 6.2多线程的互斥与同步 临界资源问题 前面所提到的线程都是独立的，而且异步执行，也就是说每个线程都包含了运行时所需要的数据或方法，而不需要外部的资源或方法，也不必关心其它线程的状态或行为。但是经常有一些同时运行的线程需要共享数据，此时就需考虑其他线程的状态和行为，否则就不能保证程序的运行结果的正确性。例6.4说明了此问题。 例6.4 class stack{ int idx=0. //堆栈指针的初始值为0 char[ ] data = new char[6]. //堆栈有6个字符的空间 public void push(char c){ //压栈操作 data[idx] = c. //数据入栈 idx . //指针向上移动一位 } public char pop(){ //出栈操作 idx - -. //指针向下移动一位 return data[idx]. //数据出栈 } } 两个线程A和B在同时使用Stack的同一个实例对象，A正在往堆栈里push一个数据，B则要从堆栈中pop一个数据。如果由于线程A和B在对Stack对象的操作上的不完整性，会导致操作的失败，具体过程如下所示： 1) 操作之前 data = | p | q | | | | | idx=2 2) A执行push中的第一个语句，将r推入堆栈； data = | p | q | r | | | | idx=2 3) A还未执行idx 语句，A的执行被B中断，B执行pop方法，返回q： data = | p | q | r | | | | idx=1 4〕A继续执行push的第二个语句： data = | p | q | r | | , | | idx=2 最后的结果相当于r没有入栈。产生这种问题的原因在于对共享数据访问的操作的不完整性。 6.2.1 互斥锁 为解决操作的不完整性问题，在Java 语言中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。每个对象都对应于一个可称为" 互斥锁" 的

标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。 关键字synchronized 来与对象的互斥锁联系。当某个对象用synchronized 修饰时，表明该对象在任一时刻只能由一个线程访问。 来源：www.examda.com public void push(char c){ synchronized(this){ //this表示Stack的当前对象 data[idx]=c. idx . } } public char pop(){ synchronized(this){ //this表示Stack的当前对象 idx--. return data[idx]. } } 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