线程池总结

1，线程池的好处

- 降低资源的消耗。通过重复利用已创建的线程降低线程的创建和销毁造成的消耗。
- 提高相应的速度。当任务到达时，任务可以不需要等到线程的创建就能立即执行。
- 提高线程的课管理性。线程时稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，调优和监控。

2，线程池的原理

- **线程在线程池中处理的流程**

  - 线程池判断核心线程池里面的线程是否都在执行任务。如果不是，则创建一个新的工作线程来执行任务(需要获取全局锁)。如果核心线程里的线程都在执行任务，则进入下一个流程。
  - 线程池判断工作队列是否已满，如果工作队列还没有满，则将新提交的任务存储在这个工作队列里面，如果工作队列满了，则进入下个流程。
  - 线程池判断线程池的线程是否都处于工作状态，如果没有，则创建一个新的工作线程来执行任务(需要获取全局锁)。如果已经满了，则交给饱和策略来处理

  采用上述的流程是为了让线程尽可能地避免获取全局锁(影响效率的瓶颈),在ThreadPoolExecutor完成预热之后，(当前运行的线程数大于核心线程数),几乎所有的execute()方法调用都是执行第二个不揍，也就是将线程加入到工作队列当中。

- **工作线程：线程池创建线程的时候，会将线程封装成工作线程Worker，在线程执行完之后，它不会被销毁，而是从阻塞队列中获取任务来执行。**

- **饱和策略**

  - 直接抛出异常
  - 只用调用者所在线程来运行任务
  - 丢弃队列中最近的一个任务，并执行当前任务
  - 不处理，丢弃掉
- **阻塞队列**

  - ArrayBlockingQueue：是一个基于数组结构的有界阻塞队列，按照先进先出的原则对元素进行排列。
  - LinkedBlockingQueue：一个基于来链表结构的无界阻塞队列，按照先进先出规则。
  - SynchronousQueue：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。
  - PriorityBlockingQueue：一个具有优先级别的无线阻塞队列。
- **线程池参数的构成**

  - corePoolsize
  - 任务队列
  - maximumPoolSize
  - ThreadFactory
  - 饱和策略
  - 线程活动保持时间
  - 线程活动保持时间的单位

- **execute()和submit()的区别?**
  - execute适用于不需要关注返回值，只需要将线程丢到线程池中去执行就行了。
  - submit适用于需要关注返回值的场景。

```
public class ThreadPool{
    public static void main(String[] args){
        ExecutorService pool = Executors.newFixedTHreadPool(10);
        Future<String> future = pool.submit(new Callable<String>(){
           @Override
            public String call() throws Exception{
                return "Hello World!";
            }
        });
        String result = future.get();
        System.out.println(result);
    }
}
```

3，线程池的使用

- **线程池的创建**

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MyThreadPool{
    public static void main(String[] args){
        ExecutorService exe=Executors.newCachedThreadPool();  //创建线程池
        exe.execute(()->{
            System.out.println("asd");
        });
        exe.shutdown();  //关闭线程池
    }
}
```

  - **四种常见的线程池**
    - FixThreadPool：用于创建使用固定线程数的ThreadPool,corePoolSize=maximmumPoolsize=n(固定)，阻塞队列位LinkedBlockingQueue(基于链表结构的阻塞队列，按照先进先出的原则)。
    - SingleThreadExecutor：用于创建一个单线程的线程池，corePoolSize=maximumPoolsize=1,阻塞队列为LinkedBlockedQueue。
    - CachedThreadPool：用于创建一个可缓存的线程池，corePoolSize=0,maximumPoolSize=Integer.MAX_VALUE,阻塞队列为SynchronousQueue(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)。
    - ScheduledThreadPoolExecutor：用于创建一个大小无限的线程池，此线程支持定时以及周期性执行任务的需求。
  - **四种线程池的使用场景**
    - FixThreadPool：一个固定大小的线程池，适用于已知并发压力的情况下，对线程数做限制。

- SinggleThreadExecutor：一个单一线程的线程池，可以用于保证顺序执行的场景，并且只有一个线程在执行。
- CachedThreadPool：一个可以无线扩大的线程池，比较适合处理时间较小的任务。
- SchedThreadPoolExecutor：可以延时启动，适用于多个后台线程执行周期人物的场景。