Java

面向对象程序设计

软件学院 贾伟峰

复习


第五次课的内容
进程
线程
并发
概念
继承Thread
实现Runnable
区别
创建线程
线程状态
新建
就绪
阻塞
运行
死亡
线程调度
优先级
休眠
让步
插队

多线程模拟4个窗口卖票

```
1  public class Example05
2  {
3      public static void main(String[] args){
4          TicketWindow tw = new TicketWindow();
5          new Thread(tw, "窗口1").start();
6          new Thread(tw, "窗口2").start();
7          new Thread(tw, "窗口3").start();
8          new Thread(tw, "窗口4").start();
9      }
10 }
11 class TicketWindow implements Runnable
12 {
13     private int tickets = 100;
14     public void run(){
15         while(true){
16             if(tickets > 0){
17                 Thread th = Thread.currentThread();
18                 String th_name = th.getName();
19                 System.out.println(th_name + "正在发售" + tickets-- +"张票");
20             }
21         }
22     }
23 }
```

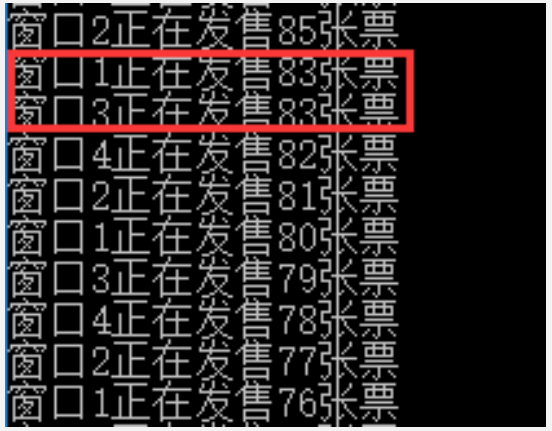
借助tw同时开启了4个线程，共享余额tickets。

```
窗口1正在发售100张票
窗口1正在发售98张票
窗口1正在发售95张票
窗口1正在发售94张票
窗口2正在发售99张票
窗口1正在发售93张票
窗口4正在发售96张票
窗口3正在发售97张票
窗口4正在发售90张票
窗口1正在发售91张票
窗口2正在发售92张票
窗口1正在发售87张票
窗口4正在发售88张票
窗口3正在发售89张票
窗口4正在发售84张票
窗口1正在发售85张票
窗口2正在发售86张票
```

理论上讲，这个线程程序是不安全的，为什么呢?

同一张票被发售多次、票卖完了还会再卖（为把进度放慢，用了sleep方法），为什么?

```
11  class TicketWindow implements Runnable
12  {
13      private int tickets = 100;
14      Object lock = new Object();
15      public void run(){
16          while(true){
17
18              if(tickets > 0){
19                  try{
20                      Thread.sleep(10);
21                  }catch(InterruptedException e){
22                      e.printStackTrace();
23                  }
24                  Thread th = Thread.currentThread();
25                  String th_name = th.getName();
26                  System.out.println(th_name + "正在发售" + tickets-- +"张票");
27              }
28
29          }
30      }
31  }
```



有什么好的解决办法吗？

同步synchronized

加了同步（synchronized）的代码

```
11  class TicketWindow implements Runnable
12  {
13      private int tickets = 100;
14      Object lock = new Object();
15      public void run(){
16          while(true){
17              synchronized(lock){
18                  if(tickets > 0){
19                      try{
20                          Thread.sleep(10);
21                      }catch(InterruptedException e){
22                          e.printStackTrace();
23                      }
24                      Thread th = Thread.currentThread();
25                      String th_name = th.getName();
26                      System.out.println(th_name + "正在发售" + tickets-- +"张票");
27                  }
28              }
29          }
30      }
31  }
```

lock是什么？lock只是一把锁。谁先用了这把锁，其他都得等！

或者，提出代码，写到方法中，给方法加synchronized

```java
public class Example13
{
    public static void main(String[] args){
        TicketWindow tw = new TicketWindow();
        new Thread(tw, "窗口1").start();
        new Thread(tw, "窗口2").start();
        new Thread(tw, "窗口3").start();
        new Thread(tw, "窗口4").start();
    }
}
class TicketWindow implements Runnable
{
    private int tickets = 100;
    Object lock = new Object();
    public void run(){
        while(true){
            saleTicket();
            if(tickets <= 0){
                break;
            }
        }
    }
    private synchronized void saleTicket(){
        if(tickets > 0){
            try{
                Thread.sleep(10);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
            Thread th = Thread.currentThread();
            String th_name = th.getName();
            System.out.println(th_name + "正在发售" + tickets-- +"张票");
        }
    }
}
```

这里lock的，是调用该方法的对象！谁调用saleTicket，就"lock"谁，其他想调用saleTicket的，就得等。

同步使用不当，会造成“死锁”！

死锁

```
class DeadLockThread implements Runnable{
    static Object chopsticks = new Object();
    static Object knifeAndFork = new Object();
    private boolean flag;
    DeadLockThread(boolean flag){
        this.flag = flag;
    }
    public void run(){
        if(flag){
            while(true){
                synchronized(chopsticks){
                    System.out.println(Thread.currentThread().getName()+"---if---chopsticks");
                    synchronized(knifeAndFork){
                        System.out.println(Thread.currentThread().getName()+"---if---knifeAndFork");
                    }
                }
            }
        }else
        {
            while(true){
                synchronized(knifeAndFork){
                    System.out.println(Thread.currentThread().getName()+"---if---knifeAndFork");
                    synchronized(chopsticks){
                        System.out.println(Thread.currentThread().getName()+"---if---chopsticks");
                    }
                }
            }
        }

    }
}
```

```
public class Example14{
    public static void main(String[] args){
        DeadLockThread d1 = new DeadLockThread(true);
        DeadLockThread d2 = new DeadLockThread(false);
        new Thread(d1, "Chinese").start();
        new Thread(d2, "American").start();
    }
}
```

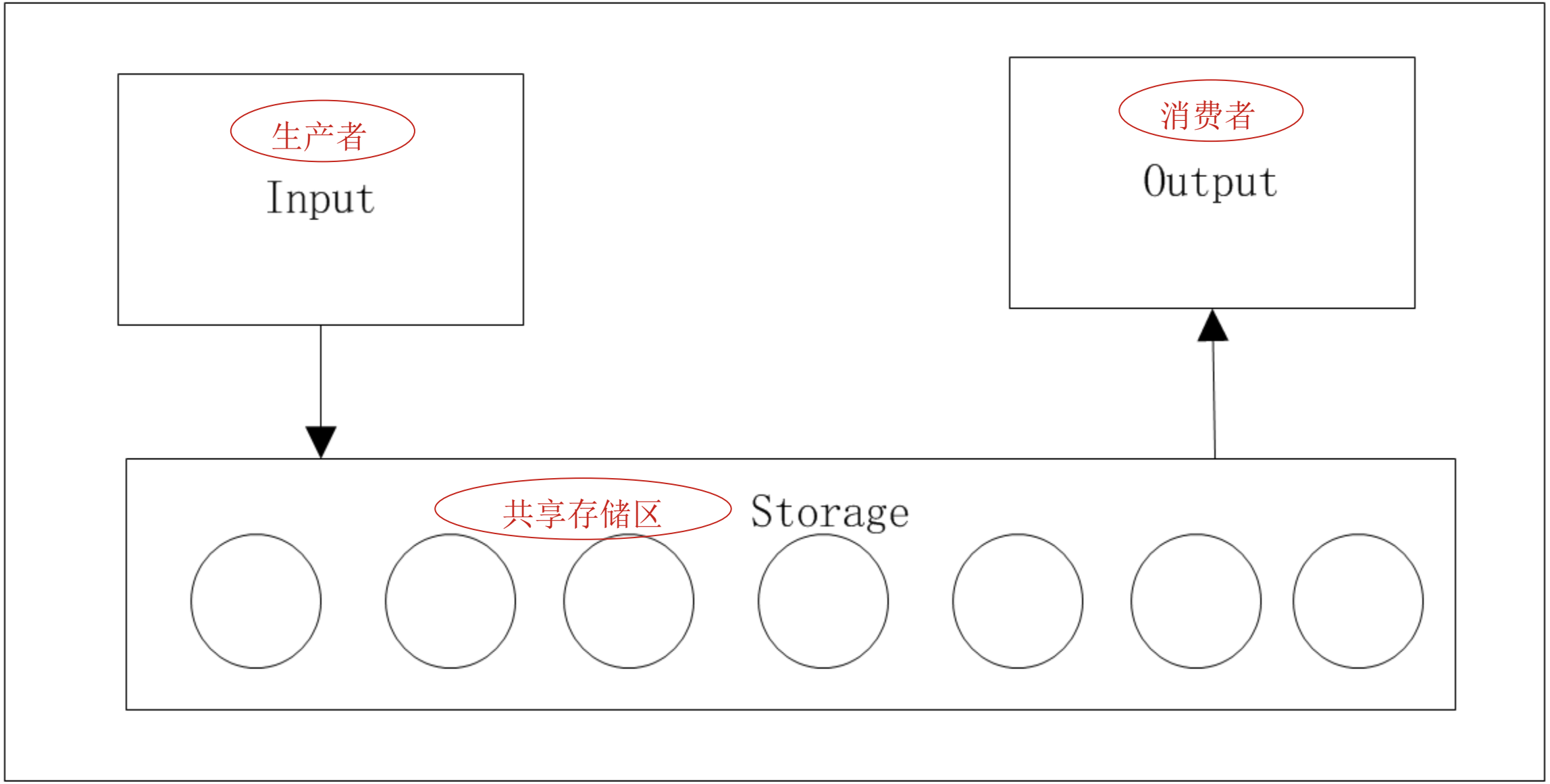
拓展思考：如何解决死锁?

死锁预防与诊断*

一次封锁法，顺序封锁法；超时法，等待图法……

线程间通信

先来看看生产者和消费者问题



初级代码实现：不加任何控制的生产和消费

```java
class Storage{
    private int[] cells = new int[10];
    private int inPos, outPos;
    public void put(int num){
        cells[inPos] = num;
        System.out.println("在cells["+inPos+"]中放入数据---"+cells[inPos]);
        inPos++;
        if(inPos == cells.length){
            inPos = 0;
        }
    }
    public void get(){
        int data = cells[outPos];
        System.out.println("从cells["+outPos+"]中取出数据"+data);
        outPos++;
        if(outPos == cells.length)
            outPos = 0;
    }
}
```

放入数据

取出数据

```java
class Output implements Runnable{
    private Storage st;
    Output(Storage st){
        this.st = st;
    }
    public void run(){
        while(true){
            st.get();
        }
    }
}
```

模拟消费者，只从Storage中取数据

```java
class Input implements Runnable{
    private Storage st;
    private int num;
    Input(Storage st){
        this.st = st;
    }
    public void run(){
        while(true){
            st.put(num++);
        }
    }
}
```

模拟生产者，只放数据到Storage

```java
public class Example17{
    public static void main(String[] args){
        Storage st = new Storage();
        Input input = new Input(st);
        Output output = new Output(st);
        new Thread(input).start();
        new Thread(output).start();
    }
}
```

生产者

消费者

比如还没生产就有消费，
后果很严重。

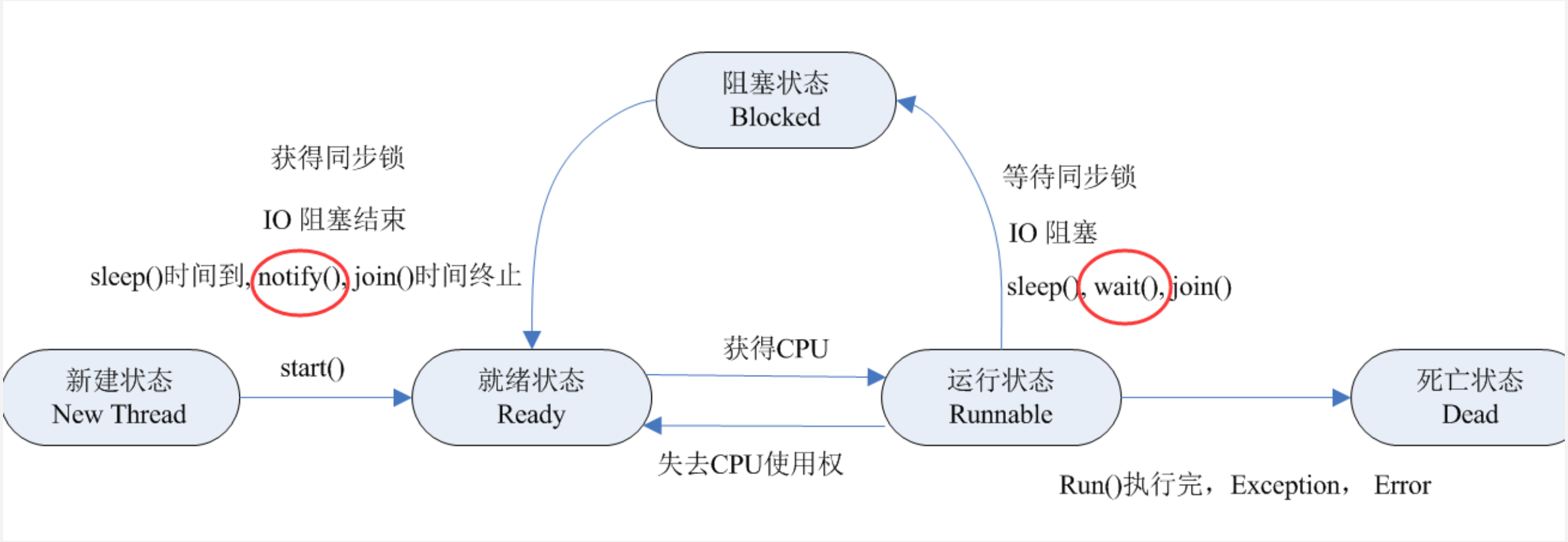
```
从cells[0]中取出数据0
在cells[0]中放入数据---0
从cells[1]中取出数据0
在cells[1]中放入数据---1
从cells[2]中取出数据0
在cells[2]中放入数据---2
从cells[3]中取出数据0
在cells[3]中放入数据---3
从cells[4]中取出数据0
在cells[4]中放入数据---4
从cells[5]中取出数据0
```

如果Storage中没数据，当然不能提取！如果Storage中数据满了，肯定无法继续存入！

如何表示Storage无数据？如何表示Storage已满？如何不让提取？如何不让存入？

在Storage中引入计数的count，要注意同步啊！

不让提取和存入，那就（让调用它的线程）wait了。待有条件的时候，就notify它继续运行啦！



改进后的代码

```
1  class Storage{
2      private int[] cells = new int[10];
3      private int inPos, outPos;
4      private int count;
5      public synchronized void put(int num){
6          try{
7              while(count == cells.length){
8                  this.wait();
9              }
10             cells[inPos] = num;
11             System.out.println("在cells["+inPos+"]中放入数据---"+cells[inPos]);
12             inPos++;
13             if(inPos == cells.length){
14                 inPos = 0;
15             }
16             count++;
17             this.notify();
18         }catch(Exception e){
19             e.printStackTrace();
20         }
21     }
22     public synchronized void get(){
23         try{
24             while(count == 0){
25                 this.wait();
26             }
27             int data = cells[outPos];
28             System.out.println("从cells["+outPos+"]中取出数据"+data);
29             outPos++;
30             if(outPos == cells.length)
31                 outPos = 0;
32             count--;
33             this.notify();
34         }catch(Exception e){
35             e.printStackTrace();
36         }
37     }
38 }
```

生产者如果遇到Storage满了，则等待

生产一个数据后，个数加1，通知消费者（因为消费者可能因为Storage中没有数据了，从而在等待）。

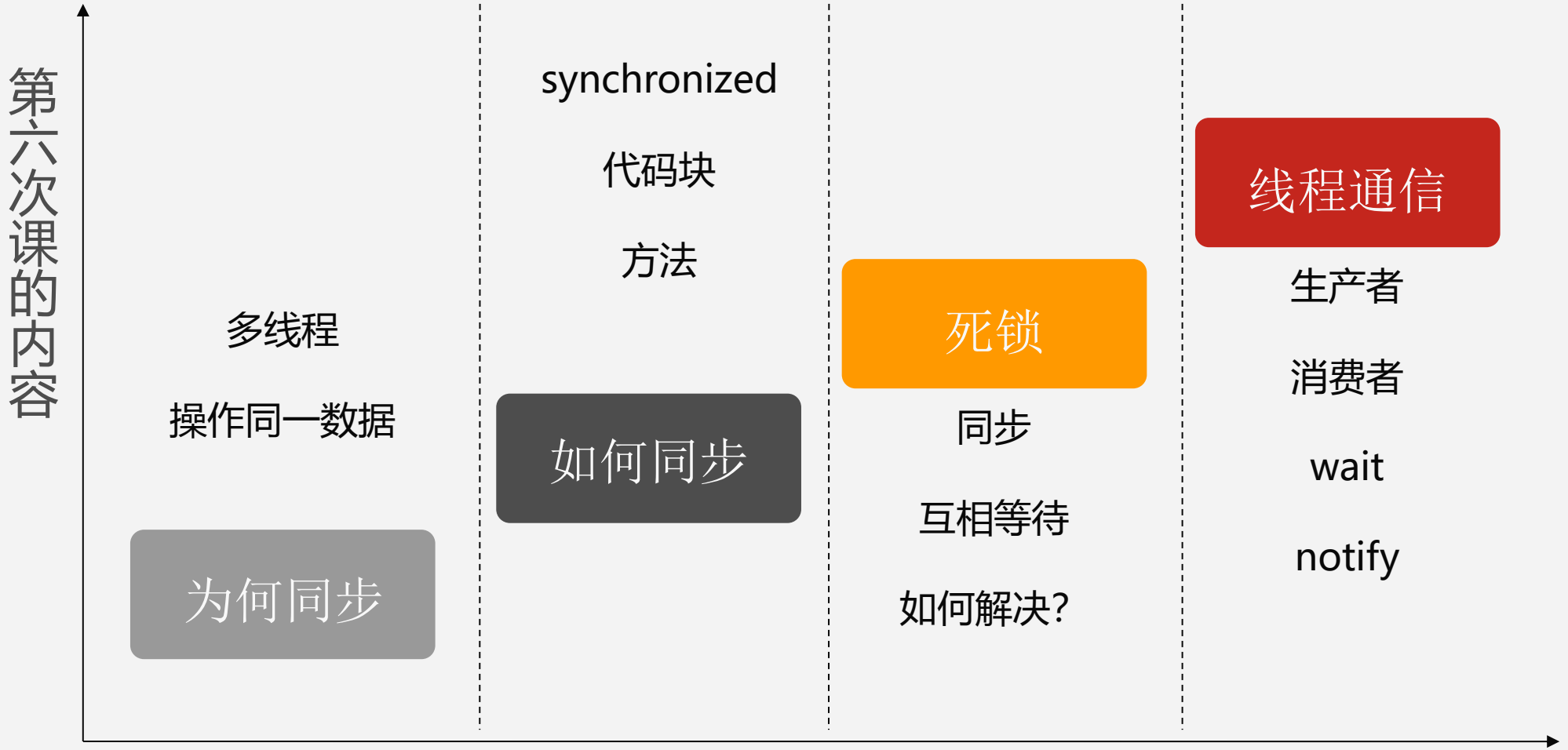
消费者如果遇到Storage空了，则等待

消费一个数据后，个数减1，通知生产者（因为生产者可能因为Storage中满了，从而在等待）。

```
在cells[0]中放入数据---0
在cells[1]中放入数据---1
在cells[2]中放入数据---2
在cells[3]中放入数据---3
在cells[4]中放入数据---4
在cells[5]中放入数据---5
在cells[6]中放入数据---6
在cells[7]中放入数据---7
在cells[8]中放入数据---8
在cells[9]中放入数据---9
从cells[0]中取出数据0
从cells[1]中取出数据1
从cells[2]中取出数据2
从cells[3]中取出数据3
从cells[4]中取出数据4
从cells[5]中取出数据5
从cells[6]中取出数据6
从cells[7]中取出数据7
从cells[8]中取出数据8
从cells[9]中取出数据9
```

这，就是线程间的通信！wait后，被别的线程notify

总结


第六次课的内容
多线程
操作同一数据
为何同步
synchronized
代码块
方法
如何同步
死锁
同步
互相等待
如何解决?
线程通信
生产者
消费者
wait
notify