### Java多线程总结

## 实现线程的方式及其常用方法与属性

### 进程与线程的概念及线程的有点

进程可以说是操作系统的基础，是程序的一次运行，进程是操作系统进行资源分配和调度的最小单元。

线程可以理解为进程的一个或多个子任务，如果一个进程只有一个线程，可以理解为单任务进程，单任务的特点就行排队执行，也就是同步。使用多线程的目的就是在线程安全的情况下进行异步执行，尽可能的提高CPU及系统资源的利用率，这也是其优点。

### 1.2 实现线程的两种方式

Java实现多线程常用的两种方式有继承Thread类、实现Runnable接口，实现Runnable接口相对来说具有优势，突破了Java单继承的局限性，维持了程序的健壮性，尤其是在多个线程需要造作同一资源时，实现接口的方式是首选。在将多线程交由线程池管理的情况下也必须是实现接口的方式。

实现多线程还有其他方式，给自己留个疑问后续在深究一下。

### 1.3 Java多线程中常用方法

currentThread():指明代码段正在被哪个线程调用；

isAlive():判断当前线程是否处于活动状态，线程处于就绪状态或运行状态为活动状态；

sleep(long millisecond):指定毫秒数让当前正在执行的线程暂停执行，当线程持有锁的情况下，执行此方法可达到暂停执行作用但不释放锁，因此在有锁的情况下慎用此方法；

停止线程的方法：

1. 使用退出标志，当线程执行完run()方法中程序时线程终止。
2. 使用stop()方法强行终止线程，不推荐使用，容易出现脏数据。
3. 使用interrupt()方法中断线程。

使用退出标志即为常用的while（flag）{}死循环，当flag变为false时，程序正常执行完毕，即线程终止。

使用stop()方法可以达到退出线程的效果，由于在调用此方法时会释放锁，这就有可能使得其他线程拿到脏数据，造成数据不同步，因此此方式已过时不推荐使用。

interrupt()方法配合抛异常或return，都能达到终止线程得效果。当检测到线程处于中断状态时，抛出异常或return即可。在睡眠状态调用此方法中断线程会抛出异常。给出检测线程中断状态得两个接口方法：

this.interrupted():测试当前线程是否处于中断状态，当前线程时指运行此方法的线程；

this.isInterrupted():测试线程是否已中断，同样用this指定时与上一个方法意义不同，用线程对象指定时意义相同，且此方法具有清除线程中断状态的作用，当线程调用interrupt()方法后处于中断状态时，调用此方法可激活。

线程暂停及恢复：

suspend()：线程暂停

resume()：恢复已暂停线程

此二方法已过时不推荐使用，由于suspend()具有独占的特性，当拥有公共资源时调用此方法不释放会造成后续线程无限时间等待；使用此方法还易造成数据不同步。

yield():让出CPU使其重新调度

### 1.4 线程的常用属性

## 对象及变量的并发访问

### 2.1 synchronized关键字

### 2.2 volatile关键字

## 线程通信

## Lock的使用

## 定时器Timer

## 单例模式下的多线程