**线程池**

private static void testExecutors() {

//创建一个线程池

ThreadPoolExecutor executor = new ThreadPoolExecutor(4, 10, 10, TimeUnit.SECONDS, new LinkedBlockingDeque<>());

//创建一个任务

Runnable task = () -> System.out.println("this is a task");

//提交任务

executor.execute(task);

}

corePoolSize: 核心线程数  
maximumPoolSize: 最大线程数  
keepAliveTime: 空闲线程存活时间，0表示永久  
TimeUnit: 空闲时间存活时间单位  
BlockingQueue: 任务存放队列  
ThreadFactory: 线程工厂，用来定制线程，一般用默认的Executors.defaultThreadFactory()就行  
RejectedExecutionHandler: 拒绝策略，默认是AbortPolicy，也就是直接抛出异常

#### 阻塞队列

所谓阻塞队列，指的是任务轮不到执行时的存放地方

 LinkedBlockingQueue: 基于链表的阻塞队列，可以指定最大长度，默认无界

 ArrayBlockingQueue: 基于数组的有界队列

 PriorityBlockingQueue: 基于堆(heap)的无界优先级队列

 SynchronousQueue: 没有存储空间的同步阻塞队列

这里面的**有界**和**无界**很重要，直接关系到线程池的逻辑，如果用的无界队列，线程个数最多只能达到corePoolSize，达到corePoolSize后，新任务全部排队(因为无界队列是无限的，永远不会满)，最大线程数(maximumPoolSize)参数就没有意义了。如果是有界队列，线程个数达到corePoolSize后，将会进入有界队列，有界队列满之后，就会创建新线程，直到线程数达到maximumPoolSize为止。对于SynchronousQueue比较特殊，因为没有存储空间(可以理解为有界队列，但是容量是0)，所以任务来临时，如果没有空闲线程，则会一直创建新线程，直到达到maxmimumPoolSize为止。

#### 线程工厂

线程工厂就是用来创建执行线程的，只有一个创建线程的方法。

public interface ThreadFactory {

Thread newThread(Runnable r);

}

#### 拒绝策略

如果任务得不到执行，而且阻塞队列也满了，就会处罚拒绝策略，拒绝策略都是ThreadPoolExecutor的静态内部类。

 ArbotPolicy: 默认的处理方式，直接抛出RejectedExecutionException异常

 DiscardPolicy: 直接丢弃新任务

 DiscardOldestPolicy: 直接丢弃最老的任务，也就是等待时间最长的那个任务

 CallerRunsPolicy: 直接在提交任务的线程执行

Tips: 核心线程不会预先创建，只有有任务时才会创建，但是可以通过prestartAllCoreThreads()函数来预先创建。核心线程不会因为空闲被终止，但是可以通过allowCoreThreadTimeOut(true)来允许空闲终止。

#### Executors

Executors是一个创建线程池的工具类，类似于Collections，Arrays，以后凡是找工具类优先找Xxxxs。

//创建一个单线程的线程池，核心线程为1，最大线程为1，永不超时。适用于排队执行任务

public static ExecutorService newSingleThreadExecutor();

//创建一个指定线程的线程池，核心线程和最大线程都为nThreads，永不超时。适用于少量长时间执行的任务

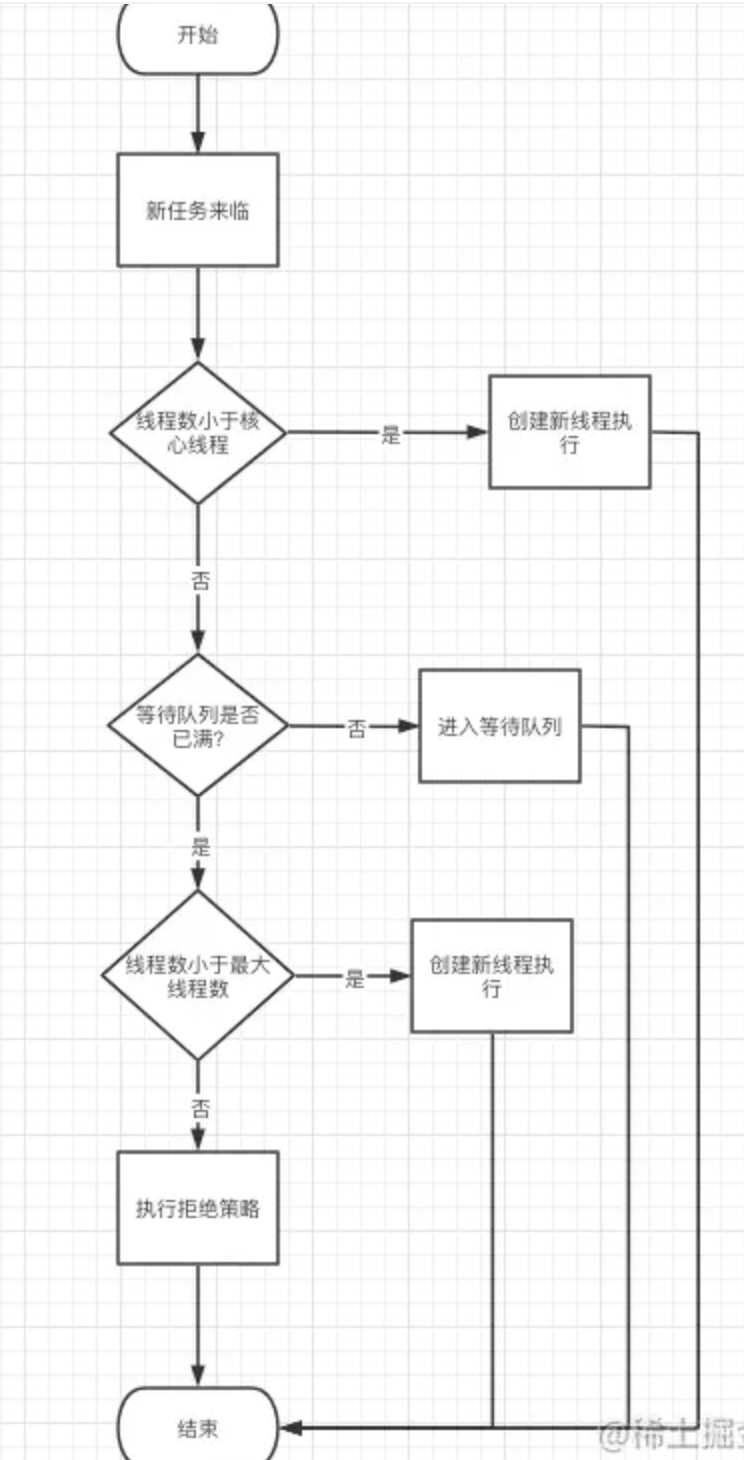
public static ExecutorService newFixedThreadPool(int nThreads);

//创建一个缓存的线程池，核心线程为0，最大线程为Integet.MAX\_VALUE，超时时间是60s。适用于大量段时间执行的任务

public static ExecytorService newCachedThreadPool();

#### 执行流程

源码流程: 当有任务要执行时，如果当前线程个数小于"核心线程数(corePoolSize)"，就会创建一个新线程(核心线程)来执行该任务，即使有空闲线程也会创建新线程。如果线程个数大于"核心线程数"，就会尝试排队，也就是加入阻塞队列，如果队列满了或者其他原因导致不能入队，就会检查线程个数是否达到了"最大线程数(maximumPoolSize)"，如果没达到，就会继续创建新线程(非核心线程)，否则将会执行拒绝策略。如果一个"非核心线程"在"存活时间(keepAliveTime)"时间内没有任务执行，那么会被终止，如果keepAlivetime=0，那么所有线程都不会超时终止。核心线程不会超时终止，但是可以通过函数allowCoreThreadTimeOut(boolean)来指定核心线程也可以超时终止。



Tips: 线程池执行任务由于是并行的，所以可能存在死锁，这里一定要注意。

**异步任务**

异步任务是基于线程池的，线程池就是异步任务的执行器。

不带返回结果的使用Runnable

private void test() {

//创建线程池

ExecutorService executor = Executors.newSingleThreadExecutor();

//创建任务

Runnable task = () -> System.out.println("this is a task");

//执行任务

executor.execute(task);

}

带返回结果的使用Callable，返回结果是Future

private void test() {

//定义任务

Callable<String> callable = () -> {

//模拟耗时操作

Thread.sleep(3000);

return "hello";

};

//创建执行器

ExecutorService executorService = Executors.newSingleThreadExecutor();

//提交任务

Future<String> future = executorService.submit(callable);

try {

//阻塞获取执行结果，只要结果没返回，就会一直阻塞

String result = future.get();

//关闭执行器

executorService.shutdown();

} catch (InterruptedException | ExecutionException e) {

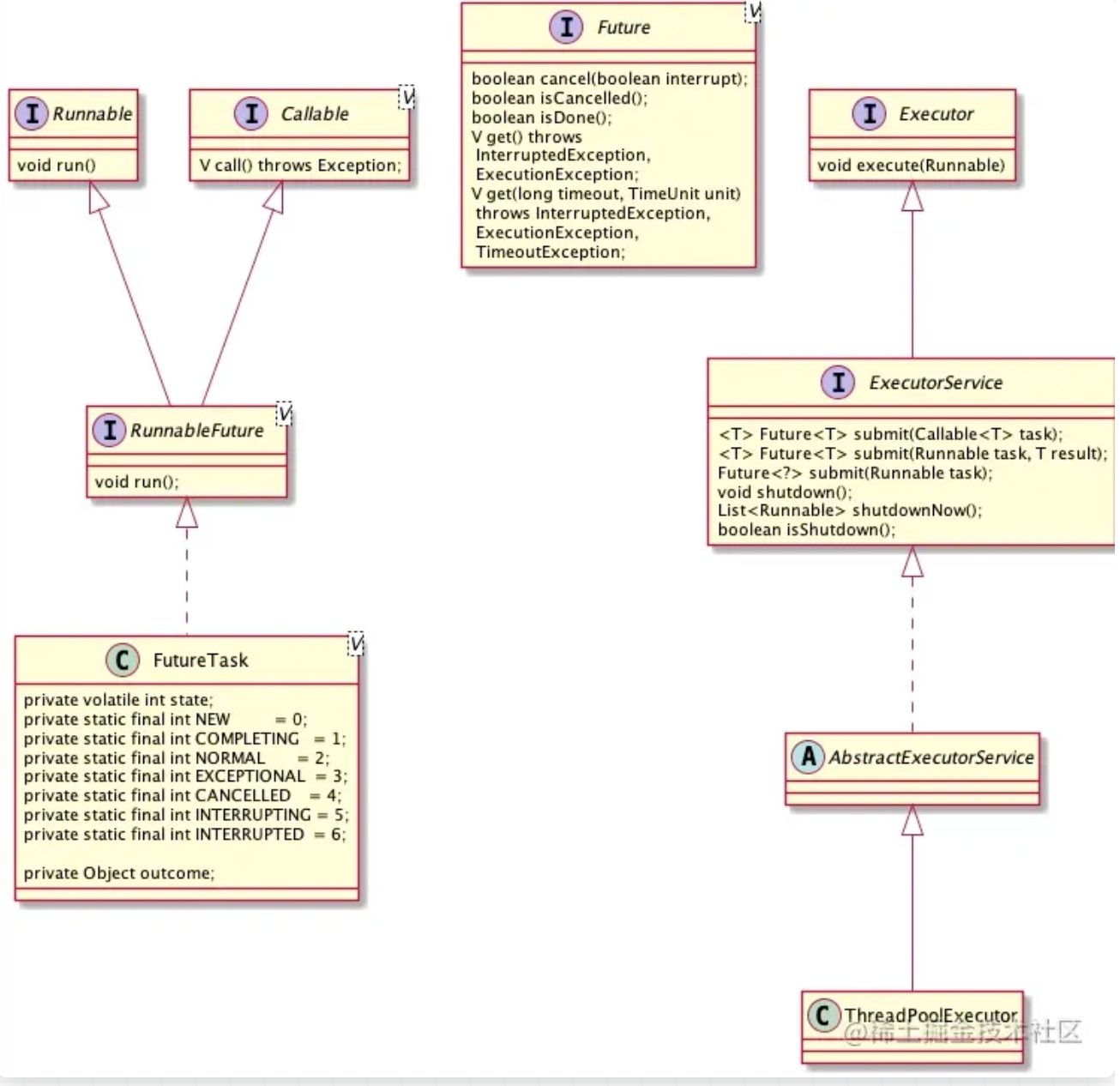
e.printStackTrace();

}

}

* 任务: Runnable无返回结果，不会抛出异常；Callable有返回结果，会抛出异常
* 执行器: Executor，是个接口，有各种实现，用来执行任务
* 结果: Future，表示异步任务的结果，可以使用get()获取执行结果

我们来看下异步任务家族图谱:



我们使用Callable创建一个任务，然后使用Executor来执行任务，得到一个Future，使用Future.get()来阻塞获取结果，如果正常完成，会获取到执行结果，如果抛出了异常或者被取消或者任务被中断，则会抛出相应异常。

异步任务的源码流程很简单，提交一个任务后，会创建一个FutureTask，里面包含了要执行的任务Runnable和需要的结果Object，使用线程池去执行Runnable同时更新state，如果执行完毕或者抛出异常，就会给Object赋值同时更新状态state，是否Future.get()获取结果的时候，先检测state，如果state>COMPLETING(表示已经执行完毕)，就返回结果Object，否则阻塞等待，等待过程中如果执行完毕，就会唤醒等待的线程，此时Future.get()就返回了结果。

Future实现了"任务的提交"和"任务的执行"以及"任务的结果"互相分离的功能，这是一种"粒度细化"的思想。