

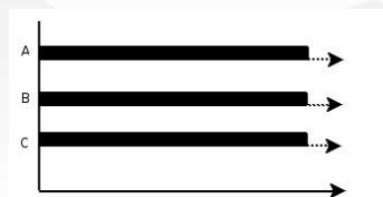
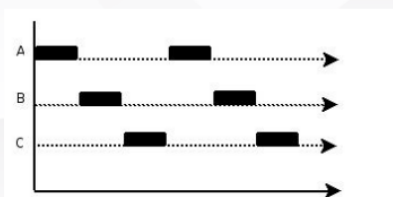
Java Thread

1、线程与进程

- 什么是进程？
 - 进程是指运行中的应用程序，每一个进程都有自己独立的内存空间。一个应用程序可以同时启动多个线程。进程也是程序的一次执行过程，是系统运行程序的基本单位。
- 什么是线程？
 - 线程是指进程中的一个执行流程。一个进程可以由多个线程组成。
- 进程和线程的区别？
 - 一个程序运行后至少有一个进程，一个进程中可以包含多个线程。
 - 每个进程都需要操作系统为其分配独立的内存地址空间，而同一进程中的所有线程共享同一地址空间。

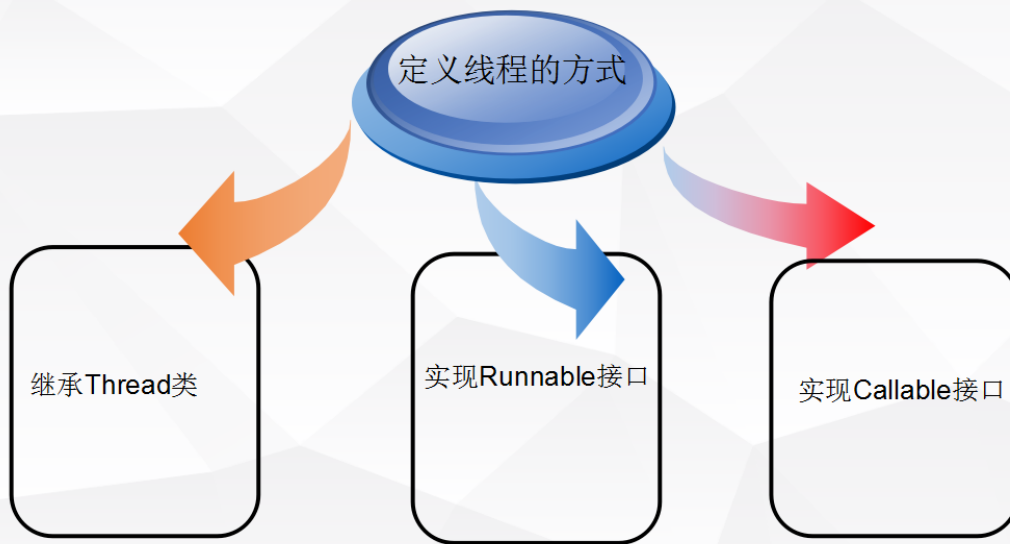
2、并发与并行

- 什么是并发？
 - 指在同一时刻只能有一条指令执行，但多个进程指令被快速的轮换执行，使得在宏观上具有多个进程同时执行的效果，但在微观上并不是同时执行的，只是把时间分成若干段，使多个进程快速交替的执行。
- 什么是并行？
 - 指在同一时刻，有多条指令在多个处理器上同时执行



3、线程的创建方式

线程创建方式



3-1、Thread 类

```
1 public class Test01_thread
2 {
3     public static void main(String[] args)
4     {
5         //常规运行
6         //     Mythread m1 = new Mythread("A程序");
7         //     Mythread m2 = new Mythread("B程序");
8         //
9         //     m1.work();
10        //     m2.work();
11
12        //使用线程
13        Thread t1 = new Mythread("A线程");
14        Thread t2 = new Mythread("B线程");
15        //run方法并不能开启线程，只是一个普通的方法
16        //     t1.run();
17        //     t2.run();
18        //开启线程，抢夺CPU资源
19        t1.start();
20        t2.start();
21    }
22 }
23
24
25 class Mythread extends Thread
26 {
27     private String name;
28     public Mythread(String name)
29     {
30         this.name = name;
31     }
32 }
```

```

32     public void work()
33     {
34         while(true)
35         {
36             System.out.println(name+"执行了");
37         }
38     }
39
40     //需要重写父类中的run方法
41
42     @Override
43     public void run()
44     {
45         work();
46     }
47 }

```

3-2、Runnable 接口

```

1  public class Test01_runnable
2  {
3      public static void main(String[] args)
4      {
5          //启动runnable线程
6          Runnable r = new Son();
7
8          //线程的启动都是依靠thread类
9          //把我们写的runnable实现类，作为参数传递给线程thread
10         //这个时候thread线程就会自动的去执行我们的run方法
11         Thread t = new Thread(r);
12         Thread t2 = new Thread(r);
13         //我们需要线程的名字，来区分哪个线程执行了，哪个线程没有执行
14         t.setName("线程1");
15         t2.setName("线程2");
16         //线程启动
17         t.start();
18         t2.start();
19     }
20 }
21 class Father
22 {
23
24 }
25 //如果我们自己写的类出现了下面这种情况，这时候，就不能再去继承thread类了，因为Java是单继承
26 //这种情况下，我们只能去使用runnable接口
27 class Son extends Father implements Runnable
28 {
29
30     @Override
31     public void run()
32     {
33         int num = 0;

```

```

34     while(num < 100)
35     {
36         //获取当前执行的线程
37         String name = Thread.currentThread().getName();
38
39         //我们可以认为的操作线程的执行
40         if(name.equals("线程1") && num < 30)
41         {
42             System.out.println(name+":正在执行"+num);
43         }
44         else if (name.equals("线程2") && num > 30)
45         {
46             System.out.println(name+":正在执行"+num);
47         }
48         num++;
49     }
50 }
51 }

```

3-3、Callable 接口

可以有返回值

```

1  public class Test01_callable
2  {
3      public static void main(String[] args) throws ExecutionException, InterruptedException
4      {
5          MyCallable my = new MyCallable();
6
7          FutureTask<String> ft = new FutureTask<>(my);
8
9          Thread t = new Thread(ft);
10         //启动
11         t.start();
12         //获取到返回值
13         String result = ft.get();
14
15         System.out.println(result);
16     }
17 }
18 class MyCallable implements Callable<String>
19 {
20
21     @Override
22     public String call() throws Exception
23     {
24         String msg = Thread.currentThread().getName()+"说你好";
25         return msg;
26     }
27 }

```

3-4模拟多文件下载(简单版)

```
1 public class Test02_模拟文件下载
2 {
3     public static void main(String[] args) throws Exception
4     {
5
6         long begin = System.currentTimeMillis();
7         String a_url = "c:\\video\\a.mp4";
8         String b_url = "c:\\video\\b.mp4";
9
10
11         DownloadFile f1 = new DownloadFile(a_url);
12         DownloadFile f2 = new DownloadFile(b_url);
13
14         // f1.download();
15         // f2.download();
16         //给线程设置名字, setName来源父类Thread
17         f1.setName("A线程");
18         f2.setName("B线程");
19
20         f1.start();
21         f2.start();
22
23         long end = System.currentTimeMillis();
24         System.out.println("main主线程运行时间: " + (end - begin) + "毫秒");
25
26     }
27
28
29
30 }
31 class DownloadFile extends Thread
32 {
33     private String old_url = "src\\download\\file.mp4";
34     private String new_url;
35
36     public DownloadFile(String new_url)
37     {
38         this.new_url = new_url;
39     }
40
41     public void download() throws Exception
42     {
43         System.out.println(getName() + "开始下载文件...");
44         long begin = System.currentTimeMillis();
45         //根据要操作的文件的类型 (不是处理字符串的, 统一使用字节流)
46         File old_file = new File(old_url); //要下载的文件
47         File new_file = new File(new_url); //要存放的文件位置
48         //如果有该文件, 就删掉, 方便我们多次测试
49         if(new_file.exists())
50         {
```

```

51         new_file.delete();
52     }
53     //创建文件，一定可以创建出来
54     new_file.createNewFile();
55
56     FileInputStream fis = new FileInputStream(old_file); //从该文件中读取数据给内存
57     FileOutputStream fos = new FileOutputStream(new_file); //向新文件输出内容
58
59     //为了提高效率
60     BufferedInputStream bufferedInputStream = new BufferedInputStream(fis);
61     BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(fos);
62
63     //每次读写的大小
64     byte[] bs = new byte[1024*1024*4]; //一次读取4MB的数据
65
66     //边读边写
67     int num;
68     while((num = bufferedInputStream.read(bs)) != -1)
69     {
70         //在内存中展示数据
71         System.out.println(num);
72         //写入到新文件中
73         bufferedOutputStream.write(bs,0,num);
74     }
75
76     bufferedInputStream.close();
77     //内存中清空资源
78     bufferedOutputStream.flush();
79     bufferedOutputStream.close();
80
81     long end = System.currentTimeMillis();
82
83     System.out.println(getName()+"运行时间:"+(end - begin)+"毫秒");
84
85 }
86
87
88 @Override
89 public void run()
90 {
91     try
92     {
93         download();
94     } catch (Exception e)
95     {
96         e.printStackTrace();
97     }
98 }
99 }

```

3-5模拟多文件上传与下载

```
1 1.先编写UrlHelper
2 2.FileService
3 3.main(只能完成单线程，main线程操作)
4 单线程测试没有问题后
5 4.FileUploadThread 文件上传的多线程
6 5.FileDownloadThread文件下载的多线程
```

```
1 public class UrlHelper
2 {
3     //服务器端（就是咱们创建的项目）文件保存路径
4     public static final String BASEURL = "src\\download";
5 }
```

```
1 package less4;
2
3 import java.io.*;
4
5 import static less4.UrlHelper.*;
6 public class FileService
7 {
8     //用户输入的文件地址（包含了文件名）
9     private String url; //用户填写的地址
10    private String cunfang_url; //用户填写硬盘路径
11
12    //上传：指的是从c盘或D盘将文件拷贝到项目中（BASEURL）下
13    public void upload() throws IOException
14    {
15        baseMethod(url,BASEURL);
16    }
17
18    public void download() throws IOException
19    {
20        baseMethod(url,cunfang_url);
21    }
22
23    /*
24    oldUrl:读取的文件
25    newUrl:写入的文件
26    */
27    public void baseMethod(String oldUrl,String newUrl) throws IOException
28    {
29        //oldUrl：我们需要在内存解读的文件（FileInputStream）
30        File old_file = new File(oldUrl);
31
32        //FileOutPutStream,只提供文件夹
33        //我们提供的只有文件夹的名字，没有文件名，但是我们可以通过old_file.getName()获取到用户上传的文件名
34        File new_file = new File(newUrl+File.separator+old_file.getName());
35
36        //download文件夹 不包含文件名
```

```

37     File doloadFile = new File(BASEURL);
38
39     System.out.println("用户输入的存放路径：" + newUrl);
40     System.out.println("存放的完成路径：" + new_file.getAbsolutePath());
41
42     //判断用户输入的地址是否存在
43     if(!old_file.exists())
44     {
45         System.out.println("用户输入文件路径有误，请重新输入");
46         return;
47     }
48     //检查项目的文件夹是否存在
49     if(!doloadFile.exists())
50     {
51         System.out.println("服务器端路径有误，请检查服务器");
52         return;
53     }
54     //如果用户已经上传一次这个文件了，我们就将它删除
55     if(new_file.exists())
56     {
57         new_file.delete();
58     }
59
60     //只是创建了一个空的文件
61     new_file.createNewFile();
62
63     //拷贝文件:将用户输入的文件中的内容拷贝到我们服务器上的文件中
64     FileInputStream fis = new FileInputStream(old_file);
65     BufferedInputStream bis = new BufferedInputStream(fis);
66
67     FileOutputStream fos = new FileOutputStream(new_file);
68     BufferedOutputStream bos = new BufferedOutputStream(fos);
69
70     int num;
71     byte[] bs = new byte[1024*1024*5]; //一次向新文件中存入5MB的内容
72     while( (num = bis.read(bs)) != -1)
73     {
74         bos.write(bs,0,num);
75     }
76
77
78     bis.close();
79     bos.flush();
80     bos.close();
81
82
83     System.out.println(Thread.currentThread().getName()+"操作【"+new_file.getName()+"】文件完成");
84
85 }
86
87 // ~ get And Set
88

```



```

89     public String getUrl()
90     {
91         return url;
92     }
93
94     public void setUrl(String url)
95     {
96         this.url = url;
97     }
98
99     public FileService(String url)
100    {
101        this.url = url;
102    }
103
104    public String getCunfang_url()
105    {
106        return cunfang_url;
107    }
108
109    public void setCunfang_url(String cunfang_url)
110    {
111        this.cunfang_url = cunfang_url;
112    }
113 }

```

```

1  package less4;
2
3  import java.util.Scanner;
4
5  public class Test
6  {
7      public static void main(String[] args) throws Exception
8      {
9          Scanner sc = new Scanner(System.in);
10         while(true)
11         {
12             System.out.println("1.上传文件");
13             System.out.println("2.下载文件");
14             System.out.println("请选择：");
15             int choose = sc.nextInt();
16             switch (choose)
17             {
18                 case 1:
19                     System.out.println("请输入您想要上传的文件路径：");
20                     String url_upload = sc.next();
21                     FileService uploadSvr = new FileService(url_upload);
22
23                     //          uploadSvr.upload(); 单线程
24
25                     FileUploadThread thread = new FileUploadThread(uploadSvr);
26
27                     thread.start();

```

```

27         break;
28     case 2:
29         System.out.println("请输入你想要下载的文件名称：");
30         String url_download = sc.next(); //src\\download\\a.mp4
31
32         System.out.println("请输入您想要存放的路径："); // c:\\download
33         String url_cunfang = sc.next();
34
35         FileService downloadSvr = new FileService(url_download);
36         downloadSvr.setCunfang_url(url_cunfang);
37
38         //main下载
39         downloadSvr.download();
40         FileDownloadThread downloadThread = new FileDownloadThread(downloadSvr);
41         downloadThread.start();
42         break;
43
44     }
45 }
46
47 }
48 }

```

```

1 package less4;
2
3 import java.io.IOException;
4
5 public class FileUploadThread extends Thread
6 {
7     FileService svr;
8     public FileUploadThread(FileService svr)
9     {
10         this.svr = svr;
11     }
12     @Override
13     public void run()
14     {
15         try
16         {
17             svr.upload();
18         } catch (IOException e)
19         {
20             e.printStackTrace();
21         }
22     }
23 }

```

```

1 package less4;
2
3 import java.io.IOException;
4

```

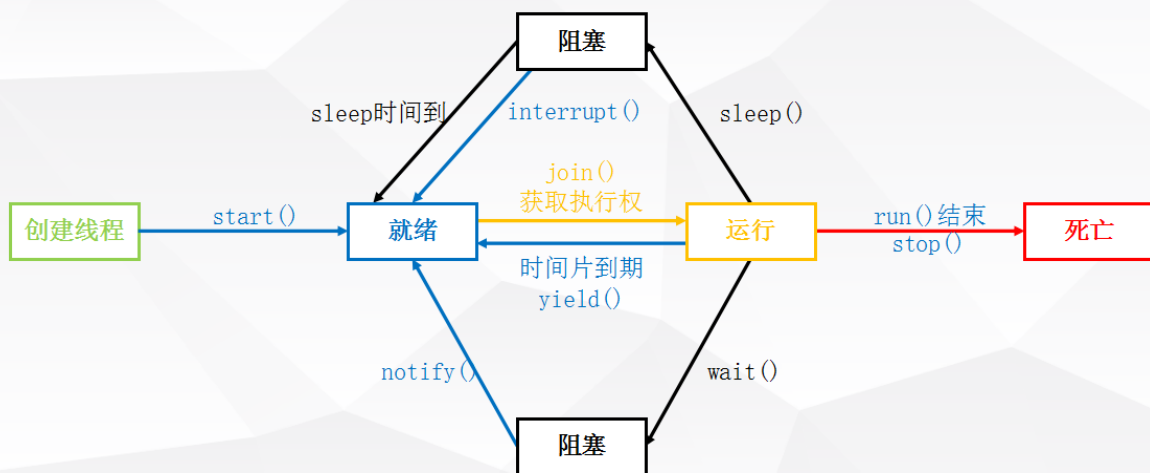
```

5 public class FileDownloadThread extends Thread
6 {
7     private FileService svr;
8     public FileDownloadThread(FileService svr)
9     {
10         this.svr = svr;
11     }
12
13     @Override
14     public void run()
15     {
16         try
17         {
18             svr.download();
19         } catch (IOException e)
20         {
21             e.printStackTrace();
22         }
23     }
24 }

```

4、线程状态

线程的状态



线程的状态

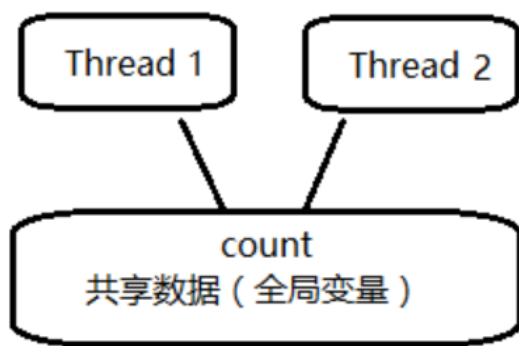
- 1、线程刚创建时，是new状态
- 2、线程调用了start()方法后，进入runnable状态，此时并未真正执行，需要和其他线程竞争cpu资源
- 3、当该线程竞争到了CPU资源，进入running状态
- 4、线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态之间处于blocked状态
 - (1) 等待阻塞：运行的线程执行wait()方法，该线程会释放占用的所有资源，JVM会把该线程放入“等待池”中，进入这个状态后，是不能自动唤醒的，必须依靠其他线程调用notify()或notifyAll()方法才能被唤醒，
 - (2) 同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入“锁池”中。
 - (3) 其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。
当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。
- 5、当线程正常执行结束会进入dead状态（一个未捕获的异常也会使线程终止）

方法	描述
getName()	获取线程名字
setName(String name)	s设置线程名字
currentThread()	获取当前线程对象
start()	开启线程
sleep(long time)	线程睡眠
join()	线程礼让
stop()	线程停止
wait()/wait(long time)	线程等待
notify()/notifyAll()	线程唤醒

5、线程同步

5-1、什么是线程同步

当多个线程同时共享同一个全局变量或静态变量，做写的操作（修改变量值）时，可能会发生数据冲突问题，也就是线程安全问题。但是做读操作时不会发生数据冲突问题。



两个线程同时修改
count变量(写操作),可能会导致数据错误

<https://blog.csdn.net/yz2015>

5-2、线程同步代码块

```

1 package less5;
2
3 public class Test01_线程同步
4 {
5     /*
6     为啥有线程同步?
7     因为可能出现多个线程在抢占资源时, 同时进行, 对一个变量同时进行了修改, 那么程序显示上就会出现不可
8     预测的问题, 我们管这个问题叫做
9     线程的并发
10    线程的同步就是为了解决并发问题, 让关键性数据在同一时间, 只能被一个线程所操作, 就可以保证数据的准
11    确性
12    */
13    public static void main(String[] args)
14    {
15        Ticket t = new Ticket(); //使用都是t对象中的number
16
17        //构建3个线程, 模拟3个窗口, 每个窗口卖4张票
18        Thread t1 = new Thread(t);
19        Thread t2 = new Thread(t);
20        Thread t3 = new Thread(t);
21        t1.setName("A窗口");
22        t2.setName("B窗口");
23        t3.setName("C窗口");
24
25        t1.start();
26        t2.start();
27        t3.start();
28    }
29 }
30 class Ticket implements Runnable
31 {
32     int number = 10; //一共有10张票
33
34     @Override
35     public void run()
36     {
37         //让每个线程只循环4次, 模拟每个线程卖4张票
  
```

```

39     for(int i = 0 ; i < 4 ; i++)
40     {
41         //同步代码块：保证代码块中的代码都执行完毕了，再让下一个线程使用
42         synchronized (this) //锁对象：要保证访问的线程都使用同一个对象
43         //this表示的是当前对象，就是谁调用了这个方法，对象就是谁
44         {
45             if(number <= 0)
46             {
47                 System.out.println("卖完了");
48                 break;
49             }
50
51             number--; //1 - 1 = 0 A(1-1),B(0-1)
52             System.out.println(Thread.currentThread().getName()+"卖了一张票，还
剩"+number+"张票");
53         }
54
55
56         //模拟现实操作，进入等待售票售票环节
57         try
58         {
59             Thread.sleep(1000);
60         } catch (InterruptedException e)
61         {
62             e.printStackTrace();
63         }
64     }
65 }
66 }

```

5-3、线程同步方法

```

1  package less5;
2
3  public class Test02_线程同步方法
4  {
5      public static void main(String[] args)
6      {
7          //保证调用对象的方法，是同一个，同步才能使用
8          Ticket2 t = new Ticket2();
9
10         Thread t1 = new Thread(t);
11         Thread t2 = new Thread(t);
12         Thread t3 = new Thread(t);
13
14         t1.start();//Thread-0
15         t2.start();//Thread-1
16         t3.start();//Thread-2
17     }
18 }
19
20 class Ticket2 implements Runnable

```

```

21 {
22     int number = 10;
23
24     public synchronized void saleTicket()
25     {
26         if (number <= 0)
27         {
28             System.out.println("票卖完了");
29             return;
30         }
31
32         number--;
33         System.out.println(Thread.currentThread().getName() + ":卖了一张票，还剩" + number +
"张");
34     }
35
36     @Override
37     public void run()
38     {
39         //不需要同步的代码写在run里面
40         for (int i = 0; i < 4; i++)
41         {
42             //要求线程执行完毕后，在开放CPU的抢占
43             saleTicket();
44         }
45     }
46 }

```

5-4、线程同步失效

```

1 package less5;
2
3 public class Test03_线程同步失效
4 {
5     public static void main(String[] args)
6     {
7         //同步失效的原因:没有保证对象的统一
8         Ticket3 ti1 = new Ticket3();//10
9         Ticket3 ti2 = new Ticket3();//10
10        Ticket3 ti3 = new Ticket3();//10
11
12        Thread t1 = new Thread(ti1);
13        Thread t2 = new Thread(ti2);
14        Thread t3 = new Thread(ti3);
15
16        t1.start();
17        t2.start();
18        t3.start();
19
20    }
21 }
22 class Ticket3 implements Runnable

```

```

23 {
24     int number = 10;
25
26     public synchronized void saleTicket()
27     {
28         if (number <= 0)
29         {
30             System.out.println("票卖完了");
31             return;
32         }
33
34         number--;
35         System.out.println(Thread.currentThread().getName() + ":卖了一张票，还剩" + number +
"张");
36     }
37
38     @Override
39     public void run()
40     {
41         //不需要同步的代码写在run里面
42         for (int i = 0; i < 4; i++)
43         {
44             //要求线程执行完毕后，在开放CPU的抢占
45             saleTicket();
46         }
47     }
48 }

```

5-5、线程同步静态代码块

```

1  package less5;
2
3  public class Test04_静态同步问题
4  {
5      public static void main(String[] args)
6      {
7
8          Ticket4 ti1 = new Ticket4();//10
9          Ticket4 ti2 = new Ticket4();//10
10         Ticket4 ti3 = new Ticket4();//10
11
12         Thread t1 = new Thread(ti1);
13         Thread t2 = new Thread(ti2);
14         Thread t3 = new Thread(ti3);
15
16         t1.start();
17         t2.start();
18         t3.start();
19
20         //静态变量是通过类型.属性
21         System.out.println(Ticket4.number);
22     }

```



```

23 }
24 class Ticket4 implements Runnable
25 {
26     //number就不再属于对象了，而属于类，所有对象共有这个属性
27     static int number = 10;
28
29
30
31
32     @Override
33     public void run()
34     {
35         for(int i = 0 ; i < 4 ;i++)
36         {
37             //这里锁的还是this，如果this的对象不一样，那么同步将会失效
38             //如果锁的是静态属性，那么不要去锁对象，转而锁类名
39             synchronized (this.getClass()) //Ticket4.class表示的类名
40             {
41                 if (number <= 0)
42                 {
43                     System.out.println("票卖完了");
44                     return;
45                 }
46
47                 number--;
48                 System.out.println(Thread.currentThread().getName() + ":卖了一张票，还剩" +
number + "张");
49             }
50         }
51     }
52 }

```

5-6、线程同步静态方法

```

1 package less5;
2
3 public class Test05_静态同步方法
4 {
5     public static void main(String[] args)
6     {
7
8         Ticket5 ti1 = new Ticket5();//10
9         Ticket5 ti2 = new Ticket5();//10
10        Ticket5 ti3 = new Ticket5();//10
11
12        Thread t1 = new Thread(ti1);
13        Thread t2 = new Thread(ti2);
14        Thread t3 = new Thread(ti3);
15
16        t1.start();
17        t2.start();
18        t3.start();

```

```

19
20
21     }
22 }
23 class Ticket5 implements Runnable
24 {
25     static int number = 10;
26
27     //如果要去同步静态变量，那么就把该方法变成静态同步方法
28     public static synchronized void saleTicket()
29     {
30         if (number <= 0)
31         {
32             System.out.println("票卖完了");
33             return;
34         }
35
36         number--;
37         System.out.println(Thread.currentThread().getName() + ":卖了一张票，还剩" + number +
"张");
38     }
39
40     @Override
41     public void run()
42     {
43         //不需要同步的代码写在run里面
44         for (int i = 0; i < 4; i++)
45         {
46             //要求线程执行完毕后，在开放CPU的抢占
47             saleTicket();
48         }
49     }
50 }

```

5-7、同步锁

```

1  package less5;
2
3  import java.util.concurrent.locks.Lock;
4  import java.util.concurrent.locks.ReentrantLock;
5
6  public class Test06_同步锁
7  {
8      public static void main(String[] args)
9      {
10         Ticket6 t = new Ticket6();
11
12         for(int i = 0 ; i < 3 ; i++)
13         {
14             Thread thread = new Thread(t);
15
16             thread.start();

```

```

17     }
18
19     }
20 }
21
22 class Ticket6 implements Runnable
23 {
24     int number = 10;
25
26     Lock lock = new ReentrantLock();
27     @Override
28     public void run()
29     {
30         //让每个线程只循环4次，模拟每个线程卖4张票
31         for (int i = 0; i < 4; i++)
32         {
33             //上锁，保证后续代码执行完毕，其他线程才可以使用
34             lock.lock();
35             if (number <= 0)
36             {
37                 System.out.println("卖完了");
38                 break;
39             }
40
41             number--; //1 - 1 = 0 A(1-1),B(0-1)
42             System.out.println(Thread.currentThread().getName() + ":卖了一张票，还剩" + number
+ "张票");
43             lock.unlock();//解锁，其他线程可以抢占资源了
44         }
45
46         //模拟现实操作，进入等待售票售票环节
47         try
48         {
49             Thread.sleep(1000);
50         } catch (InterruptedException e)
51         {
52             e.printStackTrace();
53         }
54     }
55 }
56 }
57 }

```

5-8、死锁问题

1. 使用了同步机制，资源同一时间只能被单个线程占有
2. 占有的资源不可以强行剥夺(外部无法操作资源)
3. 多线程互相等待彼此占有的资源锁释放

```

1 package less5;
2
3 import java.util.Date;

```

```
4
5 public class Test07_线程死锁
6 {
7     public static void main(String[] args)
8     {
9         LockA a = new LockA();
10        LockB b = new LockB();
11
12        Thread t1 = new Thread(a);
13        Thread t2 = new Thread(b);
14
15        t1.start();
16        t2.start();
17    }
18 }
19 class MyObject
20 {
21     public static Object o1 = new Object();
22     public static Object o2 = new Object();
23 }
24 class LockA implements Runnable
25 {
26     @Override
27     public void run()
28     {
29         //2021/8/21 14:31:00
30         System.out.println(new Date().toString() + ">>>LockA开始执行");
31         //加长程序运行时间
32         while(true)
33         {
34             synchronized (MyObject.o1)
35             {
36                 System.out.println(new Date().toString()+">>>LockA锁住了o1对象");
37
38                 try
39                 {
40                     Thread.sleep(3000);
41                 } catch (InterruptedException e)
42                 {
43                     e.printStackTrace();
44                 }
45
46                 synchronized (MyObject.o2)
47                 {
48                     System.out.println(new Date().toString()+">>>LockA锁住了o2对象");
49
50                     try
51                     {
52                         Thread.sleep(3000);
53                     } catch (InterruptedException e)
54                     {
55                         e.printStackTrace();
56                     }
57                 }
58             }
59         }
60     }
61 }
```

```

57         }
58     }
59 }
60
61
62
63
64
65     }
66 }
67 class LockB implements Runnable
68 {
69     @Override
70     public void run()
71     {
72         System.out.println(new Date().toString() + ">>LockB开始执行");
73
74         while(true)
75         {
76             synchronized (MyObject.o2)
77             {
78                 System.out.println(new Date().toString() + ">>LockB锁住了o2对象");
79
80                 try
81                 {
82                     Thread.sleep(3000);
83                 } catch (InterruptedException e)
84                 {
85                     e.printStackTrace();
86                 }
87
88                 synchronized (MyObject.o1)
89                 {
90                     System.out.println(new Date().toString() + ">>LockB锁住了o1对象");
91
92                     try
93                     {
94                         Thread.sleep(3000);
95                     } catch (InterruptedException e)
96                     {
97                         e.printStackTrace();
98                     }
99                 }
100             }
101         }
102     }
103 }

```

6、线程通信

6-1、同步通信

```

1  package less5;
2
3  public class Test08_同步通信
4  {
5      public static void main(String[] args)
6      {
7          //如果使用匿名内部类，那么对象一定是不同的，在同步代码的时候需要按照静态同步的处理方式
8          Thread t1 = new Thread(new Runnable()
9          {
10             @Override
11             public void run()
12             {
13                 MyTest.say();
14             }
15         });
16
17         Thread t2 = new Thread(new Runnable()
18         {
19             @Override
20             public void run()
21             {
22                 MyTest.hello();
23             }
24         });
25
26         t1.start();
27         t2.start();
28     }
29 }
30
31 class MyTest
32 {
33     public static synchronized void say()
34     {
35         for(int i = 0 ; i < 3;i++)
36         {
37             System.out.println("say");
38         }
39     }
40
41     public static synchronized void hello()
42     {
43         for(int i = 0 ; i < 4;i++)
44         {
45             System.out.println("hello");
46         }
47     }
48 }

```

6-2、轮询通信

```
1 package less5;
2
3 public class Test09_轮询通信
4 {
5     public static void main(String[] args)
6     {
7         Thread t1 = new Thread(new Runnable()
8         {
9             @Override
10             public void run()
11             {
12                 for(int i = 60 ; i >= 0 ; i--)
13                 {
14                     MyTest2.number++;
15
16                     System.out.println("还剩"+i+"秒结束");
17
18                     try
19                     {
20                         Thread.sleep(1000);
21                     } catch (InterruptedException e)
22                     {
23                         e.printStackTrace();
24                     }
25                 }
26             }
27         });
28
29         Thread t2 = new Thread(new Runnable()
30         {
31             @Override
32             public void run()
33             {
34                 while(true)
35                 {
36                     System.out.println(Thread.currentThread().getName()+">>在卖货物");
37
38                     if(MyTest2.number == 60)
39                     {
40                         System.out.println(Thread.currentThread().getName()+">>结束");
41
42                         break;
43                     }
44
45                     try
46                     {
47                         Thread.sleep(800);
48                     } catch (InterruptedException e)
49                     {
50                         e.printStackTrace();
51                     }
52                 }
53             }
54         });
55     }
56 }
```

```

54     });
55
56     //让多个线程去操作同一个变量，一个线程负责修改这个变量，另一个线程判断当该变量满足了一定条
    件，就把自己这个线程结束掉
57     t1.start();
58     t2.start();
59 }
60 }
61 class MyTest2
62 {
63     public static int number = 0;
64 }

```

6-3、等待与唤醒通信

```

1  package less5;
2
3  public class Test10_等待与唤醒通信
4  {
5      public static void main(String[] args) throws InterruptedException
6      {
7          Thread t1 = new Thread(new Runnable()
8          {
9              @Override
10             public void run()
11             {
12                 synchronized (MyTest3.o)
13                 {
14                     System.out.println("T1线程执行了");
15
16                     try
17                     {
18                         System.out.println("T1线程进入等待，状态【阻塞】");
19                         MyTest3.o.wait();
20                     } catch (InterruptedException e)
21                     {
22                         e.printStackTrace();
23                     }
24
25                     System.out.println("T1线程执行完毕");
26                 }
27             }
28         });
29
30     }
31 }
32
33 Thread t2 = new Thread(new Runnable()
34 {
35     @Override
36     public void run()
37     {

```



```

38         synchronized (MyTest3.o)
39         {
40             for(int i = 0 ; i < 10;i++)
41             {
42                 MyTest3.number++;
43
44                 System.out.println("T2线程执行了,number为"+MyTest3.number);
45
46                 if(MyTest3.number == 5)
47                 {
48                     System.out.println("通知T1线程准备执行,状态【就绪】");
49                     MyTest3.o.notify(); //唤醒等待的线程 t1
50                     //但是有同步代码块的原因,会保证T2线程先执行完毕,再执行T1线程
51                 }
52
53                 try
54                 {
55                     Thread.sleep(100);
56                 } catch (InterruptedException e)
57                 {
58                     e.printStackTrace();
59                 }
60             }
61         }
62
63     }
64 });
65
66 //waite与notify
67 //1、保证多个线程必须有同步代码块或者同步方法
68 //2、保证多个线程锁的是同一个对象
69 //3、一个被同步代码锁住的线程,可以使用waite方法主动让出cpu使用权
70 //4.在其他同步线程中,可以使用notify方法唤醒等待线程,等待的线程会在该同步线程执行完毕后,再
71 执行
72
73 //5、总体而言符合同步要求
74
75     t1.start();
76     Thread.sleep(1000);
77     t2.start();
78 }
79 }
80 class MyTest3
81 {
82     public static int number = 0;
83     public static Object o = new Object();
84 }
85 }

```

6-4、高效率通信

```

1 package less5;
2
3 public class Test11_高效率通信
4 {
5     public static void main(String[] args) throws InterruptedException
6     {
7         MyTest4 thread=new MyTest4();
8
9         thread.start();
10
11         Thread.sleep(3000);
12         //修改一个正在运行线程中的变量，正在运行的线程可以发生变化么？
13         thread.flag = false;//属性 ：私有空间中，加上volatile，会去修改公共区域了
14     }
15 }
16 class MyTest4 extends Thread
17 {
18     //可以修改正在运行线程中的变量，运行中的线程会立刻就可以获取修改内容
19     volatile boolean flag = true;
20     @Override
21     public void run()
22     {
23         //线程中的flag会去存放公共区域中
24         while(flag)
25         {
26             System.out.println("执行了");
27
28             try
29             {
30                 Thread.sleep(1000);
31             } catch (InterruptedException e)
32             {
33                 e.printStackTrace();
34             }
35         }
36     }
37 }

```

7、线程池

➤线程池：是一个容纳多个线程的容器，其中的线程可以反复使用，省去了频繁创建线程对象的操作，无需反复创建线程而消耗过多资源，是对线程使用的一种优化。

➤使用线程池中线程对象的步骤：

1. 创建线程池对象。
2. 创建Runnable接口子类对象。
3. 提交Runnable接口子类对象。
4. 关闭线程池（一般不关闭）。

```
1 package less5;
2
3 import java.util.concurrent.ExecutorService;
4 import java.util.concurrent.Executors;
5
6 public class Test12_线程池
7 {
8     public static void main(String[] args)
9     {
10         MyTest5 m = new MyTest5();
11         //数字表示要创建几个线程
12         ExecutorService es = Executors.newFixedThreadPool(3);
13
14         for(int i = 0 ; i < 10;i++)
15         {
16             //只能接受Runnable接口，线程池会自动创建，自动启动，自动销毁
17             //我们只需告诉线程池执行哪个线程，执行几次
18             es.submit(m);
19         }
20
21         //关闭线程池，原则不用关闭，系统会以最低的代价维护
22         es.shutdown();
23     }
24 }
25
26 class MyTest5 implements Runnable
27 {
28     @Override
29     public void run()
30     {
31         System.out.println(Thread.currentThread().getName());
32     }
33 }
```

