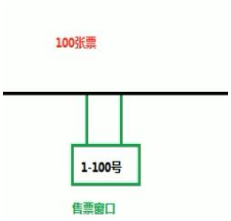
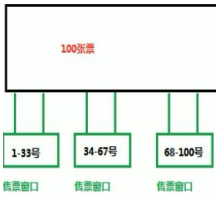
线程安全问题概述

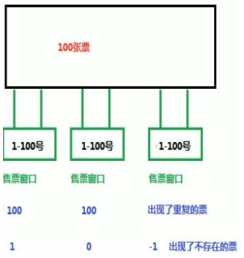
单线程程序是不会出现线程安全问题的



多线程程序，没有访问共享数组，不会产生问题



多线程访问共享的数据，会产生线程安全问题



同步代码块

卖票案列出现了线程安全问题

卖出了不存在的票和重复的票

解决线程安全问题的一种方案；使用同步代码块格式

synchronized（锁对象）{

可能会出现线程安全问题的代码（访问了共享数据的代码）

}

注意

1.同步代码块中的锁对象，可以是任意的对象

2.但是必须保证多个线程使用的锁对象是同一个

3.锁对象作用

把同步代码块锁住，只让一个线程在同步代码块中执行

```
                实现卖票案例*/
public class ShoupiaoDemo1 implements Runnable {
    /**定义一个多个线程共享的票源*/
    private int ticket = 100;
    /**创建一个锁对象**/
    Object obj = new Object();

    /**设置线程任务；卖票*/
    @Override
    public void run() {
        /**先判断票是否存在
         if (ticket>0){
         //票存在 卖票ticekt
         System.out.println(Thread.currentThread().getName()+"正在卖"+ticket+"张票");
         ticket--;    }*/
        /**使用死循环，让卖票操作次重复执行*/
        /*while(true){
         *//**先判断票是否存在*//*
            if (ticket>0){
                *//**提高安全问题出现的概率，让程序睡眠*//*
                try {
                    Thread.sleep(10);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                *//**票存在，卖票 ticket*//*
```

```
System.out.println(Thread.currentThread().getName()+"正在
卖"+ticket+"张票");
                ticket--;*/
        while (true) {
            /**同步代码块**/
            synchronized (obj) {
                if (ticket > 0) {
                    //**提高安全问题出现的概率，让程序睡眠*//*
                    try {  Thread.sleep(10); } catch
(InterruptedException e) {
                        e.printStackTrace();    }
                    //**票存在，卖票 ticket*//*
    System.out.println(Thread.currentThreaa().getName() + "正在卖"
+ ticket + "张票");
                    ticket--;
public static void main(String[] args) {
   //创建Runnable接口实现类对象
    ShoupiaoDemo1 run = new ShoupiaoDemo1();
   //创建Thread类对象，构造方法中传递Runnable();
   Thread t01 = new Thread(run);
   Thread t02 = new Thread(run);
   Thread t03 = new Thread(run);
   //调用start方法开启多线程
     t01.start();
   t02.start();
   t03.start();
```

静态同步方法：

```
public static synchronized void method(){
    // 可能会产生线程安全问题的代码
}
```

静态同步方法的锁对象：当前类的字节码对象

获取一个类的字节码对象的3种方式：

1. 对象名.getClass()
2. 类名.class
3. Class.forName("类的全路径");

calss .forname

```
"com.itheima.test05.RunnableImpl");
```

字节码的特点，

同一个类，他的字节码对象只有唯一的一个

lock.lock（ ）

lock.unlock（ ）

使用步骤

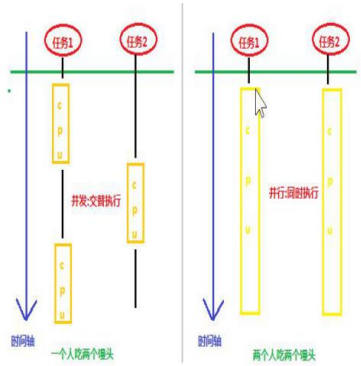
1.在成员位置创建一个Reentrantlock对象

2.在有可能出现安全问题的代码前

并发与并行

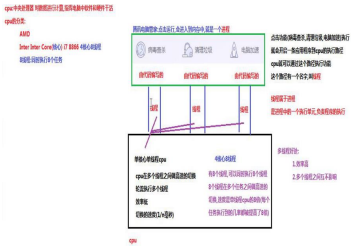
并发：指两个或多个事件在**同一个时间段内**发生。

并行：指两个或多个事件在**同一时刻**发生（同时发生）。



进程：是指一个内存中运行的应用程序。

线程：线程是进程中的一个执行单元（代码的执行路径），负责程序的执行



程序>进程>线程

线程调度

分时调度；所有线程轮流使用cpu，平分占用cpu的时间

抢占式调度；优先让优先级高的线程使用CPU，如果优先级相同，则随机选择一个线程执行。

主线程；执行主方法的线程

单线程程序；JAVA程序中只有一个线程

执行从main方法开始，从上到下依次执行

实现多线程的第一种方式:

1. 定义类, 继承 Thread 类
2. 重写 run() 方法, run方法内部是线程要执行的任务
3. 创建Thread子类的对象, 调用 start() 方法启动线程

java.lang.Thread类: 表示线程. 实现了Runnable接口

void start(): 启动线程, 即让线程开始执行run()方法中的代码

CPU高速随机切换（本质）

线程抢夺CPU资源

设置线程的两种方式

1.使用Thread类中的方法setName（名字）

void setName（String name）改变线程名称，使之参数name 相同

2.创建一个带参数的构造方法，参数传递线程的名称；调用父类的带参构造方法，把线程名称传递给父类让父类给线程起名字

Thread（String name）分配新的Thread对象

```
public class setName extends Thread {
    public setName(){}
    public setName(String name){
        super(name);

    }
    @Override
    public void run() {
        //获取线程名称
            System.out.println(Thread.currentThreaa().getName());
public static void main(String[] args) {
    //开启多线程
      MyTeread mt = new MyTeread();
```

```
    mt.setName("输入名称"); 把线程名称传递给父类，让父类（Thread）给子线程起一个名字
    mt.start();
    //开启多线程
      setName name = new setName("名称");
    name.start();
```

sleep的使用

public static void sleep (long millis) ；使当前正在执行的线程以指定的毫秒数暂停

毫秒数结束之后，线程继续执行

```
public static void main(String[] args) {
    //模拟秒表
      for (int i = 0; i < 60; i++) {
       System.out.println(i);
       //使用Thread类的sleep方法让程序休眠1秒
            try {
           Thread.sleep(1000);
       } catch (InterruptedException e) {
           e.printStackTrace();
```

Runnable接口

java.lang.Runnable

Runnable接口应该由那些打算通过某一线程执行其实例的类来实现

类必须定义一个称为run的无参方法。

java.lang.Thread类的构造方法

Thread (Runnable target) 分配新的Thread对象

Thread (Runnable target , String name) 分配新的 Thread 对象

实现步骤

1.创建一个Runnable接口的实现类

2.在实现类中重写Runnable接口的run方法，设置线程任务

3.创建一个Runnable接口的实现类对象

4.创建Thread类对象，构造方法中传递Runnable接口的实现类对象

5.调用Thread类中的start方法，开启新的线程执行run方法

```
/** 创建一个Runnable接口的实现类**/
public class RunnableImpl implements Runnable {
  /** 2.在实现类中重写Runnable接口的run方法， 设置线程任务**/
   @Override
   public void run() {
      for (int i = 0; i < 20; i++) {
         System.out.println(Thread.currentThread().getName()+"--
"+i);
public static void main(String[] args) {
   /***   3.创建一个Runnable接口的实现类对象*/
   RunnableImpl run = new RunnableImpl();
   /** 4.创建Thread类对象， 构造方法中传递Runnable接口的实现对象*/
   Thread t = new Thread(run);
   /**5.调用Thread类中的start方法， 开启新的线程执行run方法*/
   t.start();
   for (int i = 0; i < 20; i++) {
```

```
        System.out.println(Thread.currentThreao().getName() + "--" +
i);
```

实现Runnable接口创建多线程程序的好处

1.避免了单继承的局限性

一个类只能继承一个类（一个人只能有一个亲爹），类如果继承了Thread类就不能继承其他类，实现了Runnable接口，还可以继承其他类，实现其他接口

2.增强程序的扩展性，降低了程序的耦合性（解耦）

实现了Runnable接口的方式，把设置线程任务和开启新线程进行了分离（解耦）

实现类中重写了run方法，用来设置线程任务

创建Thread类对象，调用start方法；用来开启新线程

匿名内部了方式实现线程的创建

匿名；没有名字

内部类；写在其他类内部的类

匿名内部类作用

简化代码

把子类继承父类，重写父类的方法，创建子类对象合成一步

把实现类实现接口，重写接口中的方法，创建实现类对象合成一步完成

匿名内部类的最终产物；子类/实现类对象，而这个类没有名字

格式

new父类/接口（ ）{

重复父类/接口中的方法

```
public static void main(String[] args) {
```

```
   //线程的父类是Thread
   //new  MyThread（）.start（）；
      new Thread() {
       //重写run方法，设置线程任务
            @Override
       public void run() {
          for (int i = 0; i < 20; i++) {
             System.out.println(Thread.currentThread().getName() + "--
" + "黑马");}} }.start();
   //线程的接口Runnable

  Runnable runnable=  new Runnable() {
       //重写run方法  设置线程任务
@Override public void run() { for (int i = 0; i < 20; i++) {
System.out.println(Thread.currentThread().getName() + "程序员");}}};
  new Thread(runnable).start();
  //简化接口的方式
      new Thread(new Runnable() {
       //重写run方法  设置线程任务
            @Override
       public void run() {
          for (int i = 0; i < 20; i++) {
System.out.println(Thread.currentThread().getName() + "程序员
888888888"); }} }).start();
```