1. Executor框架
2. 概述

从JDK5开始，java线程的工作单元和执行机制分离开来，工作单元包括Runnable和Callable，执行机制由Executor框架提供。

1. Executor框架结构
2. 两级调度模型

在HotSpot VM的线程模型中，Java线程被一对一地映射为本地操作系统的线程，操作系统会调度所有线程并将它们分配给可用的CPU。在上层，Java多线程通常将应用分解为若干个任务，然后使用用户级的调度器（Executor框架）将任务映射为固定数量的线程；在底层，操作系统将这些线程映射到硬件处理器上。

1. Executor结构和成员

Executor框架主要由三大部分组成：

1. 任务

被执行任务需要实现如下的接口，Runnable接口和Callable接口，都可以被ThreadPoolExecutor或者ScheduledThreadPoolExecutor执行。

1. 任务的执行

任务执行机制的核心接口Executor，以及子接口ExecutorService，同时还有该接口的实现类ThreadPoolExecutor和ScheduledThreadPoolExecutor，用来执行被提交的任务。

1. 异步计算的结果

结果接口Future和实现类FutureTask类。

1. Executor执行过程

主线程首先创建实现Runnable或者Callable接口的任务对象，工具类Executors可以把一个Runnable对象封装为Callable对象，然后将Runnable对象交给ExecutorService执行，调用execute()方法或者submit()方法，submit()方法将返回一个Future接口的对象，最后主线程执行FutureTask.get()方法等待任务执行完成，同时可以执行cancel()方法来取消任务的执行。

1. ThreadPoolExecutor详解
2. 概述

Executor框架最核心的类是ThreadPoolExecutor，它是线程池的实现类，可以通过构造函数构造一个线程池，也可以通过Executors工具类来创建三种类型的线程池，FixedThreadPool，SingleThreadExecutor，CachedThreadPool。

1. FixedThreadPool

可重用固定线程数的线程池，当线程池中的线程数大于核心线程数，keepAliveTime为多余的空闲线程等待新任务的最长时间，超过这个时间的多线程则被终止。采用的工作队列为无界队列LinkedBlockingQueue，最大的容量为Integer.MAX\_VALUE，那么最大线程数就是一个无意义的参数，因为工作队列永远不会满，也永远不会拒绝任务。

1. SingleThreadExecutor详解

使用单个worker线程的Executor，核心线程数和最大线程数都为1，工作队列为无界队列，最大线程数也是无意义的参数，因为工作队列永远不会满，也永远不会拒绝任务。

1. CachedThreadPool详解

根据需要创建新的线程的线程池，核心线程数为0，最大线程数为Integer.MAX\_VALUE，使用SynchronousQueue，队列容量为1，如果提交任务的速度高于处理的速度，则会一直创建线程，消耗内存和CPU。

1. FutureTask详解
2. FutureTask概述

FutureTask除了实现了Future接口，还实现了Runnable接口，因此FutureTask可以交给Executor执行，也可以由调用线程直接执行(FutureTask.run())。

1. FutureTask使用

通过ExecutorService.submit()方法返回一个FutureTask，然后执行get()方法获取返回结果或者cancel()方法取消任务执行，还可以单独使用FutureTask。

1. FutureTask的实现

基于AQS实现的，juc中的很多同步组件都是通过AQS实现的，AQS是一个同步框架，提供通用机制来原子性地管理同步状态，阻塞和唤醒线程，以及维护被阻塞线程的队列。基于AQS实现的同步组件有ReentrantLock, Semaphore, ReentrantReadWriteLock, CountDownLatch和FutureTask。

1. 线上问题定位

在Linux上使用TOP命令查看每个线程的状态，如果某个线程的CPU利用率一直为100%，则该线程可能是死循环；如果某线程一直在TOP10的位置则该线程可能有性能问题；也可以将线程通过jstack命令dump到文件中查看。