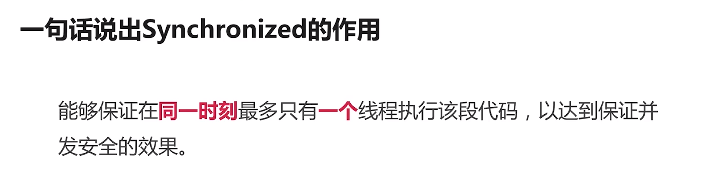
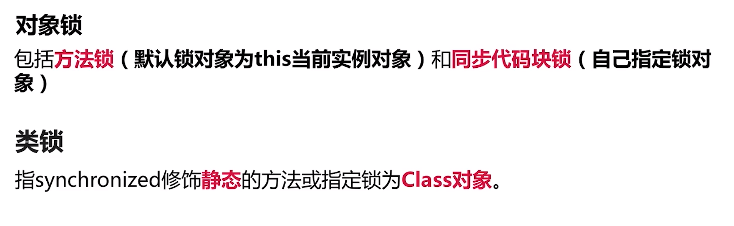
# Synchronized

## 1 Synchronized的作用



## 2 Synchronized的两个用法



## 3 线程的join()方法

main是主线程，在main中创建了thread线程，在main中调用了thread.join()，那么等thread结束后再执行main代码。

在很多情况下，主线程生成并起动了子线程，如果子线程里要进行大量的耗时的运算，主线程往往将于子线程之前结束，但是如果主线程处理完其他的事务后，需要用到子线程的处理结果，也就是主线程需要等待子线程执行完成之后再结束，这个时候就要用到join()方法了。

---------------------

作者：frankarmstrong

来源：CSDN

原文：https://blog.csdn.net/frankarmstrong/article/details/55504161

版权声明：本文为博主原创文章，转载请附上博文链接！

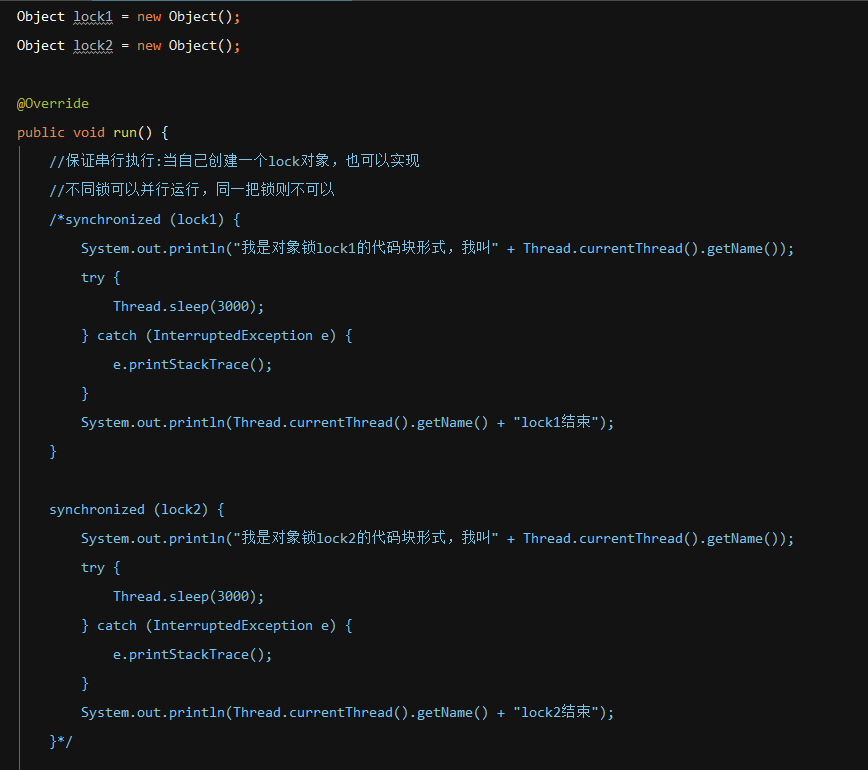
## 4 对象锁

### 4.1 代码块形式

This表示当前调用的对象



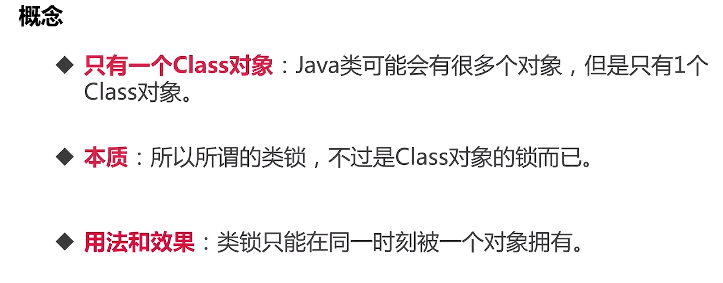
另一种方式：当有多个同步方法，不要求同时占用时，自己创建锁对象



### 4.2 方法修饰符形式（锁默认是this）

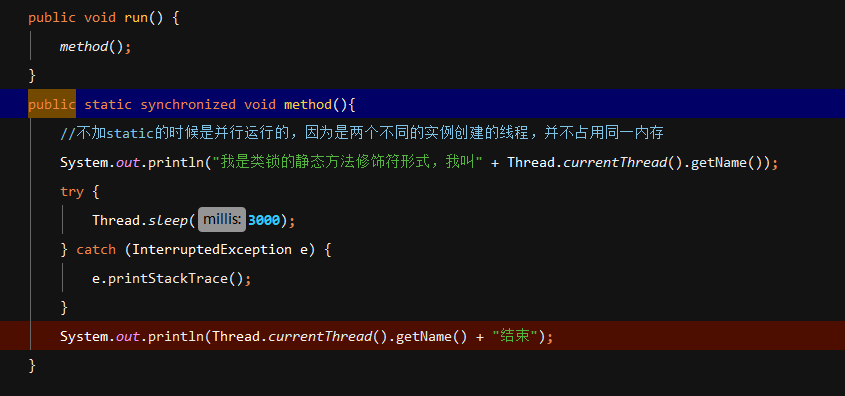


## 5 类锁



### 5.1 static方法

不加static的时候是并行运行的，因为是两个不同的实例创建的线程，并不占用同一内存。

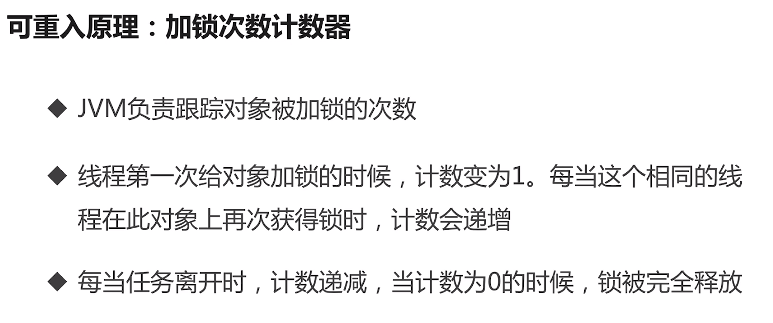


### 5.2 (\*.class)代码块

## 6 反编译获取同步锁的原理

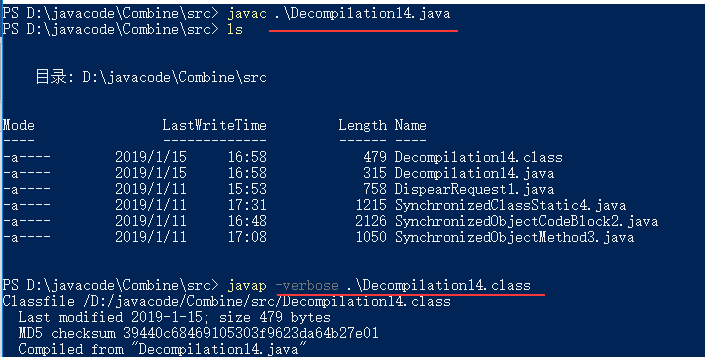
Monitorenter:1.monitor获得锁，锁计数+1;

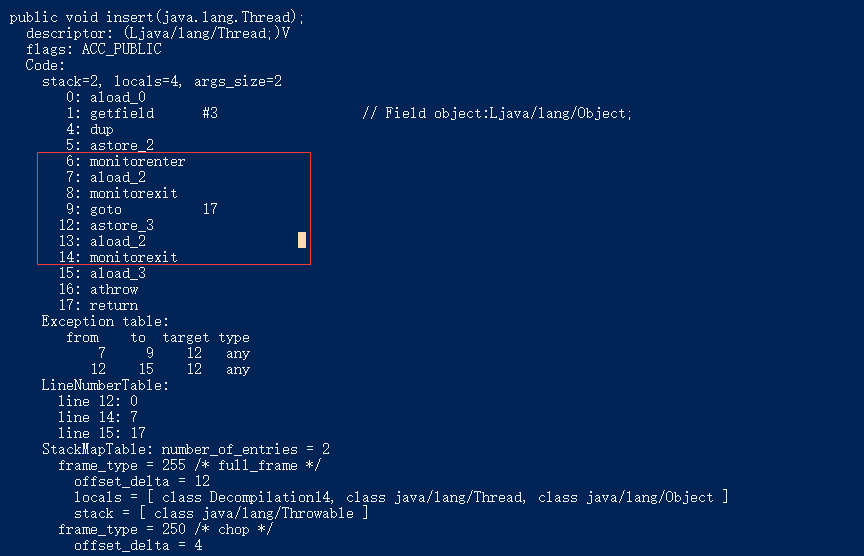
2.monitor重入锁，计数器继续+1;



3.monitor锁已被其他线程获取,等待

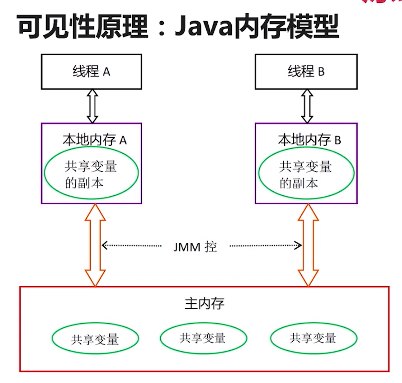
Monitorexit:释放锁：-1；如果>1:是可重入;

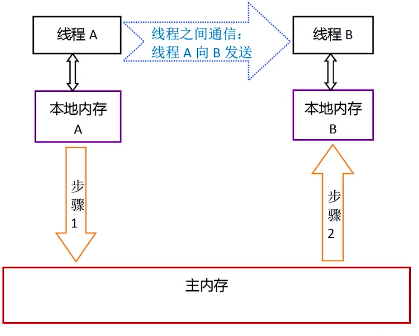




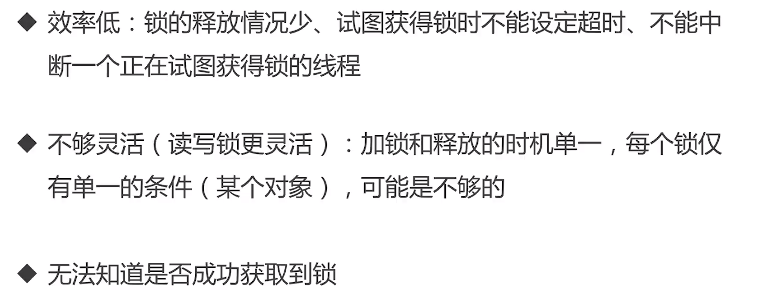
## 7 可见性原理

Synchronized:进入先读取内存，出去前再写入；





## 8 Synchronized缺陷

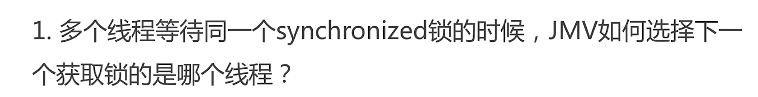


## 9 问题

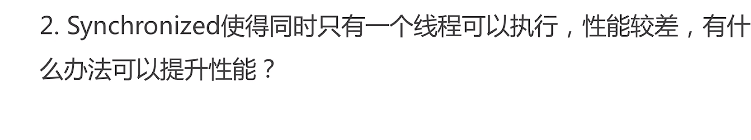
### 9.1 锁对象不能为空

因为锁的相关内容是存放再对象头中的。

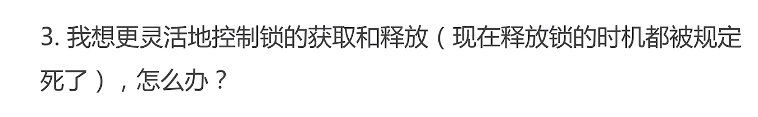
### 9.2 JVM如何调度线程



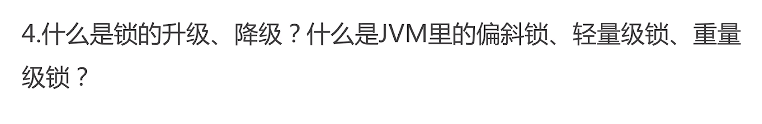
### 9.3 Synchronized的性能提升（尽可能地缩小作用域）



### 9.4 自定义锁



### 9.5 深入锁



## 10 线程的方法

### 10.1 停止方法

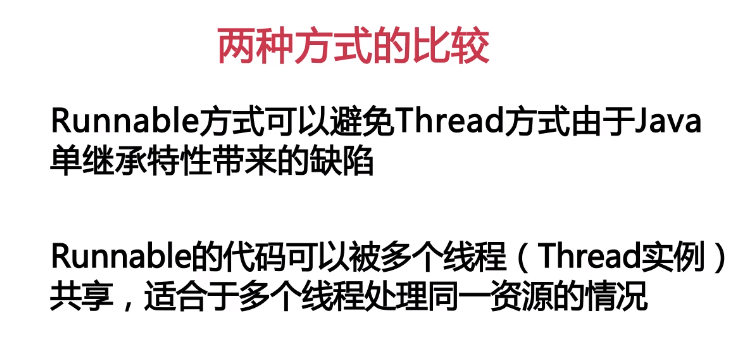
stop():该方法太多问题，线程戛然而止，还未对其进行清理，所以基本被弃用。

在方法中常见使用条件判断停止线程业务；

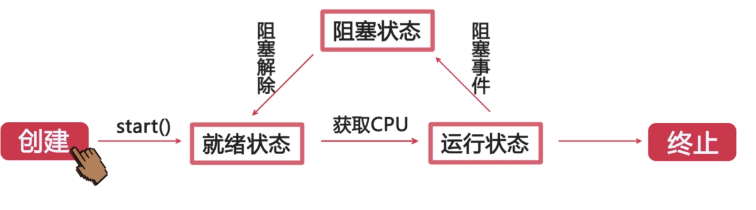
Interrupt():初衷不是停止线程，当遇到sleep和join方法时，中断状态被清除，只会输出异常，线程不停止，解决方法：在异常捕获中继续中断一次。

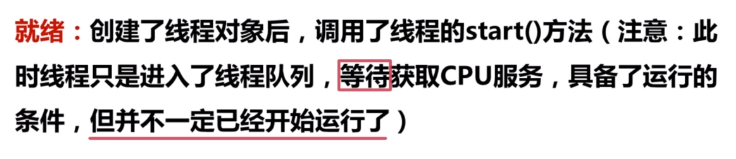


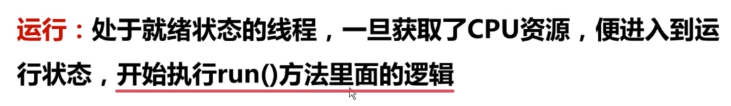
## 11 两种创建线程的方式的比较

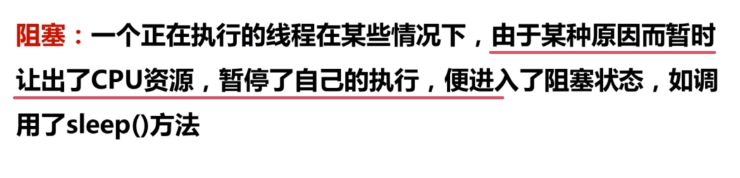


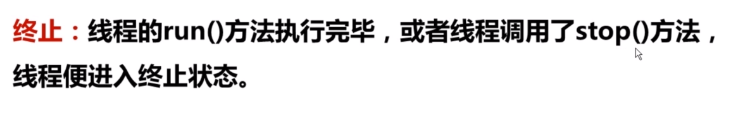
## 12 线程的状态



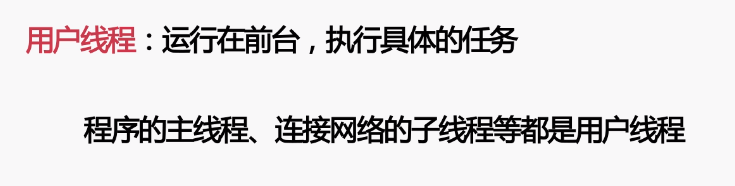


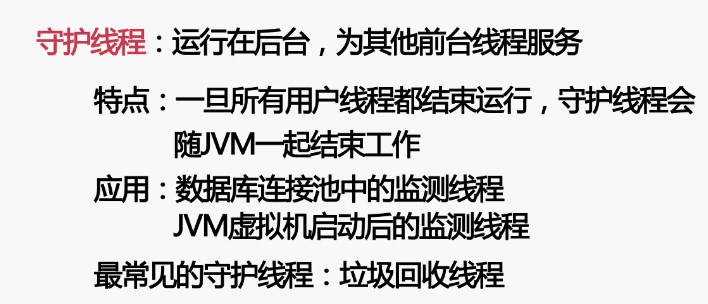




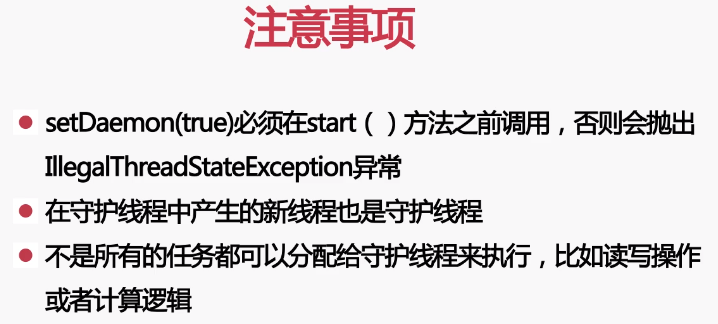


## 13 线程的种类







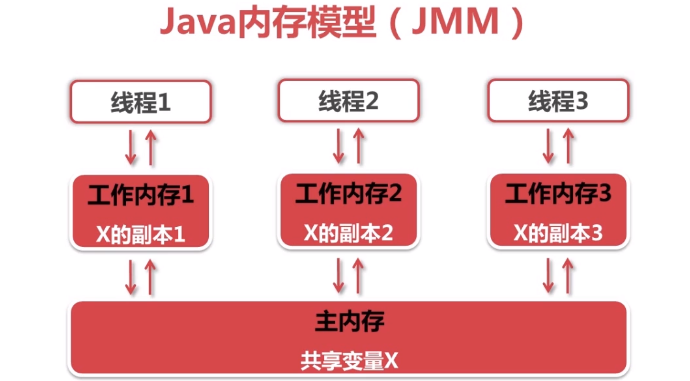


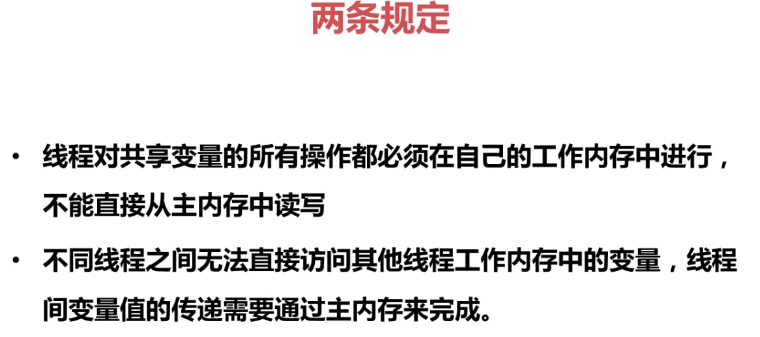
守护线程随着用户线程结束，那么有可能执行到一般就gg了。



## 13 数据争用

## 14 Java内存模型（JMM）





## 15 可见性的实现方式



