Java 第十章 多线程编程上机练习

22009200601 汤栋文

2023 年 5 月 21 日

目录

1. 个人信息

姓名：汤栋文
学号：22009200601
日期：2023 年 5 月 21 日

2. 任务重述

1. 练习 PPT 中的全部小练习，尝试对小练习中各部分进行修改，并观察修改后的执行效果。
2. 编写一个程序，程序模拟某电影院三个售票窗口同时出售电影票的过程。
   a) 电影票顺序出售，程序模拟显示售票的详细过程。
   b) 三个窗口同时出票，出票间隔采用随机控制。
   c) 不能重复出售相同的电影票。
   d) 程序应具有良好的人机交互性能。

3. PPT 练习

3.1 CreateThread on P12,P14,P16

JAVA 中有两种创建线程的方法如下，分别在 P12 和 P14 展示：

1. 构造一个 Runnable 接口的实现类并将其作为参数传入 Thread 类的构造方法中。
2. 继承 Thread 类并重写其 run 方法。

两种方法并没有太大的不同。不过继承接口应该是最正经的方法。而且利用接口可以省下来继承位（在第 16 页的 PPT 中所见），这应该是官方更希望我们使用的方法。

3.2 ControlThread on P26,P28,P32

1. start

线程创建后并不会立即执行，必须在调用 start 方法后才可以执行。调用 start 方法实质上是讲线程设置为 Runnable 状态，真正的执行还是得看虚拟机。start 方法会单开一个线程运行 run 方法。而直接运行 run 方法就是普通的函数调用，我们可以只关心 run 方法的实现本身，而不必去理会线程的具体管理。

2. join

调用 t.join 方法可以使当前线程等待，直到线程 t 结束为止，当前线程才恢复到 runnable 状态。这个方法可以方便的协调线程之间的相互关系，不至于发生一些冲突。

3. stop

调用这个方法就会在线程外部强制终止，你在终止线程的时候并不知道线程这时在干嘛，容易发生错误。建议使用 flag 标志，在线程内部结束运行并自动退出。

3.3 Lock&Synchronized on P41

（个人认为这是很重要的东西，不知道为什么 PPT 里面讲的并不是很多。）

多个线程对同共享数据进行操作时，由于多个线程的执行顺序不确定，所以可能导致数据的一致性遭到破坏（这种情况需要一点巧合才能被模拟出来，并不容易那么容易发生，但是确实是一种潜在的 Bug），甚至可能产生严重后果。

比如两个线程对同一个栈进行操作，一个进行压栈，一个进行弹栈。在压栈过程将会有放入数据和栈顶指针移动两个过程。在这放入数据之后指针移动之前另一个线程进行了弹栈操作，那么先弹出栈顶的元素不是新放入的元素。这将导致数据不一致甚至是新进栈数据的丢失。更严重地可能导致空栈弹栈或者栈元素溢出等可能导致程序崩溃的情况。

为了解决这种情况，JAVA 引入了锁的概念。我们可以利用 synchronized 来将一段代码标记为“临界区”并将一个对象设置为锁，同一时间只能有一个线程获得锁（个人理解说成只有一个线程能给这个对象解锁更恰当一点）。线程获得对象锁后才能执行临界区代码片段（即拥有对该对象的操作权）。此时，其他任何线程无法获得对象锁，也无法执行临界区代码。

通俗一点理解就是，我要修改这个对象，那么我就先把它和我自己都锁在自己家，在我修改完这个对象之前，不管谁来都不开门，保证整个过程安全完成。当然由于反复获得锁和释放锁，效率有稍微的降低也是在所难免的。

3.4 Interaction on P47,P48,P49

1. wait

顾名思义就是要去线程在此处等待其他线程执行到某个地方。在 synchronized 块中调用锁对象的该方法将阻塞该线程，释放锁，并进入该对象锁的 wait pool，直到被 notify 唤醒。

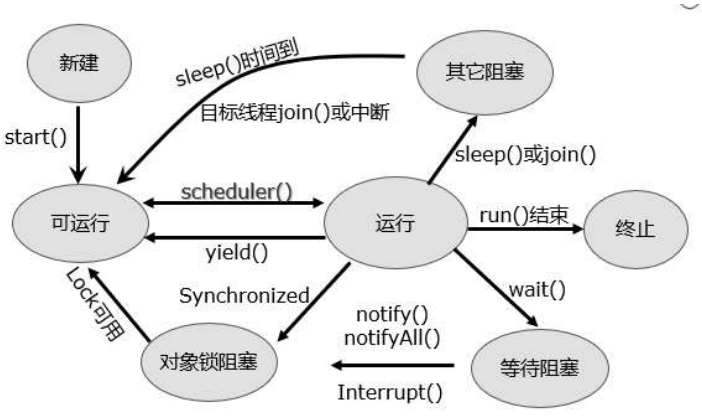
2. notify

对某一对象调用 notify 方法会将该对象 wait pool 中的一个线程唤醒，并等待该锁（这时候线程不一定马上执行，一定要重新获得锁以后才能继续执行）。注意这个唤醒是随机的。

3. notifyAll

唤醒该对象 wait pool 中所有的线程。这时候并不一定会执行到哪一个，并不一定按照之前暂停时的顺序继续执行。

(我觉得 PPT 上这张图其实很好地总结了整个线程的工作原理)



4. 模拟购票系统

4.0 整体说明

这次上机题目是比较简单的。这里面有一个地方应该要稍微注意一下。

synchronized 修饰的内容	应该锁的内容	作用效果
具体对象	被修饰的对象	这整个对象被保护
成员方法	对象自己(this)	这整个对象被保护
静态方法/变量	这个对象.class	这个对象类变量类方法被保护

这里采用第三种方式，锁的是 Window.class，做的过程中详细查了一下锁的工作方式，这实在是非常复杂，而且 Java 中锁的机制有很多种，于是暂时放一下不做具体探究。

(下面是核心部分的代码)

```
while(true){
    synchronized (Window.class) {
        if (now <= total)
            System.out.println("Sell ticket number "+now+" in window "+this.ID);
        else break;
        now++;
    }
```

4.1 遇到问题

竟然所有的票都在一个窗口被销售了？这有点奇怪，按照我的理解当线程 1 获得锁以后，线程 2 没办法获得锁就会在此处等待，当线程 1 释放以后，线程 2 马上获得，就好像大家在排队一样。但是实际上，这里的锁仍然又一次被线程 1 获得，这也说明了，JVM 底层有一套复杂的而且比较聪明的调节机制。

```
Sell ticket number 1 in window 1
Sell ticket number 2 in window 1
Sell ticket number 3 in window 1
Sell ticket number 4 in window 1
Sell ticket number 5 in window 1
Sell ticket number 6 in window 1
Sell ticket number 7 in window 1
Sell ticket number 8 in window 1
Sell ticket number 9 in window 1
Sell ticket number 10 in window 1
Sell ticket number 11 in window 1
Sell ticket number 12 in window 1
Sell ticket number 13 in window 1
Sell ticket number 14 in window 1
Sell ticket number 15 in window 1
```

4.2 按给定字符匹配查找

在此加入一个买票间隔的时间扰动后，三个线程就很好的协调工作了。

```
try { Thread.sleep((int)(Math.random()*100)); }
catch (InterruptedException e) { throw new RuntimeException(e); }
```

5. 总结

这次是除了大作业最后一次 JAVA 报告了，说实话一开始写报告挺反感的，后来发现其实在写报告的过程中学到的东西远远比上课学到的要多，而且老师给定的任务中知识点的覆盖比较全面，比自己瞎敲效率要高一点。感谢老师一学期的教导，如果有机会还会再选老师的课。