不要重新分配被锁定对象的对象引用 PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/145/2021_2022__E4_B8_8D_E8_A6_81_E9_87_8D_E6_c104_145652.htm synchronized 关键字

锁定对象。对象是在 synchronized 代码内部被锁定的，这一点对此对象以及您对其对象引用所作的更改意味着什么呢？
对一个对象作同步处理只锁定该对象。但是，必须注意不要重新分配被锁定对象的对象引用。那么如果这样做会发生什么情况呢？请考虑下面这段代码，它实现了一个 Stack：

```
class Stack { private int StackSize = 10; private int[] intArr = new int[StackSize]; private int index; //Stack 中的下一个可用位置  
    public void push(int val) { synchronized(intArr) { //如果已满，则重新分配整数数组（即我们的 Stack）。 if (index == intArr.length) { StackSize *= 2; int[] newintArr = new int[StackSize]; System.arraycopy(intArr, 0, newintArr, 0, intArr.length); intArr = newintArr; } intArr[index] = val; index++; } }  
    public int pop() { int retval; synchronized(intArr) { if (index > 0) { retval = intArr[index-1]; //检索值，index--。//并使 Stack 减少 1 个值。 return retval; } } throw new EmptyStackException(); } //... } 
```


这段代码用数组实现了一个 Stack。创建了一个初始大小为 10 的数组来容纳整数值。此类实现了 push 和 pop 方法来模拟 Stack 的使用。在 push 方法中，如果数组中没有更多的空间来容纳压入的值，则数组被重新分配以创建更多的存储空间。（故意没有用 Vector 来实现这个类。Vector 中不能储存基本类型。）
请注意，这段代码是要由多个线程进行访问的。push 和 pop 方法每次对该类的共享实例数据的访问都是在

synchronized 块内完成的。这样就保证了多个线程不能并发访问此数组而生成不正确的结果。这段代码有一个主要的缺点。它对整数数组对象作了同步处理，而这个数组被 Stack 类的 intArr 所引用。当 push 方法重新分配此整数数组时，这个缺点就会显露出来。当这种情况发生时，对象引用 intArr 被重新指定为引用一个新的、更大的整数数组对象。请注意，这是在 push 方法的 synchronized 块执行期间发生的。此块针对 intArr 变量引用的对象进行了同步处理。因此，在这段代码内锁定的对象不再被使用。请考虑以下的事件序列：线程 1 调用 push 方法并获得 intArr 对象的锁。线程 1 被线程 2 抢先。线程 2 调用 pop 方法。此方法因试图获取当前线程 1 在 push 方法中持有的同一个锁而阻塞。线程 1 重新获得控制并重新分配数组。intArr 变量现在引用一个不同的变量。push 方法退出并释放它对原来的 intArr 对象的锁。线程 1 再次调用 push 方法并获得新 intArr 对象的锁。线程 1 被线程 2 抢先。线程 2 获得旧 intArr 对象的对象锁并试图访问其内存。现在线程 1 持有由 intArr 引用的新对象的锁，线程 2 持有由 intArr 引用的旧对象的锁。因为两个线程持有不同的锁，所以它们可以并发执行 synchronized push 和 pop 方法，从而导致错误。很明显，这不是所希望的结果。这个问题是因 push 方法重新分配被锁定对象的对象引用而造成的。当某个对象被锁定时，其他线程可能在同一个对象锁上被阻塞。如果将被锁定对象的对象引用重新分配给另一个对象，其他线程的挂起锁则是针对代码中已不再相关的对象的。您可以这样修正这段代码，去掉对 intArr 变量的同步，而对 push 和 pop 方法进行同步。通过将 synchronized 关键字添加为方法修饰符即可实

现这一点。正确的代码如下所示：`class Stack{//与前面相同...public synchronized void push(int val){//如果为空，则重新分配整数数组（即我们的 Stack）。if (index == intArr.length){stackSize *= 2.int[] newintArr = new int[stackSize].System.arraycopy(intArr, 0, newintArr, 0, intArr.length).intArr = newintArr.}intArr[index]= val.index.}public synchronized int pop(){int retval.if (index > 0){retval = intArr[index-1].index--.return retval.}throw new EmptyStackException().}`这个修改更改了实际上获取的锁。获取的锁是针对为其调用方法的对象的，而不是锁定 `intArr` 变量所引用的对象。因为获取的锁不再针对 `intArr` 所引用的对象，所以允许代码重新指定 `intArr` 对象引用。作者简介 Peter Haggart 是 IBM 的高级软件工程师。他目前正在研究新兴的 Java 和因特网技术，并且是 IBM 实时 Java 参考实现的项目主持人。他有丰富的编程经验，从事过开发工具、类库和操作系统等方面的工作。在许多行业研讨会上，他也经常就 Java 和其他技术作技术性发言。他于 1987 年在纽约获得 Clarkson 大学计算机科学学士学位。可以通过 haggart@us.ibm.com 与他联系。100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com