

---

实践篇：

# 第八讲 左手画圆、右手画方



## 8.1 引言

在前面讲述的游戏程序中，其中主要工作都集中在主类（JFrame的某个子类）的paint()方法中：  
**更新、绘制、暂停、重绘**；游戏动画是一个隐式循环：在paint()方法中，不断呼叫系统重新调用自己paint()方法。这种方式只是一个权宜之计，因为它有很多缺点：



## 8.1 引言

(1) 把逻辑上不同的任务都放在同一个地方——`paint()`方法；并且这里在逻辑上是绘制图形的地方。不应该搅和在一起的东西放在一起将会导致麻烦。

(2) `paint()`方法占据了几乎全部CPU处理时间，我们的这个程序将没有时间腾出手来处理别的事情，比如最重要的事情：**接受玩家输入！**



## 8.1 引言

我们希望程序具有这样的能力：分别并且同时干几件事情！比如在我们的游戏中，程序一边进行着“更新循环”，一边进行着“绘制”，并且还能接受“用户输入”，等。

现代的软件平台都提供给程序这种能力：

**在运行时同时完成多个不同的任务！**

术语叫做“多进程”和“多线程”！



## 8.2 进程与线程

### 8.2.1 进程

**进程**是计算机科学中最深刻最成功的概念之一！

当我们在一个现代系统上运行一个程序时，我们会得到一个假象，就好像我们的程序是系统中当前运行的唯一程序。我们的程序好像是独占计算机的处理器和存储器。处理器好像是无间断地一条接一条地执行我们的程序中的指令。最后，我们程序中的代码和数据显得好像是系统存储器中唯一的对象。这些假象都是通过进程的概念提供给我们的。



## 8.2 进程与线程

进程的经典定义就是：一个执行中的程序的实例。

系统中的每个进程都是运行在某个进程的上下文（**context**）中的。上下文是由程序正确运行所需要的状态组成的。这个状态包括存放在存储器中的程序的代码和数据、它的栈、它的通用目的的寄存器的内容、它的程序计数器、环境变量以及打开文件描述符的集合。



## 8.2 进程与线程

每当用户双击Windows资源浏览器中的一个可执行文件时，操作系统就会创建一个新的进程，然后在这个新进程的上下文中运行这个可执行文件。这个运行中的进程还能创建其他的进程。



## 8.2 进程与线程

提供多进程服务的原因是为了提供计算机资源的利用率，因为：几乎所有的程序在其运行过程中都会出现很多“停滞时间”。例如，交互式程序在等待用户输入（键鼠等）时会处于停滞状态（CPU空载）；如果某个程序访问数据库，则在处理数据之前，必须等待I/O操作，而此时便产生了CPU资源等待；等等。所以当某个程序进入停滞状态时，操作系统就会换出该程序，让其他程序获得CPU资源进入运行状态。这是的计算机系统的利用率显著提高。





## 8.2 进程与线程

但是，进程的创建和进程的切换需要很大的系统开销（如CPU周期、内存等），用多个进程进行并行处理的成本太高。



## 8.2.2 线程

实现并行工作的效果，但系统开销更小的办法是创建一种叫“线程”的次级进程。线程是一种子进程，由主进程而不是操作系统进行管理。线程包含自己的局部变量副本、自己的程序计数器以及生存周期。创建和销毁线程的系统开销远低于进程。



## 8.2.2 线程

程序需要多线程的其他原因举例如下：

- ♠ 一些任务必须由线程来执行。例如前面我们写的那种类型的程序是一个单线程程序，这个唯一的线程被用来忙于动画的播放，程序（进程）没有办法来响应处理用户的其它操作。
- ♠ 一些任务可以在后台执行，这使得用户能执行其它任务。例如在早期的字处理程序中，打印文档时，不能进行其它操作。现代的字处理程序则通常在后台执行打印操作。
- ♠ 一些任务本身就应是多线程的。例如视频软件需要一个线程来播放视频，另一个线程来播放声音。
- ♠ 一些仿真和游戏程序为了模拟现实世界而允许发生多个事件。例如在实时战略游戏。



## 8.2.2 线程

### 8.2.2.1 Java程序中的默认线程

每个Java程序在默认情况下至少存在三个线程：

- ♠ **主线程**：main()方法的执行线程；
- ♠ **UI线程**：当main()方法执行到最后，主线程结束，使java程序持续存在的就是UI线程，它负责程序用户界面的各种工作，比如：绘制图形（paint()方法就是由UI线程调用的）、接受用户输入
- ♠ **垃圾收集线程**：用来在后台进行无用内存收集和释放工作。



## 8.2.2 线程

### 8.2.2.2 使用多线程

在Java程序中使用线程，需要以下若干步骤：

**(1) 声明某个类具有运行另外一个线程的能力：**

如果想让一个类，比如在此是主类，能运行其它的线程，以便完成某项工作，应首先声明这个类实现了多线程的接口：

```
public MultiThread extends JFrame implements  
Runnable{.....}
```



## 8.2.2 线程

注释：接口interface是一种java语言的类型，它声明了一组公开的抽象方法（不需要任何方法体），这个类型自己没有任何实例。任何想实现这些功能的类，都应该声明实现这个接口，并把那些抽象方法一一实现。

接口的定义与类的定义很相似，其一般的语法格式如下：

```
interface interfaceName {  
    return-type method1(para-list); //没有方法体的抽象方法。  
    .....;  
    return-type methodN(para-list); //没有方法体的抽象方法。  
}
```

一旦定义了一个接口，一个或多个类就可以实现该接口。实现一个接口的类的语法格式如下：

```
class classname extends superclass  
implements interfaceName{  
    return-type method1(para-list){//真正的方法实现}  
    .....  
    return-type methodN(para-list){//真正的方法实现}  
}
```



## 8.2.2 线程

Runnable接口是Java系统中的一个重要接口，其定义如下：

```
interface Runnable{  
    public void run(); //运行线程中要处理的工作  
}
```



## 8.2.2 线程

### (2) 声明新线程对象

线程在Java中也是对象，是Thread类型。为了让主类能运行另外一个线程，应该在主类的属性区中声明一个Thread类型的对象。

```
Thread newThread;
```





## 8.2.2 线程

### (3) 创建新线程对象

在主类的构造方法中，创建线程对象：

```
newThread = new Thread(this);
```



## 8.2.2 线程

### (4) 激发线程开始运行

在主类的构造方法中，激发线程开始运行。

```
newThread.start();
```



## 8.2.2 线程

### (5) 实现线程接口的run()方法

要实现“implements Runnable”这个承诺，必须在主类MultiThread类中，按照我们的需要**添加并实现Runnable**接口的公开方法run()：

把新线程要完成的工作放在run()方法中。在游戏应用中，往往把“游戏状态更新”、“游戏暂停”和“重画”等工作转移到run()方法中。此外，为了能让游戏不停地运行下去，应把“游戏状态更新”、“游戏暂停”和“重画”等工作放在一个while循环中。



## 8.2.2 线程

//在主类MultiThread中添加下述代码。

//在主类构造方法中新添加的代码

```
public MultiThread (){  
    .....; //其它的初始化代码  
    newThread = new Thread(this); //创建线程对象  
    newThread.start();           //启动新线程，激发run()方法运行。  
}
```

//实现Runnable接口的run()方法

```
public void run(){  
    while(animationThread != null){           //更新循环  
        for(int i =0; i < actors.size(); i++){  
            ((Spirit)actors.get(i)).update(this);  
        }  
    }
```



## 8.2.2 线程

```
try{
    Thread.sleep(40);           //暂停
}
catch(InterruptedException E){ }
repaint();                     //重画
}
}

//主类的paint()方法就变得单纯了。
public void paint(Graphics g){
    Container pane = getContentPane(); //取得内容窗格
    Graphics pg = pane.getGraphics();  //取得内容窗格的图像对象
    pg.setColor(Color.white);
    pg.fillRect(0,0,pane.getWidth(),pane.getHeight()); //用白色刷屏
    for(int i =0; i < actors.size(); i++){
        ((Spirit)actors.get(i)).draw(pg);
    }
}
```