若某一任务需要频繁创建和销毁线程，而任务执行的时间又非常短，这样线程创建和销毁的带来的开销就不容忽视。如网络缓冲区内的数据包，若按传统单线程顺序执行，系统处理速度无法达到要求，若每份数据包都创建一个线程去处理，那么创建和销毁该线程的时间可能远大于处理该数据包的时间。此时，若使用线程池，就能够使得线程和任务分离，提升线程的可重用性；降低服务器压力，控制线程数量，集中管理全部线程；既并行处理数据包达到吞吐量要求，又避免重复创建和销毁线程导致额外开销。设计一个线程池，重点在于合理设置线程池的以下4个参数：

1. **核心线程数（corePoolSize）**

核心线程数需要依据单个任务处理时间T\_process和每秒产生的任务数量N来确定，计算公式如下：

corePoolSize = T\_process \* N

例如：某一任务的处理时间T\_process = 0.1s，系统大部分时间里每秒产生的任务数量为N = 100pkt/s，那么，要想在1秒内处理完这100个任务，需要的线程数为：

corePoolSize = T\_process \* N = 0.1 \* 100 = 10

此时我们将系统的核心线程数设计为10。实际系统肯定存在任务数量的波动，我们按照能够满足80%的情况来设计核心线程数即可，其余20%的情况开启临时线程（受最大线程数限制）进行处理。

1. **任务队列长度（workQueue）**

任务队列长度依据核心线程数corePoolSize和单个任务处理时间T\_process而定，计算公式如下：

workQueue = 2 \* corePoolSize / T\_process

上述环境中，核心线程数corePoolSize 设计为10，单个任务处理时间T\_process为0.1s，则队列长度设计为：

workQueue = 2 \* corePoolSize / T\_process = 2 \* 10 / 0.1 = 200

1. **最大线程数（maximumPoolSize）**

最大线程数需要依据系统每秒产生的最大任务数N\_max、任务队列长度workQueue和单个任务处理时间T\_process来确定，计算公式如下：

maximumPoolSize = (N\_max - workQueue) \* T\_process

上述环境中，假设系统每秒最大产生的任务数N\_max = 1000，那么最大线程数设计为：

maximumPoolSize = (N\_max - workQueue) \* T\_process = (1000 - 200) \* 0.1 = 80

1. **最大空闲时间（keepAliveTime）**

该参数没有固定的参考值和计算公式，由系统运行环境和硬件压力设定，系统设计者根据系统产生任务的时间间隔合理设置一个值即可。



图2 线程池基本结构

对于线程池运行的基本原理，以下做简要介绍（假设核心线程数为2，最大线程数为3，任务队列最大任务数为1）：

A任务需要处理，通知线程池，但程序刚运行时，线程池中初始线程数量为0。于是管理线程创建1号线程执行A任务。

在A任务尚未处理完毕时，B任务需要处理，于是管理线程创建2号线程执行B任务。

在A、B任务都未处理完毕时，C任务需要处理，但此时核心线程数已满，于是管理线程将C任务放入任务队列中排队等候。在A、B业务中的任一业务处理完毕后，C业务就可以执行。

此时，D任务需要处理，而A、B任务都未处理完毕，C任务尚在排队等待，但此时任务队列已满，于是管理线程创建3号线程（临时线程）处理D任务。

假设前面的任务都未处理完毕，E任务又需要处理，此时只能按照饱和处理机制，拒绝处理E任务，管理线程按照设置参数报警或直接忽略。

当D任务处理完后，3号线程（临时线程）开始计时，若空闲时间超过最大空闲时间，3号线程将被销毁。

但为了保证接下来的业务正常执行，即使1号和2号线程（核心线程）空闲，也不能够销毁，线程池内必须保持核心线程数量。