10 有福同享，有难同当—原子性

更新时间：2019-10-01 20:52:06

> 耐心和恒心总会得到报酬的。
>
> ——爱因斯坦

从本节开始，我们进入新一章的学习–《并发的问题和原因详解》。关于如何实现并发，前文已经做了详尽的讲解。现在我们可以轻松的启动多个线程来完成工作，但是同样也会面临各种各样的问题。例如前一节的例子，学生和老师都要访问任务列表这个共享资源的时候，我们的程序必须加上同步才能正常运行。其实这只是问题之一，还有更多的问题等待着我们。

所有并发程序都需要保证线程的安全性，那么什么是线程的安全性呢？其实很难给出一个非常正式的定义。有些定义虽然没有错误，但好像说的又是废话。例如，线程安全是指一个类在多线程并发的情况下可以安全使用。这种定义没有任何指导价值。

其实线程安全中的安全，是指程序的正确性。程序不但要在单线程的时候保证正确，在多线程并发的时候也要保证程序计算的正确性。比如我们最初几版抄写单词的代码，只有一个线程运行是正确的，但多线程并发会使得抄写次数超过要求次数，也就是说程序运行结果不正确，那么就是非线程安全的。最后一版我们经过修改，确保多线程并行，抄写次数的总和等于要求的次数，那么就是线程安全的。所以我们线程安全可以这样定义：某个类，在多线程并发访问时，始终能够确保运行的正确性，那么这个类就是线程安全的。

确保线程安全，会面对诸多挑战。在本章中，我们将分析多线程开发中会遇到的典型问题以及其产生的根本原因。解决了这些问题，也就保证了线程安全。最后为了帮助大家对多线程问题产生的原因有更为深入的理解，我会用一节来介绍Java的内存模型。只有深刻理解了我们所使用语言的底层原理，才能够从容应对任何问题，万变不离其宗。

1. 并发编程的三大特性

所有讲并发编程的书籍都会讲到并发编程的三大特性，这是并发编程中所有问题的根源，我们只有深刻理解了这三大特性，才不会编写出漏洞百出的并发程序，才不会遇到问题时无从下手，才不会对自己的程序没有信心。

这三大特性是：

1、原子性

所有操作要么全部成功，要么全部失败。

2、可见性

一个线程对变量进行了修改，另外一个线程能够立刻读取到此变量的最新值。

3、有序性

代码在执行阶段，并不一定和你的编写顺序一致。

以上是对三大特性的简单解释。不理解也没有关系，本章中会一一进行讲解。在本节中我们重点来看原子性。

2. 什么是原子性

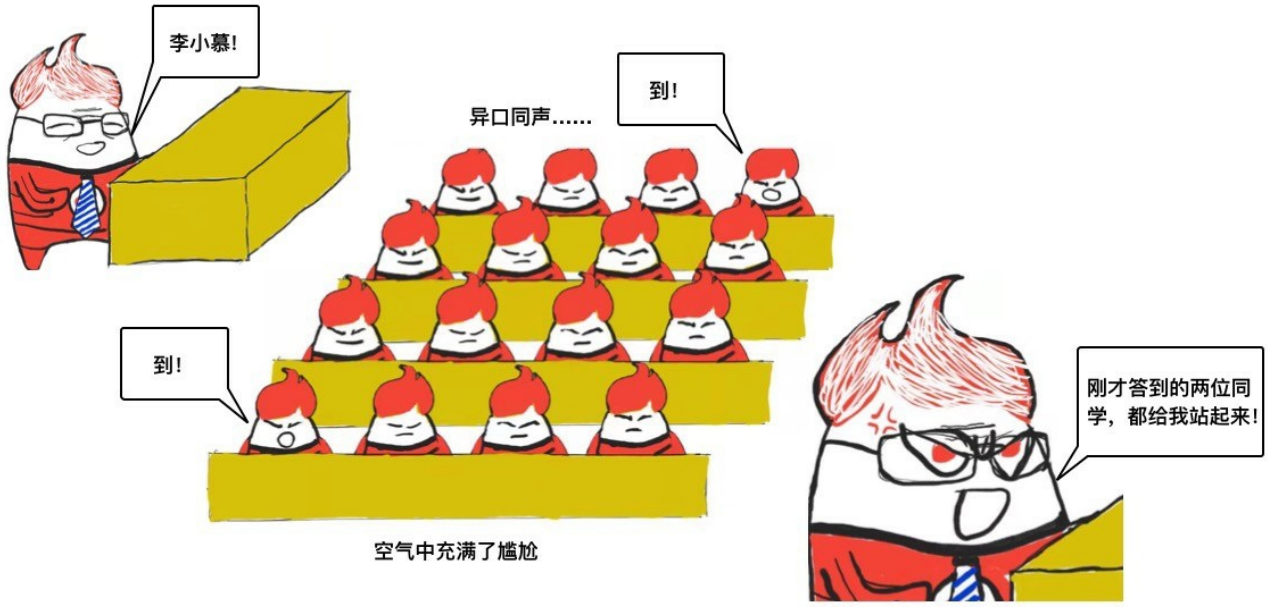
原子性是三大特性中最好理解的一个。只要你做过程序开发，应该都会听说过原子性。如果没有，那么至少听说过事务吧？原子性是事务的四大特性—ACID 之一，并且位居首位，可见其重要性。那么到底什么是原子性呢？原子性的重点在原子上。如果你的初中物理和化学，还没有因全身心投入到计算机行业，而全部还给老师，那么应该还记得原子在化学反应中不可以再分割。其实所谓的原子性就是不可分割性。做为一个整体的N次操作不可分割，一荣俱荣，一损俱损。

我们抄写单词的例子中有三步操作。第一步，查询剩余抄写次数。第二步，如果剩余次数大于零，把次数-1。第三步，把新的剩余次数更新到 punishment 对象中。这三步操作是原子操作。在操作期间，别的线程不能读取剩余抄写次数，以免别的取到更新前的旧值而重复抄写。这里我们引入一个新的概念：竞态条件。

3. 竞态条件

竞态条件是指，在多线程的情况下，由于多个线程执行的时序不同，而出现不正确的结果。上文的例子是典型的先检查后执行，这也是最常见的竞态条件类型。上面例子的问题出现在第 2、3 步操作依赖于第1步的检查，而第一步的检查结果并不能保证在执行 2、3 步的时候依旧有效。这是因为其它线程可能在你在执行完第一步时已经改变了剩余次数。此时 2，3 步依旧会按照已经失效的检查结果继续执行，那么线程安全问题就出现了。

其实现实中，我们也会经常遇到竞态条件。举个例子，你的室友中午要出去办事，可能赶不上下午第一节课。他拜托你，如果老师点名时他还没回来，帮他答一下到。下午第一节课果然老师点名了，眼看就要点到你的室友，你环顾了下四周，确认室友没有赶回来，然后紧张的等待老师点到室友的名字。老师又点了几个名字后，终于点到了你室友的名字。你故作镇定，沉稳、大方的喊了声：到！但令人尴尬的是，几乎同时，教室后面也传出了一声铿锵有力的到！你回头一看，就这几秒钟的时间，室友已经赶回了教室，从后门溜进来坐在了最后一排。



这就是竞态条件，你观察到室友没有来上课的结果，在你替室友答到的时候已经失效了。但你并不知道，依旧按照失效的观测结果执行答到。最后造成了尴尬的局面。你们精心设计好的程序执行错误，穿帮了。这多像我们精心编写一段并发程序，信心满满的去执行，却发现执行结果是错误的。

竞态条件并不一定会造成问题，正如我们前面的程序，在第一版改进后，抄写 1000 次单词，并不会出现错误。但是抄写 1 万次以上时，就会出现问题。这是因为在多线程执行时，不同线程的不同步骤在特定时序执行才会出问题。而执行次数越多就越可能碰上导致出错的特定时序。回到例子，如果你的室友没有偏偏赶在你观察和点名之前那个时间段回到教室，就不会出现任何问题。

单例是个经典的话题。仅是单例有几种写法，都够程序员们争论几天。其中有一种写法如下：

```
public class Singleton {
    private static Singleton singleton = null;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if(singleton==null){
            singleton = new Singleton();
        }
        return singleton;
    }
}
```

这段代码在非并发的情况下没有任何问题。但是在并发的情况下，因为竞态条件有可能引发错误。如果线程 A 在判断 singleton 为空并且创建 singleton 对象之前，线程B也开始执行这段代码，它同样会判断 singleton 为空去创建 singleton，这样本来的单例却变成了双例，和我们期望的正确结果不一致。

3. 总结

如果在需要保证原子性的一组操作中，有竞态条件产生，那么就会出现线程安全的问题。我们可以通过为原子操作加锁或者使用原子变量来解决。原子变量在 java.util.concurrent.atomic 包中，它提供了一系列的原子操作。后面的章节我们会深入讲解原子变量的使用和原理。

}

09 使用多线程实现分工、解耦、缓冲—生产者、消费者实战

11 眼见不实—可见性