



第 9 章

图形用户界面设计

第9章 图形用户界面设计

- ◆ Swing和AWT
- ◆ 创建一个基本GUI程序
- ◆ 常用的时间及其相应的监听器接口
- ◆ 菜单

9.1 Swing 和 AWT

图形界面就是用户界面元素的有机合成。

设计和实现图形用户界面时主要包含两项内容：

- (1) **创建**图形界面中需要的**元素**，**进行**相应的**布局**。
- (2) **定义**界面元素对用户交互**事件的响应**以及对**时间的处理**。

9.1 Swing 和 AWT

- 9.1.1 AWT 组件

AWT（抽象窗口工具集） 是Java为**GUI（图形用户界面）** 设计提供的最原始的工具包，它在Java技术的每个版本上都成为了一种**标准配置**，在任何一个Java运行环境中都可以使用它。

弱点： 由于AWT要依赖于主机的对等体控件来实现GUI，因此GUI的外观和行为在**不同的主机上会有所不同**。因此就开发了**Swing组件**。

9.1.2 Swing组件

- ◆ Java Swing组件在**javax.swing**包中，它是试图解决AWT缺点的一个尝试。Swing组件也是AWT组件的一部分。
- ◆ Swing是在AWT组件基础上构建的，所有Swing组件也是AWT组件的一部分。
- ◆ 同样功能的组件，为了在名称生能够区分，Swing组件多以【 J 】开头，例如：**JFrame**、**JDialog**、**JPanel**等。
- ◆ **Swing**使用了AWT的事件模型和支持类，例如：**Colors**、**Images**和**Graphics**。

9.1.2 Swing组件

- ◆ 为了克服在不同机器上行为会不同的缺点，Swing将对主机空间的依赖性降到了最低。
- ◆ Swing只为诸如窗口和框架之类的顶层组件使用对等体，大部分组件都是使用纯Java代码来模拟的。
- ◆ Swing可以再所有主机之间很好地进行移植。
- ◆ 除了具有更多的组件、布局管理器和事件之外，Swing还有很多特性使得比AWT的功能更加强大。

9.1.3 容器类组件

◆ 窗口 (Window)

- 窗口类产生一个顶级窗口 (Window) 。
- Window对象是一个没有边界和菜单栏的顶层窗口，它直接出现在桌面上。。
- 通常不会直接产生Window对象。将使用Window类的子类，即 Frame，窗口的默认布局是 BorderLayout。
- 构造窗口时，它必须拥有窗体、对话框或其他作为其所有者定义的窗口。

9.1.3 容器类组件

◆ 框架 (Frame)

- Frame是带有标题和边框的顶层窗口。
- 窗口的大小包括为边框指定的所有区域。
- 当一个Frame窗口被程序创建时就创建了一个通常的窗口。
- 与Frame不同当用户视图关闭窗口时，JFrame知道如何进行相应。
- 用户关闭窗口时，默认的行为只是简单地隐藏JFrame。
- 要更改默认的行为，可调用方法 `setDefaultCloseOperation(int)`。

9.1.3 容器类组件

◆ 面板 (Panel)

- Panel类是Container类的一个具体子类。它没有添加任何新的方法，只是简单地实现了Container类。
- 一个Panel对象可以被看作是一个递归嵌套的具体的屏幕组件，Panel类是Applet类的子类。
- 当屏幕输出直接传递给一个小应用程序时，它将一个Panel对象的表面被画出。
- 实际上，一个Panel对象是一个不包含标题栏、菜单栏以及边框的窗口。

9.1.3 容器类组件

◆ 画布 (Canvas)

- 虽然画布不是小应用程序和Frame窗口的层次结构的一部分，但是Canvas这种类型的窗口是很有用的。
- Canvas类封装了一个可以用来绘制的空白窗口。

9.2 创建一个基本GUI程序

■ 9.2.1 使用JFrame类创建一个框架

JFrame主要构造方法：

- ◆ **JFrame()** 构造一个初始时不可见的新窗体。
- ◆ **JFrame(String title)** 创建一个新的、初始时不可见的、具有指定标题的 Frame
- ◆ **JFrame(String title, GraphicsConfiguration gc)** 创建一个具有指定标题和指定屏幕设备的 GraphicsConfiguration 的 JFrame。

JFrame类的几个常用的方法

- ◆ **setSize**方法用来设置窗口的大小

void setSize(int newWidth, int newHeight)

- ◆ **setVisible**方法用来隐藏和显示一个窗口

void setVisible(boolean visibleFlag)

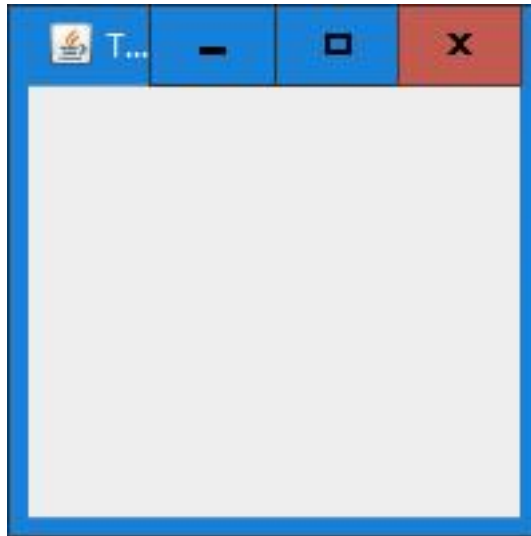
- ◆ **setTitle**方法用来设置或改变窗口标题

void setTitle(String newTitle)

【例9.1】 如何创建一个GUI框架

```
import java.awt.*;
import javax.swing.*;
public class TestFrame_1 {
    public static void main(String args[]) {
        //创建一个JFrame的实例
        JFrame frame = new JFrame();
        //将JFrame设置成200*200
        frame.setSize(200,200);
        //设置框架的标题
        frame.setTitle("TestFrame");
        //显示JFrame
        frame.setVisible(true);
    }
}
```

【例9.1】 如何创建一个GUI框架



9.2.2在框架中添加组件

- **FlowLayout**布局管理器
- **BorderLayout**布局管理器
- **CardLayout**布局管理器
- 网格布局管理器

9.2.2在框架中添加组件

【例9.2】如何在框架中添加按钮

```
import java.awt.*;
import javax.swing.*;
public class TestFrame_2 {
    public TestFrame_2 (){}
    public static void main(String args[]) {
        JFrame frame = new JFrame("TestJFrame");
        //创建一个JFrame的实例
        frame.setSize(300,300); //将JFrame设置成300*300
        frame.setVisible(true); //显示JFrame
        JButton button1 = new JButton("确定");
        //创建一个JButton的实例
        frame.add(button1);
    }
}
```


9.2.2在框架中添加组件

【例9.2】如何在框架中添加按钮



由于上例的框架不能重用，所以做修改。

```
public class TestFrame_3 extends JFrame {  
    public TestFrame_3(){  
        //创建一个JButton的实例  
        JButton button1 = new JButton("确定");  
        //将组件button1添加到本类对象框架中  
        this.add(button1);  
        //将JFrame设置成300*300  
        this.setSize(300,300);  
        this.setTitle("TestFrame");  
    }  
    public static void main(String args[]) {  
        TestFrame_3 f=new TestFrame_3 ();  
        f.setVisible(true);           //显示JFrame  
    }  
}
```

测试如何重用上述框架，代码如下：

```
import javax.swing.*;

public class UseTestFrame {

    public static void main(String []args){

        //调用了TestFrame类的构造方法

        JFrame f=new TestFrame_3();

        f.setVisible(true);

    }

}
```

由于上例的框架不能重用，所以做修改。



9.2.3 设置界面布局

【例9.3】 试图在框架中添加多个按钮，但没有做任何布局设置

```
import javax.swing.*;
```

```
public class TestFrame_4 {
```

```
    public static void main(String args[]) {
```

```
        //创建一个JFrame的实例
```

```
        JFrame frame = new JFrame("TestJFrame");
```

```
        frame.setSize(200,200);           //将JFrame设置成200*200
```

```
        frame.setVisible(true);           //显示JFrame
```

```
        //创建一个JButton的实例
```

```
        JButton button1 = new JButton("确定");
```

```
        JButton button2 = new JButton("取消");
```

```
        frame.add(button1);
```

```
        frame.add(button2);
```

```
    }
```

```
}
```

9.2.3 设置界面布局

【例9.3】 试图在框架中添加多个按钮，但没有做任何布局设置



FlowLayout布局管理器

- ◆ FlowLayout定义在java.awt包中。
- ◆ 对容器中组件进行布局的方式是将组件逐个安放在容器中的一行上，一行放满后另起一个新行。
- ◆ 在默认情况下，将组件居中放置在容器的某一行上。
- ◆ 不强行设置组件的大小，允许组件拥有它们自己所希望的尺寸。
- ◆ 每个组件都有一个getPreferredSize()方法，布局管理器会调用这一方法取得每个组件希望的大小。


FlowLayout布局管理器

FlowLayout定义在java.awt包中，对容器中组件进行布局的方式是将组件逐个安放在容器中的一行上，一行放满后另起一个新行。

- FlowLayout布局管理器
- BorderLayout布局管理器
- CardLayout布局管理器
- 网格布局管理器

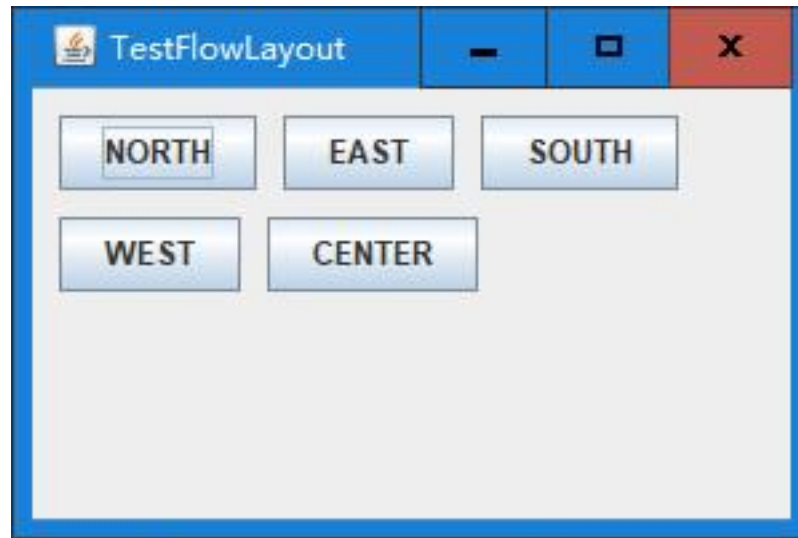
【例9.4】如何同时为多个组件设置布局

```
import java.awt.*;
import javax.swing.*;
public class TestLayout {
    public TestLayout(){
    }
    public static void main(String args[]) {
        JFrame frame=new JFrame();
        frame.setTitle("TestFlowLayout");
        frame.setSize(300,200);           //将JFrame设置成500*500
        frame.setVisible(true); //显示Jframe
        //用户单击窗口的关闭按钮时程序执行的操作
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //设置间距为水平10，垂直10的边界布局
        frame.setLayout(new FlowLayout(FlowLayout.LEFT,10,10));
        JButton button1 = new JButton("NORTH");//创建JButton实例
```



```
JButton button2 = new JButton("EAST");  
JButton button3 = new JButton("SOUTH");  
JButton button4 = new JButton("WEST");  
JButton button5 = new JButton("CENTER");  
frame.add(button1);  
frame.add(button2);  
frame.add(button3);  
frame.add(button4);  
frame.add(button5);  
  
}  
  
}
```

布局管理器



【例9.5】如何创建FlowLayout布局框架

```
package com.Gui;
import java.awt.*;
import javax.swing.*;
public class TestLayout extends JFrame{
    public TestLayout(){
        setLayout(new FlowLayout(FlowLayout.LEFT,10,10));
        //设置间距为水平10，垂直10的边界布局
        JButton button1 = new JButton("NORTH");
        //创建一个JButton的实例
        JButton button2 = new JButton("EAST");
        JButton button3 = new JButton("SOUTH");
        JButton button4 = new JButton("WEST");
        JButton button5 = new JButton("CENTER");
        add(button1);
        add(button2);
        add(button3);
        add(button4);
        add(button5);
    }
}
```

【例9.5】 如何创建FlowLayout布局框架

```
public static void main(String args[]) {  
    TestLayout frame = new TestLayout();  
    //创建一个JFrame的实例  
    frame.setTitle("TestFlowLayout");  
    frame.setSize(300,200); //将JFrame设置成500*500  
    frame.setVisible(true); //显示Jframe  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

BorderLayout布局管理器

- ◆ BorderLayout是顶层容器中内容窗格的默认布局管理器。
- ◆ 容器被划分成北(North)、南(South)、西(West)、东(East)、中(Center)五个区域，分别代表容器的上、下、左、右、中不。
- ◆ BorderLayout 定义在 java.awt 包中，使用 add(BorderLayout, index) 方法可以将组件加到 BorderLayout 中。
- ◆ Index是一个常量，取值为 BorderLayout.NORTH、BorderLayout.SOUTH、BorderLayout.WEST、BorderLayout.EAST、BorderLayout.CENTER。

【例9.6】如何创建BorderLayout布局框架

```
package com.Gui;
import java.awt.*;
import javax.swing.*;
public class TestLayout extends JFrame{
    public TestLayout(){
        setLayout(new BorderLayout(20,30));
        //设置间距为水平10，垂直20的边界布局
        JButton button1 = new JButton("NORTH");
        //创建一个JButton的实例
        JButton button2 = new JButton("EAST");
        JButton button3 = new JButton("SOUTH");
        JButton button4 = new JButton("WEST");
        JButton button5 = new JButton("CENTER");
        add(button1,BorderLayout.NORTH);
        add(button2,BorderLayout.EAST);
        add(button3,BorderLayout.SOUTH);
        add(button4,BorderLayout.WEST);
        add(button5,BorderLayout.CENTER);
    }
}
```

【例9.6】 如何创建BorerLayout布局框架

```
public static void main(String args[]) {  
    TestLayout frame = new TestLayout();  
    //创建一个JFrame继承类 TestLayout的实例  
    frame.setTitle("TestJFrame");  
    frame.setSize(300,200); //将JFrame设置成500*500  
    frame.setVisible(true); //显示Jframe  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```


CardLayout布局管理器

- ◆ CardLayout是一种卡片式的布局管理器。
- ◆ 将容器的组件处理为一系列的卡片，每一时刻只显示出其中的一张。
- ◆ javax.swing包中定义了JTabbedPane类，它的使用效果与CardLayout类似，但更为简单。
- ◆ CardLayout()创建一个间距大小为0的新卡片布局。
- ◆ CardLayout(int hgap, int vgap)创建一个具有指定水平间距和垂直间距的新卡片布局。

CardLayout布局管理器

- ◆ 常用的方法有：
- ◆ `last(Container parent)` 翻转到容器的最后一张卡片
- ◆ `next(Container parent)` 翻转到指定容器的下一张卡片
- ◆ `previous(Container parent)` 翻转到指定容器的前一张卡片

【例9.7】 使用JTabbedPane实现CardLayout布局

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JTabbedPane;
public class TestCardlayout extends JFrame {
    public TestCardlayout() {
        this.setTitle("使用JTabbedPane实现卡片式布局");
        setSize(400, 300);
        JTabbedPane card = new JTabbedPane();
        card.addTab("大数据", null, new JLabel("大数据内容"));
        card.addTab("云计算", null, new JLabel("云计算内容"));
        add(card);
    }
    public static void main(String[] args) {
        TestCardlayout test = new TestCardlayout();
        test.setVisible(true);
    }
}
```

网格布局管理器

- ◆ 网格布局管理器(GridLayout)将容器空间划分成若干行乘若干列的网格
- ◆ 组件依次放入网格中，每个组件占据一格网格
- ◆ 组件被放入容器的次序决定了它所占据的位置
- ◆ 每行网格从左至右依次填充，
- ◆ 容器的大小改变时，GridLayout所管理的组件的相对位置不会发生变化，但组建的大小会随之改变。

【例】

```
import java.awt.*;  
import javax.swing.*;  
public class TestCardlayout extends JFrame {  
    public TestCardlayout() {  
        this.setTitle("GridLayout");  
        this.setLayout(new GridLayout(3,3));  
        //将界面设置成3×3的网格  
        b1=new JButton("云计算");  
        b2=new JButton("大数据");  
        b3=new JButton("5G");  
        b4=new JButton("数据中心");  
        b5=new JButton("信息消费");  
        b6=new JButton("软件产业");
```

【例】

```
        this.add(b1);
        this.add(b2);
        this.add(b3);
        this.add(b4);
        this.add(b5);
        this.add(b6);
    }
    public static void main(String[] args) {
        TestCardlayout f = new TestCardlayout();
        f.setVisible(true);
    }
}
```

9.2.4事件处理

- **事件**：用户对组件的一个操作,称之为一个事件，也就是说按钮被按下事件，还是鼠标被拖动事件；
- **事件源**：发生事件的组件就是事件源，明确到底由哪个组件来完成功能；
- **事件处理器**：某个java类中负责处理事件的成员方法，对由哪个组件发出的什么样的动作进行对应的响应

【例9.8】如何例9.2中的“确定”按钮执行关闭窗口的工作

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestEvent extends JFrame {
    JButton button1;
    public TestEvent(){
        setSize(300,300);    //将JFrame设置成300*300
        //创建一个JButton的实例
        button1 = new JButton("确定");
        add(button1);
    }
}
```


//事件处理方法

```
public static void main(String args[]) {
```

```
    //创建框架并完成添加组件在框架中
```

```
    TestEvent frame=new TestEvent();
```

```
    frame.setVisible(true);        //显示JFrame
```

```
    //将组件与事件响应程序链接在一起
```

```
    frame.button1.addActionListener(new EventResponse());
```

```
}
```

```
}
```

//定义一个监听器类完成响应组件按钮按下的事件

```
class EventResponse implements ActionListener{
```

```
    //事件响应程序
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        System.exit(0);
```

```
    }
```

```
}
```



9.3 常用的事件及其相应的监听器接口

■ 9.3.1 Java中事件

事件：用户对程序的某一种功能性操作，Java中的事件类都包含在JDK的`Java.awt.event`包中。

■ Java中的事件主要有两种：

- (1) 组件类事件
- (2) 动作类事件

(1) 组件类事件

- **ComponentEvent**

操纵某组件时发生的一个高层事件

- **ContainerEvent**

向容器添加或删除组件时发生

- **WindowEvent**

操作窗口时发生的事件，如最大化或最小化某一窗口

- **FocusEvent**

获取或失去焦点

- **PaintEvent**

描绘组件时发生的一个事件

- **MouseEvent**

操作鼠标时发生

(2) 动作类事件

- **ActionEvent**

激活组件时发生的事件

- **TextEvent**

更改文本时发生

- **AdjustmentEvent**

调节可调整的组件（如移动滚动条）时发生的事件

- **ItemEvent**

在选择项、复选框或列表中选择时发生

9.3.2 Windows 事件处理

- **Windows**类的任何子类都可能触发当通过打开、关闭、激活或停用、图标化或取消图标化而改变了窗口状态窗口事件。
- 接口 **WindowListener**是用于接收窗口事件的侦听器接口，旨在处理窗口事件的类要么实现此接口（及其包含的所有方法），要么扩展抽象类 **WindowAdapter**（仅重写所需的方法）。然后使用窗口的 **addWindowListener**方法将从该类所创建的侦听器对象向该Window注册。

@Override

◆ @Override的作用是：

如果想重写父类的方法，比如toString()方法的话，在方法前面加上@Override系统可以帮你检查方法的正确性。

◆ Override的用法：

Override:java.lang.Override 是一个 marker annotation类型，它被用作标注方法。它说明了被标注的方法重载了父类的方法，起到了断言的作用。

【例9.9】 演示当窗口图标化或取消图标化，文本框中文字的变化

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class TestWindowsEvent extends JFrame{
    static JTextField textfield1;
    private Font f=new Font("sanserif",Font.PLAIN,30);
    public TestWindowsEvent(){
        setTitle("测试window事件处理");
        setSize(300,300);
        textfield1=new JTextField(50);
        textfield1.setLocation(200, 100);
        textfield1.setText("我将变化");
        textfield1.setFont(f);
        add(textfield1);
    }
}
```


【例9.9】 演示当窗口图标化或取消图标化，文本框中文字的变化

```
public static void main(String args[]){  
    TestWindowsEvent testwindow1=  
        new TestWindowsEvent();  
    testwindow1.setVisible(true);  
    testwindow1.addWindowListener(new  
        WindowsEventListener() );  
}  
}  
class WindowsEventListener implements WindowListener{  
    @Override  
    public void windowActivated(WindowEvent e) {  
        // TODO Auto-generated method stub  
    }  
}
```

【例9.9】 演示当窗口图标化或取消图标化，文本框中文字的变化

@Override

```
public void windowDeactivated(WindowEvent e) {  
    // TODO Auto-generated method stub  
}
```

@Override

```
public void windowClosed(WindowEvent e) {  
}
```

@Override

```
public void windowClosing(WindowEvent e) {  
    // TODO Auto-generated method stub  
}
```

```
public void windowDeiconified(WindowEvent e) {  
    // TODO Auto-generated method stub  
    TestWindowsEvent.textfield1.setText("我回来了");  
}
```

【例9.9】 演示当窗口图标化或取消图标化，文本框中文字的变化

@Override

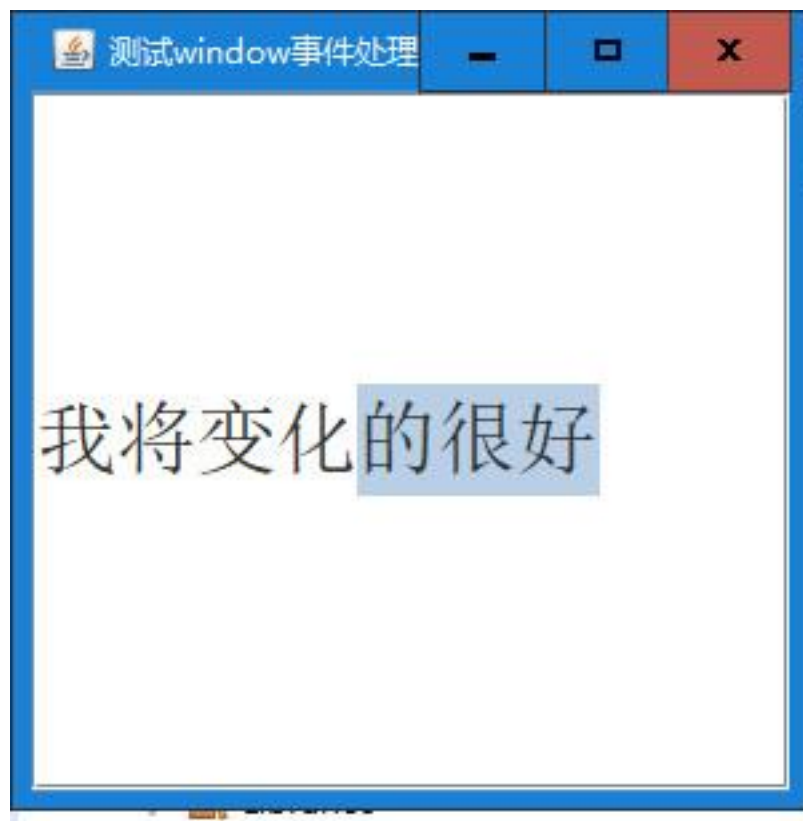
```
public void windowIconified(WindowEvent e) {  
    // TODO Auto-generated method stub  
}
```

@Override

```
public void windowOpened(WindowEvent e) {  
    // TODO Auto-generated method stub  
}
```

```
}
```

9.3.2 Windows 事件处理



9.3.3 键盘事件处理

- 键盘事件可以利用键来控制 and 执行一些操作，或从键盘上获取输入，Java提供KeyListener来处理键盘事件，KeyEvent对象描述事件的特性。

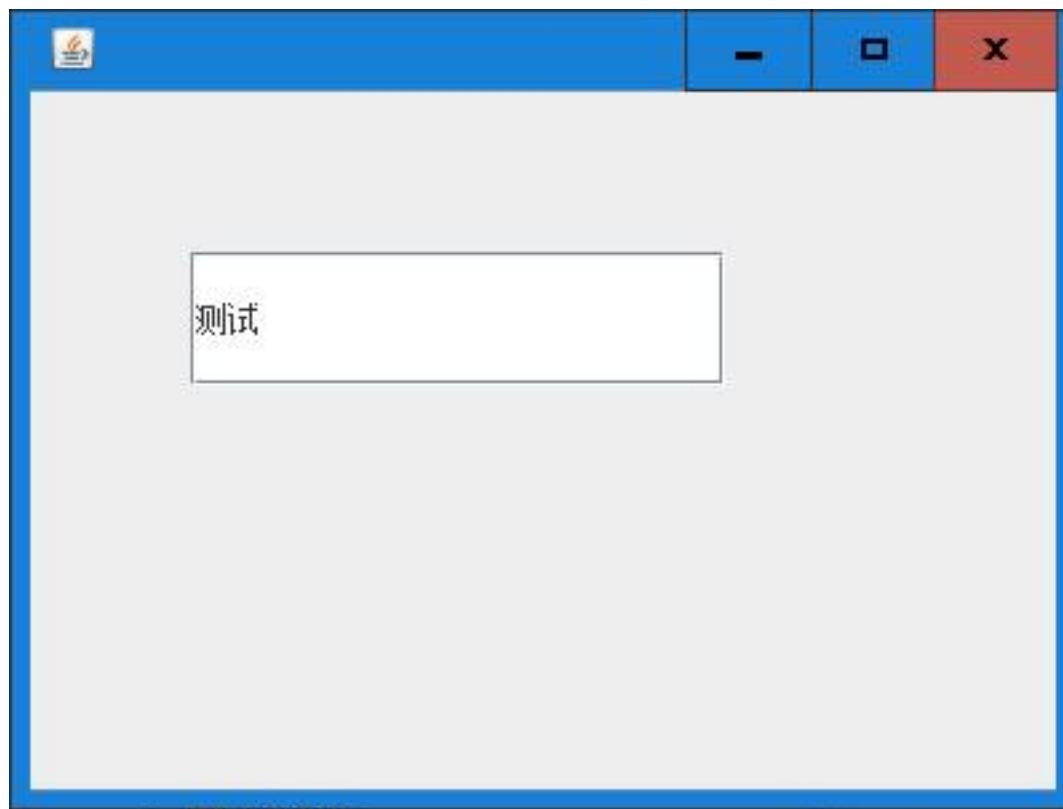
9.3.3 键盘事件处理

```
public class TestKeyEvent extends JFrame {  
    private JTextField textField;  
    Font f=new Font('sanserif',Font.PLAIN,20);  
    public TestKeyEvent() {  
        super();  
        final JPanel panel = new JPanel();  
        panel.setLayout(null);  
        add(panel, BorderLayout.CENTER);  
        textField = new JTextField();  
        textField.setBounds(60, 60, 200, 50);  
        panel.add(textField);  
        this.setSize(400,300);  
        this.setVisible(true);  
    }  
}
```

9.3.3 键盘事件处理

```
textField.addKeyListener(new KeyAdapter() {  
    public void keyPressed(KeyEvent e) {  
        if (e.getKeyCode()==KeyEvent.VK_ENTER){  
            textField.setFont(f);  
        }  
    }  
});  
}  
public static void main(String[] args) {  
    new TestKeyEvent();  
}  
}
```

9.3.3 键盘事件处理



9.3.4 鼠标事件处理

- **MouseEvent**类用于描述鼠标事件。鼠标事件主要指鼠标按下、释放、点击、移动或拖动时产生的事件，Java提供了两个监听器接口**MouseListener**和**MouseMotionListener**，用于实于监听鼠标事件。

9.3.4 鼠标事件处理

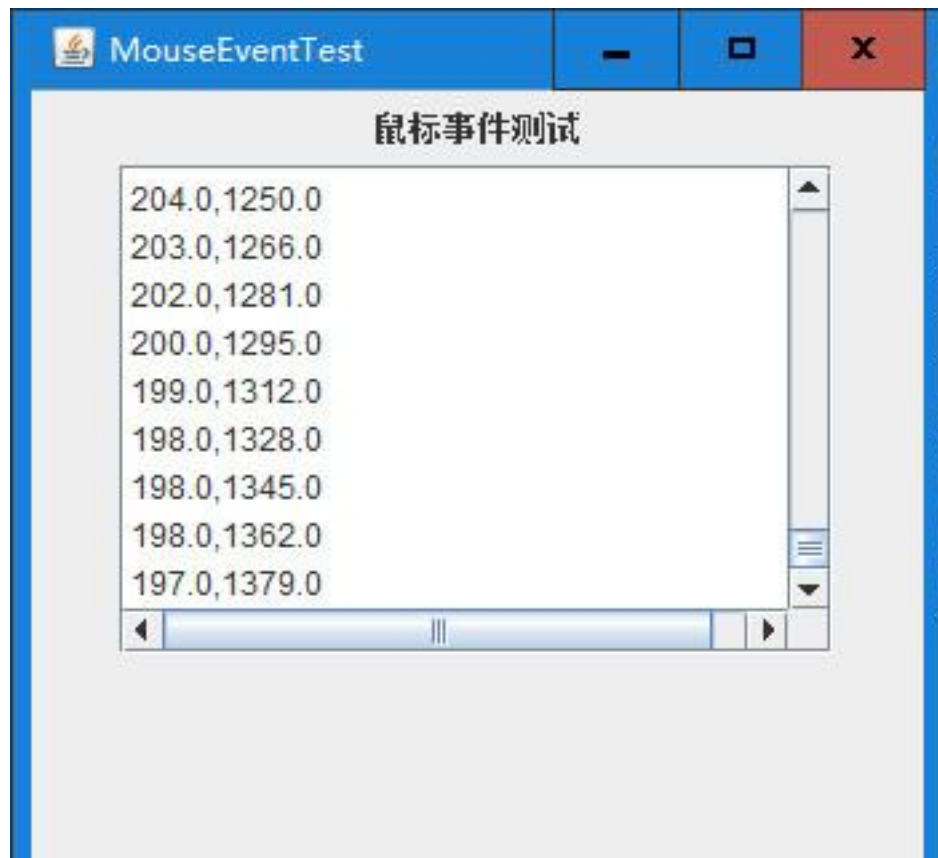
```
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
public class TestMouseEvent extends JFrame implements
MouseMotionListener {
    JTextArea jtextarea01;
    JScrollPane jscrollpanet01;
    JLabel lable01;
    TestMouseEvent() {
        setTitle('MouseEventTest');
        setLayout(new FlowLayout());
        lable01=new JLabel('鼠标事件测试');
        this.add( lable01);
        jtextarea01=new JTextArea(10,24);
        jscrollpanet01 = new JScrollPane(jtextarea01);
        this.add( jtextarea01);
    }
}
```

帶有滚动条的面板

9.3.4 鼠标事件处理

```
this.add(jscrollpanet01);
this.setSize(350, 2100);
this.setVisible(true);
jtextarea01.addMouseMotionListener(this);
}
public void mouseDragged(MouseEvent e){
}
public void mouseMoved(MouseEvent e){
    float x=e.getX();
    float y=e.getY();
    jtextarea01.setText(jtextarea01.getText()+"\n "+x+", "+y);
}
public static void main(String[] args) {
    new TestMouseEvent();
}
}
```

9.3.4 鼠标事件处理



9.4菜单

一个完整的菜单系统由菜单条、菜单和菜单项组成，Java中与菜单相关的类主要有3个MenuBar、Menu、MenuItem。

■ 9.4.1 菜单的设计与实现

//创建一个菜单栏对象jmb

```
JMenuBar jmb = new JMenuBar();
```

//将菜单栏与框架关联

```
Frame.setJMenuBar(jmb);
```

(2) 创建菜单，然后把菜单添加到菜单栏上。

```
JMenu fileMenu = new JMenu("File"); //创建菜单
JMenu editMenu = new JMenu("Edit");
JMenu helpMenu = new JMenu("Help");
jmb.add(fileMenu); //将菜单加到菜单栏
jmb.add(editMenu);
jmb.add(helpMenu);
```

(3) 创建菜单项并把它们添加到菜单上，如下面代码所示：

//创建一个菜单项并添加到菜单上

```
fileMenu.add(new JMenuItem("New"));
```

```
fileMenu.add (new JMenuItem ("Open"));
```

(4) 创建子菜单项。

```
JMenuItem jmiNew = new JMenuItem("New")
```

```
JMenuItem jmiOpen = new JMenuItem("Open")
```

```
fileMenu.add(jmiNew);
```

```
fileMenu.add(jmiOpen);
```

(4) 创建子菜单项。

```
JMenuItem jmiNew = new JMenuItem("New");  
JMenuItem jmiOpen = new JMenuItem("Open");  
fileMenu.add(jmiNew);  
fileMenu.add(jmiOpen);
```

(5) 创建复选框菜单项。使用JCheckBoxMenuItem类：

```
editMenu.add (new JCheckBoxMenuItem("Font"));
```


(6) 创建单选按钮菜单项。使用JRadioButtonMenuItem类

```
JMenu colorMenu = new JMenu("Color");
```

```
editMenu.add(colorMenu);
```

```
JRadioButtonMenuItem jmbRed = new JRadioButtonMenuItem("Red");
```

```
JRadioButtonMenuItem jmbGreen = new JRadioButtonMenuItem("Green");
```

```
JRadioButtonMenuItem jmbBlue = new JRadioButtonMenuItem("Blue");
```

```
ButtonGroup colorgroup = new ButtonGroup();
```

```
colorgroup.add(jmbRed);
```

```
colorgroup.add(jmbGreen);
```

```
colorgroup.add(jmbBlue);
```

```
colorMenu.add(jmbRed);
```

```
colorMenu.add(jmbGreen);
```

```
colorMenu.add(jmbBlue);
```

(7) 为菜单设置图标，使用setIcon()方法，例如：

```
jmiNew.setIcon(new ImageIcon("image/new.gif");  
jmiOpen.setIcon(new ImageIcon("image/open.gif");
```

(8) 为菜单或者菜单项设置热键快捷键，使用setAccelerator()方法

```
jmiOpen.setAccelerator(KeyStroke.getKeyStroke('O',  
java.awt.Event.CTRL_MASK);
```

9.4.2 实现菜单项事件处理代码

当菜单项被选中时，会触发ActionEvent事件，因此要处理该事件，程序必须实现 ActionListener接口。例如：

```
public class MenuActionTest implements ActionListener{  
    public void actionPerformed(ActionEvent e){  
        String m = e.getActionCommand();  
        if(m.equals("exit")){  
            do  
                .....  
        }  
    }  
}
```