用管程实现生产者消费者、JAVA 中的类似机制

管程的应用

下面呢，我们来看一下管程是怎样使用的

管程的实现

管程实现的两个主要途径：

* 直接构造 → 效率高
* 间接构造
  → 用某种已经实现的同步机制去构造

例如：用信号量及P、V操作构造管程

P、V 操作来构造一个管程 有了管程之后，如何用管程

用管程解决生产者消费者问题

```
monitor ProducerConsumer
    condition full, empty;
    integer count;

    procedure insert (item: integer);
    begin
        if count == N then wait(full);
        insert_item(item); count++;
        if count ==1 then signal(empty);
    end;

    function remove: integer;
    begin
        if count ==0 then wait(empty);
        remove = remove_item; count--;
        if count==N-1 then signal(full);
    end;

    count:=0;
end monitor;
```

```
procedure producer;
begin
    while true do
    begin

        item = produce_item;
        ProducerConsumer.insert(item)
    end
end;

procedure consumer;
begin
    while true do
    begin
        item=ProducerConsumer.remove
        consume_item(item);

    end
end;
```

6:38 / 11:23

JAVA中的类似机制(1/2)

```
public class ProducerConsumer {
    static final int N = 100;          // constant giving the buffer size
    static producer p = new producer( );    // instantiate a new producer thread
    static consumer c = new consumer( ); // instantiate a new consumer thread
    static our_monitor mon = new our_monitor( );     // instantiate a new monitor

    public static void main(String args[ ]) {
        p.start( );       // start the producer thread
        c.start( );       // start the consumer thread
    }

    static class producer extends Thread {
        public void run( ) {// run method contains the thread code
            int item;
            while (true) {      // producer loop
                item = produce_item( );
                mon.insert(item);
            }
        }
        private int produce_item( ) { ... }      // actually produce
    }
```

JAVA中的类似机制(2/2)

```
static class consumer extends Thread {
   public void run( ) {run method contains the thread code
      int item;
      while (true) {      // consumer loop
         item = mon.remove( );
         consume_item (item);
      }
   }
   private void consume_item(int item) { ... }// actually consume
}

static class our_monitor {  // this is a monitor
   private int buffer[ ] = new int[N];
   private int count = 0, lo = 0, hi = 0;  // counters and indices

   public synchronized void insert(int val) {
      if (count == N) go_to_sleep( );     // if the buffer is full, go to sleep
      buffer [hi] = val; // insert an item into the buffer
      hi = (hi + 1) % N;          // slot to place next item in
      count = count + 1;          // one more item in the buffer now
      if (count == 1) notify( );          // if consumer was sleeping, wake it up
   }
```

11:22 / 11:23

0:00 下面呢，我们来看一下管程是怎样使用的 怎么样利用管程来解决 我们碰到的同步互斥问题 在这个介绍这个管程使用之前呢，我们简单地提一下 管程的实现。 管程的实现呢主要有两种途径 第一种途径呢，是直接构造 因为它是语言机制，所以我们可以 在某个语言当中加入这样一个管程成分 然后呢，去编写相应的编译器 好，那么第二种方法呢，是间接构造 也就是如果我已经有了一种同步机制 我可以用这种同步机制去构造管程这种新的同步机制 所以可以用已有的同步机制，比如说我们已经学过的是信号量及 P、 V 操作 所以我们完全可以用信号量及 P、 V 操作来构造一个管程 那在这里头我们只是做了这样一个简要的介绍 那么感兴趣的同学呢可以去看一下相关的资料，或者是试着 用信号量、 P、 V 操作来构造一个管程 有了管程之后，如何用管程 来解决进程间的同步互斥问题呢？ 我们用一个例子 用管程解决生产者消费者为例，我们来介绍 怎么样去用管程来解决问题 有了管程之后 由于管程把生产者消费者 所要放数据、 取数据的这个缓冲区 统一管起来了，提供了相应的操作，因此 对于生产者和消费者进程，它们的编程就变得简单了 我们来看一下，生产者它生产了一个产品 它只需要调用管程的相应操作把这个产品放到相应的缓冲区 而消费者呢调用管程相应的操作 从缓冲区去取产品，然后做相应的其它操作 那我们来看看，在生产者消费者这个问题当中，管程是怎么设计的 管程 呢是有这么一个起的一个名字，首先管程要有一个名字 然后管程呢里头要有相应的条件变量，当条件不满足的 时候呢，要有一个等待的这样一个条件变量的一个机制，所以呢这里头设置了两个条件变量 那么管程还有一些其它的数据结构变量，比如说我们这有一个 count count 的初值是 0，这初始化是 0 那么 count 是记录了这个缓冲区里头有没有数据，有几个数据 那么下面就是管程当中的几个典型的操作 两个过程，一个过程呢是 insert ，往缓冲区里送数据 另外一个呢，是从缓冲区里取数据 那么其它的就和生产者消费者的这样一个问题就很相似了 比如说，当判断 count == N， count == N 说明缓冲区满了 那么这个时候，就要去调用 wait 操作，使得调用 这个操作的这个进程处于等待，等在某个条件变量上，所以呢我们来看看 wait 操作 wait 到 full 这个条件变量上 表示缓冲区已经满了。 那么同样 当一个生产者调用了 insert 操作，把一个数据放到了缓冲区里以后 当然不是缓冲区满吗，就可以放。 这时候放入缓冲区 count++，那么缓冲区里的这个数据就多了一个 那么如果之前可能有消费者来消费，没有 得到相应的数据，所以这时候呢它就等在条件变量上，所以这里头呢我们要判断一下 那么看看有没有进程等在条件变量上，所以要做一个 signal 操作。 那么什么情况下做

呢？就是当 count == 1 的时候 也就是说可能刚才 count 因为等于 0 有消费者来没有取到数据，那么它进入等待状态 那么等到 count = 1 我放了一个数据之后，就把它唤醒了 这个呢和用信号量及 P、V 操作解决问题的这个思路是完全一致的 那我们再看看取数据 remove，remove 也是一样 如果 count == 0，没有数据可取，那么就去 等在条件变量 empty 上。如果 count == N- 1 也就是刚才是 N，我取走了一个 count- 1 那么空出来一个缓冲区，那么就可能有 其它的进程正好等这个缓冲区，所以要做一个 signal 操作 所以用管程来解决生产者消费者问题，对于 生产者和消费者来讲，编程简单了 那么怎么简单呢？我们来看看，生产者生产完产品之后只需要调用管程的 insert 这个过程操作就可以完成了，消费者也是一样 只需要调用 remove，管程的 remove 操作就可以完成了 那么这就是用一个管程，设计的一个管程来管理 缓冲区这样一些资源，然后呢生产者消费者这些进程呢就可以 调用相应的操作来对缓冲区进行相应的这个 送数据和取数据这样一个过程。 我们刚才介绍了 管程是程序设计语言的一个成分 但是我们常用的 C 语言、C++ 没有支持管程 但是我们常用的 Java 却对管程有类似的机制 我们来看一下，我们用 Java 来实现 生产者消费者这个问题时，怎么样利用这样一个机制 那么这里有两个线程，生产者和消费者线程 它们都继承了 thread 类。 然后呢 我们看到了一个类，这个类呢是 有 monitor，实际上它是一个管程，是一个管程类 那么生产者我们要启动 生产者这个线程，通过调用 run 这样一个函数 那么这里头给出了生产者线程的 主要的工作，那么生产者主要做的一件事就是 先申请，先生产一个 数据，生产一个产品。当它生产完产品之后，它需要调用管程 相应的操，那么来把这个产品放到缓冲区里头，那么我们这里头示意就是这样 因为我们的那个类呢是这样一个 monitor 这个类，所以我们这里头呢 管程，调用管程的一个操作，这和刚才那个非常相似，就管程的名字，这就 不一样，其它的都很相似。 那么消费者呢也是通过 调用管程相应的 remove 操作来完成取数据的工作 然后再去做相应的消费。 下面 我们呢看一下，在 Java 当中这个管程是怎么设计的 我们首先看到这是一个类，这是一个类，那么这个类呢实际上就是代表的管程 管程。 那么这个管程呢管理的相应的一些共享资源，一些数据结构 那么管程里提供了相应的 insert 操作和 remove，那在这里头呢，我们只给出了 insert 操作的代码 那么在教材里头还给出这个 remove 的代码，大家可以去看一下 那么 insert 这个操作 具有什么性质呢？我们来看一下，有一个关键

词 synchronized 通过这个关键词，表达出说这个 insert 操作 只允许一个线程来调用，也就是说在这个管程类当中的 这个操作只允许一个线程来调用 那么在它没有结束之前，其它的线程是不能调用的 所以 synchronized 就是表示的一个互斥使用，这样一个 含义。 那么 insert 操作其它的和我们前面介绍的很相似，比如说我们 看一下，如果 count == N ，就缓冲区满了 那么这个线程就要，生产者线程就要去等待、 去睡眠，那么它呢 编了一个 go to sleep 这么一个函数，这个呢函数呢 是一个这个自己编的一个函数，那么它呢实际上是调用了 Java 提供的这个 wait， wait 操作，但是这里头就没把 wait 写在这，因为一条，一个函数呢是不能解决很多问题，所以呢这里头是 go_to_sleep 用这么一个来表示这个线程呢就去睡眠了 好，那么如果 缓冲区刚开始是空的，放进去一个数据之后，那么 count == 1 了 这个时候呢可能会有其它的线程等待这个数据，所以这时候要做一个 notify 所以我们看到， notify 的意思呢就是说 通知一个等待的这个线程，相当于把它从睡眠呢给唤醒 那么这里头呢，我们知道就 wait 和 notify 呢实际上是 Java 当中 对管程，在管程里头提供的这两个相应的操作