## **一、多线程的创建方式一**

方式一、使用继承Thread 类，重写run方法，调用start方法执行。 来创建线程。  
二、学习线程常用方法  
 Thread.currentThread() 返回当前线程的对象  
 Thread.currentThread().getName() 获取当前线程的线程名  
 Thread.currentThread().setName() 设置当前线程的线程名  
 join() 在线程a中调用另一个线程b的join方法 ，此时a会进入阻塞状态，会一直执行b这个线程到结束 ，才会执行a这个线程  
 yield() 释放当前cpu的执行权  
 stop() 强制结束线程  
 isAlive() 判断线程是否存存活

## **三、线程的优先级**

currentThread().getPriority() 获取当前线程的优先级

currentThread().setPriority(Thread.MAX\_PRIORITY) 设置当前线程的优先级

高优先级的线程要抢占低优先级的cpu的执行权，但是只是从概率上讲，高优先级的线程执行的概率比较高，但是不是意味着高优先级的线程执行完后低优先级的线程才被执行。

## **二、多线程的创建方式二**

方式二：1、创建一个实现了runnable 接口的类，实现run（）方法

2、创建该类的对象

3、将该类作为参数传入 Thread 的构造方法中

优先使用实现runnable接口的方法

原因一、java 单继承的局限性

原因二、多个线程间共享数据的问题

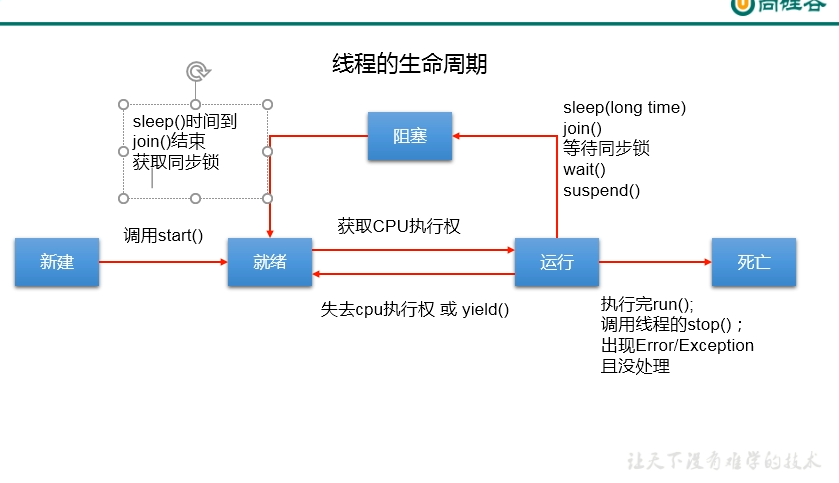
## **线程的生命周期**

新建、

就绪（执行start()方法）、

运行（失去cpu执行权或者yield() 会回到就绪状态）、

阻塞（执行sleep、join、等待同步锁、wait、suspend）、

死亡（执行完run()、调用线程的stop()、出现error或者exception且没处理）  
 

## 一、理解线程安全问题

多个线程操作共享的数据时，会出现线程安全问题（买票案例的重票和错票问题，当某个线程在操作车票的过程中，尚未操作完成时，其他的线程参与进来，也操作车票。）

## 二、解决线程安全问题

方式一：同步代码块

synchronized（锁）{}

synchronized 代码块中的代码只有一个线程可以操作，在一个线程操作的过程中其他线程处于阻塞状态。锁可以是任意的对象，锁必须多个线程共用一把，也就是说这个锁对象对多个线程对象来说是唯一的。

方式二：同步方法

private synchronized void sale(){}

加了synvchroized 关键字的方法同一时间只有一个线程可以执行

不用显式的写出锁来，非静态方法锁默认的是this，静态方法的锁默认的是 class对象

方式三：使用Lock对象

不同：1、synchroized 是隐式锁，Lock是显式锁（手动开启和关闭）

2、lock 只有代码块锁，synchronized有代码块锁和方法锁。

3、使用Lock锁，JVM将花费较少的时间来调度线程，性能更好，并且具有良好的扩展性

优先使用顺序：Lock锁 --> synchronized代码块 -->synchroized方法

## 三、线程通信问题

三个方法：

1、wait() 一旦使用此方法，当前线程会立即阻塞, 并会释放锁（同步监视器）

2、notify() 使用此方法会唤醒被wait的一个线程，如果有多个线程wait，就唤醒优先级高的哪个线程。

3、notifyAll() 唤醒所有被wait的线程

说明：

1、三个方法必须使用在同步代码块和同步方法中

2、方法的调用者必须是同步代码块或同步方法中的同步监视器对象

sleep 和 wait 的区别

一、sleep 在 Thread 类中声明，wait 在object中的声明。

二、sleep可以在任何场景下使用，wait只能在同步代码块中使用。

三、sleep不会释放同步锁，wait会释放同步锁。

## 多线程的创建方式三

**步骤**：

1、实现callable接口 重写call方法  
 2、将实现类作为参数传入 FutureTask的构造方法中，构造一个FutureTask对象启动一个线程。  
 3、 将FutureTask 对象作为参数传入Thread 的构造方法中 并调用Thread的start方法  
 4、使用 FutureTask的get方法获得 call方法的返回值

**好处：**

1. 可以接收返回值
2. 可以在外部捕获异常
3. 可以支持泛型

## **使用线程池的好处**

背景：经常创建和销毁**使用量**特别大的资源，在高并发的情况下对性能影响很大。

思路：提前创建好线程放入池中，要用的时候拿出来，使用完放回池中，可以避免频繁的创建销毁，实现重复利用。

好处：

1. 提高响应速度。（减少了线程创建的时间）
2. 降低资源的消耗。（重复使用线程，不用每次使用都创建）
3. 便于线程管理

CorePoolSize ：核心池的大小

1. Maximunpoolsize：最大线程数
2. Keepalivetime：线程没有任务的时候最多保持多长时间后会终止

## 使用线程池

**步骤：**

1. 使用Executors 工具类构造线程池 ,线程池对象是ExecutorService接口 的实现类

2、ExecutorService 可以强制转型为ThreadPoolExecutor 对象，来设置线程池的属性

1. 使用executorService.submit(Runnable runnable) 方法把实现了Runnable 或Callable接口的类的实例放入线程池并启动一个线程，返回Future对象。Submit 一次启动一个线程。
2. 使用Future 获取 call 方法中的执行结果