并发基础篇（7）：Thread 类的 sleep()、yeild()、join()

1、Thread.sleep(long millis)

sleep( )是一个静态方法，让当前正在执行的线程休眠（暂停执行），而且在睡眠的过程是不释放资源的，保持着锁。

在睡眠的过程，可以被中断，注意抛出 InterruptedException 异常；

作用

1、暂停当前线程一段时间；

2、让出 CPU，特别是不想让高优先级的线程让出 CPU 给低优先级的线程

```
try {        //单位是毫秒，睡眠 1 秒
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

2、Thread.yeild（ ）

同样也是一个静态方法，暂停当前正在执行的线程，线程由运行中状态进入就绪状态，重新与其他线程一起参与线程的调度。

作用

线程让步，顾名思义，就是说当一个线程使用了这个方法之后，它就会把自己 CPU 执行的时间让掉，让自己或者其它的线程运行。但是，这种让步只对同优先级或者更高优先级的线程而言，同时，让步具有不确定性，当前线程也会参与调度，即有可能又被重新调度，那么就没有达到让出 CPU 的效果了。

3、Thread.join（ ）

JDK 中提供三个版本的 join 方法：

1. join( )：等待该线程终止。
2. join(long millis) :等待该线程终止的时间最长为 millis 毫秒。超时为 0 意味着要一直等下去。
3. join(long millis, int nanos)：等待该线程终止的时间最长为 millis 毫秒 + nanos 纳秒。

作用

join 方法的作用是父线程等待子线程执行完成后再执行 ,换句话说就是将异步执行的线程合并为同步的线程。

```
public static void main(String[] args) {
    Thread childThread = new Thread("childThread"){
        @Override
        public void run() {
            int a = 1;
            for(int i=1;i<5;i++){
                a += i;
            }
            System.out.println("线程"+getName()+"结束，Count a = "+a);
        }
    };
    //线程启动
    childThread.start();
    try {
     //main 线程要等待 childThread 线程的结束，才可以往下执行
        childThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
        System.out.println("线程"+Thread.currentThread().getName()+"结束");
    }
运行结果：
线程 childThread 结束，Count a = 11
线程 main 结束
```