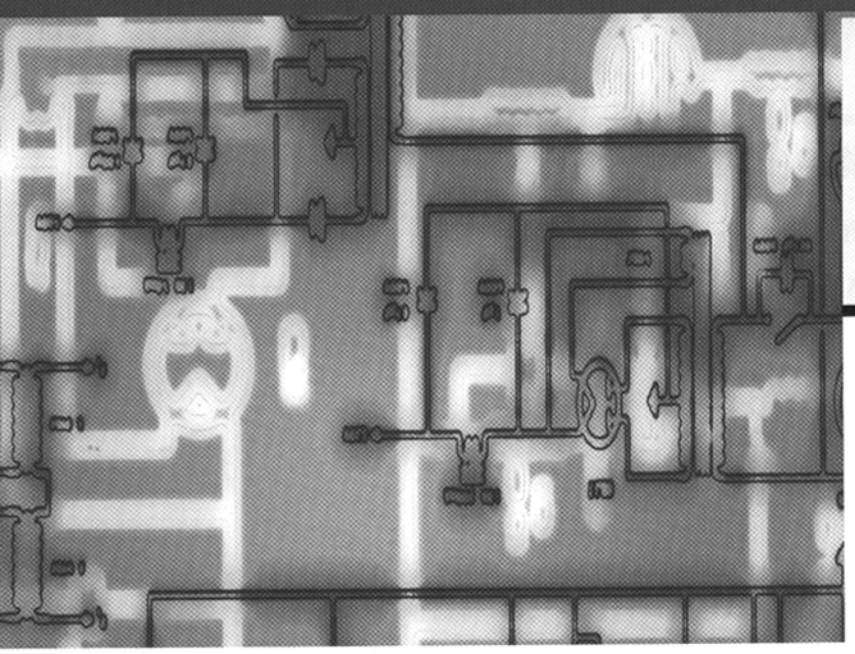

基于Java的网络多媒体制作的关键技术

吕梦雅
(河北秦皇岛 燕山大学信息科学与工程学院 066004)

摘要：本文阐述了Java在网络多媒体声像设计中的应用，详细论述了使用类和多线程技术、双缓冲区技术及异常处理技术实现播放声音、编辑图像、动画等。

关键词：网络　多媒体　多线程技术　双缓冲区技术

1　Java语言实现多媒体的几个关键技术

1.1 Java语言中类的使用

Java语言是一种跨平台、面向对象的语言，类是Java中的一个重要的概念，每个对象均有一个相应的类，通过类来确定对象如何创建，对象包含哪些方法以及对哪些消息作出怎样的响应等。

在Jave语言中有一个重要的类库－－浏览器库(Java. applet)，它主要用于针对WWW浏览器，特别是HotJava的程序设计，Java applet可提供几个方法，如Audio (声音)、Applets(小应用程序)、Links (链接) 等，通过这些方法，可以不依赖网关接口 (CGI) 就可以给浏览器带来动态和交互特性，在Java语言中，Java小应用程序(applet) 是需通过支持Java语言的WWW浏览器调用来执行的可执行模块。

1.2 使用缓冲技术

解决动画效果的关键技术是使用缓冲技术，因为动画效果过程是通过不断刷新实现的，这中间屏幕会出现闪烁，解决这个问题可使用双缓冲技术，所谓双缓冲技术就是定义两块屏幕，一块是显示屏幕 (看得见的屏幕)，另一块是虚拟屏幕 (看不见的屏幕)，显示动画时，不直接向显示屏幕输出图片而是先把它画到虚拟屏幕上，然后再把虚拟屏幕映射到显示屏幕，完成图片的显示。

1.3 使用异常处理技术

异常，即Exception，是指当Java程序违反Java语言的语义限制时，Java Visual Machine就向程序发出错误讯号。在程序执行期间，可能会有许多意外的事件发生，例如，程序中对象还未创建就被使用，程序申请内存时没申请到等。而要一个程序可靠地运行必须进行异常处理，在Java中使用try - catch -finally进行异常处理，我们把可能发生异常情况的代码段放在try语句的程序段中，利用try语句对部分代码进行监视，如果发生异常就进入到catch语句中处理。

1.4 使用多线程

所谓"线程"是"进程"中某个单一顺序的控制流，是操作系统分配时间的基本实体，每个程序至少有一个线程。多线程就是通过系统的调度使几个具有不同功能的程序流即线程同时并行地运行，提高CPU的利用率。每个线程都是和生命周期相联系的，一个生命周期含有多个状态，这些状态间可以互相转化。Java的线程的生命周期可以分为4个状态：创建 (new) 状态、可运行 (runnable) 状态、不可运行 (not runnable) 状态、消亡 (dead) 状态。为达到多媒体设计的动态效果，我们在程序中使用了多线程，因为动画效果是通过重复地刷新显示区域，同时显示内容每次都产生变化而形成的，要重新刷新显示区域，并不是用无限循环语句一刻不断地进行重复刷新，如果这样，程序就会象死循环一样，其他工作就无法做了；最好的办法就是定期启动程序刷新显示区域，而线程正好很适合这一工作，它可以用睡眠sleep (time) 方法每隔一段时间启动程序刷新显示区域。

2　网络多媒体制作的设计与基本技术

2.1 声音播放技术的实现

Java只能支持一种格式的声音文件，即.au格式的声音文件，au是unix操作系统中一种标准的声音文件格式，我们为了实现任意类型的声音文件播放，需要创建Java .applet .Audio clip接口；而且由于在Java .applet .Applet类上无法继承FileDialog类，所以需先定义一新类继承Frame类，以便可以在新类的窗体上实现对声音文件编辑，同时为使界面更友好，操作更方便，使用了文件流与数组，在数组中存放声音文件，并使用文件流的操作来存储用户

Java-based Key Technology for Internet Multimedia Production

对声音的编辑，其中利用了1秒钟的CPU延迟。下面是几种装载声音文件的方法：

Public AudioClip getAudioClip (URL url, String name)//根据给定的参数获取并返回一声音片段。

Public AudioData getAudioData (URL url, String name)// 根据给定的参数获取并返回一个AudioData声音片段。

获取了声音对象后，就可以播放声音文件了，播放声音文件的方法为：在Applet类中定义声音片段的实例变量：

```
AudioClip Sound = null;
```

然后在int( )方法中获取声音片段，run( )方法中调用play( )方法。部分代码如下：

```
... ...
protected void init( ){
... ...
sound = getAudioClip(getCodeBase( ), "soundname.au");
... ...}
... ...
public void run( ){ ... ... play(sound); ... ...}
```

2.2 图形技术编辑技术的实现

(1) 绘图：需要先在画板 (Canvas) 上定义两个类变量ImageScreenBuf与GraphicsScreenBufGraphics，以便每次在画图操作时使用Graphics类而不必再调用一次GetGraphics( )，ScreenBuf存放绘图操作，然后在画板上加入绘图所需的各种方法，如MouseDown( )等。Graphics类的dodraw( )方法中提供了各种绘图方法，用switch进行判断即可实现各种绘图方法。另外，对dodraw( )方法中的整型变量进行操作即可实现对线宽控制。

(2) 图像显示：创建一个新类继承canvas类，使用新类下的getToolkit( )方法中的getImage( )方法获得图像资源，再使用Graphics类提供的drawImage ( )方法在指定的位置上显示图像。在显示图像过程中，为保证图像在装完之后再执行显示，需使用ATW软件包中的MediaTracker类中WaitForAll( )方法等待图像完全载入后再调用paint( )方法显示图像。

(3) 图像加载中的切换问题 Java读取图像时有一个严重的问题，就是程序在图像完全加载之前就开始显示，这就使图片在第一次播放时产生断续的画面，要解决这个问题，需使用Java的AWT提供的MediaTracker类检测图片的载入，图像完全加载后再返回该图像对象，就可解决图像完全加载之前就开始显示的问题。

```
Public Image LoadImage(String imageFile)
{ Image image;
MediaTracker mediaTracker=new MediaTracker(this);
Image = getImage(getCodeBase( ),imageFile);
Media Tracker.add(image,0);
... ...}
```

双缓冲技术可以消除画面的闪烁，它在后台缓冲区内创建一个图形对象，将需要绘制的图像绘于其上，然后传给paint( )方法，由paint( )方法绘于前台，同时，后台图形缓冲区进入下一帧图像的准备。

2.3 动画播放技术的实现

动画播放技术中要循环调用多幅照片，并且要解决画面的闪烁问题。播放动画必须用到线程类Thread。线程是进程中一个控制执行流，其要经历开始、执行和结束三个阶段。线程是通过线程run( )方法实现的。当run( )方法建立并初始化后，执行时系统会自动调用run( )方法。通常情况下，run( )方法是一个循环体，循环调用多幅图片，实现动画效果。创建线程的方法是使用接口Runnable并在小程序的start方法中调用线程start方法。线程start方法被调用后，系统执行run( )方法。最后在小程序stop方法中调用线程的stop方法，该线程结束。

Updata( )方法：双缓存机制，在一幅图的同时，将另一幅图调入内存，但不显示。作用是使动画尽量变得平滑。部分程序如下：

```
... ...
public void run( )  /* 线程体,实现动画 */
{thread. Current Thread( ). Setpriority(Thread. MIN PRIORITY);
/* 设置线程最小优先级 */
imgs = getImage(getCodeBase( ), " images lmy01.gif ";
... ... }
public void update(Graphics g ) /* 实现双缓存 */
{if (erase (sequence [seqslot] =3))
{g. setcolor (color. LightGray); /* 设置背景颜色 */
... ... }
```

3 结束语

由于使用多线程技术，使用缓冲技术和类等，使得我们可以对复杂的多媒体元素进行处理和控制。实践证明，Java的多线程机制和缓冲技术具有很高的实用价值，可以增强多媒体元素的动态性和交互性效果；但如何将这些技术应用到分布式系统中，将是需要深入讨论的问题。

参考文献

1 Vanhelsuwe L 著，邱仲潘译，Java从入门到精通 [M]，电子工业出版社，1997。

2 史惠康，JAVA使用编程技术 [M]，中国水利水电出版社，1998。

3 The AWT Tutorial. Sun Microsystems, Inc. 1997。