**线程安全问题及解决**

线程安全定义：

如果多个线程同时运行，而这些线程会同时运行一段代码。程序每次运行的结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。

造成线程不安全问题的三个关键点：

1.共享资源：当一个任务开启时，一个进程就被开启了，这个时候会分配一定的资源给进程，这个资源对于进程之下的子任务（线程）来说就是共享资源。

2.多线程：在多线程的情况下才有可能发生线程安全问题，单线程不会。

3.并发访问：只有多线程同时运行才有可能导致执行时出错，如果线程一个一个的来执行任务是不会有问题的；

# 并发：

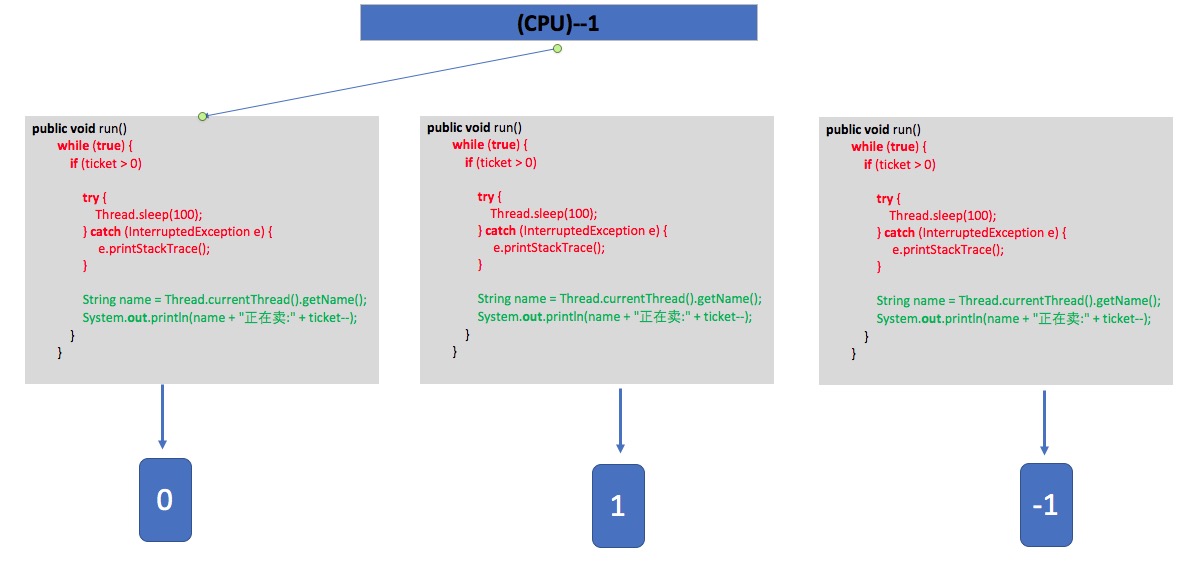
进程并发：在过去，只有单核cpu的情况下，要同时执行多个程序，就是利用并发，cpu在多个程序间切换运行；

线程并发：在一个任务内，多个子任务要同时执行，就是利用并发，cpu在多个子任务间切换运行；

# 并行：

进程并行：在有多核cpu的情况下，每个内核都能单独执行一个程序，使得程序进程并行轻而易举；

线程并行：在a程序的一个子线程和b程序的一个子线程同时运行是，可以成为两个线程在并行；



上图以三个窗口（线程）同时卖一百张票为例：

while（true）语句是执行的一个死循环，用来循环卖票。

假设卖到还剩最后一张票的时候：

1号线程抢到cpu资源先执行，执行完上图红色部分进入了休眠，空置状态出现，2号3号开始争抢cpu；

以此类推，2号和3号也会执行完红色部分进入100毫秒的休眠。

此时，1号重新开始执行后面的绿色部分代码，这个时候会执行代码并打印“1号窗口卖出了第1张票”；按理说此时就应该停止，因为已经没有票可以再卖了。但是此时2号3号线程都是已经启动了无法关闭，在他们的休眠时间结束后，也会开始执行绿色部分的代码。打印输出“2号窗口卖出了第0张票”、“3号窗口卖出了第-1张票”，这显然是错的！

这种现象就是线程不安全导致的并发问题；

要解决这种问题可以怎么办呢？

解决方法1：解决线程不安全的方式一：同步方法锁(锁:this)

class TicketRannable implements Runnable{

private int ticket = 1000;

@Override

//什么时候释放锁？ 在run方法结束才会释放锁。

public synchronized void run() {

while(true) {

if(ticket>0) {

//模拟出票的耗时

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"卖出第"+ticket--+"张票！！！");

}

}

}

}

这就是在run方法加了个synchronized 关键字，然后就给这个方法加了一个锁，在这个方法没有结束前，别的线程都无法抢cpu。哪怕是进入了休眠也不会释放锁，因为此处while是个死循环，所以如果哪个窗口抢到了第一次卖票的机会，之后的的所有票也会由这个窗口进行卖出；

这样肯定是不好的，其他两个窗口都没有事情可做；虽然这样做也已经解决了线程安全问题；

那么我们就把synchronized 所能影响的区域减小，

这时候我们可以用：解决线程不安全的方式一：同步块锁(this)

class TicketRannable1 implements Runnable{

private int ticket = 1000;

@Override

public void run() {

while(true) {

//同步块锁 锁（this）

//什么时候释放锁？块执行完就释放锁

synchronized (this) {

if(ticket>0) {

//模拟出票的耗时

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"卖出第"+ticket--+"张票！！！");

}

}

}

}

}这里的synchronized 关键字是通过影响一个块，然后再把代码放在这个被影响的块中来达到给其上锁的目的。这个时候快中的代码表示的是一次卖票的过程，执行完以后就会释放锁，然后三个窗口再进行争抢。

这样也解决了线程不安全的问题；

**思考**

上面两种方法虽然都能解决线程的安全问题但是实际效率甚至比单线程都要慢，因为从本质上，每一次执行卖票的的窗口仍然只有一个，也就是说和单线程（一个窗口卖100张票）是没有区别的，甚至会因为多线程并发使得整个过程还多出一些线程间切换的时间消耗。那么解决这种问题的话，可以通过另一种解决方法———————— ConcurrentHashMap:分段锁

分段锁（举例）

\* 窗口1 1（1~10-锁）

\*

\* 窗口2 2(11~20-锁)

\*

\* 窗口3 3（21~30-锁） 这里相当于给每十张票加一个锁，这样做的话就能让卖不同分段票的线程并发而且互不干扰，从而不会引发线程安全问题。

此处需要注意的是锁的数量不宜过多，这就涉及到了优化的问题。