热点透视

Java 语言的基本编程原理(下)

王克宏　孙　元　丁　锂

下面我们用一个经典的例子来说明线程间的同步。

举例:生产者——消费者问题

```
class CubbyHole {
    private int seq; // this is the condition variable.
    private boolean available = false;
    public synchronized int get(){
    while( available==false ){
       try{
       wait( );
         }catch( InterruptedException e ){
       }
    }
    available = false;
    notify( );
    return seq;
    }
    public synchronized void put(int value){
       while( available==true ){
       try{
       wait( );
       }catch( InterruptedException e ){
       }
    }
    seq = value;
    available = true;
    notify( );
    }
}
class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Producer(CubbyHole c, int number) {
         cubbyhole = c;
         this.number = number;
    }
    public void run() {
         for (int i = 0; i < 10; i++) {
           cubbyhole.put(i);
           System.out.println("Producer #" +
           this.number + " put: " + i);
           try {
              sleep((int)(Math.random() * 100));
           } catch (InterruptedException e) {
           }
         }
    }
}
class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Consumer(CubbyHole c, int number) {
    cubbyhole = c;
    this.number = number;
    }
    public void run() {
         int value = 0;
         for (int i = 0; i < 10; i++) {
              value = cubbyhole.get();
              System.out.println("Consumer #" + this.
              number + " got: " + value);
           }
         }
    }
class ProducerConsumerTest {
    public static void main(String[] args) {
         CubbyHole c = new CubbyHole();
         Producer p1 = new Producer(c, 1);
         Consumer c1 = new Consumer(c, 1);
         p1.start();
         c1.start();
    }
}
```

在本例中,通过监控器 Cubbyhole 及其定义的同步方法 put()和 get(),以及同步方法中使用的 wait()和 notify()的同步控制,能够保证生产者线程每向存储单元中写一个数据,消费者线程就能立即取回数据并只取一次,从而实现同步。

7 Applet 编程

前面我们向大家介绍了 Java 程序的两种编程方式，下面我们就网络上广泛应用的 Java Applet 程序的特点和应用作一个初步的探讨。

7.1 Applet 的特点

Applet 是 Java 的小应用程序，它是动态、安全、跨平台的网络应用程序。Java Applet 嵌入 HTML 语言，通过主页发布到 Internet。网络用户访问服务器的 Applet 时，这些 Applet 从网络上进行传输，然后在支持 Java 的浏览器中运行。由于 Java 语言的安全机制，用户一旦载入 Applet，就可以放心地来生成多媒体的用户界面或完成复杂的计算而不必担心病毒的入侵。虽然 Applet 可以和图象、声音、动画等一样从网络上下载，但它并不同于这些多媒体的文件格式，它可以接收用户的输入，动态地进行改变，而不仅仅是动画的显示和声音的播放。归纳起来，applet 具有如下的一些特点：

基本的绘画功能：applet 程序能够利用 Java 丰富的类库创造出多姿多彩的图形效果，它能够对所显示的文字的字体和大小进行设置；能够通过库函数对一些规则的图形进行描绘，以及通过几何变换关系绘制不规则图形；能够直接调用主机内存储的现成的图形文件

动态效果的显示：applet 能够通过多线程的功能来实现程序的动态显示。

动画和声音的播放。

交互功能的实现。

窗口开发环境。

网络交流能力的实现：通过 Java 类库中所提供的通用资源定位指针 URL 等类实现通过网络进行交流的功能。

7.2 Applet 的执行框架

当在用户与浏览器的交互过程中发生下载 applet 程序、离开或返回 applet 所在的页、退出浏览器等与 applet 相关的事件时，applet 程序需要完成自己相应的行为，这些行为由 Java 兼容浏览器调用 Applet 类中的 init( )、start( )、stop( )、destroy( ) 和 paint( )这 5 个主要的方法来完成。下面，我们就来逐一介绍一下这 5 个方法：

(1) public void init( )

当一个 applet 程序被网络浏览器下载时，Java 兼容浏览器首先调用 init( )方法进行初始化，对 applet 程序进行初始的设置工作。在实际的使用中，用户可以在这个方法当中编写一些数据进行装入，或者对系统进行设置工作。

(2) public void start( )

在执行完 init( )方法后，系统调用 start( )方法进行 applet 程序的启动工作。该方法是 Applet 的主体，在其中可以启动相关的线程来执行任务。

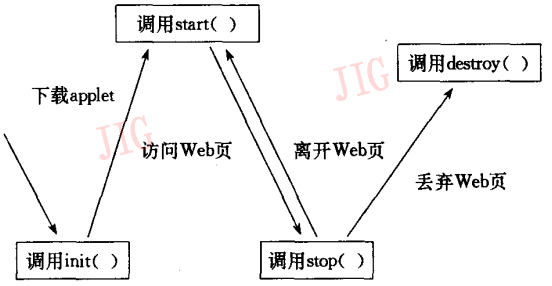
(3) public void stop( )

该方法与 start()方法相对应，用于终止 Applet 的执行。

(4) public void destroy( )

当用户退出浏览器时，通过使用 destroy( )方法释放 Applet 所分配的系统资源。

我们用下图来简单描述一下一个 applet 程序从创建到执行的各个步骤以及所调用的各个方法。



Applet 的创建和调用过程

7.3 Applet 的多媒体支持

人们喜爱 Java 的一个重要的原因在于它能够支持多媒体的应用，在 Java 中提供了很多的类来进行声音的播放、图形的绘制、文字的显示以及图像的处理等，限于篇幅，不能一一尽述。有关 Java 中的图形图像处理，我们将在下一讲中向大家详细介绍。

在这儿我们通过一个动画的例子来说明 Applet 的执行框架以及对图像的简单处理。

举例：

```
import java.aw t. * ;
import java.applet.Applet;
public class SunLogo extends Applet implements Runnable
{
    int frameNumber;
    int delay;
    int allFrame = 24;
```

```
    Thread animatorThread;
    Dimension offDimension;
    Image offImage;
    Graphics offGraphics;
    Image images[];
    MediaTracker tracker;
    public void init() {
        String str;
        str = getParameter("fps");
        int fps = (str != null) ? Integer.parseInt(str)
                : 5;
        delay = (fps > 0) ? (1000 / fps) : 100;
        images = new Image[60];
        tracker = new MediaTracker(this);
        for (int i = 1; i <= 13; i++) {
        images[i-1] =getImage(getCodeBase(),
                    "cc1-"+i+".gif");
        tracker.addImage(images[i-1], 0);
    }
    for (int i = 14; i <= allFrame; i++) {
        images[i-1] = images[allFrame-i+1];
        tracker.addImage(images[i-1], 0);
    }
}
public void start() {
    if (animatorThread == null) {
        animatorThread = new Thread(this);
        animatorThread.start();
    }
}
public void stop() {
    animatorThread = null;
    offImage = null;
    offGraphics = null;
}
public boolean mouseDown(Event e, int x, int y) {
    if (animatorThread == null) {
        start();
    }
    else {
        animatorThread = null;
    }
    return false;
}
public void run() {
    long startTime = System.currentTimeMillis();
    while (Thread.currentThread() == animatorThread)
{
        repaint();
        try {
          startTime += delay;
          Thread.sleep(Math.max(0, startTime Sys-
          tem.currentTimeMillis()));
        } catch (InterruptedException e) {
          break;
        }
        frameNumber++;
    }
}
public void paint(Graphics g) {
if (offImage != null) {
        g.drawImage(offImage, 0, 0, this);
    }
}
public void update(Graphics g) {
    Dimension d = size();
    if ( (offGraphics == null)
        || (d.width != offDimension.width)
        || (d.height != offDimension.height) ) {
          offDimension = d;
          offImage = createImage(d.width, d.height);
          offGraphics = offImage.getGraphics();
        }
        offGraphics.setColor(getBackground());
        offGraphics.fillRect(0, 0, d.width, d.height);
        offGraphics.setColor(Color.black);
        if (tracker.statusID(0, true) == MediaTrack-
er.COMPLETE) {
            offGraphics.drawImage(images[frameNumber
% allFrame], 0, 0, this);
        }
        g.drawImage(offImage, 0, 0, this);
    }
}
```

下面结合这个例子来讨论如何改善动画效果：

(1) 生成动画线程 由于显示动画是一个循环的过程，通常生成一个动画线程来完成图象的显示和更新。同时应该能让用户控制动画的启动和停止。

(2) 显示频率 在由显示频率确定了延迟时间以后，通过线程的方法 sleep()来实现延迟，同时要注意由于动画通过多线程来实现，所以使用绝对的延迟时间可能会使延迟过长，可以用我们例中的方法来解决该问题。

(3) 消除闪烁 要消除闪烁，可以重载 update( ) 方法，从而去掉不必要的清屏操作。另外，我们还可

以用双缓冲技术生成一幅后台图象,然后把后台图象一次显示与屏幕的方法来解决闪烁问题。

(4) 跟踪图象的载入 在刚启动程序时,由于没有全部载入图象,屏幕上显示的是残缺不全的画面,这时可以使用例中的 MediaTracker 或者 ImageObserver 来跟踪图象的载入,并进行相应的处理。

以上介绍的一些用 Java 语言实现交互功能的基本事件和实现方法。程序员可以通过使用 Java 语言来为自己的 Web 页上加入交互功能。除了另浏览者感到赏心悦目以外,也为程序增添了使用价值。

8 网络应用

对许多程序设计者来说,网络通讯似乎是很高深的事情。而在 Java 的环境下,实现网络通信却是件很容易的事,通过包 java.io 可以方便地进行网络通信。

Java 开发环境中最高一级的网络通讯是从网络上下载 Applet,用户浏览器通过 HTML 文件中的<APPLET>标记来识别 Applet,并解析 Applet 的属性,通过网络获取 Applet 的字节码文件。

这一部分,我们将讲述较低级的网络通讯,即不使用浏览器或 Java 开发环境中已提供的功能,而是通过 java.net 包中的类直接在程序中实现网络通讯。

8.1 通过 URL 及其相关的类来实现与 WWW 的连接

URL(Uniform Resource Locator)是通用资源定位指针的简称,它表示 Internet 上某一资源的地址。通过 URL,我们可以访问 Internet 和 WWW。浏览器通过解析给定的 URL 可以在网络上查找相应的文件或其它资源。

当我们得到一个 URL 对象后,就可以通过它从 WWW 读取指定的资源。这时我们将使用 URL 的方法 openStream( ),其接口为:

```
InputStream openStream( );
```

方法 openStream( )与指定的 URL 建立连接并返回 InputStream 类的对象以从这一连接中读取数据。

如果我们同时还想输出数据,就要用到类 URLConnection 了。URLConnection 类也在 java.net 包中定义,它表示 Java 程序和 URL 在网络上的通讯连接。

当与一个 URL 建立连接时,首先要在一个 URL 对象上通过方法 openConnection( )生成对应的 URLConnection 对象,URLConnection 对象设置一些连接参数,如 AllowUserInteraction、RequestProperty、UseCaches、DoOutput、DoInput 等,然后和远程对象建立实际的连接,最后,我们得到远程对象,并可获取各种连接参数,如 ContentLength、ContentType、HeaderField、InputStream、OutputStream 等。

URLConnection 类提供了很多方法来设置或获取连接参数,程序设计时最常使用的是 getInputStream( )和 getOutputStream( ),其接口为:

```
InputStream getInputStream( );
OutputStream getOutputStream( );
```

通过返回的输入/出流我们可以与远程对象进行通讯。

8.2 Socket 通讯

网络上的两个程序通过一个双向的通讯连接实现数据的交换,这个双向链路的一端称为一个 socket。socket 通常用来实现客户方和服务方的连接。下面我们来介绍如何用 socket 进行 client/server 程序设计。

8.2.1 socket 通讯的步骤

无论一个 socket 通讯的功能多么齐全,程序多么复杂,其基本结构都是一样的,都包括以下 4 个基本的步骤。

(1) 打开 socket。

(2) 打开连接到 socket 的输入/出流。

(3) 按照一定的协议对 socket 进行读/写操作。

(4) 关闭 socket。

通常,程序员主要是针对所要完成的功能在第 3 步进行编程,第 1、2、4 步对任何程序几乎都是一样的。

8.2.2 打开 socket

Java 在 java.net 包中提供了 2 个类:Socket 和 ServerSocket,分别用来表示双向连接的客户端和服务端。对于客户端程序,可以通过生成一个 Socket 对象打开 socket,例如:

```
Socket client;
client=new Socket ("host name", Port number);
```

对于服务方,我们通过生成一个 ServerSocket 对象打开 socket,然后调用方法 accept( )准备接收客户发来的连接请求,同样, (下转 249 页)

作为一个个输入变量进行处理，并按公式中规定的顺序和规则形成逻辑的数据流，直到输出。

在算法方面，ER Mapper 一向是为用户免费加入新的算法，不管是用户自己研制的而由 ER Mapper 帮助开发的算法，还是 ER Mapper 自己新研制的算法，对用户都一律免费。

5　输入/输出数据兼容和设备可扩性

用户最关心的问题之一是数据输入/输出是否兼容性强、是否灵活、外设增加是否方便等等，ER Mapper 系统的答案是：

支持 187 种硬拷贝输出的图象格式，包括已经光栅化的图象和待光栅化的矢量文件。这一特生几乎包括了当前国际上流行的所有设备和数据格式，因此，用户无论期望什么样的输出设备，还是期望将自己有的外设加入 ER Mapper，您都会如愿以偿。

ER Mapper 所支持的数据格式不仅来自于遥感（Landsat、SPOT、ERS-1、JERS-7、NOAA AVHRR、航空多光谱等等），而且还来自地球物理勘探的各个领域，如测井、地震、航磁、重力等等，因而形成发综合地学信息分析的基础。

ER Mapper 在硬拷贝制图方面，用户只需要感觉逻辑设备的存在，规定了制图的比例尺和设备类型之后，就不需要再作任何操作，相应的软件会把图象自动分条分幅，一次批量拷贝，这样用户可以迅速得到优质的、符合制图标准的输出结果。

ER Mapper 中丰富的图形和文字编辑、注记、地理信息和其它数据的动态连接，使用户可以将传统制图的各种要素和图元迅速、灵活地编排成图，并使用漂亮的动态图象例库及边框图案自动整饰技术来修整全图，配合丰富的外设接口种类，实现专题制图的全自动化。

（北京特灵顿自控系统公司）

（上接 242 页）

选择端口号时也要防止发生冲突。

```
ServerSocket server = new ServerSocket (Port number);
Socket socket=server.accept();
```

方法 accept( )等待客户的请求，直到有一个客户启动并请求连接到相同的端口，然后 accept( )返回一个对应于客户的 socket。这时，客户方和服务方都建立了用于通讯的 socket，注意在建立 socket 连接时，必须处理例外事件。接下来就是由各个 socket 分别打开各自的输入/出流。

8.2.3　打开输入/出流

Socket 类提供了方法 getInputStream( )和 getOutputStream( )来得到对应的输入/出流以进行读/写操作，这两个方法分别返回 InputStream 和 Outputstream 类对象。为了便于读/写数据，我们还可以在返回的输入/出流对象上建立过滤流，如 DataInputStream、DataOutputStream 或 PrintStream 类对象。

8.2.4　关闭 socket

在关闭 socket 之前，应将与 socket 相关的所有的输入/出流全部关闭，这是因为一个良好的程序应该在执行完毕时清除所有的资源。

需要注意的是，在关闭 socket 时，一定要注意关闭的顺序，与 socket 相关的所有的输入/出流应该首先关闭，然后再关闭 socket。

8.3　多客户程序

一个典型的多客户机制包含运行在某台主机上的服务程序和其它机器上的多个客户程序，服务程序一般作为一个永久的进程，等待客户的请求，并提供相应的服务。多客户程序的实现是通过多线程的并发执行完成的。

在多客户程序中，客户方程序与单客户是一样，而服务方程序只需在每接收一个用户的 socket 请求时，就为该客户生成一个新的线程来处理该用户的事务即可。

这一讲我们向大家介绍了 Java 语言基本类库的编程，限于篇幅，每一部分都只作了简单的介绍，掌握了这些基本编程原理以后，您就可以得心应手的运用 Java 中丰富的类库来编写各种程序了。