Java 2 基于自定义事件的动画标签和按钮的实现

Java 2 according to from the animation label and buttons of the definition affairs

曹大有 王 瑜 （郧阳师范高等专科学校 计算机科学系 湖北 丹江口 442700）

摘要：在 Java 2 的 Swing 组件体系下，JLabel 和 JButton 两个控件能显示动画 GIF 图片。本文则从自定义事件的角度出发，探讨了在 AWT 和 Swing 组件体系下实现这一过程的另一机制。

关键词：事件 监听器 AWT Swing

在 Java 2 的 Swing 组件体系中的 JLabel 和 JButton 两个控件的构造器中，由于可以指定实现了接口 Icon 的图标，我们不但可以在 JLabel 和 JButton 上显示已有 GIF 图片、自绘的图片，而且也可以显示动画 GIF 图片，从而实现动画标签和按钮[2]，如 ButtonImageTest. java 和 LabelImageTest. java 两个程序所示那样。但那是通过已知动画 GIF 图形所形成的，这里能否让 JLabel 和 JButton 连续显示几幅静止 GIF 图片来形成动画呢？本文则从 Java 2 自定义事件的角度出发来实现这一构想。

1 Java 2 预定义事件的分析

为了在 Java 2 中实现自定义事件并能响应自定义事件，我们首先必须弄清 Java 2 的预定义事件实现的机制，由于在 Java 2 中不论是 AWT 组件还是 Swing 组件，它们的事件响应机制都是一样的。下面我们就以行动事件 ActionEvent 的定义和响应过程为例来对 Java 2 的预定义事件进行分析。

首先，我们先来看 ActionEvent 的定义主要过程：

```
public class ActionEvent extends AWTEvent {//行动事件类的定义
    ……;
private static final long serialVersionUID = -7671078796273832149L;
public ActionEvent ( Object source, int id, String command) {
        this ( source, id, command, 0 ) ;}//构造函数
public ActionEvent ( Object source, int id, String command, int modifiers) {
        this( source, id, command, 0, modifiers) ;}//构造函数
public ActionEvent ( Object source, int id, String command, long when,
        int modifiers) {//构造函数
    super( source, id) ;
    ……;}
……;}
```

从以上可以看出：ActionEvent 的定义是从扩展 AWTEvent 开始，这说明 Java 2 的事件队列中的事件一般都是 AWTEvent 的子类，ActionEvent 事件构造器中必须指定引发行动事件的事件源对象 source，最后要给行动事件指定一个 ID 号码。当然为了正确实现回调，还必须为行动事件定义相应的事件监听器接口 ActionListener，其定义很简单，实现过程为：

```
public interface ActionListener extends EventListener {
    public void actionPerformed( ActionEvent e) ;}
```

接着我们来看一看能引发行动事件的组件类本身是如何响应和管理行动事件的。由于能引发行动事件的事件源很多，下面我们以 Button 类为例来加以介绍，在 Button 中与此有关的代码为：

```
transient ActionListener actionListener;
public synchronized void addActionListener( Action-
```

```
Listener l) {
    if (l == null) { return;}
    actionListener = AWTEventMulticaster.add(actionListener, l);
    newEventsOnly = true;} //增加事件监听器方法
    public synchronized void removeActionListener(ActionListener l) {
    if (l == null) { return;}
    actionListener = AWTEventMulticaster.remove(actionListener, l);} //删除事件监听器方法
    public synchronized ActionListener[ ] getActionListeners( ) {
    return (ActionListener[ ]) (getListeners(ActionListener.class));}
    //通过调用下面方法来求出组件的事件监听器队列
    public EventListener[ ] getListeners(Class listenerType) {
    ……;}//求出组件的事件监听器队列,代码在此省略
    boolean eventEnabled(AWTEvent e) {
    ……;}//允许组件将相应的事件插入到事件队列中,代码在此省略
    protected void processEvent(AWTEvent e) {//事件处理方法
    if (e instanceof ActionEvent) {
    processActionEvent((ActionEvent)e);
    return;}
    super.processEvent(e);}
    protected void processActionEvent(ActionEvent e)
{//事件处理方法
    ActionListener listener = actionListener;
    if (listener != null) {
    listener.actionPerformed(e);}}
```

在以上代码中,方法 addActionListener(ActionListener l)、removeActionListener(ActionListener l)、getActionListeners( )和 getListeners(Class listenerType)的作用是为 Button 类管理监听器队列,eventEnabled(AWTEvent e)方法的功能是为了将行动事件加入到系统事件队列中去的,保护方法 processEvent(AWTEvent e)是 Java 系统只要从事件队列中删除一个事件对象,系统便会自动调用它,它的作用是判断事件如果是一个行动事件,便会自动调用 processActionEvent(ActionEvent e)来通过回调实现事件的响应。

2 Java 2 自定义事件的实现

有了以上预定义事件的分析,下面我们就可以实现自定义事件。由于本文主题是要在 JLabel 和 JButton 上通过固定的时间间隔显示静止的 GIF 图片来实现动画,所以我们定义一个简单的计时器 Timer[1]。首先我们从必需的计时器事件的监听器开始,它的定义为:

```
interface TimerListener extends EventListener
{public void timeElapsed(TimerEvent evt);}// 计时器事件的监听器
```

其次来定义计时器事件:TimerEvent,它的定义为:

```
class TimerEvent extends AWTEvent{
 public TimerEvent(Timer t) { super(t, TIMER_EVENT);}//TimerEvent 构造方法
 public static final int TIMER_EVENT = AWTEvent.RESERVED_ID_MAX + 5555;}
```

最后来定义计时器对象本身:Timer,它的定义为:

```
class Timer extends Component implements Runnable{
 public Timer(int i){
 interval = i;// 计时间隔时间
 Thread t = new Thread(this);//生成线程
 t.start( );//线程启动
 evtq = Toolkit.getDefaultToolkit( ).getSystemEventQueue( );
 enableEvents(0);}//求系统事件队列,并允许计时器事件插入事件队列
 public void addTimerListener(TimerListener l){
 listener = l;}//管理监听器队列
 public void run( ){//生成计时器事件并将事件发送出去
 while(true){
 try{Thread.sleep(interval);}
 catch(InterruptedException e){}
```

```
TimerEvent te = new TimerEvent(this); //生成计时器事件
evtq.postEvent(te);}//发送计时器事件
public void processEvent(AWTEvent evt){ //事件处理方法
if(evt instanceof TimerEvent){
if(listener! =null)
listener.timeElapsed((TimerEvent)evt);
}else
super.processEvent(evt);}
private int interval;
private TimerListener listener;
private static EventQueue evtq;}
```

它之所以要扩展 Component 类，是因为 Java 的事件机制要求每个事件源都要对 Component 类进行扩展，另外我们也想把计时器定义为不可视组件。为什么要实现 Runnable 接口，是因为它要构造出时间间隔，只要到达规定的计时周期，便生成一个新的计时器时间，并将其插入系统事件队列中去。

3 Swing 的动画标签和按钮

现在我们就来实现能显示动画的 Swing JLabel 和 JButton 组件。其基本原理是：由于 JLabel 和 JButton 组件都有接收 Icon 图标的功能，具体体现为：在构造器方法中指定或通过方法 setIcon(Icon icon)设定。所以我们对 JLabel 和 JButton 进行扩展，在扩展的类中加入计时器事件并在计时器事件处理方法中通过 setIcon(Icon icon)方法来变化图标，从而实现动画显示。

对 JLabel 的扩展为：

```
class CustomEventLabel extends JLabel implements TimerListener{
public CustomEventLabel(){//扩展 JLabel 类的构造方法
super("动画标签演示");//调用父类的构造方法
Timer t = new Timer(500);//生成一个计时器对象
t.addTimerListener(this);// 增加计时器对象监听器}
public void timeElapsed(TimerEvent evt){//计时事件处理方法
icon = new ImageIcon("su0" +(ticks% 4 +1) +".gif");//由图像生成图标
setIcon(icon);//设置标签对象的图标
ticks + +;//计数器增加 1}
private int ticks =0;
private Icon icon;}
```

对 JButton 的扩展为：

```
class CustomEventButton extends JButton implements TimerListener{
public CustomEventButton(){//扩展 JButton 类的构造方法
super("动画按钮演示");//调用父类的构造方法
Timer t = new Timer(500); //生成一个计时器对象
t.addTimerListener(this); // 增加计时器对象监听器}
public void timeElapsed(TimerEvent evt){ //计时事件处理方法
icon = new ImageIcon("su0" +(ticks% 4 +1) +".gif");//由图像生成图标
setIcon(icon); //设置按钮对象的图标
ticks + +;//计数器增加 1}
private int ticks =0;
private Icon icon;}
```

上面我们是通过在扩展有 JLabel 和 JButton 中连续显示 4 幅 GIF 图片形成动画，要注明的是必须将 GIF 图片通过方法：ImageIcon()变成 Icon 图标才可以用。(具体可参见附带程序 CustomEventTest1.java 和 CustomEventTest.java 所示)

4 AWT 的动画标签和按钮

我们同样也可以对 AWT 的 Label 和 Button 组件实现动画显示，由于它们不能指定图标，所以只能通过连续显示字符数据来达到动画效果。

对 Label 的扩展为：

```
class CustomEventLabel extends Label implements TimerListener{
public CustomEventLabel(){//扩展 Label 类的构
```

```
造方法
    super("动画标签演示");//调用父类的构造方法
    Timer t = new Timer(500); //生成一个计时器对象
    t.addTimerListener(this); // 增加计时器对象监听器}
  public void timeElapsed(TimerEvent evt){ //计时事件处理方法
    setText("标签" + ticks% 4); //设置标签对象的图标
    ticks + +;//计数器增加1
   }
  private int ticks = 0;}
```

对 Button 的扩展为：

```
class CustomEventButton extends Button implements TimerListener{
  public CustomEventButton(){//扩展 Button 类的构造方法
    super("动画按钮演示");//调用父类的构造方法
    Timer t = new Timer(500); //生成一个计时器对象
    t.addTimerListener(this); // 增加计时器对象监听器}
  public void timeElapsed(TimerEvent evt){ //计时事件处理方法
    setLabel("按钮" + ticks% 4); //设置按钮对象的图标
    ticks + +;//计数器增加1}
  private int ticks = 0;}
```

（具体可参见附带程序 CustomEventTest11.java 和 CustomEventTest0.java 所示）

5 总结

本文通过分析 Java 2 预定义事件的实现机制，讨论了如何实现自定义事件。并通过自定义事件实现了 AWT 下和 Swing 下的动画标签、按钮，实际上只要是能接收 ICON 图标的控件都可以通过此方法实现动画显示。另外通过本文也可以使我们详细了解 Java 2 在这几方面实现的全过程。

参考文献

1 （美）Cay s. Horstmann、Gary Cornell 著，京京工作室译，Java 2 核心技术（卷 I：基础知识）[M]，北京 机械工业出版社，2000：260－265。

2 （美）John Zukowski 著，邱仲潘译，Java 2 从入门到精通[M]，北京 电子工业出版社，2000：319－323。