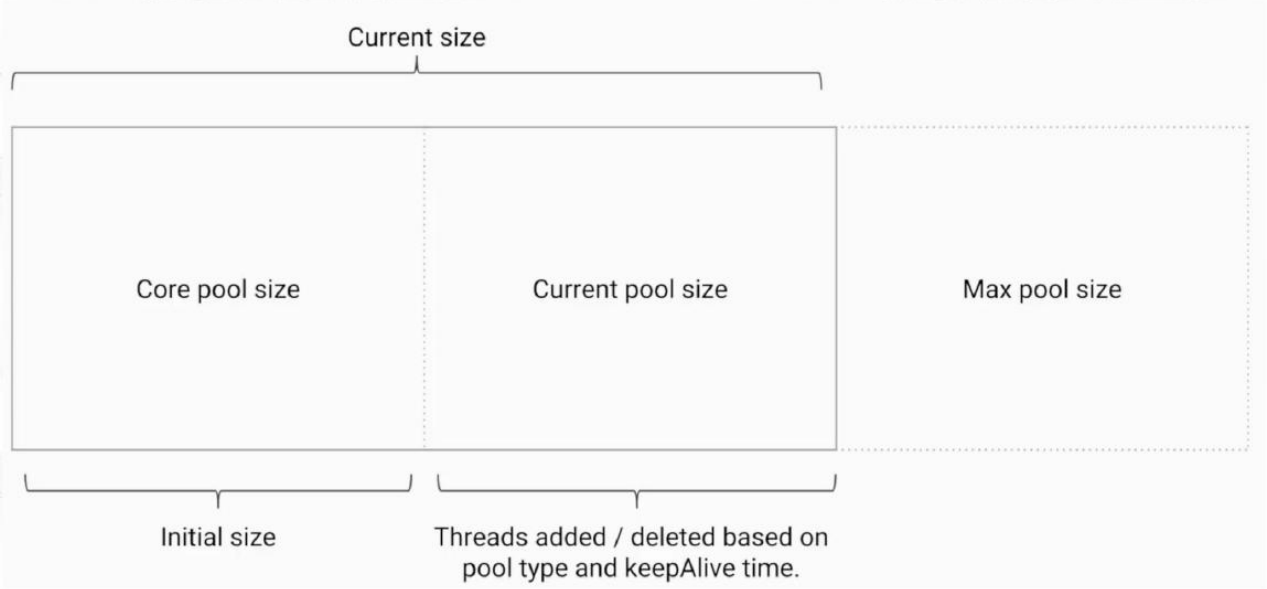
◆ corePoolSize指的是核心线程数

- 线程池在完成初始化后，默认情况下，线程池中并没有任何线程，线程池会等待有任务到来时，再创建新线程去执行任务

◆ 最大量maxPoolSize

- 在核心线程数的基础上，额外增加的线程数的上限

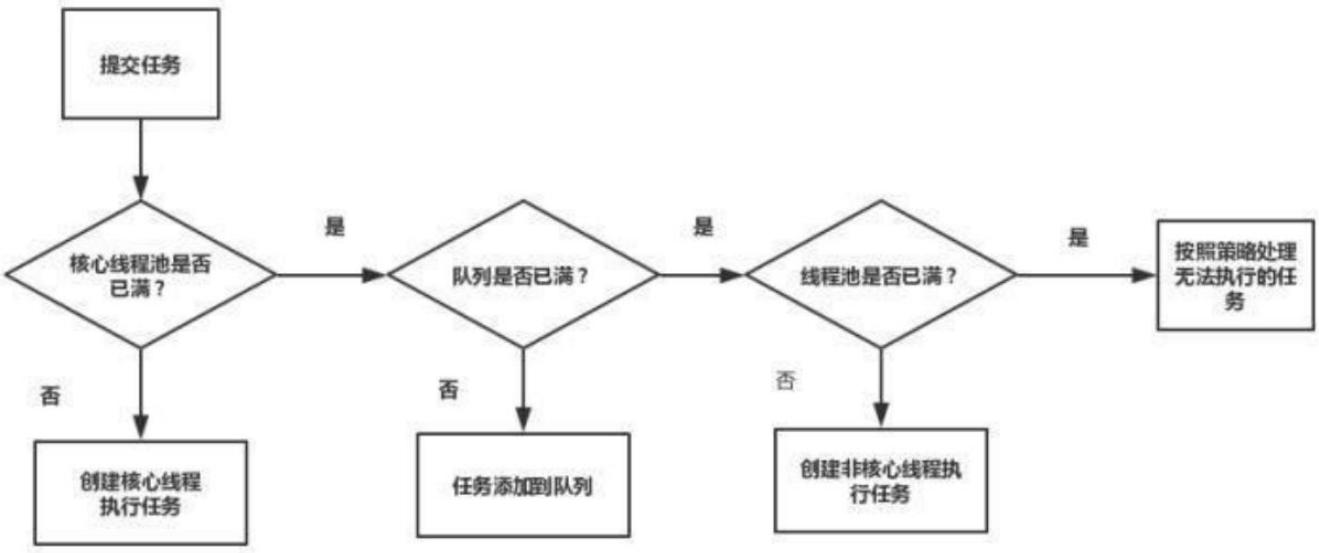
corePoolSize和maxPoolSize



添加线程规则

1. 如果线程数小于corePoolSize，即使其他工作线程处于空闲状态，也会创建一个新线程来运行新任务。
2. 如果线程数等于（或大于）corePoolSize但少于maximumPoolSize，则将任务放入队列。
3. 如果队列已满，并且线程数小于maxPoolSize，则创建一个新线程来运行任务。
4. 如果队列已满，并且线程数大于或等于maxPoolSize，则拒绝该任务。

添加线程规则



添加线程规则

- 是否需要增加线程的判断顺序是：
  - corePoolSize
  - workQueue
  - maxPoolSize

- 比喻：烧烤店的桌子

举个例子

- 线程池：核心池大小为5，最大池大小为10，队列为100。

- 因为线程中的请求最多会创建5个，然后任务将被添加到队列中，直到达到100。当队列已满时，将创建最新的线程maxPoolSize，最多到10个线程，如果再来任务，就拒绝。

增减线程的特点

1.通过设置corePoolSize和maximumPoolSize 相同，就可以创建固定大小的线程池。

2.线程池希望保持较少的线程数，并且只有在负载变得很大时才增加它。

增减线程的特点

3.通过设置maximumPoolSize为很高的值，例如 Integer.MAX_VALUE，可以允许线程池容纳任意数量的并发任务。

4.只有在队列填满时才创建多于corePoolSize的线程，如果使用的是无界队列（例如LinkedBlockingQueue），那么线程数就不会超过corePoolSize。

keepAliveTime

- 如果线程池当前的线程数多于corePoolSize，那么如果多余的线程空闲时间超过keepAliveTime，它们就会被终止

ThreadFactory 用来创建线程

- 新的线程是由ThreadFactory创建的，默认使用Executors.defaultThreadFactory()
- 创建出来的线程都在同一个线程组，拥有同样的NORM_PRIORITY优先级并且都不是守护线程。
- 如果自己指定ThreadFactory，那么就可以改变线程名、线程组、优先级、是否是守护线程等。
- 通常使用默认的ThreadFactory就可以了

工作队列

- 有3种最常见的队列类型：

  1) 直接交接：SynchronousQueue

  2) 无界队列：LinkedBlockingQueue

  3) 有界的队列：ArrayBlockingQueue

线程池应该手动创建还是自动创建

- 手动创建更好，因为这样可以更加明确线程池的运行规则，避免资源耗尽的风险

自动创建线程池（即直接调用JDK封装好的构造方法）可能带来哪些问题？

线程池应该手动创建还是自动创建

- newFixedThreadPool
  - 容易造成大量内存占用，可能会导致OOM
- newSingleThreadExecutor
  - 当请求堆积的时候，可能会占用大量的内存

线程池应该手动创建还是自动创建

- newCachedThreadPool
  - 弊端在于第二个参数maximumPoolSize被设置为了Integer.MAX_VALUE，这可能会创建数量非常多的线程，甚至导致OOM
- newScheduledThreadPool
  - 原因和newCachedThreadPool一样

线程池应该手动创建还是自动创建

- 正确的创建线程池的方法
  - 根据不同的业务场景，设置线程池参数
  - 比如：内存有多大，给线程取什么名字等等

线程池里的线程数量设定为多少比较合适?

- CPU密集型（加密、计算hash等）：最佳线程数为CPU核心数的1-2倍左右。
- 耗时IO型（读写数据库、文件、网络读写等）：最佳线程数一般会大于CPU核心数很多倍

参考Brain Goetz推荐的计算方法：

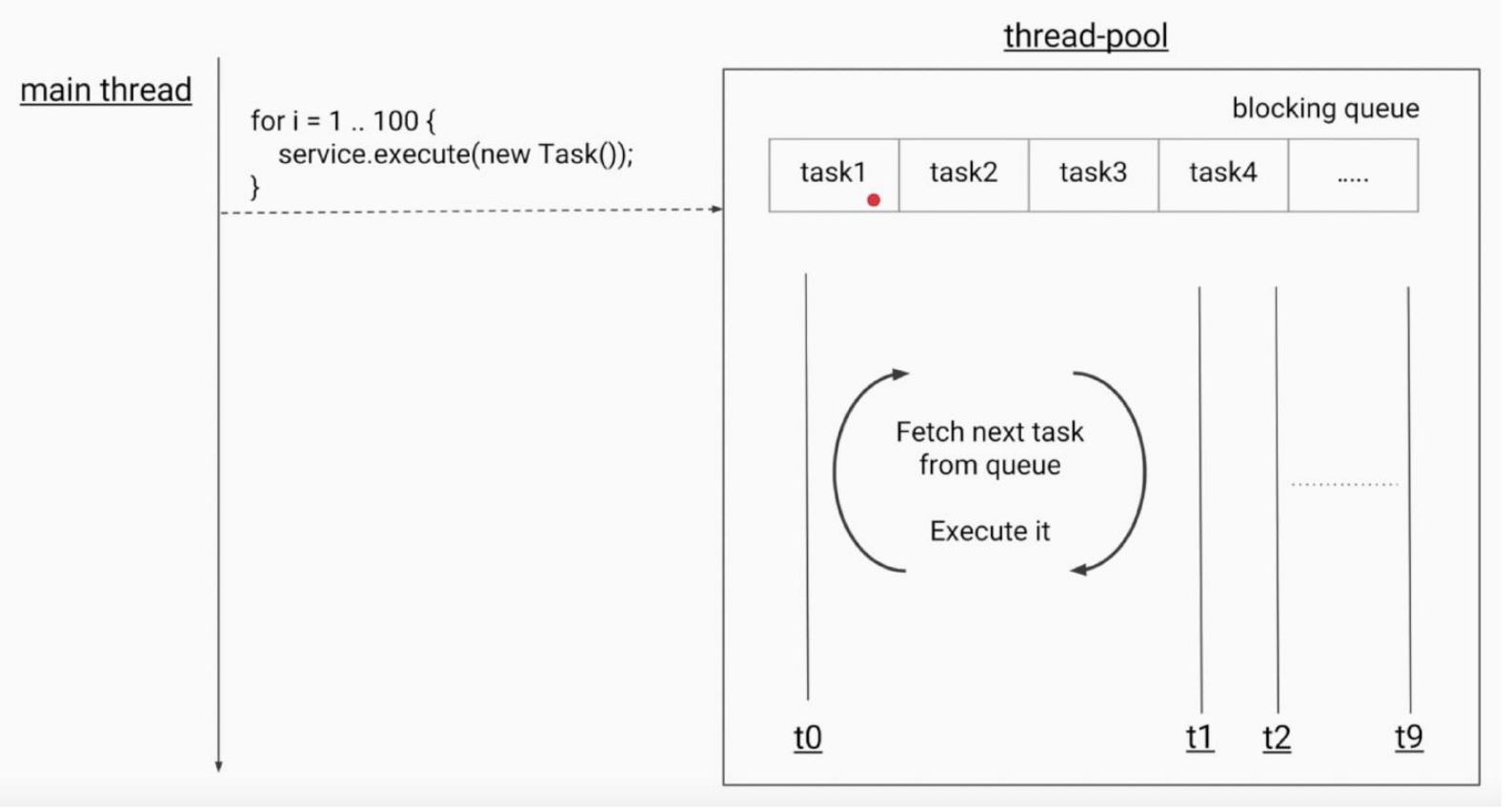
线程数=CPU核心数*（1+平均等待时间/平均工作时间）

常见线程池的特点

- FixedThreadPool
- CachedThreadPool
- ScheduledThreadPool
- SingleThreadExecutor

常见线程池的特点

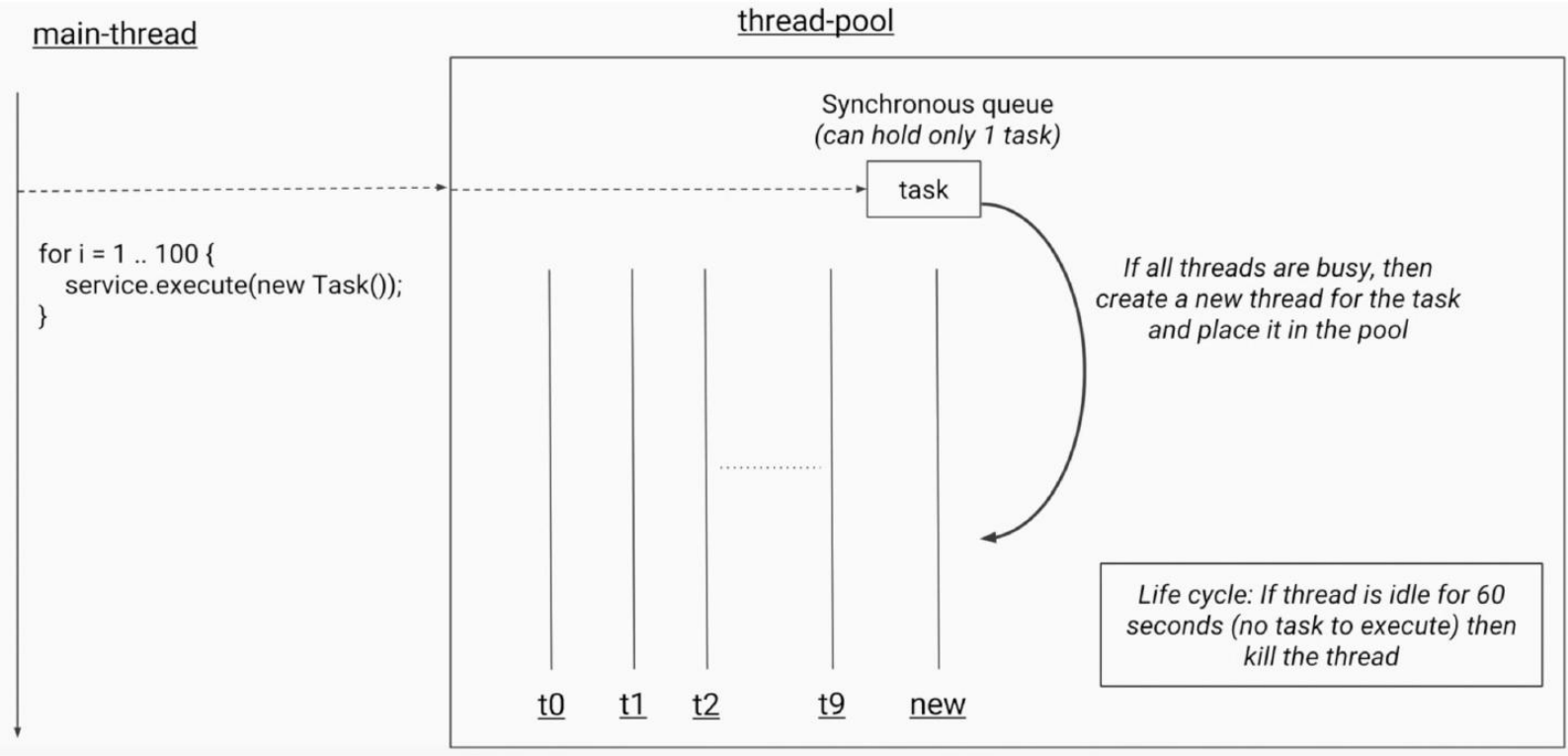
- FixedThreadPool



CachedThreadPool

- 可缓存线程池

特点: 具有自动回收多余线程的功能



ScheduledThreadPool

- 支持定时及周期性任务执行的线程池

SingleThreadExecutor

- 单线程的线程池：只会用唯一的工作线程来执行任务
- 原理和FixedThreadPool是一样的，但是此时的线程数量被设置为了1

以上4种线程池的构造方法的参数

Parameter	FixedThreadPool	CachedThreadPool	ScheduledThreadPool	SingleThreaded
corePoolSize	constructor-arg	0	constructor-arg	1
maxPoolSize	same as corePoolSize	Integer.MAX_VALUE	Integer.MAX_VALUE	1
keepAliveTime	0 seconds	60 seconds	0	0 seconds

阻塞队列分析

- FixedThreadPool和SingleThreadExecutor的Queue是LinkedBlockingQueue?
- CachedThreadPool使用的Queue是SynchronousQueue?
- ScheduledThreadPool来说，它使用的是延迟队列DelayedWorkQueue

workStealingPool是JDK1.8加入的

- 这个线程池和之前的都有很大不同
- 子任务
- 窃取

停止线程池的正确方法

1、shutdown

2、isShutdown

3、isTerminated

4、awaitTermination

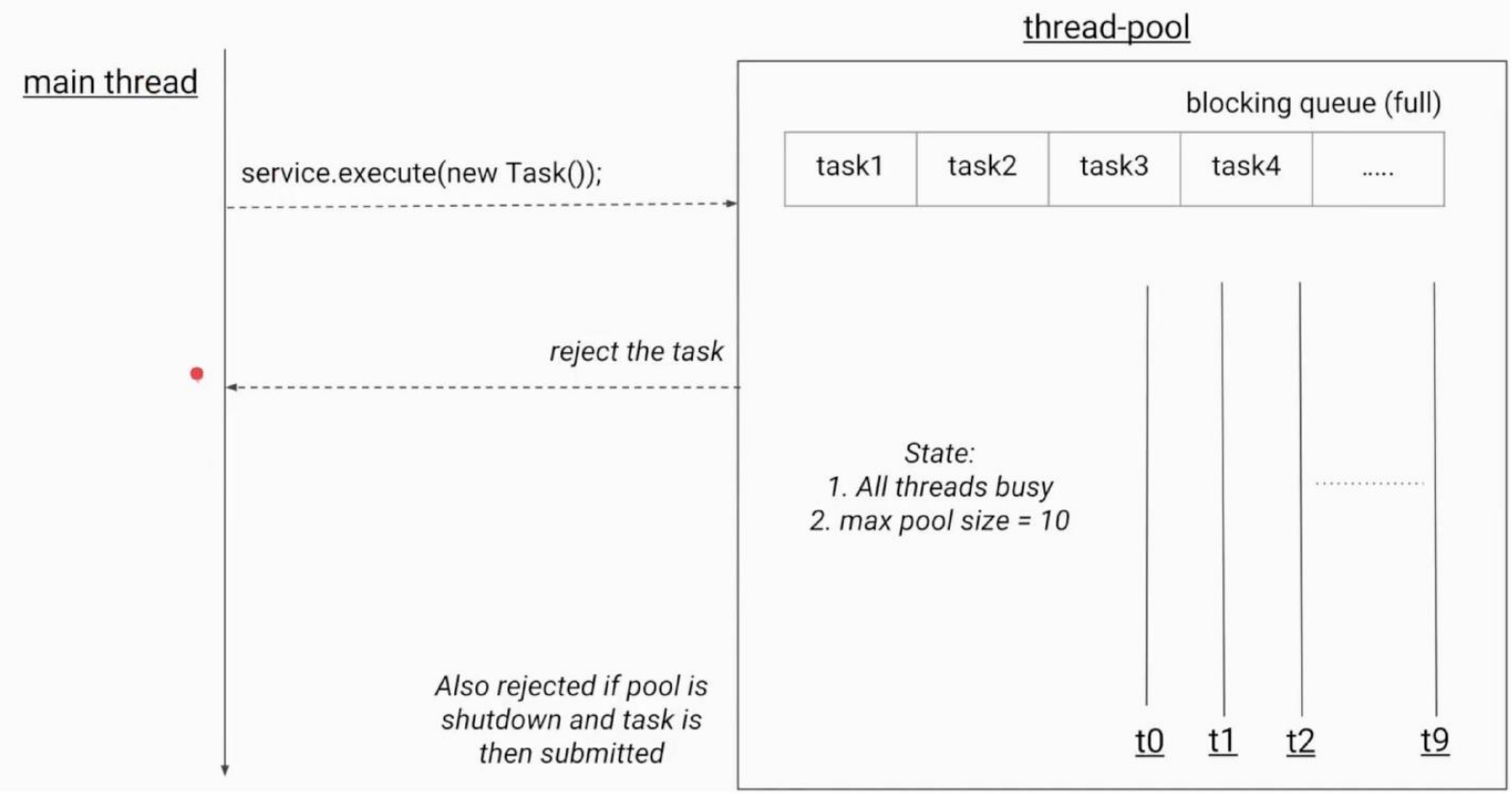
5、shutdownNow

任务太多，怎么拒绝?

◆ 拒绝时机

1.当Executor关闭时，提交新任务会被拒绝。

2.以及当Executor对最大线程和工作队列容量使用有限边界并且已经饱和时



4种拒绝策略

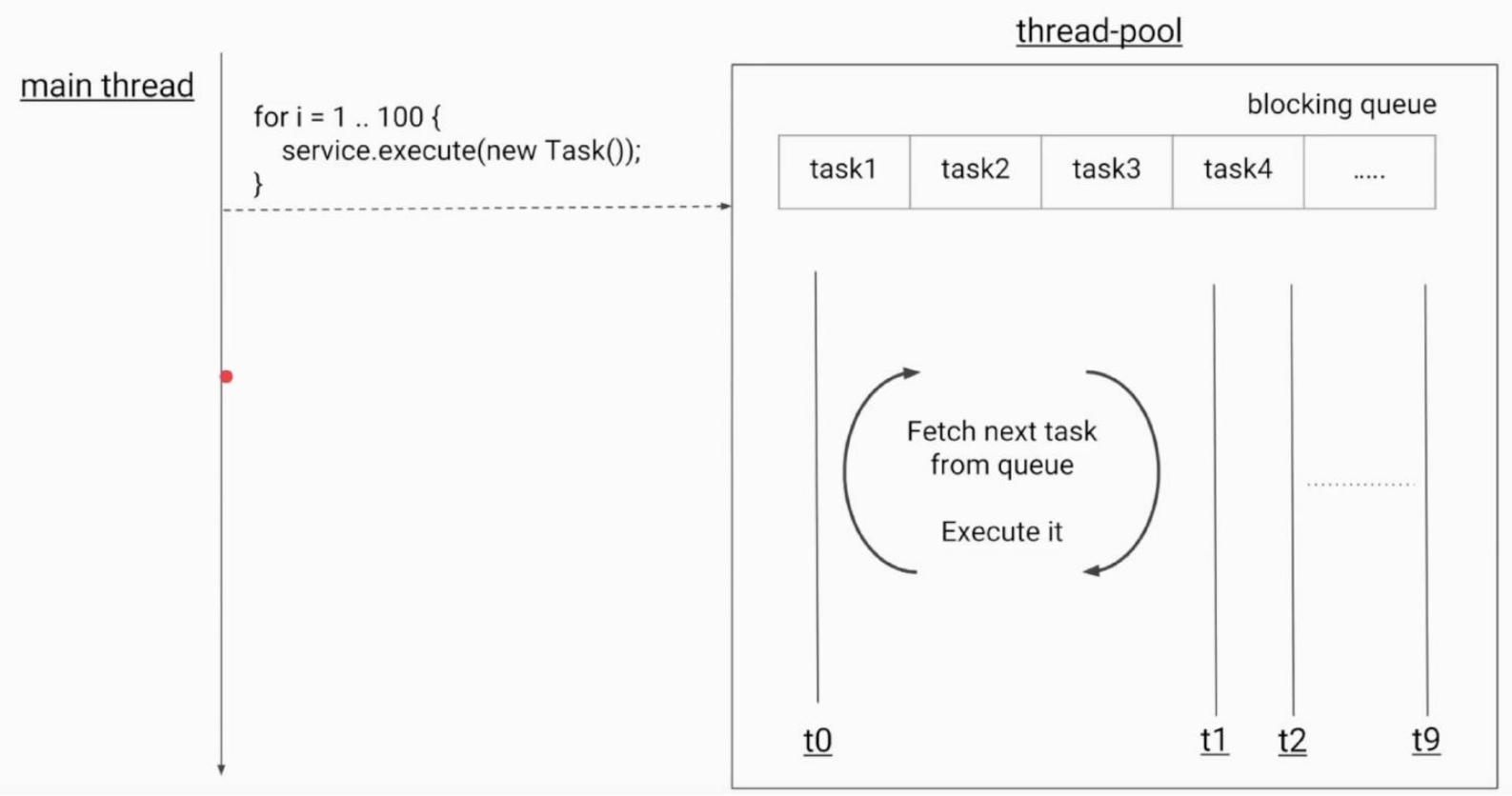
- AbortPolicy
- DiscardPolicy
- DiscardOldestPolicy
- CallerRunsPolicy

钩子方法，给线程池加点料

- 每个任务执行前后
- 日志、统计
- 代码演示

实现原理、源码分析

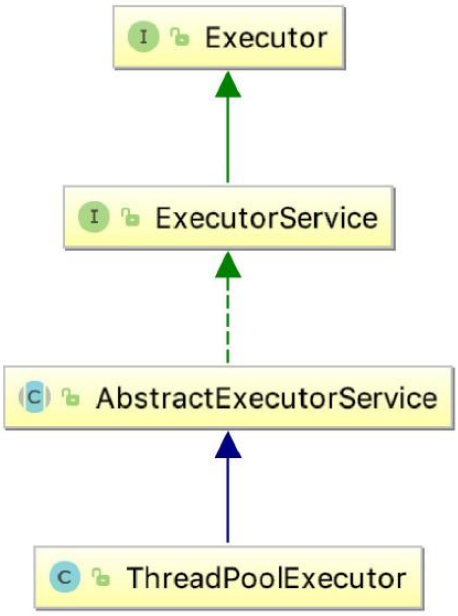
- 线程池组成部分
  - 线程池管理器
  - 工作线程
  - 任务队列
  - 任务接口（Task）



Executor家族?

- 线程池、ThreadPoolExecutor、ExecutorService、Executor、Executors等这么多和线程池相关的类，都是什么关系?

哪个是线程池?



Executor家族?

- Executor
- ExecutorService
- Executors

线程池实现任务复用的原理

- 相同线程执行不同任务
- 源码分析

线程池状态

状态	说明
RUNNING	接受新任务并处理排队任务
SHUTDOWN	不接受新任务，但处理排队任务
STOP	不接受新任务，也不处理排队任务，并中断正在进行的任务
TIDYING	所有任务都已终止，workerCount为零时，线程会转换到TIDYING状态，并将运行terminate（）钩子方法
TERMINATED	terminate（）运行完成

execute方法

- 自己实现最简单的线程池
- 然后我们再来看看ThreadPoolExecutor对execute的实现

线程工厂

- 线程工厂为线程设置了如下内容：
  - 默认名字（pool-线程池自增编号-thread-线程的自增编号）
  - 非守护线程
  - 默认优先级
  - 默认线程组

使用线程池的注意点

- 避免任务堆积
- 避免线程数过度增加
- 排查线程泄漏
- 和ThreadLocal配合

总结：线程池——治理线程的法宝

1. 线程池的自我介绍
2. 创建和停止线程池
3. 常见线程池的特点和用法

总结：线程池——治理线程的法宝

4. 任务太多，怎么拒绝？
5. 钩子方法，给线程池加点料
6. 实现原理、源码分析
7. 使用线程池的注意点