1. 为什么要使用线程池？

答：

1. 减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务。
2. 可以根据系统的承受能力，调整线程池中工作线程的数目，防止因为消耗过多的内存，而把服务器累趴下。

2、Executors提供的四种线程池:newSingleThreadExecutor,newFixedThreadPool,newCachedThreadPool,newScheduledThreadPool ，请说出他们的区别以及应用场景

答：

1. newSIngleThreadExecutor 创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有的任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。
2. newFixedThreadPool 创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常，那么线程池会补充一个新线程。
3. newCacheThreadPool 创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务，此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者JVM）能够创建的最大线程大小。
4. newScheduledThreadPool 创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

3、线程池有哪几种工作队列？

① ArrayBlockingQueue 是一个基于数组结构的有界阻塞队列，此队列按FIFO（先进先出）原则对元素进行排序。

② LinkedBlockingQueue 是一个基于链表结构的阻塞队列，此队列按FIFO（先进先出）排序元素，吞吐量通常要高于ArrayBlockingQueue。静态工厂方法Executors.newFixedThreadPool()使用了这个队列。

③ SynchronousQueue 是一个不存储元素的阻塞队列。每个插入操作必须等到另一个线程调用移除操作，否则插入操作一直处于阻塞状态，吞吐量通常要高于LinkedBlockingQueue ，静态工厂方法Executors.newCachedThreadPool(5) 使用了这个队列。

④ PriorityBlockingQueue 一个具有优先级的无限阻塞队列。

1. 线程池默认的拒绝策略有哪些
2. CallerRunsPolicy 线程调用运行该任务的execute本身。此策略提供简单的反馈控制机制，能够减缓新任务的提交速度。
3. AbortPolicy 处理程序遭到拒绝将抛出运行时RejectExecutionException，这种策略直接抛出异常，丢弃任务。
4. DiscardPolicy 不能执行的任务将被删除，这种策略和AbortPolicy几乎一样，也是丢弃任务，只不过它不抛出异常。
5. DiscardOldestPolicy 如果执行程序尚未关闭，则位于工作队列头部的任务将被删除，然后重试执行程序（如果再次失败，则重复此过程）

该策略就稍微复杂一些，在pool没有关闭的前提下首先丢掉缓存在队列中的最早的任务，然后重新尝试运行该任务。这个策略需要适当小心。

5、如何理解有界队列和无解队列

答：

有界队列：

1. 初始的poolSize < corePoolSize,提交的runnable任务，会直接做为new一个Thread的参数，立马执行。
2. 当提交的任务数超过了corePoolSize，会将当前的runnable提交到一个block queue中。
3. 当有界队列满了之后，如果poolSize < maximumPoolSize，会尝试new一个Thread的进行救急处理，立马执行对应的runnable任务。
4. 如果3中也无法处理了，就会走到第四步执行reject操作。

无界队列：

1. 与有界队列相比，除非系统资源耗尽，否则无界的任务队列不存在任务入队失败的情况。当有新的任务到来，系统的线程数小于corePoolSize时，则新建线程执行任务。当达到corePoolSize后，就不会继续增加，若后续仍有新的任务加入，而没有空闲的线程资源，则任务直接进入队列等待。若任务创建和处理的速度差异很大，无界队列会保持快速增长，直到耗尽系统内存。

6、线程池是如何实现线程回收的？ 以及核心线程能不能被回收？

工作线程回收需要满足三个条件：

1. 设置线程池线程最大空闲时间，超出这个时间，对线程进行回收。
2. 调用shutdown方法，将线程池中的空闲线程回收。该方法会使得keepALiveTime参数失效。

核心线程可以被回收。

解决办法：threadPoolExecutor.allowCoreThreadTimeOut(true);

1. FutureTask是什么

答: Future 接口代表异步计算的结果，通过Future接口提供的方法可以查看异步计算是否执行完成，或者等待执行结果，同时还可以取消执行。

FutureTask是Future的实现类，如上面类继承关系，它还继承了Runnable;所以FutureTask即可以作为Runnable被Thread执行，也可以作为Future获取Callable的结果。

1. Thread.sleep(0)的作用是什么

答： 在线程中，调用sleep(0) 可以释放cpu时间，让线程马上从重新回到就绪队列而非等待队列，sleep(0) 释放当前线程所剩余的时间片（如果有剩余的话），这样可以让操作系统切换其他线程来执行，提升效率。

在线程没退出之前，线程有三个状态，就绪态，运行态，等待态。Sleep（n）之所以在n秒内不会参与CPU竞争，是因为，当线程调用sleep（n）的时候，线程是由运行态转入等待态，线程被放入等待队列中，等待定时器n秒后的中断事件，当达到n秒计时后，线程才重新由等待态转入就绪态，被放入就绪队列中，等待队列中的线程是不参与cpu竞争的，只有就绪队列中的线程才会参与cpu竞争，所谓的cpu调度，就是根据一定的算法（优先级，FIFO等。。。）,从就绪队列中选择一个线程来分配cpu时间。

1. 如果提交任务时，线程池队列已满，这时会发生什么

由于默认是AbortPolicy拒绝策略，因此即可能是抛出RejectedExecutionException异常，如果是其他的策略是其他的处理方式。此时线程会等待，后续未进入的请求会直接占用CPU和内存资源，直到内存溢出。

1. 如果一个线程池中还有任务没有执行完成，这个时候是否允许被外部中断？

答：

Shutdown():等待那些已经开始的任务执行完毕，等待任务队列中尚未开始的任务执行完毕。

Shutdownnow()：尝试取消已经开始的任务（中断这些任务），返回任务队列中尚未开始的任务。Shutdownnow只返回任务队列中尚未开始的任务列表，但是想要记录那些已经但被取消的任务，可以使用条件检查：线程池的状态处于关闭中且任务的状态为已中断（当然前天是任务要保持中断任务）。