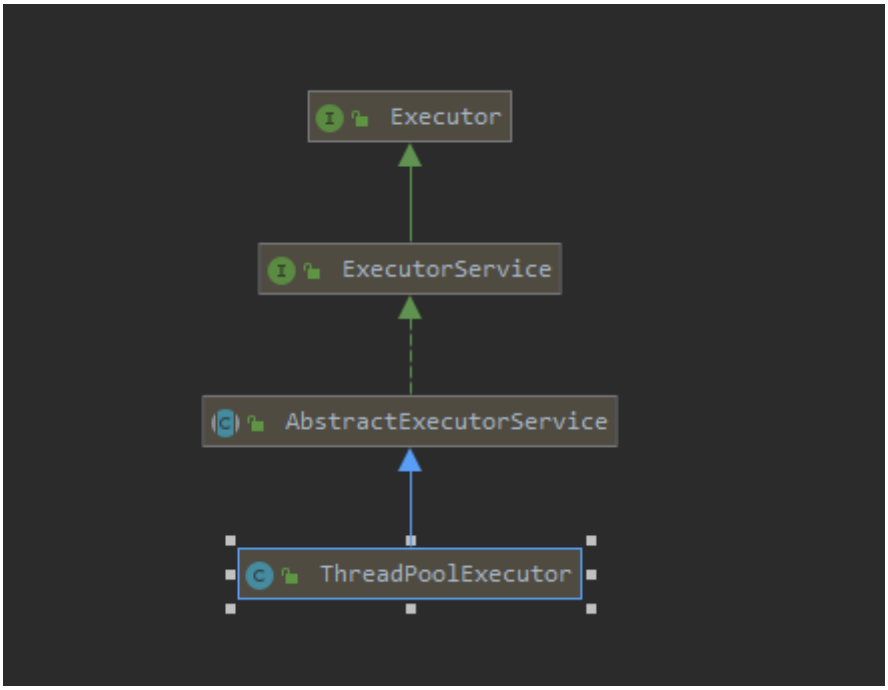
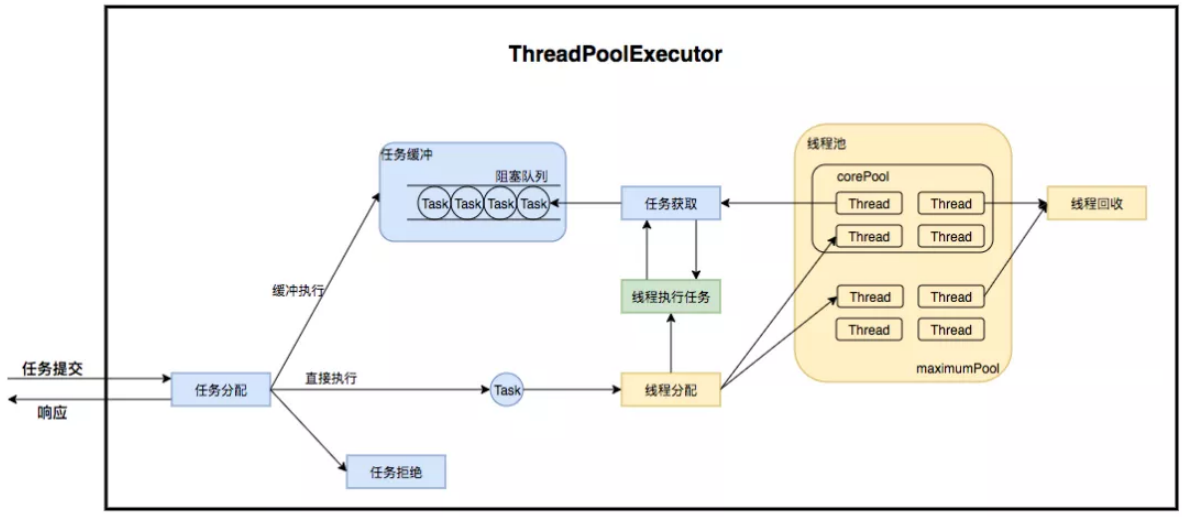
线程池

1.UML类图



> 顶层接口是Executor，只有一个方法execute，只是提供了线程池的一种思想：将任务提交与任务执行解耦，用户只需要调用execute将待执行的任务提交给线程池即可，至于线程池如何创建创建线程，如何调度线程去执行这个任务对用户来说是透明的。
>
> ExecutorService接口增加了一些能力：（1）扩充执行任务的能力，补充可以为一个或一批异步任务生成Future的方法(submit与invokeAll/invokeAny方法)；（2）提供了管控线程池的方法，比如停止线程池的运行(shutdown/shutdownNow)。
>
> AbstractExecutorService抽象类是ExecutorService的实现之一，主要实现了submit方法与invokeAll/invokeAny方法，关注的是任务的执行部分；
>
> ThreadPoolExecutor实现了最复杂的管理任务与线程，管理线程池状态的部分(核心的execute方法的就是在这部分实现的)

2.



线程池构建了一个生产者消费者模型，将任务与执行任务的线程分开，从而实现线程池的核心复用线程执行任务。任务的管理看作是生产者，任务提交到线程之后，有三种路径1)直接交由一个工作线程执行该任务；2)放入阻塞队列，等待有空闲的线程从阻塞队列中取出该任务执行；3）直接拒绝该任务

工作线程的管理就是消费者了，工作线程在线程池中统一维护，根据任务请求进行线程的调度分配，如果某个线程已经处理完一个任务，那么可能会分配给该线程另一个任务去执行。没有任务要执行了，线程就被回收了。

至此可以将线程池分为三部分：线程池自身的状态维护、任务的管理、线程的管理

3.Doug Lea设计线程池的时候，使用了一个AtomicInteger类型的变量ctl来同时表示线程池的状态与线程池中有效线程的数目，其中高三位表线程池状态，低29位表示线程数目，由于线程池的状态与线程池的数目密切相关，所以在操作时候如果单独使用两个变量来维护，那么就需要引入锁资源，而Doug Lea巧妙的使用了一个原子类来表示两种状态，避免的锁的引入(这点在读写锁中也有体现)；

```
// runState is stored in the high-order bits
private static final int RUNNING    = -1 << COUNT_BITS;
private static final int SHUTDOWN   =  0 << COUNT_BITS;
private static final int STOP       =  1 << COUNT_BITS;
private static final int TIDYING    =  2 << COUNT_BITS;
private static final int TERMINATED =  3 << COUNT_BITS;
```

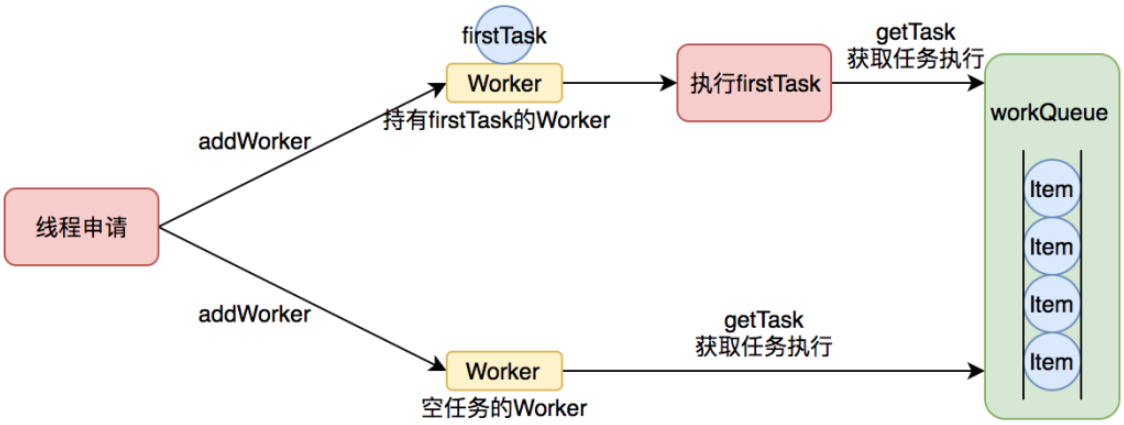
> RUNNING:线程池初始状态，处于该状态的线程可以接受新任务，处理已添加的任务，一般来说线程池被创建的时候线程池就处于该状态；
>
> SHUTDOWN：不再接受新的任务，但可以处理已经添加的任务，包括阻塞队列中的任务；
>
> 调用线程池的shutdown()接口时，线程池由RUNNING -> SHUTDOWN。
>
> STOP：不接收新任务，不再处理阻塞队列已添加的任务，并且会尝试中断正在处理的任务；
>
> 调用线程池的shutdownNow()接口时，线程池由(RUNNING or SHUTDOWN) -> STOP。
>
> TIDYING：当所有的任务已终止，ctl记录的”任务数量”为0，线程池会变为TIDYING状态。
> 当线程池变为TIDYING状态时，会执行钩子函数terminated()。
>
> 当线程池在SHUTDOWN状态下，阻塞队列为空并且线程池中执行的任务也为空时，就会由SHUTDOWN -> TIDYING。
> 当线程池在STOP状态下，线程池中执行的任务为空时，就会由STOP -> TIDYING。
>
> TERMINATED ： 线程池彻底终止，就变成TERMINATED状态。
>
> 线程池处在TIDYING状态时，执行完terminated()之后，就会由 TIDYING -> TERMINATED。

4.提交给线程池的任务有两种可能的执行路径：一个是直接由线程池新创建一个线程来处理该任务，一个是线程从阻塞队列中取出任务执行，执行完任务的空闲线程会再次去阻塞队列申请任务去执行。

5.常见的阻塞队列：ArrayBlockingQueue（有界阻塞队列） LinkedBlockingQueue（无界阻塞队列） PriorityBlockingQueue（支持优先级的无界阻塞队列） DelayQueue(支持延迟的无界队列)

SynchronousQueue`：特殊的BlockingQueue，对其的操作必须是放和取交替完成的，典型的生产者-消费者模型，它不存储元素，每一次的插入必须要等另一个线程的移除操作完成。

6.




```
private final class Worker extends AbstractQueuedSynchronizer implements
Runnable{
  final Thread thread;//Worker持有的线程
  Runnable firstTask;//初始化的任务，可以为null
}
```

Worker这个工作线程，实现了Runnable接口，并持有一个线程thread，一个初始化的任务firstTask。thread是在调用构造方法时通过ThreadFactory来创建的线程，可以用来执行任务；firstTask用它来保存传入的第一个任务，这个任务可以有也可以为null。如果这个值是非空的，那么线程就会在启动初期立即执行这个任务，也就对应核心线程创建时的情况；如果这个值是null，那么就需要创建一个线程去执行任务列表（workQueue）中的任务，也就是非核心线程的创建。

Worker是通过继承AQS，使用AQS来实现独占锁这个功能。没有使用可重入锁ReentrantLock，而是使用AQS，为的就是实现不可重入的特性去反应工作线程现在的执行状态。

7.线程池构造参数有8个，但是最核心的是3个：corePoolSize、maximumPoolSize，workQueue，它们最大程度地决定了线程池的任务分配和线程分配策略。JDK提供了setCorePoolSize(int)、setMaxmumPoolSize(int)，**可以在线程池运行时动态设置corePoolSize和maximumPoolSize的值。**

在运行期线程池使用方调用此方法设置corePoolSize之后，线程池会直接覆盖原来的corePoolSize值，并且基于当前值和原始值的比较结果采取不同的处理策略。

对于当前值小于当前工作线程数的情况，说明有多余的worker线程，此时会向当前空闲的worker线程发起中断请求以实现回收，多余的worker在下次空闲的时候也会被回收；

对于当前值大于原始值且当前队列中有待执行任务，则线程池会创建新的worker线程来执行队列任务。

如果想要动态这是阻塞队列的长度，需要自己实现一个阻塞队列，提供get/set方法来设置阻塞队列长度。

8.线程池刚创建的时候并没有线程，正常情况下只有过来一个任务才回去创建一个线程去执行，但是线程池也提供了**preStartAllCoreThreads**()、**preStartCoreThread**()方法来预热全部或者单个线程提前加载线程。

9.核心线程默认情况下是不会被回收的，即使线程池已经没有任务需要执行，这个时候核心线程会等待新的任务进来但是不会被回收(此时线程处于阻塞挂起状态等待新的任务的到来)。如果想要核心线程在空闲之后被回收可以设置**allowCoreThreadTimeOut** 为true。

10.默认的线程池

Executors 实现了以下四种类型的 ThreadPoolExecutor：

类型	特性
newCachedThreadPool	线程池的大小不固定，可灵活回收空闲线程，若无可回收，则新建线程
newFixedThreadPool	固定大小的线程池，当有新的任务提交，线程池中如果有空闲线程，则立即执行，否则新的任务会被缓存在一个任务队列中，等待线程池释放空闲线程
newScheduledThreadPool	定时线程池，支持定时及周期性任务执行
newSingleThreadExecutor	只创建一个线程，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序（FIFO-LIFO-优先级)执行

11.设置线程池大小

CPU 密集型任务

> 这种任务消耗的主要是 CPU 资源，可以将线程数设置为 N（CPU 核心数）+1，比 CPU 核心数多出来的一个线程是为了防止线程偶发的缺页中断，或者其它原因导致的任务暂停而带来的影响。一旦任务暂停，CPU 就会处于空闲状态，而在这种情况下多出来的一个线程就可以充分利用 CPU 的空闲时间。

I/O 密集型任务

> 这种任务应用起来，系统会用大部分的时间来处理 I/O 交互，而线程在处理 I/O 的时间段内不会占用 CPU 来处理，这时就可以将 CPU 交出给其它线程使用。因此在 I/O 密集型任务的应用中，我们可以多配置一些线程，具体的计算方法是 2N。

线程池种线程空闲的时候处于阻塞状态