精品课程配套教材

“双创”型人才培养优秀教材

21世纪应用型人才培养“十三五”规划教材

Java程序设计任务驱动教程

JAVA CHENGXU SHEJI RENWU QUDONG JIAOCHENG

主 编 蓝 敏 殷正坤 李 奇

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

```
        iBuffer = createImage (this. getSize () . width, this. getSize () . height);
        gBuffer = iBuffer. getGraphics ();
        paint (gBuffer);
        gBuffer. dispose ();
        scr. drawImage (iBuffer, 0, 0, this);
    }
}
```

【技术拓展】

10.4 线程的调度

1. 线程的优先级

优先级越高的线程获得 CPU 执行的机会越大，而优先级越低的线程获得 CPU 执行的机会越小。对于优先级相同的线程，则遵循队列的“先进先出”原则，即先进入就绪状态排队的线程被优先分配到处理器资源，随后才为后进入队列的线程服务。

线程的优先级用 1~10 之间的整数来表示，数字越大优先级越高。除了可以直接使用数字表示线程的优先级，还可以使用 Thread 类中提供的 3 个静态常量表示线程的优先级(见表 10-1)。

表 10-1 线程的优先级

Thread 类的静态常量	功能描述
static int MAX_ PRIORITY	表示线程的最高优先级，相当于值 10
static int MIN_ PRIORITY	表示线程的最低优先级，相当于值 1
static int NORM_ PRIORIY	表示线程的普通优先级，相当于值 5

程序在运行期间，处于就绪状态的每个线程都有自己的优先级，例如 main 线程具有普通优先级。线程优先级不是固定不变的，可以通过 Thread 类的 setPriority (int newPriority) 方法对其进行设置，该方法中的参数 newPriority 接收的是 1~10 之间的整数或者 Thread 类的三个静态常量。

下面通过一个案例来演示不同优先级的两个线程在程序中的运行情况。

【案例 10-3】不同优先级线程的运行

```
//定义类 MaxPriority 实现 Runnable 接口
package 第 10 章多线程;
class MaxPriority implements Runnable {
    public void run () {
        for (int i = 0; i < 10; i++) {
            System. out. println (Thread. currentThread () . getName () + " 正在输
出:" + i);
        }
    }
}
//定义类 MinPriority 实现 Runnable 接口
class MinPriority implements Runnable {
    public void run () {
        for (int i = 0; i < 10; i++) {
            System. out. println (Thread. currentThread () . getName () + " 正在输
出:" + i);
        }
    }
}
public class TestPriority {
    public static void main (String [] args) {
        //创建两个线程
        Thread minPriority = new Thread (new MinPriority (), " 优先级较低的线
程" );
        Thread maxPriority = new Thread (new MaxPriority (), " 优先级较高的线
程" );
        minPriority. setPriority (Thread. MIN_ PRIORITY); // 设置线程的优先级为 1
        maxPriority. setPriority (10); // 设置线程的优先级为 10
        //开启两个线程
```

```
        maxPriority. start ();
        minPriority. start ();
    }
}
```

2. 线程的基本控制

（1）线程睡眠。如果希望人为地控制线程，使正在执行的线程暂停，将 CPU 让给别的线程，这时可以使用静态方法 sleep（long millis），该方法可以让当前正在执行的线程暂停一段时间，进入休眠等待状态。

当前线程调用 sleep（long millis）方法后，在指定时间（参数 millis）内该线程是不会执行的，这样其他的线程就可以得到执行的机会了。

【案例 10-4】 线程睡眠

```
public classTestSleep {
    public static void main (String [ ] args) throws Exception {
        // 创建一个线程
        new Thread (new SleepThread ( ) ) . start ();
        for (int i = 1; i <= 10; i++) {
            if (i == 5) {
                Thread. sleep (2000);    // 当前线程休眠 2 s
            }
            System. out. println (" 主线程正在输出:" + i);
            Thread. sleep (500); // 当前线程休眠 500 ms
        }
    }
}
//定义 SleepThread 类实现 Runnable 接口
class SleepThread implements Runnable {
    public void run ( ) {
        for (int i = 1; i <= 10; i++) {
```

```
            if (i == 3) {
                try {
                    Thread.sleep (2000); // 当前线程休眠 2 s
                } catch (InterruptedException e) {
                    e.printStackTrace ();
                }
            }
            System.out.println (" 线程一正在输出:" + i);
            try {
                Thread.sleep (500); // 当前线程休眠 500 ms
            } catch (Exception e) {
                e.printStackTrace ();
            }
        }
    }
}
```

除了 sleep（）以外，wait（）也可以让线程挂起。调用 wait（）方法的线程将进入“非可执行”状态，使用 wait（）方法有两种方式。

语法格式为：thread.wait（1000）；

或为：thread.wait（）与 thread.notify（）结合。

第一种方式给定线程挂起时间，第二种方式是 wait（）与 notify（）方法配合使用，这种方式让 wait（）方法无限等下去，直到线程接收到 notify（）或则 notifyAll（）消息为止。

wait（）、notify（）、notifyAll（）不同于其他线程方法，这 3 个方法是 java.lang.Object 类的一部分，而 Object 类是所有类的父类，所以这 3 个方法会自动被所有类继承下来，wait（）、notify（）、notifyAll（）都被声明为 final，所以无法重新定义。

（2）线程让步。线程让步可以通过 yield（）方法来实现，该方法可以让当前正在运行的线程暂停。yield（）方法不会阻塞该线程，它只是将线程转换成就绪状态，让系统的调度器重新调度一次。

当某个线程调用 yield（）方法之后，只有与当前线程优先级相同或者更高的线程才

能获得执行的机会。通过一个案例来演示一下 yield（）方法的使用。

【案例 10-5】线程让步

```
//定义 YieldThread 类继承 Thread 类
class YieldThread extends Thread {
    // 定义一个有参的构造方法
    public YieldThread (String name) {
        super (name); // 调用父类的构造方法
    }
    public void run () {
        for (int i = 0; i < 5; i++) {
            System. out. println (Thread. currentThread () . getName () + " ---" +
i);
            if (i == 3) {
                System. out. print (" 线程让步:" );
                Thread. yield (); // 线程运行到此，作出让步
            }
        }
    }
}
public classTestYield {
    public static void main (String [ ] args) {
        //创建两个线程
        Thread t1 = new YieldThread (" 线程 A" );
        Thread t2 = new YieldThread (" 线程 B" );
        //开启两个线程
        t1. start ();
        t2. start ();
    }
}
```

（3）线程插队。在 Thread 类中也提供了一个 join（）方法可以实现线程的插队。当在某个线程中调用其他线程的 join（）方法时，调用的线程将被阻塞，直到被 join（）方法加入的线程执行完成后它才会继续运行。

通过一个案例来演示一下 join（）方法的使用。

【案例 10-6】 线程插队

```
public classTestJoin {
    public static void main (String [ ] args) throws Exception {
        // 创建线程
        Thread t = new Thread (new EmergencyThread (),"线程一");
        t. start (); // 开启线程
        for (int i = 1; i < 6; i++) {
            System. out. println (Thread. currentThread ( ) . getName ( ) +" 输入:" +
i);
            if (i == 2) {
                t. join (); // 调用 join ( ) 方法
            }
            Thread. sleep (500); // 线程休眠 500 ms
        }
    }
}
class EmergencyThread implements Runnable {
    public void run ( ) {
        for (int i = 1; i < 6; i++) {
            System. out. println (Thread. currentThread ( ) . getName ( ) +" 输入:" +
i);
            try {
                Thread. sleep (500); // 线程休眠 500 ms
            } catch (InterruptedException e) {
```

```
                e. printStackTrace ();
            }
        }
    }
}
```

(4) 线程状态检查。一般情况下无法确定一个线程的运行状态，对于这些处于未知状态的线程，可以通过 isAlive () 方法来确定一个线程是否仍处在活动状态。当然即使处于活动状态的线程也并不意味着这个线程一定正在运行，对于一个已开始运行但还没有完成任务的线程，这个方法返回值为 true。

isAlive () 方法用于测试线程是否处于活动状态。如果线程已经启动且尚未终止，则为活动状态。

语法格式为：

thread. isAlive ()

thread：这是一个线程对象，isAlive () 方法将判断该线程的活动状态。

(5) 结束线程。

结束线程有两种情况：

①自然消亡：一个线程从 run () 方法的结尾处返回，自然消亡且不能再被运行；

②强制死亡：调用 Thread 类中 stop () 方法强制停止，不过该方法已经被废弃。

虽然这两种情况都可以停止一个线程，但最好的方式是自然消亡，简单地说，如果要停止一个线程的执行，最好提供一个方式让线程可以完成 run () 的流程。

例如：线程的 run () 方法中执行一个无限循环，在这个循环中可以提供一个布尔变量或表达式来控制循环是否执行，在线程执行中，可以调用方法改变布尔变量的值，用这种方式使线程离开 run () 方法以终止线程。

【案例 10-7】 线程终止

```
package com;
public class HelloWorld extends Thread {
    private boolean flag=true; //跳出循环标记量
    public boolean isFlag () {//标记量取值
        return this. flag;
    }
```

```
    public void setFlag (boolean flag) {   //标记量赋值
        this.flag=flag;
    }
    public void run () {
        while (isFlag ()) {
            //执行相关业务操作
            if (! isFlag ()) {      //如果标记量为 false，结束循环
                return;
            }
        }
    }
}
```

10.5 游戏加入障碍物（砖墙和铁墙）

1. 加入障碍物——墙壁

墙壁有砖墙（brickWall）和铁墙（ironWall）之分，砖墙（type==1）子弹可以打穿，铁墙（type==2）子弹打不穿。它们的图片也不同（见图 10-3），都有碰撞检测矩阵，以便于检测坦克和子弹与墙的碰撞。

砖墙：  　　铁墙： 

图 10-3 两 种 墙 壁

```
package 第10章多线程.加入障碍物;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.Toolkit;
```