远程教学 Java语言 多线程机制 CAI (20)

计算机科学1999Vol.26№.11

53-54189

用 Java 的多线程机制实现远程教学应用

Implementation of the Application on Distance-Learning with the Mechanism of Java Multi-Thread

孙华志 G728 G434
（天津师范大学计算机科学系 天津300074）

Abstract With the development of computer science and technology, CAI has become an important subject with great demand. The network and multimedia technology is becoming matured, which established the material foundation for distance learning. So it is possible for the network teaching system to practise. In this paper, some technical means, such as multi-thread, exception handle of Java Applet, are introduced, which gave a method for increasing the speed of sound transmission under the environment of network during the process of developing distance learning system.

Keywords Distance-learning, Network, Multimedia

1. 引言

随着计算机科学的发展和普及，计算机的应用领域日益广泛，计算机辅助教学（CAI）亦成为一个迅速发展起来的热门学科，特别是近年来，由于 Internet 网络技术和多媒体技术的发展与成熟，基于 Internet/Intranet 技术的信息系统和传统的信息系统相比，无论在技术上还是在服务方式上都有很大优势，这一切都为网上远程教学奠定了物质基础，使得网上教学成为可能。为了充分发挥网络资源优势，把握信息时代的脉搏，我们开始了"大学英语训练系统"的开发和研制，力图将网络和多媒体的最新技术应用于网上 CAI 教学软件。本课题属于天津市"教育信息资源服务系统"的一个子项目。本文将就该系统在研制过程中涉及的关键技术问题进行阐述。

2. 声音控制的一般手段

1995年 Java 语言一面世，就得到了人们的认可，并很快深入人心，一个纯 Java 程序将可以在任何具有 Java 虚拟机的环境中运行。这正是 Internet 网络环境下分布式计算所需要的，也是在以往的网络环境中所缺少的。因此，Java 语言的出现大大丰富了 Internet 的应用；而 Internet 的发展又大大促进了 Java 语言的普及。

Java 程序有两种类型：Java Applet 和 Java Application。Applet 是嵌入 Web 文档的程序，而 Application 是一般意义上的应用程序。就 Java 而言，Applet 与 Application 的大小和复杂性都没有限制，但是因为 Applet 主要是用于网络通讯，由于通讯速度有限，下载时间较长，因而 Applet 一般来说规模较小，而对于 Application 则无此顾虑。

Applet 与 Application 的运行环境决定了在技术方面的差别。Applet 需要来自浏览器的大量信息：浏览器客户机的位置和大小、嵌入主 HTML 文档的参数、初始化过程（init）、启动过程（start）、停止过程（stop）、终止过程（destory）、绘图过程（paint）等等，而 Application 则相对要简单得多，它来自外部世界的唯一输入就是命令行参数。

"大学英语训练系统"中的听力部分需要播放声音，如果该系统限制用户在 IE 浏览器中运行，这个问题很容易解决，因为 IE 支持 HTML 中的 bgsound 语句，且可直接播放 WAV 格式的声音。但在 Netscape 浏览器中不支持 bgsound 语句，而且在播放声音文件时自动调用播放器，这样就不能控制声音的播放次数，因此我们使用了 Java Applet。利用 Java Applet 将其嵌入 Web 页面，通过按钮控制声音的播放、停止（au 格式文件）、限定播放次数，而且在 IE 和 Netscape 中都能正常运行，音质效果较理想，Web 页面生动、有趣，引人入胜。

通过使用 Java 包 java.applet 中的 AudioClip 接口，为 Web 页加入声音。java.applet.AudioClip 类是关于声音数据的描述。在该类中定义了三个方法：

孙华志 副教授。

play、loop 和 stop。当给定声音的 URL 时，applet 类定义了一个 getAudioClip 方法以返回包含该声音的 AudioClip。

```
AudioClip sounds[];
sounds=new AudioClip[16];
sounds[0]=getAudioClip(getCodeBase(),"ques01.
          au");
sounds[1]=getAudioClip(getCodeBase(),"ques02
          au");
.....
for(i=0;i<16;i++);
  play(i);
  ......
;
```

3. 多线程控制

如果将听力部分录制在一个文件中，文件太大，会使下载速度很慢，为解决速度问题我们将整个听力部分分成多个文件，放在数组中，依次播放这些文件，但我们发现，Applet 并不是将一个 AudioClip 对象播放完再播放下一个对象，当执行第一个对象的 play 方法后，它继续执行第二个对象的 play，结果所有的文件几乎同时播放。这是由于 Java 支持多线程，它可并行执行多个任务(即线程)，播放数个文件。因此，要想让声音依次播放，必须保证系统中同一时刻只能有一个线程为运行状态，其它为休眠状态。

3.1 线程的概念

线程是一个抽象的概念，它包含了一个计算机执行传统程序时所做的每一件事情。线程是一种在 CPU 上调度的程序状态，它是从事实际工作的"事物"。在某一瞬间看来是计算过程的一个状态，简单地说，线程是程序中的一个执行流。

多线程程序是指一个程序中包含有多个执行流，多线程是实现并发的一种有效手段，Java 支持并发，即它能同时进行多个任务的并发处理。例如要同时完成计算和处理用户输入的两个任务，在单执行流程序中，只能通过设定定时时钟中断等方法，来经常性地中断计算任务的执行，以便检查是否有用户输入，这种方法既显得笨拙，又容易造成混乱。有了多线程的支持，这种问题就很容易解决。只需创建两个线程，一个用来处理用户输入事件，一个用来进行后台计算。两个线程同时运行，由系统来实现对它们的调度执行。多线程在提高系统吞吐率、有效利用系统资源、改善用户任务之间的通讯效率以及发挥多处理机硬件性能等方面有显著作用。Java 在语言级提供了对多线程的有效支持，通过语言和运行支持系统提供的复杂的同步机制，从而极大地方便了用户，有效地减少了多线程并行程序设计的困难。

3.2 线程的创建

Java 多线程程序设计的基础是 Java.lang 中的 Thread 类，线程的行为由线程体决定，Java 的线程体是由线程类的 run()方法定义。运行系统通过调用 run()方法实现线程的具体行为。可以有两种途径提供 run()方法的实现：继承 Thread 类和使用 Runnable 接口。我们采用的是在类中使用 Runnable 接口，并在该类中提供 run()方法的实现。这种方法为线程的创建提供了更大的灵活性。在本系统使用的 Java Applet 程序中，我们在初始化方法中创建线程。当 Applet 被加载到浏览器中时，调用初始化方法，由 Applet 自己创建线程，然后执行 start()方法运行它。

```
public class sound02 extends Applet
  implements Runnable{
  Thread soundThread=null;
  ......
public void init(){
  soundThread=new Thread(this);
.....
public void start(){
    if(soundThread==null){
      soundThread.start();
    ;
  }
}
```

创建线程后，可以在程序中对线程进行各种控制操作，包括启动、终止、挂起、恢复等，这些操作是用 Thread 类中的方法实现的，而具体能对线程实现什么操作则依赖于线程当时所处的状态。例如在 sound02.java 中，Applet 开始运行，播放第一段声音，同时创建一个线程(第二段声音)，并令其处于休眠状态，此时循环也暂停，不再继续调入 AudioClip 对象。当线程的休眠时间到，它自动恢复为运行状态(此时前一个声音已经播完)，而第三段声音被置为休眠状态……。

在此程序中，我们设计了两个按钮，分别用来播放(Play)和终止(Stop)声音。当按下 Play 按钮，系统调用线程的 Start()方法，开始播放声音，并让另一个线程等待。当按下 Stop 按钮，系统会将处于休眠状态的线程撤销，声音播放停止。

```
public boolean action(Event evt,Object obj)
{ if ("Play".equals(obj)){
    soundThread.start();}
  else if ("Stop".equals(obj))
    {if (soundThread.isAlive()){
       soundThread.suspend();}
     else{soundThread.resume();}}
  return true;
}
```

4. 异常处理

在程序的运行过程中，可能会由于各种各样的原因出错。如在线程运行过程中，试图挂起一个已终止的线程；令一个空线程处于休眠状态等。Java 语言用异常为它的程序提供了错误处理方式，为方法的异常终止和出错处理提供了清晰的接口。错误处理的一般情

(下转第89页)

```
  A[i,j]=A[j,i]+A[i,j]; //求度数//
  if A[i,i]<min then min=a[i,i]; //求最小度//
}
for (k=2;min;k++)
{if !(kn[2)then {输出"kn 非偶数";返回;}
else {for (i=2;n;i++)
  for {j=1;i-1;j++}
      while (A[i,j]=0)
       if (k|2)then {if A[i,i]+A[j,j]<n+2k-4 then 返
       回;}
       else if A[i,i]+A[j,j]<n+2k-3 then 返回;
       A[i,j]=1; // 加边 v,v, 入图//
       A[i,j]++;A[j,j]++; //度数增加;
       if(生成图是完全图) then 输出"G 有 k-因子";
       if(生成图有 k-因子) then 输出"G 有 k-因子";
      }
   }
if(不再找其他 k-因子) then {结束}
}
```

四、算法分析

此算法的正确性由上边定理1可直接得出。下边我们对算法的复杂性作一简要分析。

此算法的前半部(见算法描述,即注有求最小度前)用双重循环求每点度数及最小度;后半部用双重循环判定 G 是否有某一特定的 k-因子。由于假定图 G 是 n 阶的,因此问题规模是 n。显然可见,前半部和后半部都涉及了(n+1)n/2的计算步骤,即有

$$(n+1)n/2+(n+1)n/2=(n+1)n=O(n^2)$$

因此本算法的时间复杂性是 $O(n^2)$。

若采用下三角压缩存储,占用空间(n+1)n/2,因此其空间复杂性亦为 $O(n^2)$。

五、寻找 k-因子的一些初步结果

在上述算法中,若执行到第 i 步得到的 A_i 是完全图,则 G 必有 k-因子,这一事实便得到了验证。

对于完全图 $K_n(n>=3)$,当 n 为奇数时仅有偶数阶因子,当 n 为偶数时,有2～n－1各阶因子,举例说明如下:n＝3时,K_3即2-因子,n＝4时,有2-因子和3-因子 K_4;n＝5时,有2-因子和4-因子 K_5;n＝6时,有2-因子和3-因子,4-因子,5-因子 K_6;n＝7时,有2-因子和4-因子,6-因子 K_7;n＝8时,有2-因子和3-因子、4-因子,5-因子,6-因子,7-因子 K_8。

参 考 文 献

1 田丰,马仲蕃.图与网络流理论.北京:科学出版社,1987
2 邦迪 J A,默蒂 U S R.图论及其应用.北京.科学出版社,1984
3 孟亚,俞政.闭包与图的 k-因子,电子工程师杂志,1996
4 Nishimura T. Degree condition for the existence of k-factors. J. Graph Theory,1992,16:141～151

(上接第54页)

况是方法的设计者能够获悉运行时产生的错误,但通常不知道如何处理它,而方法的调用者可能知道如何处理这种错误。因此,引出异常处理的基本观点是:由发现错误但不能处理的方法引发一个异常,由该方法的直接或间接调用者处理这个错误,且能够处理该错误的方法可以声明它愿意处理之。异常为 Java 程序提供了一致的错误处理方式,即抛出异常、捕获异常、处理异常。

Java 中的每个异常都是一个对象,它是 Throwable 类或其子类的实例对象,这个对象是用来从出现异常的方法传递信息给捕获异常并处理之的方法。Throwable 类有两个标准子类:java.lang.Error 和 java.lang.Exception,即错误和异常。错误是指与虚拟机或动态装载等相关的问题,如系统崩溃等,这类错误一般被认为是不可恢复和不可捕获的。异常是指一些可以被捕获且可能恢复的异常情况,如读文件尾或数组访问下标越界等。

Java 的异常处理机制由 try/catch/finally 三个语句组成。把可能出现异常的 Java 语句放入由 try 引导的块中,try 块管理包含于其中的语句、定义与之相关的异常指针范围;try 块后面跟零个或多个由 catch 语句引导的块,catch 块把异常指针与 try 块联系起来,它负责处理指定类型的异常;catch 块后面跟一个由 finally 引导的块,它包含清除程序现场的语句,不论 try 中的代码如何退出,都将执行 finally 块。

在"大学英语训练系统"中,由于采用多线程,需要在任一时刻只有一个线程处于活动状态,其它线程处于休眠状态,且用按钮控制线程的 start()和 stop()。这种情况可能会产生 InterruptedException 异常,即当一个线程正在等待时,另一个线程来中断这个线程,因此我们在程序中加入了下面的异常处理。

```
public void play(int n){
 try{
   if (sounds[n]!=null){
     music=sounds[n];
     music.play();
   }
   soundThread.sleep(length[n]*1000);
   }
   catch(InterruptedException e){
   }
   }
```

参 考 文 献

1 Desborough J. Intranet Web Development. Prentice Hall, 1996
2 孙燕唐,等.面向 Windows 的 Internet 网络应用与开发.北京.电子工业出版社,1997
3 吴建平,尹霞.JAVA 程序设计语言.北京.清华大学出版社,1997