

1. 常用类

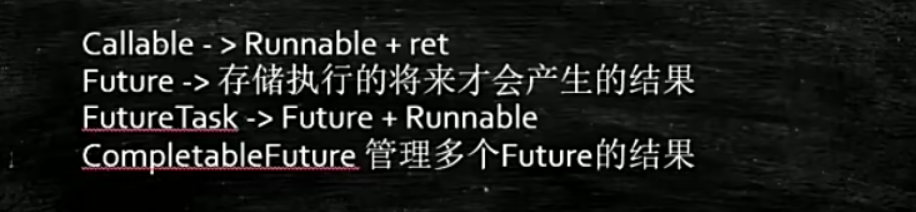
* Executor：执行器，执行Runnable

execute(Runnable)，没有返回值

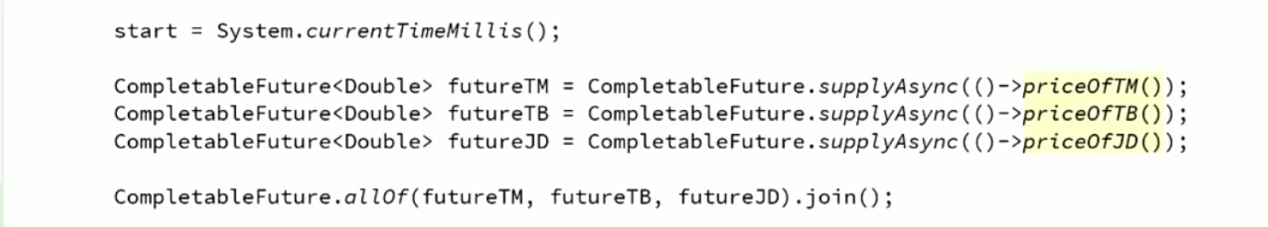
* ExecutorService：所有线程池都继承ExecutorService

submit(Runnable、Callable)，有返回值Future

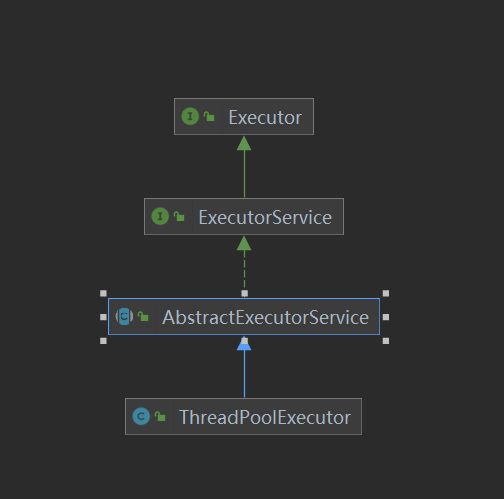
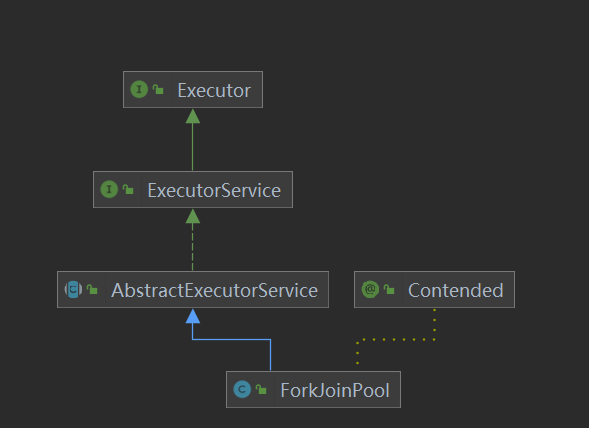
* Callable：任务（Runnable + return value）
* Future：将来产生的执行结果
* FutureTask：任务+将来产生的执行结果（Runnable+Future）



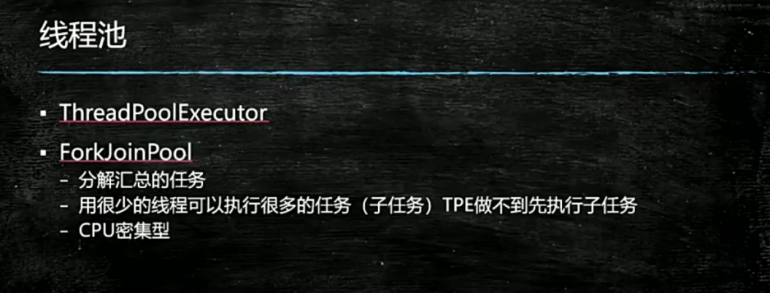
* CompletableFuture：可以管理所有任务的执行结果，不用像Future那样一个一个get()



1. 两种线程池

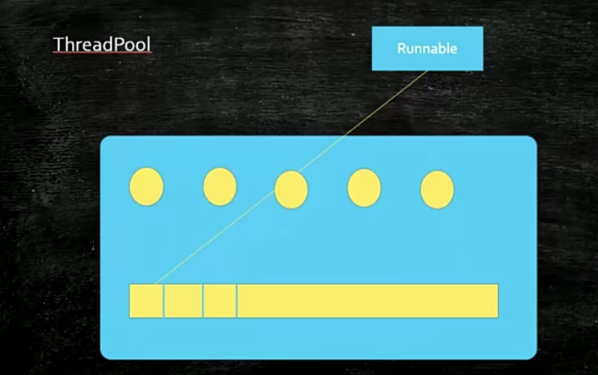
两种线程池：ThreadPoolExecutor、ForkJoinPool



1. ThreadPoolExecutor：所有线程共享一个任务队列

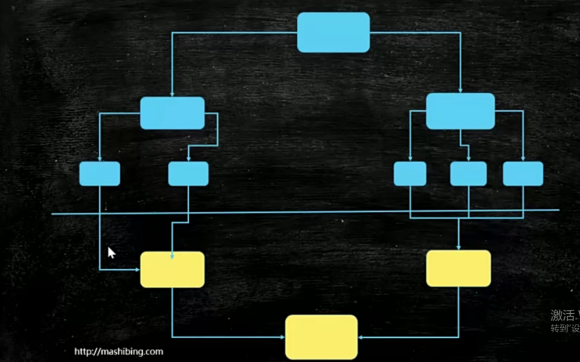
自定义7个参数的ThreadPoolExecutor

JDK自带的4个线程池：sigle、cache、fixed、scheduled



1. ForkJoinPool：每个线程都有一个任务队列，分解任务、汇总任务

ForkJoinPool



1. ThreadPoolExecutor线程池流程
2. 定义任务：Callable有返回值、Runnable无返回值、FutureTask（Runnable + Future）
3. 定义线程池：自定义参数ThreadPoolExector及四种自带常量池
4. 提交任务：把任务交给线程池异步执行，返回Future用于存储任务执行结果。

* submit(Callable、Runnable) 有返回值Future
* execute(Runnable) 没有返回值

1. 获得执行结果：可以通过future.get()阻塞获取执行结果。

如果想要有返回值，就Callable、submit

1. ForkJoinPool线程池流程
2. 定义总任务：

* 需要分解任务：RecursiveTask有返回值、RecuriveAction无返回值；

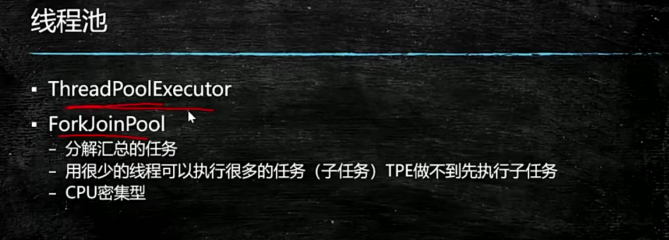
重写compute方法：分解、合并任务（fork：执行子任务；join：合并子任务结果）

* 也可以callable、Runnable

1. 定义线程池：new ForkJoinPool()
2. 提交总任务：把任务交给线程池异步执行，返回Future用于存储任务执行结果。

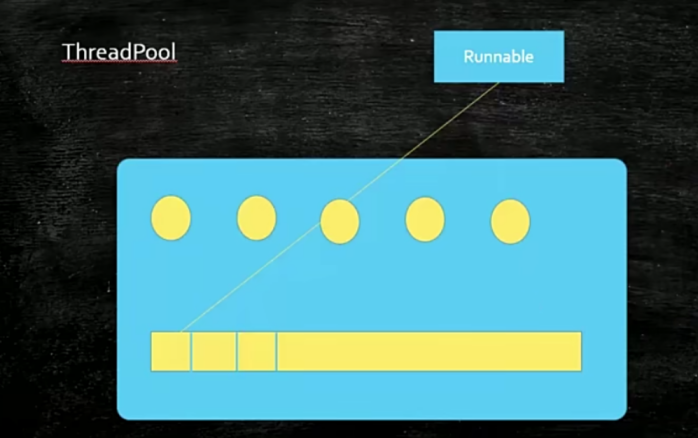
* submit() 有返回值Future
* execute() 没有返回值

1. 获得执行结果：可以通过task.join()或者Future.get()。
2. ThreadPoolExecutor线程池参数

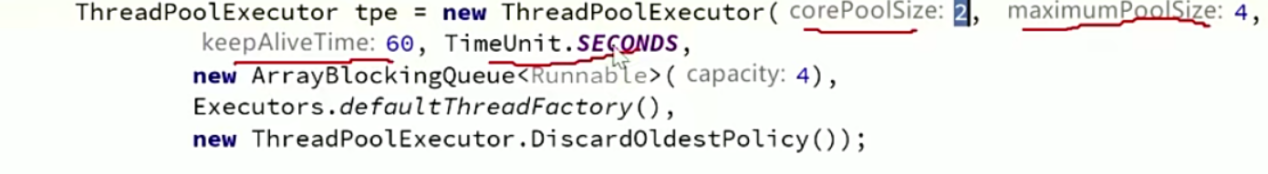


1. 线程池结构

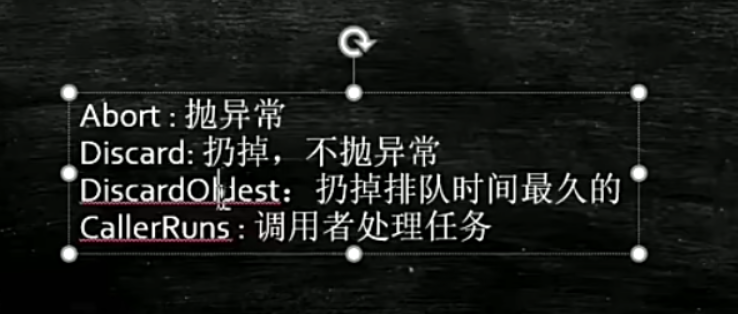
* 线程集合：用于执行任务
* 任务集合



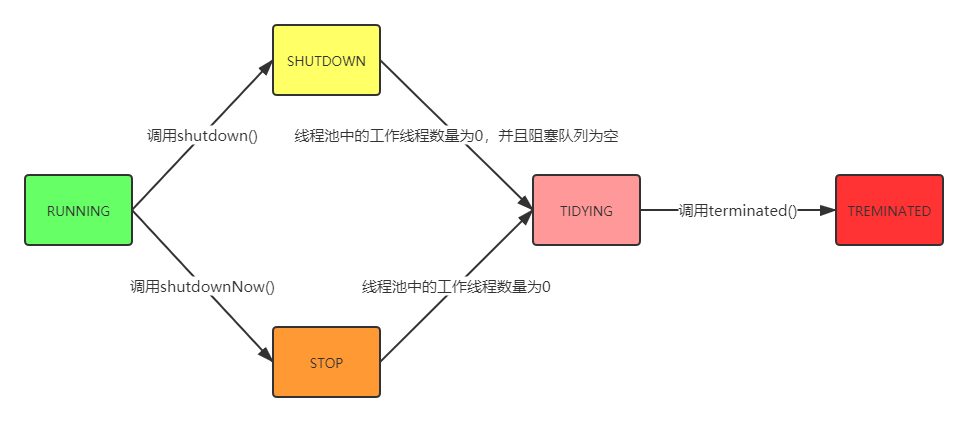
1. 线程池7个参数



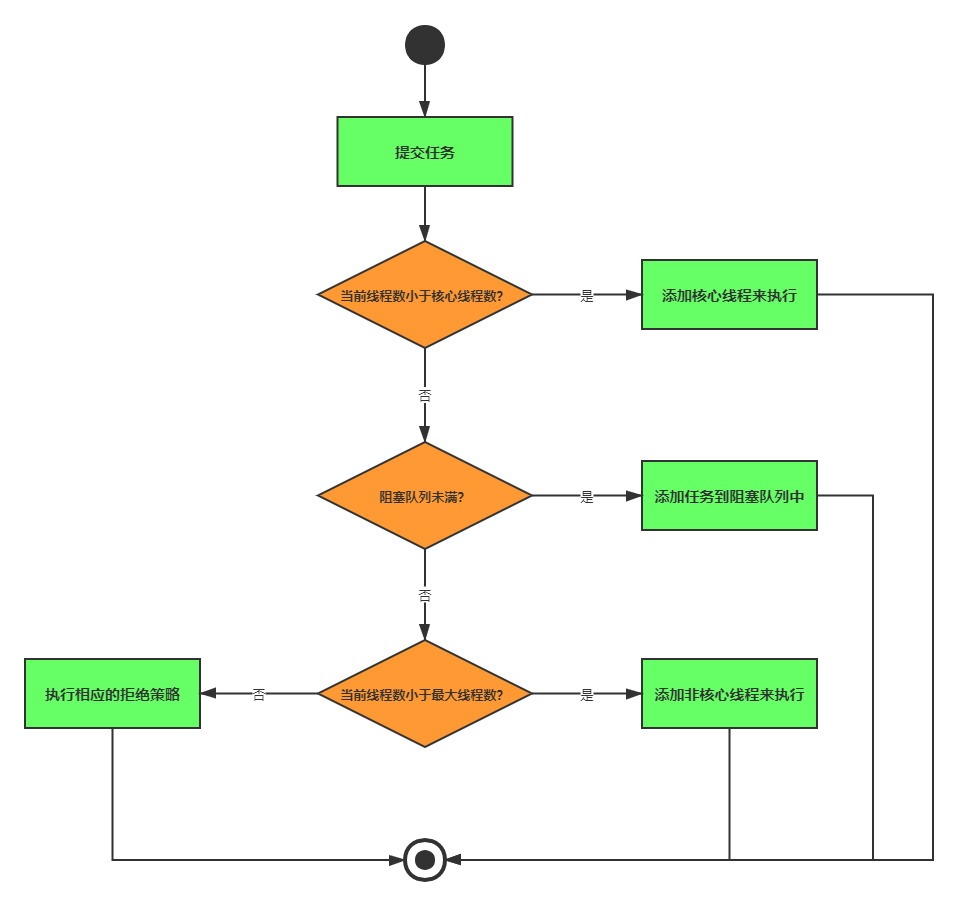
* 核心线程数
* 最大线程数
* 生存时间：非核心线程如果超过生存时间不干活，则将线程资源归还给操作系统，避免消耗资源；核心线程永远活着
* 生存时间单位：秒、毫秒、纳秒等等
* 任务队列：ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue、LinkedTransferQueue
* 线程工厂：线程创建的方式
* 拒绝策略：所有线程都没空，且任务队列都满了情况下，执行拒绝策略（JDk提供4种如下，但是可以自定义）。一般是自定义，把任务保存到Kafak或者MySQL



1. 线程池状态



1. 执行流程



手工定义线程池

阿里要求手工定义线程池，方便出错时回溯。