博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

在多线程操作之中如果要启动多线程肯定使用的是Thread类中start()方法，而如果对于多线程需要进行停止处理，Thread类原本提供有stop()方法，但是对于这些方法从JDK1.2版本开始就已经将其废除了，而且一直到现在也不再建议出现在你的代码之中，而除了stop()之外还有几个方法也被禁用了：

·停止多线程：public final void stop()；

·销毁多线程：public void destroy()；

·挂起线程：public final void suspend()、暂停执行；

·恢复挂起的线程执行：public final void resume()；

之所以废除掉这些方法，主要的原因是因为这些方法有可能导致线程的死锁，所以从JDK1.2开始就都不建议使用了。如果这个时候要想实现线程的停止需要通过一种柔和的方式来进行。

范例：实现线程柔和的停止

```
public class ThreadDemo {
	public static boolean flag = true ;
	public static void main(String[] args) throws Exception {
		new Thread(() -> {
			long num = 0;
			while (flag) {
				try {
					Thread.sleep(50);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				System.out.println(Thread.currentThread().getName() + "正在运行、num
= " + num++);
			}
		}, "执行线程").start();
```

```
            Thread.sleep(200); // 运行200毫秒
            flag = false ; // 停止线程
        }
}
```

万一现在有其它的线程去控制这个flag的内容，那么这个时候对于线程的停止也不是说停就立刻停止的，而是会在执行中判断flag的内容来完成。

2、具体内容

现在假设有一个人并且这个人有一个保镖，那么这个保镖一定是在这个人活着的时候进行守护，如果这个人已经死了，保镖没用了。所以在多线程里面可以进行守护线程的定义，也就是说如果现在主线程的程序或者其它的线程还在执行的时候，那么守护线程将一直存在，并且运行在后台状态。

在Thread类里面提供有如下的守护线程的操作方法：

·设置为守护线程：public final void setDaemon(boolean on);

·判断是否为守护线程：public final boolean isDaemon();

范例：使用守护线程

```
public class ThreadDemo {
    public static boolean flag = true;
    public static void main(String[] args) throws Exception {
        Thread userThread = new Thread(() -> {
            for (int x = 0 ; x < 10 ; x ++) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println(Thread.currentThread().getName() + "正在运行、x = "
+ x);
            }
        }, "用户线程");  // 完成核心的业务
        Thread daemonThread = new Thread(() -> {
```

```
                for (int x = 0 ; x < Integer.MAX_VALUE ; x ++) {
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(Thread.currentThread().getName() + "正在运行、x = "
+ x);
                }
            }, "守护线程");  // 完成核心的业务
            daemonThread.setDaemon(true); // 设置为守护线程
            userThread.start();
            daemonThread.start();
        }
}
```

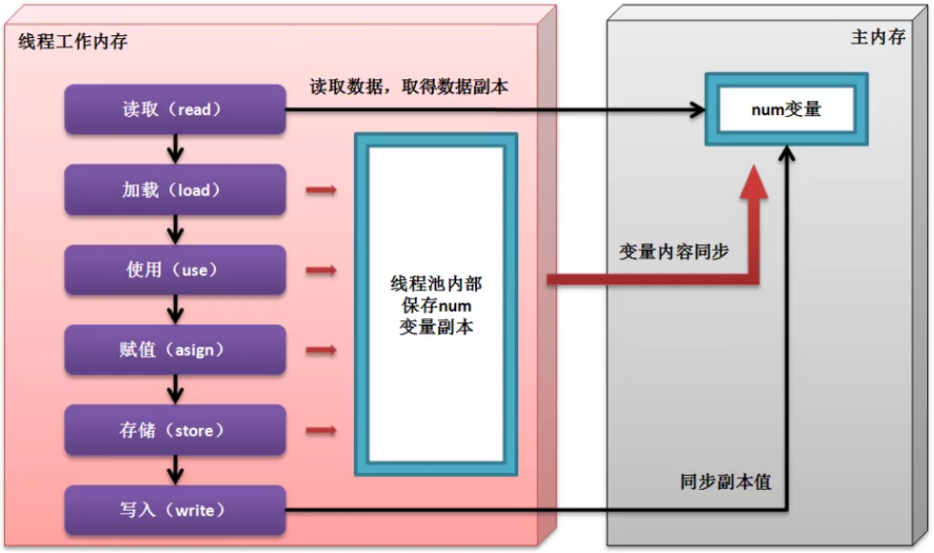
可以发现所有的守护线程都是围绕在用户线程的周围，如果程序执行完毕了，守护线程也就消失了，在整个JVM里面最大的守护线程就是GC线程。

程序执行中GC线程会一直存在，如果程序执行完毕，GC线程也将消失。

2、具体内容

在多线程的定义之中，volatile关键字主要是在属性定义上使用的，表示此属性为直接数据操作，而不进行副本的拷贝处理，这样的话在一些书上就将其错误的理解为同步属性了。

volatile



在正常进行变量处理的时候往往会经历如下的几个步骤：

·获取变量原有的数据内容副本；

·利用副本为变量进行数学计算；

·将计算后的变量，保存到原始空间之中；

而如果一个属性上追加了volatile关键字，表示的就是不使用副本，而是直接操作原始变量，相当于节约了：拷贝副本、重新保存的步骤。

```
class MyThread implements Runnable {
	private volatile int ticket = 10 ; // 直接内存操作
	@Override
	public void run() {
		while (true) {
			synchronized(this) {
				if (this.ticket > 0) {
					try {
						Thread.sleep(100);
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
					System.out.println(Thread.currentThread().getName() + "卖票，
ticket = " + this.ticket --);
				} else {
					System.out.println("***** 票已经卖光了 *****");
					break ;
				}
			}
		}
	}
}
public class ThreadDemo {
	public static void main(String[] args) throws Exception {
		MyThread mt = new MyThread() ;
```

```
            new Thread(mt,"票贩子A").start();
            new Thread(mt,"票贩子B").start();
            new Thread(mt,"票贩子C").start();
        }
}
```

面试题：请解释volatile与synchronized的区别?

·volatile主要在属性上使用，而synchronized是在代码块与方法上使用的；

·volatile无法描述同步的处理，它只是一种直接内存(操作内存)的处理，避免了副本的操作，而synchronized是实现同步的。

·volatile使用的时候不能离开synchronized