

Java软件设计基础





第9章 图形用户界面设计

1.1 GUI概述

- **GUI: Graphics User Interface**

- 设计和实现GUI的主要难点

- 创建组成界面的各成分和元素，指定它们的属性和位置关系，根据具体需要布局排列，从而构成完整的GUI的外观表象；
 - 在GUI程序中，代码所执行的路线及顺序都是很不确定的。在GUI程序中，用户可以点击任意按键并和屏幕上的其他任何部分进行交互及信息传递——即事件驱动。

- AWT(Abstract Window Tookit)包

- 当程序运行时，将这些组件的创建和动作委托给程序所在的运行平台，即JVM调用操作系统本地的图形界面来创建和平台一致的对等体(peer)。

- Swing包

- 由Sun和Netscape合作完善，组件是绘制在空白窗口上的，更少的依赖目标机器上的底层平台，是对AWT图形用户界面的极好补充和加强。

◆ 广义组件 (Component)

- 表示一个能以图形化方式显示出来，并可以与用户交互的元素。
- 常用方法
 - 组件背景色的设置方法：**public void setBackground(Color c)**
 - 组件前景色的设置方法：**public void setForeground(Color c)**
 - 组件背景色的获取方法：**public Color getBackground()**
 - 组件前景色的获取方法：**public Color getForeground()**
 - 组件字体的设置方法：**public void setFont(Font f)**
 - 组件字体的获取方法：**public void getFont()**
 - 组件边框的设置方法：**public void setBorder(Border b)**
 - 组件边框的获取方法：**public Border getBorder()**
 - 默认情况下组件的边框是黑边矩形。

- 设置组件的透明性：**public void setOpaque(boolean isOpa)**
 - isOpa取false时组件为透明， 取true时被设置为不透明；
- 获取组件的透明性：**public boolean getOpaque()**
 - 组件透明时返回值为false， 否则返回值为true；
- 设置组件的大小， width指定组件的宽度， height指定组件的高度。
public void setSize(int width, int height)
- 设置组件在容器中的位置。
public void setLocation(int x, int y)
- 返回一个Dimension对象下组件的宽度和高度值。
public Dimension getSize()
- 返回一个含有成员变量x和y的Point对象的引用， x和y就是组件在容器中的坐标。
public Point getLocation()

- 设置组件在容器中的位置和组件的大小。

public void setBounds(int x, int y, int width, int height)

- 返回一个含有成员变量x、y、width和height的Rectangle对象的引用。

public Rectangle getBounds()

- 设置组件是否可被激活，当b为true时组件可被激活，否则组件不可激活。

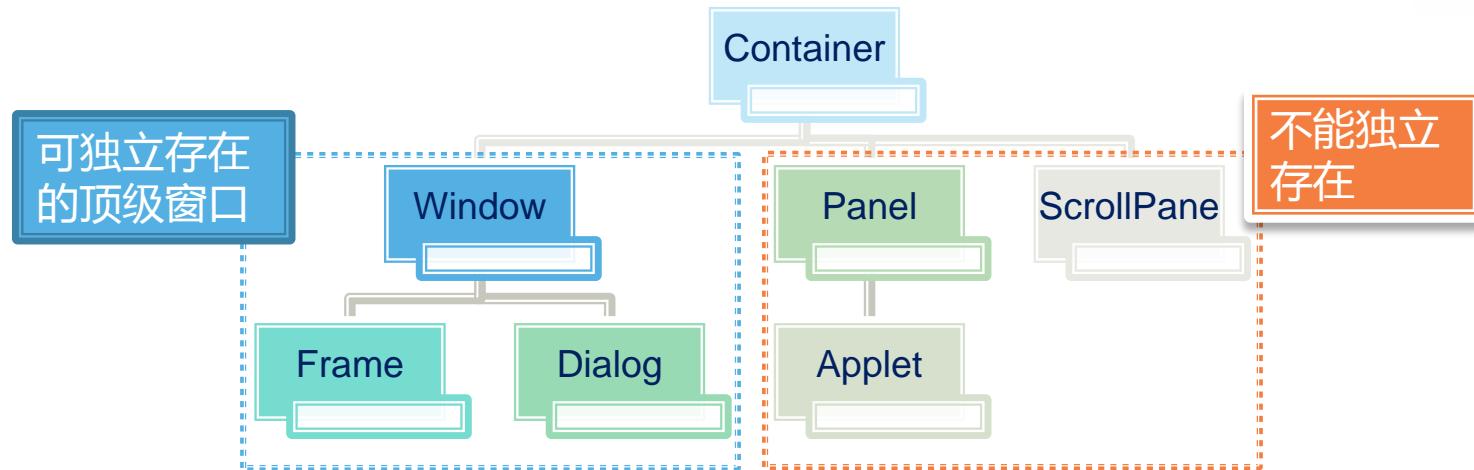
public void setEnable(boolean b)

- 设置组件在该容器中的可见性，当b为true时组件可见，否则组件不可见。

public void setVisible(boolean b)

◆ 容器(Container)

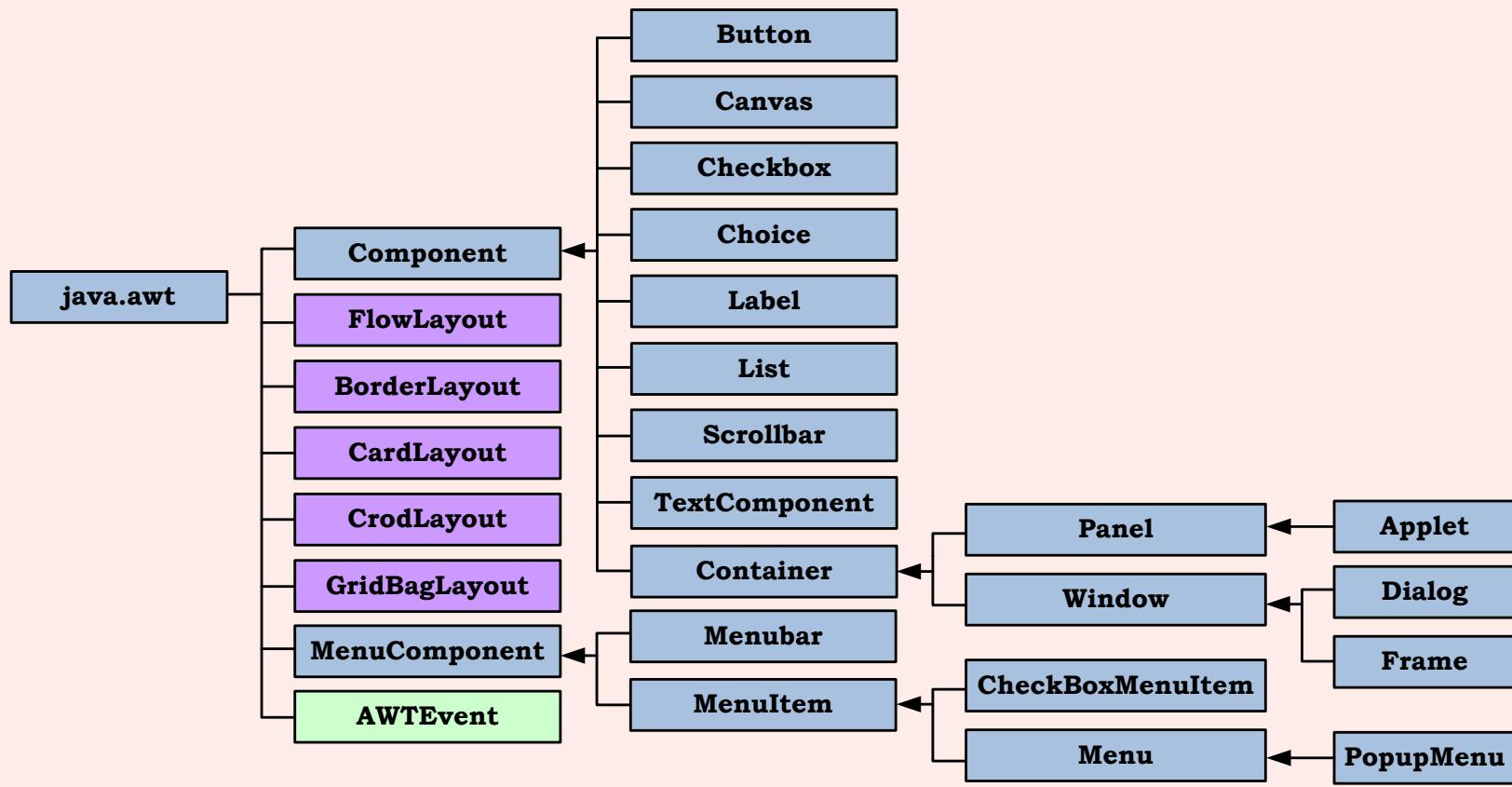
- 是Component（组件，广义）的子类，因此容器对象本身也是组件，具有组件所有的性质，可以调用Component类的所有方法。
- 容器可以盛放其他组件，提供了以下方法：
 - 向容器中添加组件 **add(Component comp)**
 - 返回指定点的组件 **Component getComponentAt(int x,int y)**
 - 返回该容器内的组件数量 **int getComponentCount()**
 - 返回该容器的所有组件 **Component[] getComponents()**



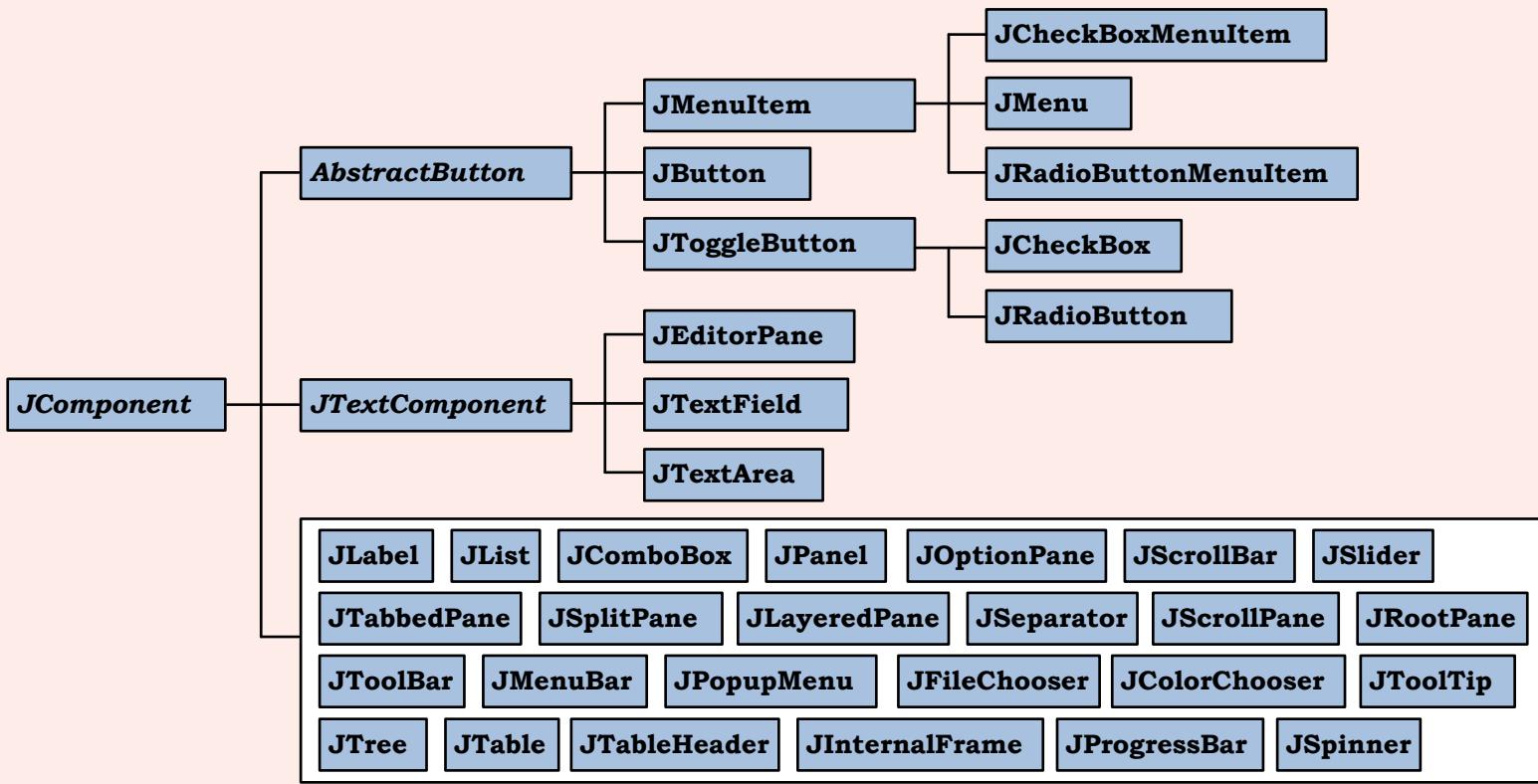
顶层容器		中间容器
AWT	Window、Frame、Dialog	Panel、ScrollPane、Applet
swing	JFrame、JApplet、JDialog、JWindow	JPanel、JScrollPane、JSplitPane、JToolBar等

◆ 狹义组件：除了容器之外的组件，包括菜单相关的组件、普通组件等。

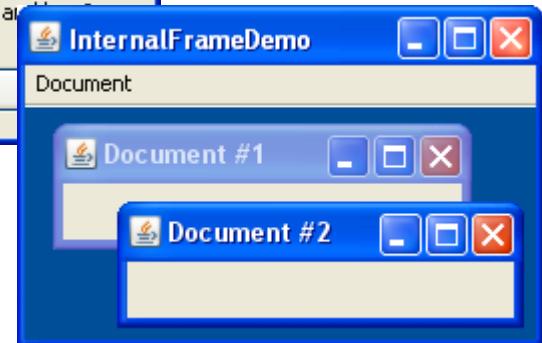
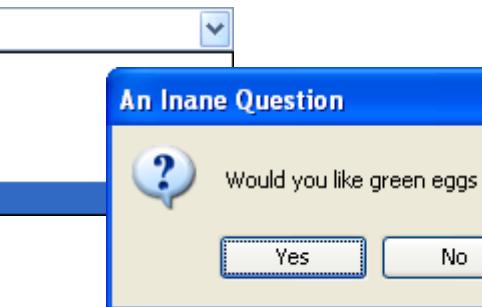
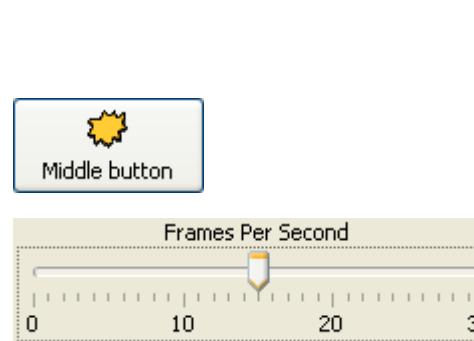
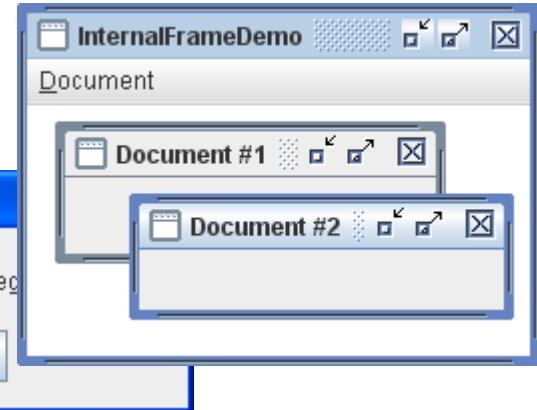
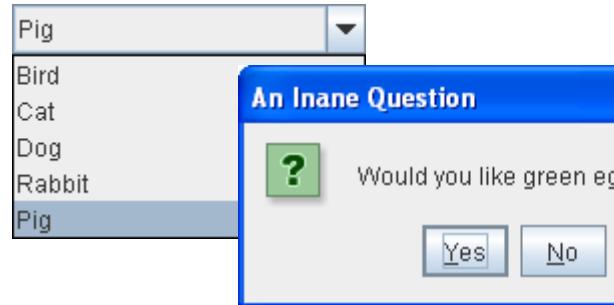
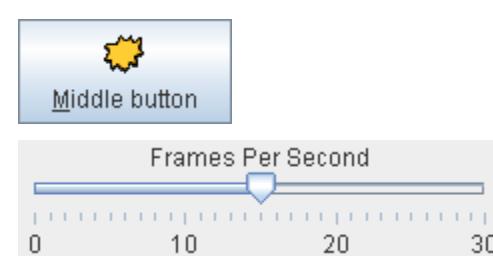
AWT



swing



- Swing外观与Windows外观



不依赖本地GUI的Swing组件称为轻组件——**Lightweight component**;
AWT组件称为重组件——**Heavyweight component**。

◆一些常用辅助类

- 辅助类都不是Component的子类，它们用来描述GUI组件的属性，例如颜色、字体、大小等。
 - Graphics类是一个抽象类，提供一个图形环境，用于绘制字符串、直线和简单几何图形；
 - Color类用来处理GUI组件的颜色；
 - Font类指定GUI组件上文本和图形的字体；
 - FontMetrics是用于获取字体属性的抽象类；
 - Dimension将组件的宽度和高度（精度为整数）封装在单个对象中；
 - LayoutManager是一个接口，指定组件在容器中的摆放方式。

◆GUI设计的简要流程-外观设计部分

- 引入java.awt包/javax.swing包/java.awt.event包；
- 利用包中提供的组件类来定义某种组件；
- 定义放置组件的容器；
- 对容器进行布局管理，将此组件添加到该容器中；
- 设置组件的外观；

1.2 常用组件与使用

◆Frame/JFrame

- 顶级窗口，该组件在Java的顶层窗口中可以独立使用。
- 包含标题、边框等，创建时不可见，需要通过setVisible(true)方法显示。
- 当Frame/JFrame被关闭后，将产生WindowEvent事件；无法直接监听键盘输入事件。
- 默认使用BorderLayout布局。
- 构造方法：

- 建立一个没有标题的新Frame对象：

Frame();

Jframe();

- 建立一个以title为标题的Frame对象：

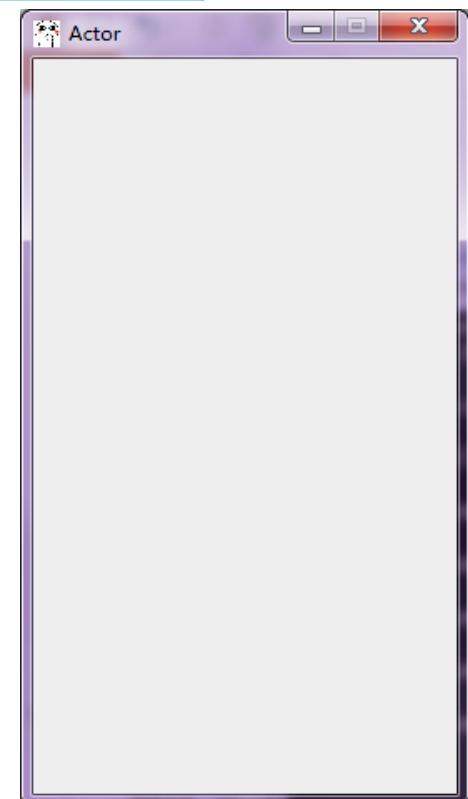
Frame(String title);

JFrame(String title);

- 普通方法：

String getTitle()	获取标题
setTitle(String title)	设置标题为title
boolean isResizable()	测试是否可以改变大小
Image getIconImage()	获取最小化图标
setIconImage(Image img)	设置最小化图标

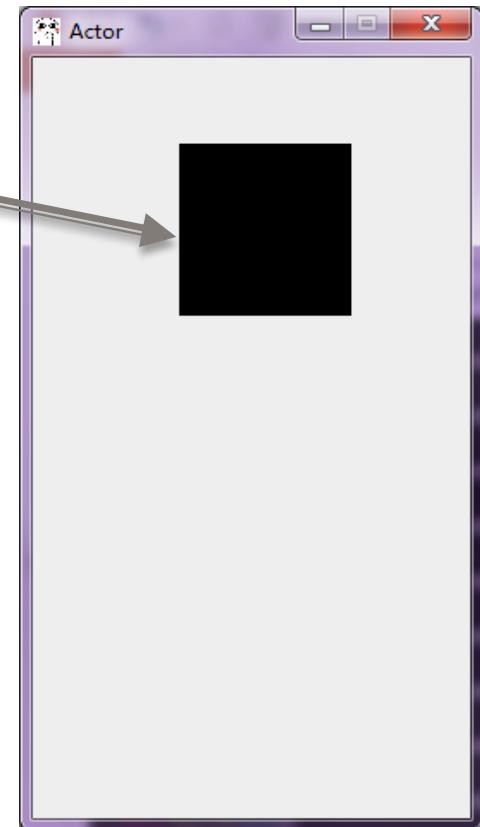
```
import javax.swing.*;
class MainFrame extends JFrame{
    static ImageIcon icon=new ImageIcon("icon.png");
    public MainFrame(String title){
        super(title);
        setBounds(0,0,270,480);
        setResizable(false);
        setIconImage(icon.getImage());
        setVisible(true);
    }
}
public class Actor{
    public static void main(String[] args){
        new MainFrame("Actor");
    }
}
```



◆ Panel/JPanel:

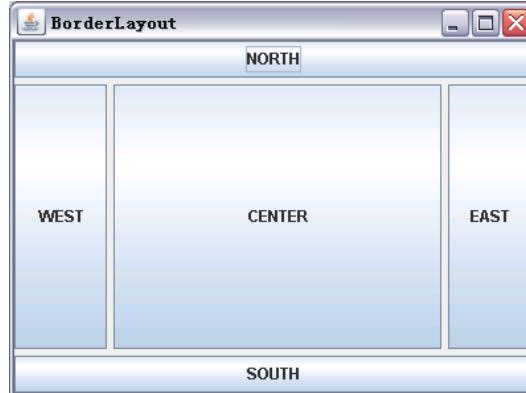
- 面板不是顶级窗口，不能独立存在，必须将它添加到其他容器中。其外在表现为一个矩形区域，该区域内可以盛装其他组件。
- 默认布局方式是FlowLayout。

```
static JPanel p=new JPanel();
{
    p.setBounds(85,50,100,100);
    p.setBackground(Color.black);
    setLayout(null);
    add(p);
}
```

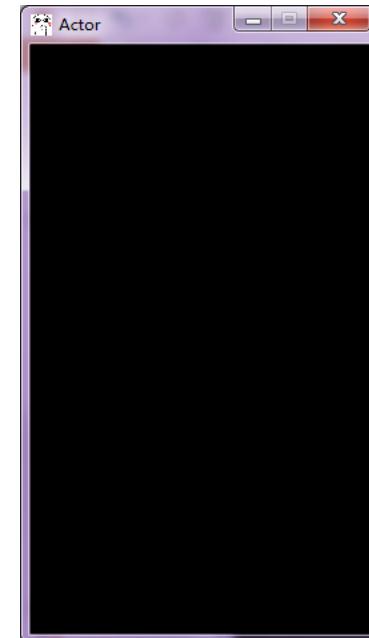


扩展 : BorderLayout

- BorderLayout也称为边界布局管理器，是Window、Frame和Dialog默认的布局管理策略。
- 将窗口分为North/East/South/West/Center五个区域，中间的区域最大。南北组件可以水平扩展，东西组件可以垂直拉伸；每加入一个组件都应该指明添加到哪个区域，若不指明则默认添加至Center区域。
- 如果上例中注释掉“setLayout(null);”则panel会按照边界布局方式加入中心区域，并占据全部窗口。



五个按钮依次加入BorderLayout布局窗口中的五个方位



◆ScrollPane (带滚动条的容器)

- 不能独立存在，也是可用于盛装组件的容器；
 - 盛装组件占用空间过大的时候自动产生滚动条，也可通过指定特定的构造方法参数来指定默认具有滚动条；
 - 默认使用BorderLayout布局，因它通常用于盛装其他容器，所以通常不允许改变其布局。
- 将上例中的JPanel改为ScrollPane

```
static ScrollPane p=new ScrollPane(ScrollPane.SCROLLBARS_ALWAYS);
```



◆Label/JLabel——标签

- 是一种用来显示说明性的静态文本的组件，它起到信息说明的作用。是用户只能查看而不能简单的修改其内容的文本显示区域，但用户可以在应用程序中通过调用Label提供的方法更换文本的内容。
- 构造方法

- 创建一个没有名字的标签对象

Label();

JLabel();

- 创建一个名字为str的标签对象

Label(String str);

JLabel(String str);

- 创建一个名字为str的标签对象，对齐方式为align

Label(String str, int align);

JLabel(String str,int align);

- 其中Label.LEFT、Label.RIGHT、Label.CENTER分别为居左、居右、居中。
- 创建带图标的标签对象

JLabel(Icon ic);

◆Button/JButton——按钮

- 是Java程序GUI设计中最常用的一个组件，是可以提供用户快速启动某一动作的类。按钮本身不显示信息，它一般对应一个事先定义好的功能操作，并对应一段程序。当用户点击按钮时，系统自动执行与该按钮相联系的程序，从而完成预先指定的功能。

- 构造方法

- 构造一个没有标题的按钮：

```
Button();
JButton();
```

- 构造一个标题为str的按钮：

```
Button(String str);
JButton(String str);
```

- 构造有图标的按钮：

```
JButton(Icon ic);
```

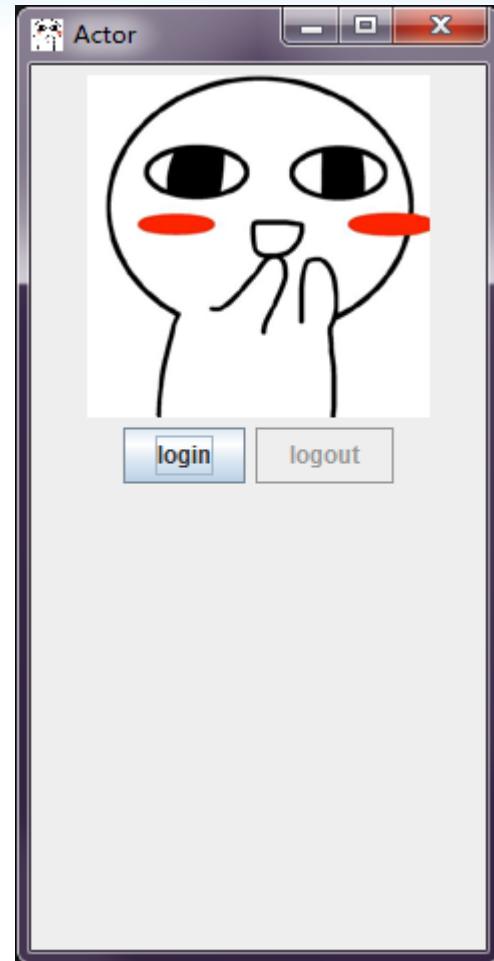
- 构造有图标有标题的按钮：

```
JButton(String str,Icon ic);
```

```
JLabel jl=new JLabel(icon);
JButton login=new JButton("login");
JButton logout=new JButton("logout");
p.add(jl);
p.add(login);
p.add(logout);
logout.setEnabled(false);
```

扩展：*FlowLayout*

将容器的组件按照加入的先后顺序从左至右依次排列，一行排满后就转到下一行继续从左至右顺序排列，每一行中的组件都居中排列，组件间默认的水平和垂直间隙是五个像素。
FlowLayout是Panel、Applet的默认布局方式。



◆ TextField/JTextField/JPasswordField (文本框)

- TextField/JTextField用于单行文本输入处理； JPasswordField是swing中提供输入密码的组件；
- 当按下回车键时，会发生事件(ActionEvent)，可以通过ActionListener中的actionPerformed方法对事件进行处理。
- 构造方法

- 创建一个初始文本串为空的文本框对象

TextField();

JTextField();

- 创建一个初始文本串为str的文本框对象

TextField(String str);

JTextField(String str);

- 创建一个初始文本串为str、列数为column长度的文本框对象

TextField(String str, int column);

JTextField(String str, int column);

- 普通方法

setText(String s)	将文本框对象的文本信息设置为str
getText()	返回文本框对象的文本信息
getSelectedText()	返回文本框对象被选的文本串信息内容
setEchoChar(char ch)	将文本框对象的回显字符设为ch, 常用于密码接收场合

```
JTextField username=new JTextField(10);
JPasswordField pw=new JPasswordField(10);
pw.setEchoChar('*');
p.add(new JLabel("User Name:"));
p.add(username);
p.add(new JLabel("Password:"));
p.add(pw);
```



◆ 复选框

- 复选框Checkbox/JCheckbox由一个方形的选择区和一个标签组成，有选中(check)和未选中(uncheck)两种状态。
- 构造方法：创建一个标签为str的复选框

Checkbox(String str);
JCheckbox(String str);

- 普通方法：

getState()	返回Checkbox对象的状态是否被选中
setState(boolean state)	设置Checkbox对象的当前状态

◆ 单选框

- awt组件中，多个Checkbox实例加入到一个CheckboxGroup组件内时，选择区变成圆形，组件内的Checkbox只能有一个被选中，此时实现的是单选按钮的功能。

Checkbox(String str, CheckboxGroup g, boolean state)

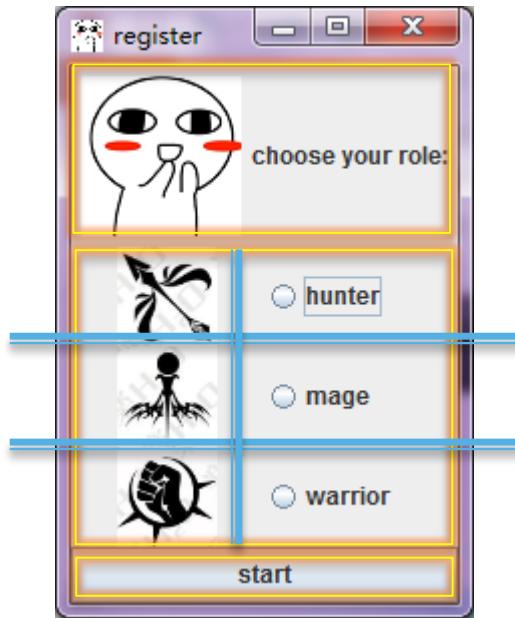
- swing组件：构造多个JRadioButton实例，加入到ButtonGroup中，既可实现单选按钮。

```
JRadioButton[] jb;
String[] role={"hunter","mage","warrior"};
p.add(new JLabel("choose your role"));
jb=new JRadioButton[3];
ButtonGroup bg=new ButtonGroup();
JLabel[] roleicon=new JLabel[3];
for(int i=0;i<3;i++){
    roleicon[i]=new JLabel(new ImageIcon(role[i]+".png"));
    jb[i]=new JRadioButton(role[i]);
    bg.add(jb[i]);
    p.add(roleicon[i]);
    p.add(jb[i]);
}
```



扩展 : GridLayout

- 在一个二维网格中布置组件。将容器划分为若干行若干列的网格区域，每个网格的大小相等，组件就位于这些划分出来的小格中，一个网格可以放置一个组件。组件可被依次的放在第一行第一列、第一行第二列……的顺序进行。
- 通过在一个网格中放置容器，可以实现嵌套。
- GridLayout也称为网格布局管理器，该布局比较灵活，划分多少网格由程序自由控制，组件定位也比较精确。



```
static JPanel actorPane=new JPanel();
static JPanel choosePane=new JPanel();
JButton start=new JButton("start");
add(actorPane, BorderLayout.NORTH);
add(choosePane, BorderLayout.CENTER);
add(start, BorderLayout.SOUTH);
choosePane.setLayout(new GridLayout(3,2));
pack();
```

◆Choice/JComboBox

- 提供一个弹出式的下拉列表让用户选择，也是“多选一”的输入界面。下拉列表的所有选项被折叠起来，在这个列表折叠时只显示最前面的或是用户选定的那一项。下拉列表节省显示空间，适用于大量选项的情形。由ItemListener接口来监听。
- 普通方法

addItem(String item)	添加一个item字符串到Choice对象中
add(String item)	将新选项item加载当前下拉列表的最后
getSelectedIndex()	返回被选中的选项的序号(0~n-1)
getItem(int index)	返回Choice对象index索引项目的字符串
getSelectedItem()	返回Choice对象所选项目的字符串
removeAll()	把下拉列表中的选项全部移除
remove(String item)	把指定标签文本的选项移除

◆List/JList

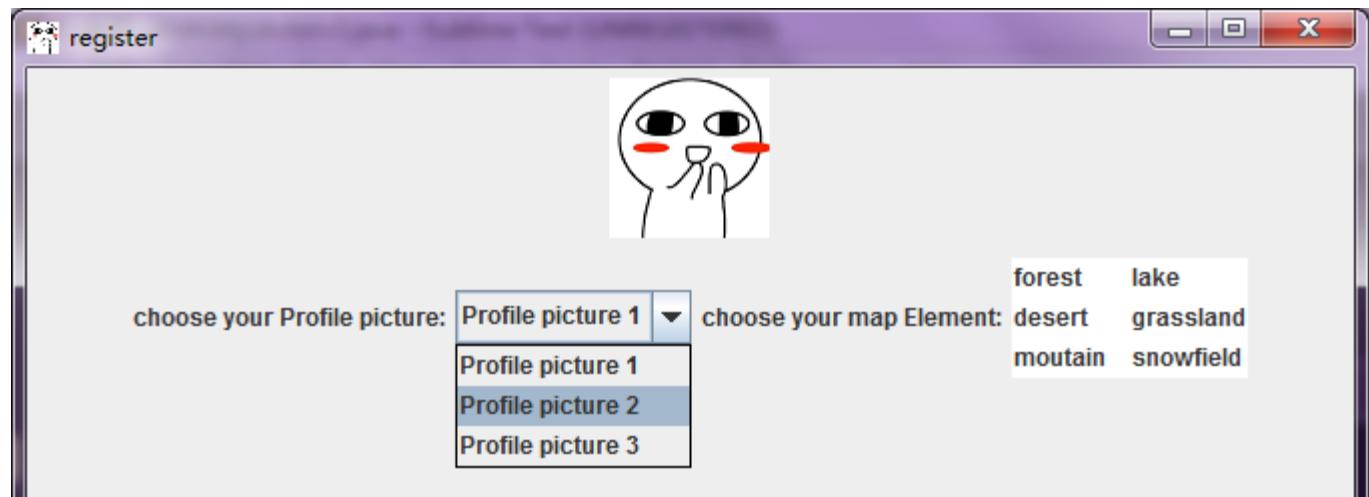
- 把所有供用户选择的项都显示出来以供选择，可以实现多选，支持滚动条，可以同时浏览多项。
- 列表可以产生两种事件：
 - 当用户单击列表中的某个选项并选中它时，将产生ItemEvent类的选择事件；
 - 当用户双击列表中的某一个选项时，将产生ActionEvent类的动作事件。
- 普通方法

getSelectedItem()	返回List对象所选项目的字符串组成的字符串数组
getSelectedIndex()	返回被选中的选项的序号(0~n-1)组成的整数数组
addItem(String item, int index)	添加一个item字符串到List对象的索引index处
delItem(int index)	删除List对象index处的选项
add(String item)	将新选项item加载当前列表的最后
remove(String item)	把指定标签文本的选项移除

```
String[] pp={"Profile picture 1","Profile picture 2","Profile picture 3"};
String[] mapElement={"forest","desert","moutain","lake","grassland","snowfield"};

JComboBox jcb=new JComboBox();
for(int i=0;i<3;i++)
    jcb.addItem(pp[i]);

JList jme=new JList();
jme.setListData(mapElement);
jme.setSelectionMode(ListSelectionMode.MULTIPLE_INTERVAL_SELECTION);
jme.setVisibleRowCount(3);
jme.setLayoutOrientation(JList.VERTICAL_WRAP);
```



◆TextArea/JTextArea（文本域）

- 多行文本组件，用于显示程序中的多行文本信息。
- 构造方法

- 创建一个初始文本串为str的文本域对象

```
TextArea(String str);  
JTextArea(String str);
```

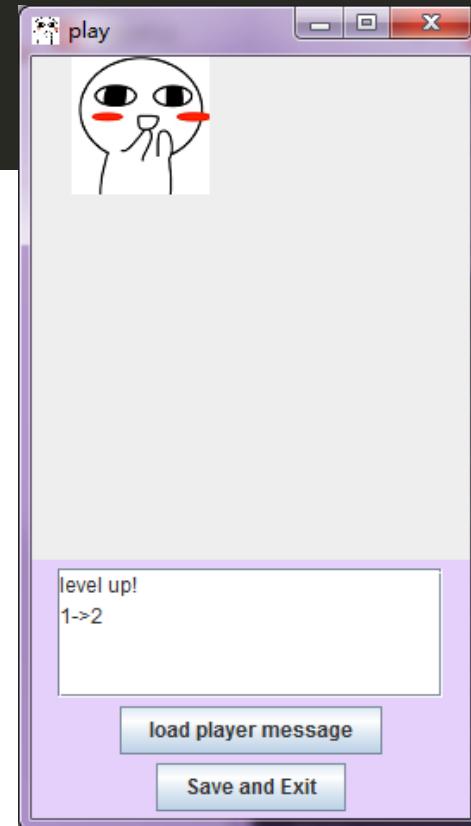
- 创建numLines行、 numChars列的文本域对象

```
TextArea(int numLines, int numChars);  
JTextArea(int numLines, int numChars);
```

- 创建一个初始文本串为str、 numLines行、 numChars列的文本域对象

```
TextArea(String str, int numLines, int numChars);  
JTextArea(String str, int numLines, int numChars);
```

```
recordPane.setPreferredSize(new Dimension(250,150));
recordPane.setBackground(new Color(229,208,252));
recordPane.setLayout(new FlowLayout());
JTextArea record=new JTextArea(4,20);
JButton load=new JButton("load player message");
JButton save=new JButton("Save and Exit");
JScrollPane jtp=new JScrollPane(record);
jtp.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
recordPane.add(jtp);
recordPane.add(load);
recordPane.add(save);
add(recordPane, BorderLayout.SOUTH);
```



◆ Dialog/JDialog (对话框)

- 对话框与Frame/JFrame类似，是有边框、有标题而独立存在的容器。通常起到与用户交互的对话窗口的作用。但它不能作为程序的最外层容器，必须隶属于某个窗口并由该窗口负责弹出。
- 对话框可以被设置为模式窗口，其特点是总是在激活窗口的最前面，即若是不关闭，则不能对其隶属的窗口进行操作。
- 构造方法

- 创建没有标题的对话框，Owner表示所有者

Dialog(Frame Owner);
JDialog(Frame Owner);

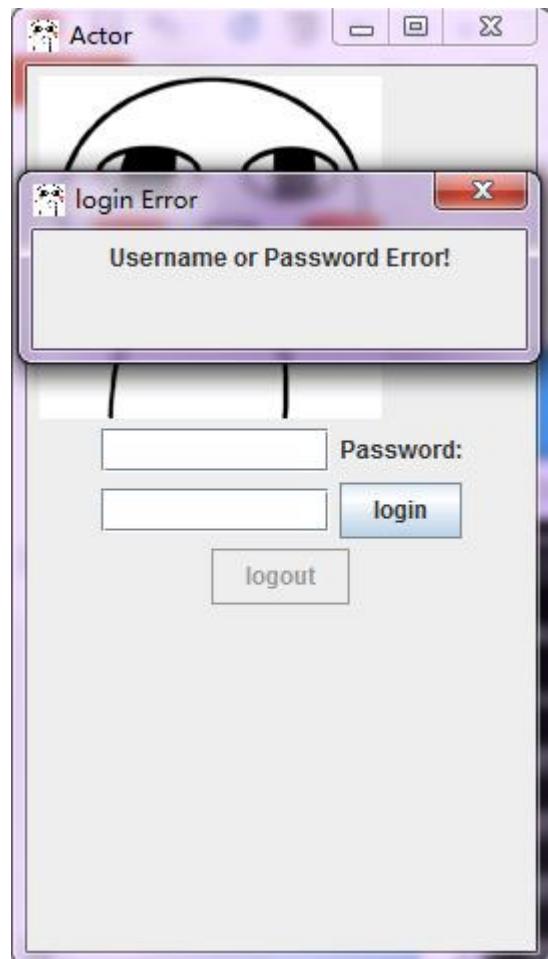
- 创建有指定标题title的对话框

Dialog(Frame Owner, String title);
JDialog(Frame Owner, String title);

- 创建指定标题的对话框并指明是否为模式窗口

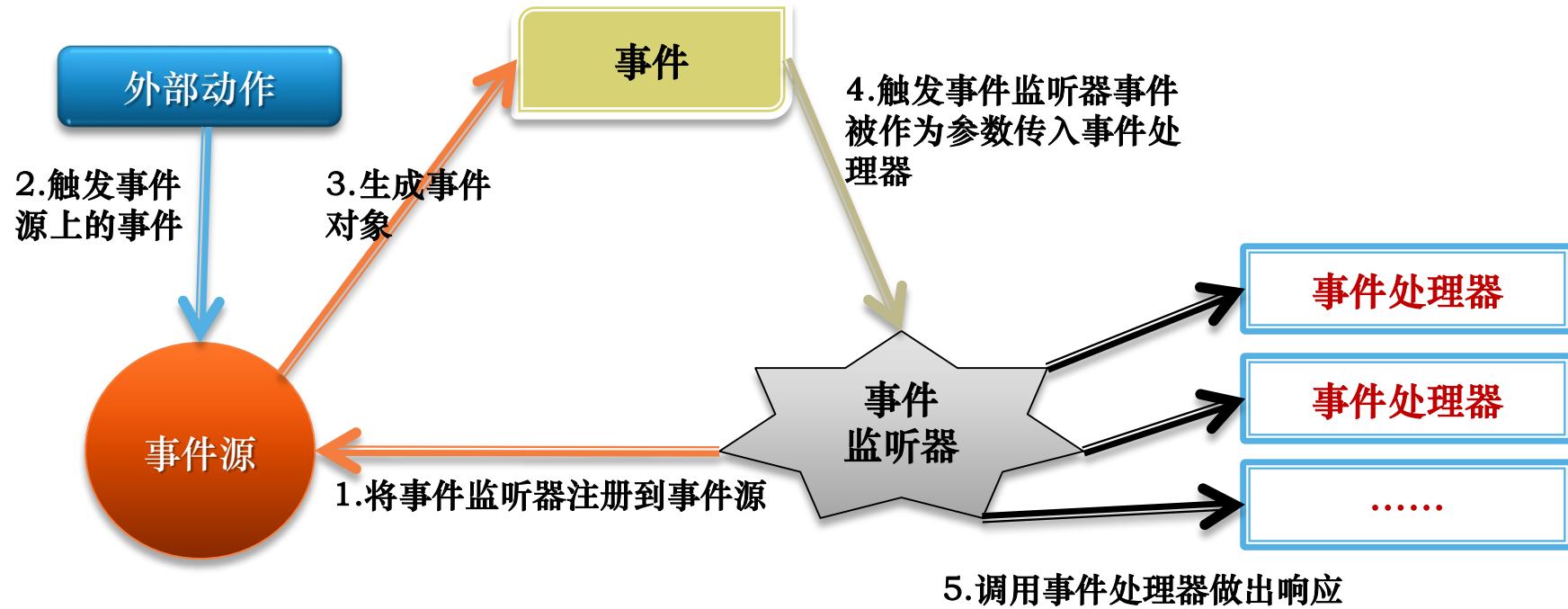
Dialog(Frame Owner, String title, boolean modal);
JDialog(Frame Owner, String title, boolean modal);

```
JDialog jd=new JDialog(this,"login Error",true);
jd.setBounds(100,100,200,60);
jd.setLayout(new FlowLayout());
jd.add(new JLabel("Username or Password Error!"));
jd.setVisible(true);
```



3. 事件处理机制

◆ AWT事件处理的流程

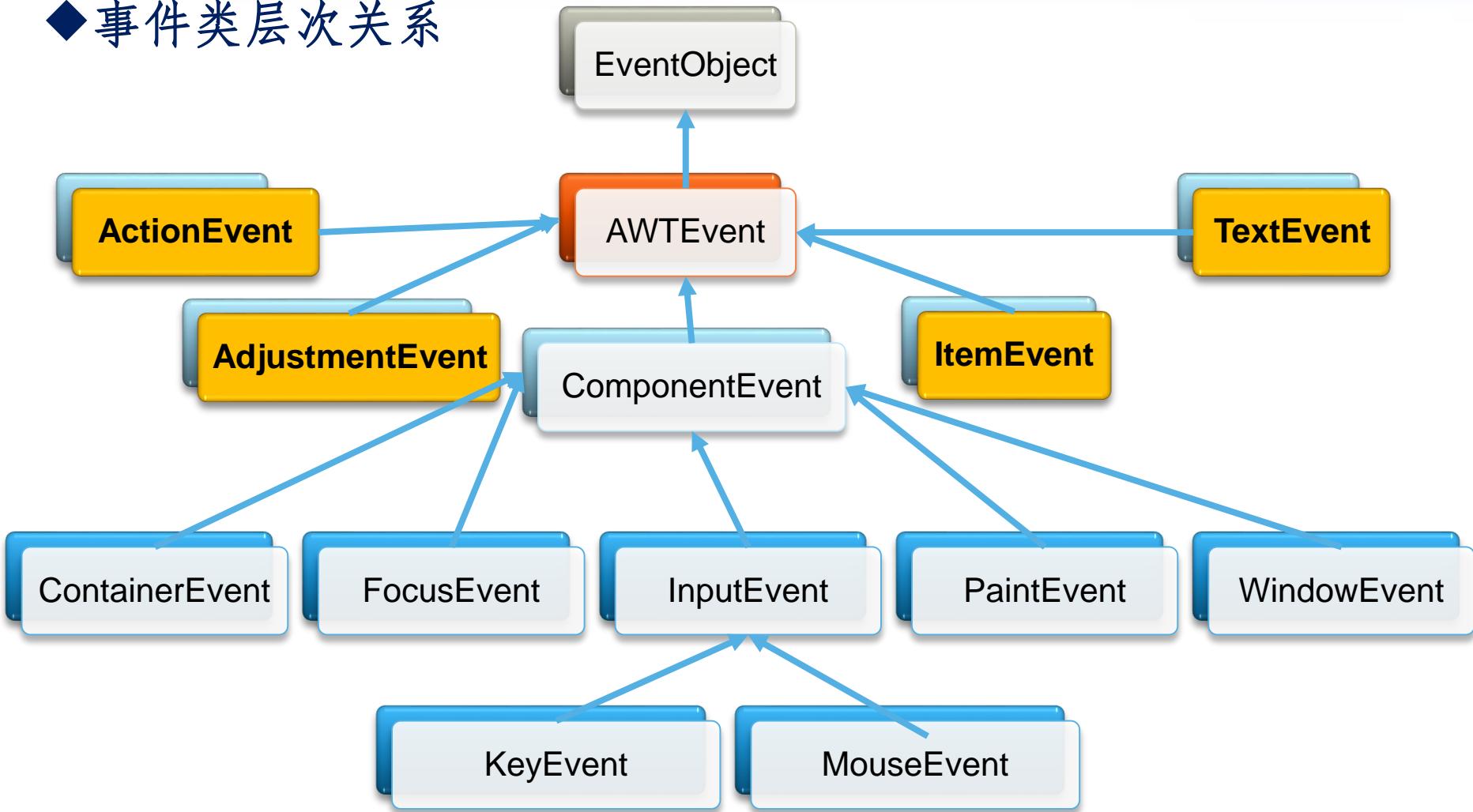


以上例为例，注册事件监听器**ActionListener**到两个按钮上，一旦在事件源b1/b2产生外部动作(用户点击)，则触发事件，生成**ActionEvent**事件对象，并作为参数传入事件处理器**actionPerformed**进行处理

◆修改上一个例子

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class MainFrame extends JFrame implements ActionListener{
    static ImageIcon icon=new ImageIcon("icon.png");
    static JPanel p=new JPanel();
    public MainFrame(String title){
        .....
        login.addActionListener(this);
        .....
    }
    public void actionPerformed(ActionEvent e){
        JDialog jd=new JDialog(this,"login Error",true);
        jd.setBounds(100,100,200,60);
        jd.setLayout(new FlowLayout());
        jd.add(new JLabel("Username or Password Error!"));
        jd.setVisible(true);
    }
}
```

◆事件类层次关系



下列事件属于高级事件，是基于语义的事件，可以不和特定动作相关联，而依赖于触发此事件的类。

ActionEvent

单击按钮、菜单项，在文本框按回车，双击列表选项

ItemEvent

点击复选框、单选按钮、列表选项及带复选框的菜单项等

addItemListener(ItemListener Listener)

itemStateChanged(ItemEvent e)

TextEvent

文本框、文本域内的文本内容发生改变

addTextListener(TextListener Listener)

textValueChanged(TextEvent e)

AdjustmentEvent

调节滚动条或滑块等

addAdjustmentListener(AdjustmentListener Listener)

AdjustmentValueChange(AdjustmentEvent e)

下列属于低级事件，如基于特定动作的事件，如进入、拖放、获得/失去焦点等。

ComponentEvent

移动、隐藏、显示组件和改变组件大小等

```
addComponentListener(ComponentListener Listener)  
ComponentMoved(ComponentEvent e)  
ComponentHidden(ComponentEvent e)  
ComponentResized(ComponentEvent e)  
ComponentShown(ComponentEvent e)
```

ContainerEvent

容器中添加、移动组件等

```
addContainerListener(ContainerListener Listener)  
ContainerAdded(ContainerEvent e)  
ContainerRemoved(ContainerEvent e)
```

FocusEvent

当组件具有焦点监视器后，组件从无输入焦点变成有输入焦点，或从有输入焦点变成无输入焦点

addFocusListener(FocusListener Listener)

FocusGained(FocusEvent e)

FocusLost(FocusEvent e)

WindowEvent

激活、打开、关闭窗口或窗口失去焦点及收到窗口级事件

addWindowListener(WindowListener Listener)

windowClosing(WindowEvent e)

windowClosed(WindowEvent e)

windowOpened(WindowEvent e)

windowIconified(WindowEvent e)

windowDeiconified(WindowEvent e)

windowActivated(WindowEvent e)

windowDeactivated(WindowEvent e)

MouseEvent

鼠标单击或鼠标移动

addMouseListener(MouseListener Listener)

mouseEntered (MouseEvent e)

mouseExited(MouseEvent e)

addMouseMotionListener(MouseMotionListener Listener)

mousePressed(MouseEvent e)

mouseReleased(MouseEvent e)

mouseClicked(MouseEvent e)

mouseDragged(MouseEvent e)

mouseMoved(MouseEvent e)

MouseWheelEvent

鼠标滚轮滑动

```
addMouseWheelListener(MouseWheelListener Listener)  
mouseWheelMoved(MouseWheelEvent e)
```

KeyEvent

键盘事件

```
addKeyListener(KeyListener Listener)  
keyPressed(KeyEvent e)  
keyReleased(KeyEvent e)  
keyTyped(KeyEvent e)
```

◆事件适配器

- Java针对大多数事件监听器接口都提供相应的实现类——事件适配器。宗旨是使监听器的创建变得更加简便。
- 适配器中系统自动实现相应监听器接口中的所有方法(只写出空方法体),但不做任何事情。编程时定义继承事件适配器类的监听器,只重写需要的方法。常用的适配器类有:
 - KeyAdapter: 键盘事件适配器
 - MouseAdapter: 鼠标事件适配器
 - MouseMotionAdapter: 鼠标运行事件适配器
 - WindowAdapter: 窗口事件适配器
 - FocusAdapter: 焦点事件适配器
 - ComponentAdapter: 组件事件适配器
 - ContainerAdapter: 容器事件适配器

◆ JOptionPane

- 上例中使用了自定义的JDialog对象来进行交互。
- Java中提供了多种可供用户使用的对话框以及相应的**静态方法**，指定该方法中的相关**参数**就可以方便地创建各种类型的简单对话框。

showMessageDialog

显示消息对话框，用于向用户显示一些信息，含有“确定”按钮，无返回值。

showConfirmDialog

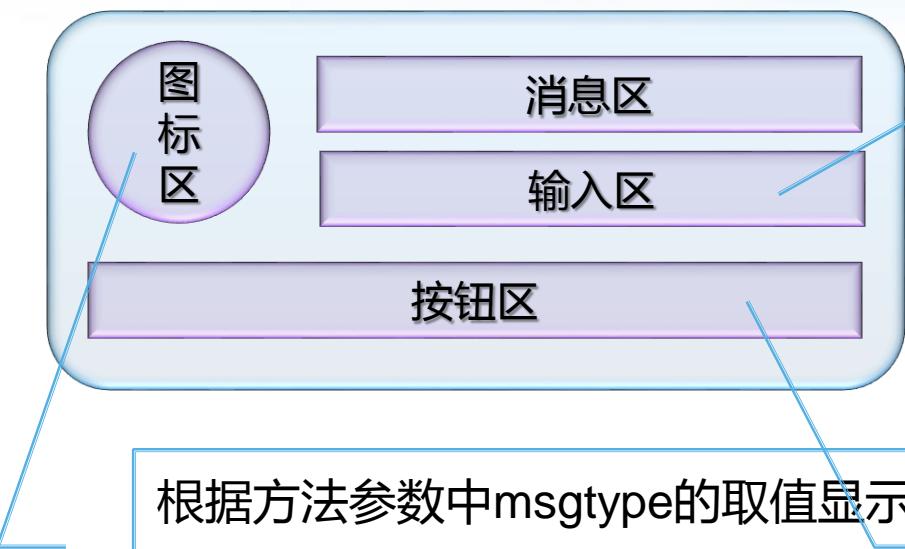
显示确认对话框，请用户选择，例如yes、no等，返回值是用户单击的按钮。

showInputDialog

显示输入对话框，要求输入某些信息，返回用户输入的字符串。

showOptionDialog

显示自定义选项对话框，可取代showComfirmDialog产生的对话框，只是使用更为复杂。



如果无需用户输入，则
输入区不存在

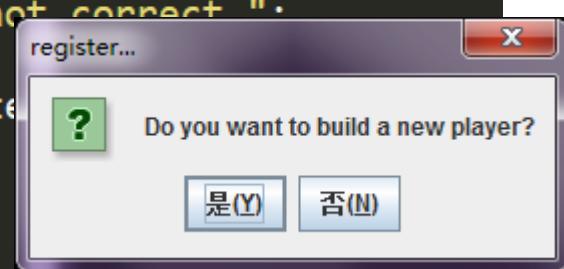
根据方法参数中msgtype的取值显示
不同图标：

JOptionPane.WARNING_MESSAGE
JOptionPane.INFORMATION_MESSAGE
JOptionPane.QUESTION_MESSAGE
JOptionPane.ERROR_MESSAGE
JOptionPane.PLAIN_MESSAGE

根据方法参数中optionType的取值显
示不同按钮：

JOptionPane.DEFAULT_OPTION
JOptionPane.YES_NO_OPTION
JOptionPane.OK_CANCEL_OPTION
JOptionPane.YES_NO_CANCEL_OPTION

```
public void actionPerformed(ActionEvent e){  
    int dialogType;  
    String title;  
    String content;  
    if(e.getSource()==login){  
        if(username.getText().length()==0 || pw.getText().length()==0)  
            title="login...";  
        content="Your username or password can not be blank!";  
        dialogType=JOptionPane.WARNING_MESSAGE;  
        JOptionPane.showMessageDialog(null,content,title,dialogType);  
    }  
    else{  
        title="login...";  
        content="Your username or Password is not correct...";  
        dialogType=JOptionPane.ERROR_MESSAGE;  
        JOptionPane.showMessageDialog(null,content,title,dialogType);  
    }  
}  
else if(e.getSource()==register){  
    title="register...";  
    content="Do you want to build a new player?";  
    dialogType=JOptionPane.YES_NO_OPTION;  
    JOptionPane.showConfirmDialog(null,content,title,dialogType);  
}
```



```
username.requestFocusInWindow();
username.addFocusListener(this);
pw.addFocusListener(this);
```

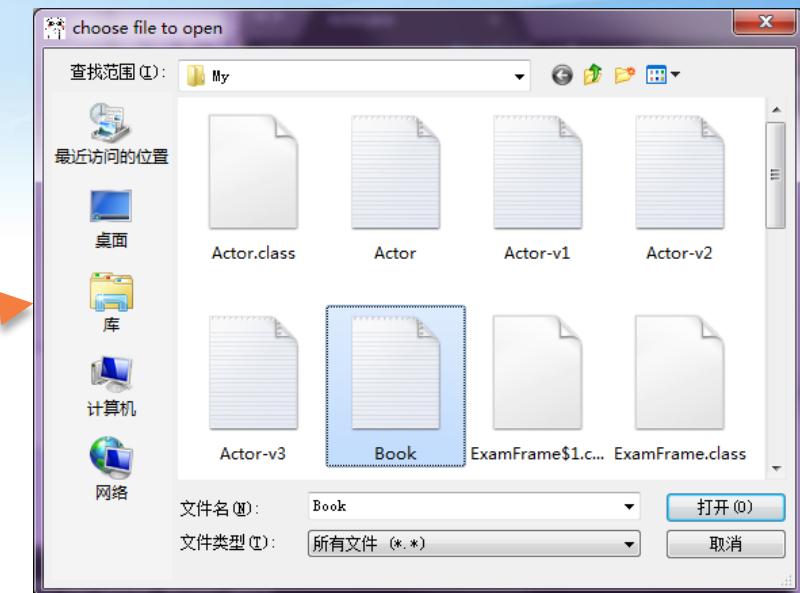
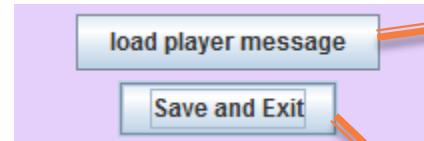
```
public void focusGained(FocusEvent e){
    ((Component)e.getSource()).setBackground(new Color(229,208,252));
}
public void focusLost(FocusEvent e){
    ((Component)e.getSource()).setBackground(Color.WHITE);
}
```



◆FileDialog

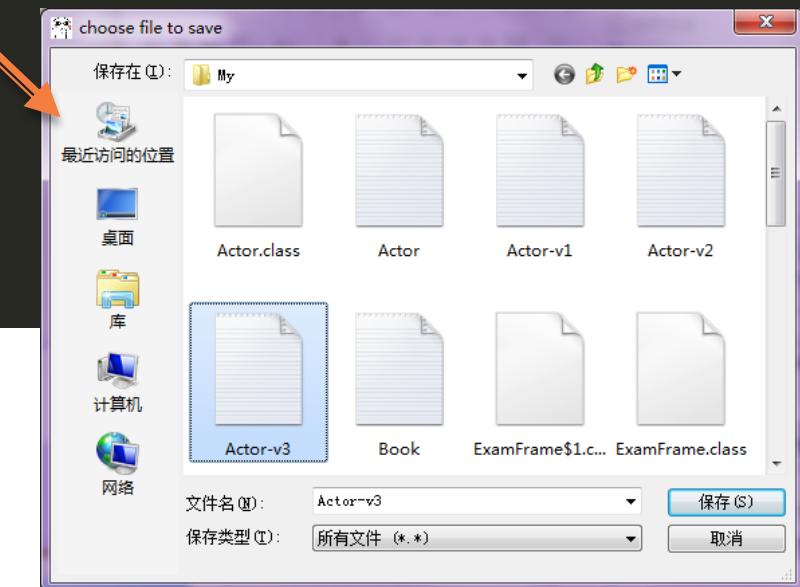
- 文件对话框，用于打开或保存文件，其中构造方法中的mode参数用于指定用于打开还是保存文件，该参数支持两个参数值：
 - FileDialog.LOAD
 - FileDialog.SAVE
- FileDialog不能指定是否为模式对话框，它依赖于运行平台的实现，若运行平台的文件对话框是模式的，那么FileDialog也是模式的，否则就是非模式的。
- 普通方法

getDirectory()	返回FileDialog被打开/保存文件的路径
getFile()	返回FileDialog被打开/保存的文件名



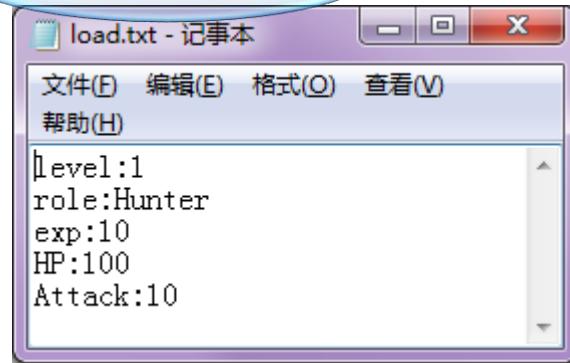
```
load.addActionListener(this);
save.addActionListener(this);
```

```
public void actionPerformed(ActionEvent e){
    FileDialog d1=new FileDialog(this,"choose file to open",FileDialog.LOAD);
    FileDialog d2=new FileDialog(this,"choose file to save",FileDialog.SAVE);
    if(e.getSource()==load){
        d1.setVisible(true);
    }
    if(e.getSource()==save){
        d2.setVisible(true);
    }
}
```

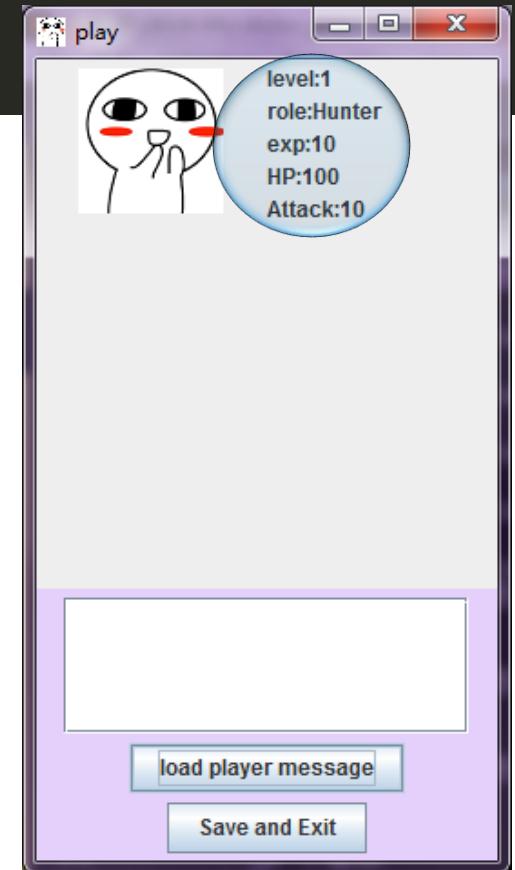


```
String s=null;
StringBuffer sb=new StringBuffer();
try{
    BufferedReader br=new BufferedReader(new FileReader(d1.getDirectory()+d1.getFile()));
} {
    while((s=br.readLine())!=null){
        sb.append(s+"<br>");
    }
    this.playerMessage.setText("<html>"+sb.toString()+"</html>");
} catch(IOException ie){
    ie.printStackTrace();
}
```

上例中读取名为load.txt的文本文件，并显示在之前
定义好的标签playerMessage中。

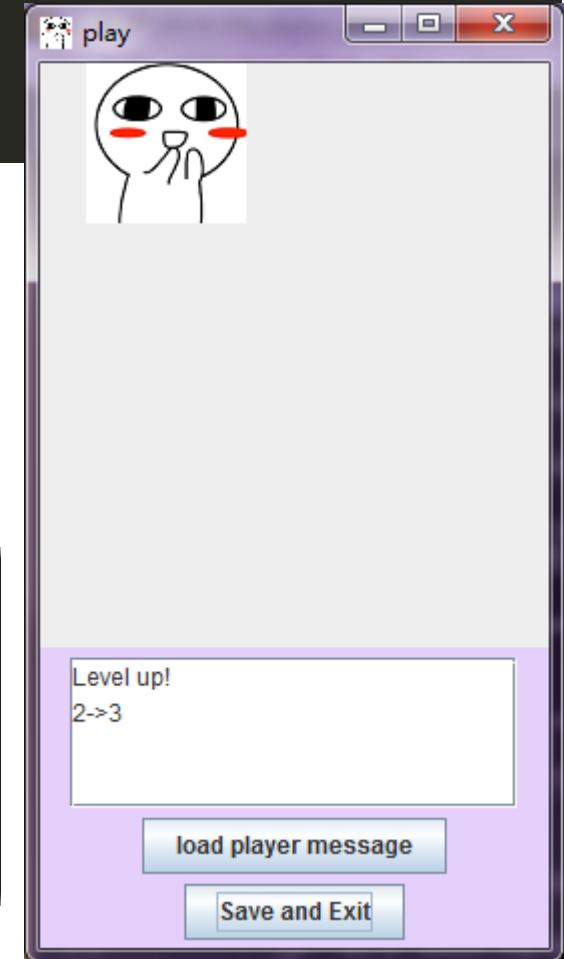
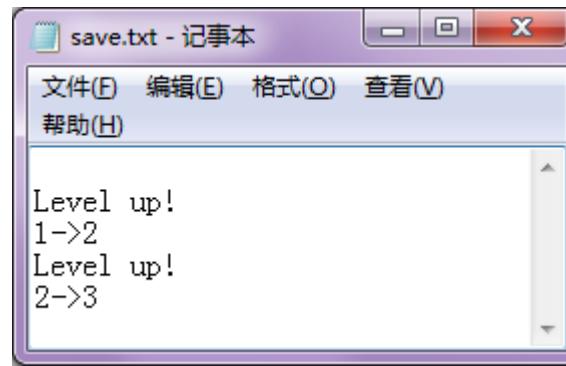


JLabel标签无法通过附加“\n”来进行多行显示，但JLabel可以解析html文本，因此可以利用html标签来实现多行显示。



```
String s;
try{
    BufferedWriter bw=new BufferedWriter(new FileWriter(d2.getDirectory()+d2.getFile(),true));
} {
    s=this.record.getText().replaceAll("\n","\r\n");
    bw.write("\r\n"+s);
} catch(IOException ie){
    ie.printStackTrace();
}
```

上例中选取名为save.txt的文本文件进行保存，会将JTextArea中的内容保存在文件中。因为选择打开文件输出流的方式是append方式，所以会在文件尾部进行续写，而不是覆盖。



- 设定一个3X3的冒险地图，当点开图标时随机产生不同的环境，并有简单的不同收益记录。

```

if(e.getSource()==go[0]||e.getSource()==go[1]||e.getSource()==go[2]||e.getSource()==go[3]||e.getSource()==go[4]){
    JButton temp=(JButton)e.getSource();
    int i=new Random().nextInt(3);
    if(i==0){
        ImageIcon ii=new ImageIcon("desert.png");
        temp.setIcon(new ImageIcon(ii.getImage().getScaledInstance(270/n,-1,Image.SCALE_SMOOTH)));
        temp.removeActionListener(this);
        this.record.append("\n"+ "you got a piece of gold");
    }
}

```

- 产生[0,3]内的整数随机数；
- 根据随机数判断是哪种环境（forest、 sea、 desert）；
- 将对应的按钮图标设为相应图片；
- 将按钮上的监听器移除；
- 记录简单收益

其中涉及到很多按钮，导致事件源的判定非常繁琐，程序臃肿。



扩展：内部类和匿名内部类

- 内部类

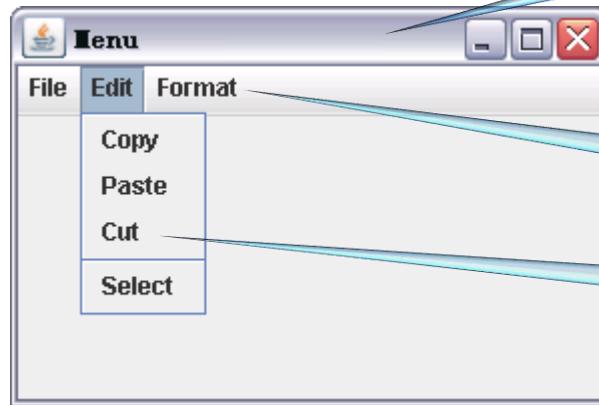
```
class MainFrame extends JFrame{  
    class loadListener implements ActionListener{  
        public void actionPerformed(ActionEvent e){  
            .....  
        }  
    }  
}
```

```
load.addActionListener(new loadListener());
```

- 匿名内部类

```
for(int i=0;i<9;i++){  
    go[i]=new JButton("Unknown");  
    go[i].addActionListener(new ActionListener(){  
        public void actionPerformed(ActionEvent e){  
            .....  
        }  
    });  
}
```

◆ 菜单



菜单栏

MenuBar/JMenuBar

菜单

Menu/JMenu

菜单项

MenuItem/JMenuItem

除了基本的菜单之外，诸如文本框、下拉列表等也可以作为菜单加入到菜单栏中；

除了基本的菜单项之外，菜单项还可以是菜单，称为子菜单；
也可以使用带复选框或单选按钮的菜单项。

CheckBoxMenuItem/ JCheckBoxMenuItem

RadioButtonMenuItem/ JRadioButtonMenuItem

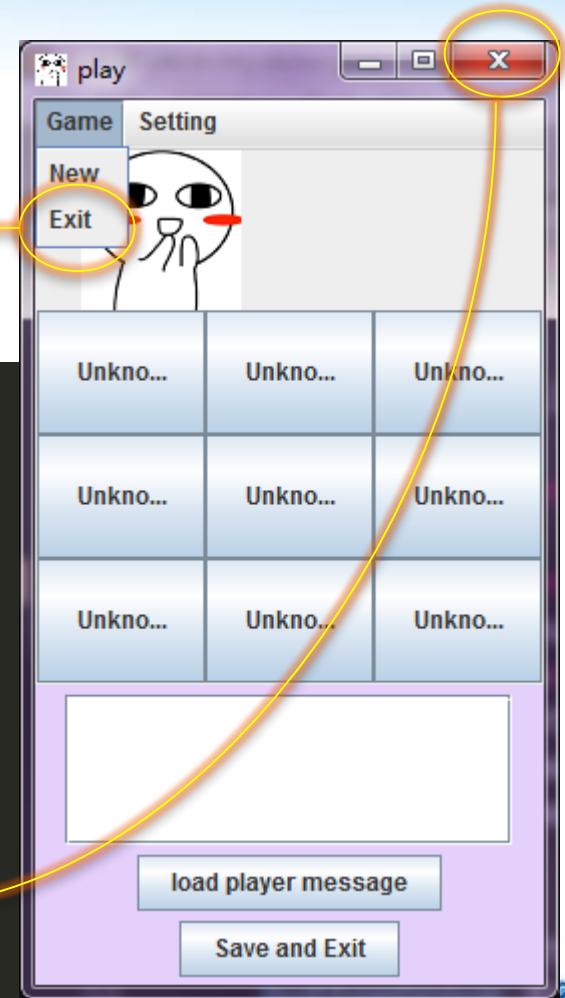
```
JMenuBar jmb=new JMenuBar();
JMenu gameMenu=new JMenu("Game");

JMenuItem gameNew = new JMenuItem("New");
JMenuItem gameExit = new JMenuItem("Exit");
```

```
jmb.add(gameMenu);
gameMenu.add(gameNew);
gameMenu.add(gameExit);
gameExit.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        System.exit(0);
    }
});
```

```
addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
```

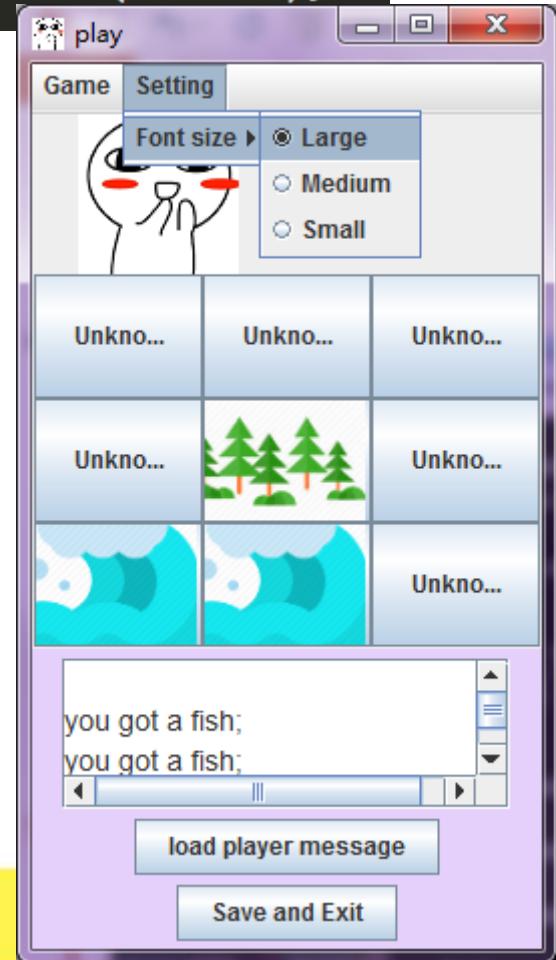
```
setJMenuBar(jmb);
```



```
JMenu setMenu=new JMenu("Setting");
JMenu fontSetting = new JMenu("Font size");
ButtonGroup fontSizeGroup=new ButtonGroup();
JRadioButtonMenuItem fontBig=new JRadioButtonMenuItem("Large");
JRadioButtonMenuItem fontNormal=new JRadioButtonMenuItem("Medium");
JRadioButtonMenuItem fontSmall=new JRadioButtonMenuItem("Small");
```

```
jmb.add(setMenu);
setMenu.add(fontSetting);
fontSizeGroup.add(fontBig);
fontSizeGroup.add(fontNormal);
fontSizeGroup.add(fontSmall);
fontSetting.add(fontBig);
fontSetting.add(fontNormal);
fontSetting.add(fontSmall);
```

```
fontBig.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        record.setFont(
            new Font(record.getFont().getName(),
                    record.getFont().getStyle(),
                    14));
    }
});
```



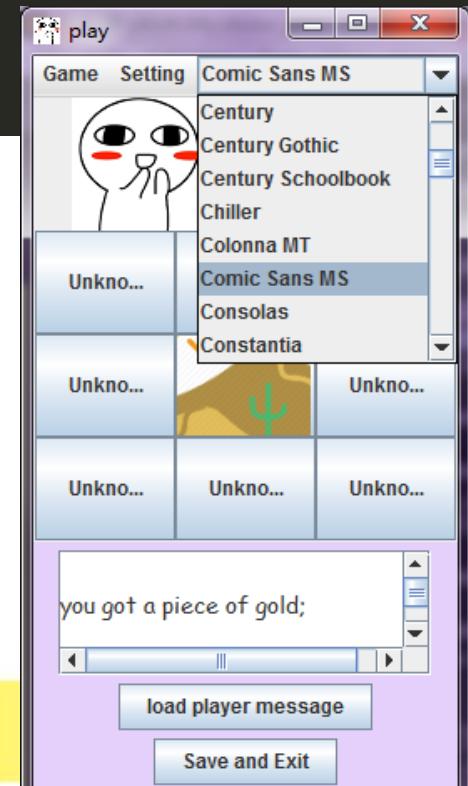
```
JComboBox fontList;
```

```
GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
String fontName[] = ge.getAvailableFontFamilyNames();
fontList=new JComboBox(fontName);
fontList.setPreferredSize(new Dimension(100,10));
fontList.addItemListener(new ItemListener(){
    public void itemStateChanged(ItemEvent e){
        String name = (String)fontList.getSelectedItem();
        Font f = new Font(name,record.getFont().getStyle(),record.getFont().getSize());
        record.setFont(f);
    }
});
```

上述功能用到了java.awt包中的Font类，该类的构造方法如下：

public Font(String name, int style, int size)

其中Style为字体样式，有效取值为：Font.BOLD、Font.PLAIN、Font.ITALIC等。



◆ JProgressBar

- 进度条组件，通过用颜色动态填充显示某任务完成的百分比。
- 进度条的最大值并不是进度条的长度，进度条的长度依赖于放置它的布局和本身是否试用了setSize()设置了大小；
- 进度条的最大值max是指将进度条平均分成max份。

```
JProgressBar jpb=new JProgressBar(0,n*n);
```

```
 jpb.setValue(jpb.getValue()+1);
```



Expression

◆ JTable

- JTable是Swing新增的组件，主要功能是将数据以二维表格的形式显示出来。注意：
 - 用户在表格单元中输入的数据都被认为是一个**Object对象**；
 - 表格中的数据将以行和列的形式显示数组data每个单元中对象的字符串表示。
 - JTable不支持滚动，要创建一个可滚动的表格，需要先创建 JScrollPane，并将 JTable的实例添加到该滚动面板中。

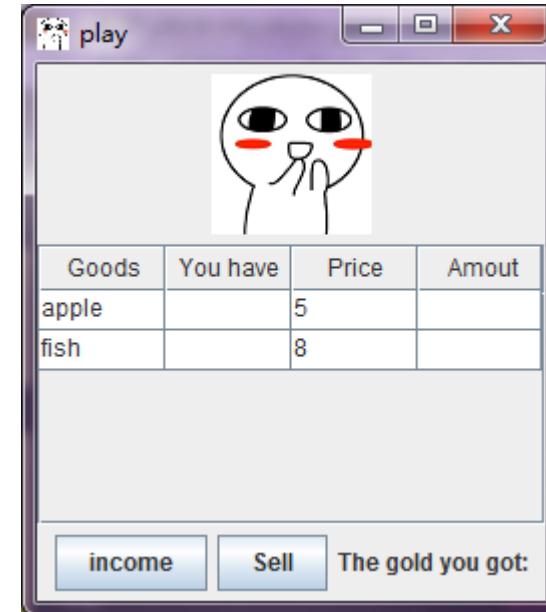
JTable()	创建一个默认模式的表格对象
JTable(Object data[][], Object columnName[])	创建一个表格对象，用来显示二维数组 data 中的值，其列名称为 columnName

getRowCount();	获取表格行数
getColumnCount();	获取表格列数
getColumnName(int col);	获取表格某列名字
setValueAt(Object value, int row, int col)	改变现有表格某行某列的数据
getValueAt(int row, int col)	获取现有表格某行某列的数据

```
Object[][] a=new Object[2][4];
Object[] columnName={"Goods","You have","Price","Amout"};
String goods[]={ "apple", "fish"};
int price[]={5,8};
JTable jt;
```

```
for(int i=0;i<2;i++){
    for(int j=0;j<4;j++){
        if(j==0) a[i][j]=goods[i];
        else if(j==2) a[i][j]=price[i]+"";
        else a[i][j]="";
    }
}

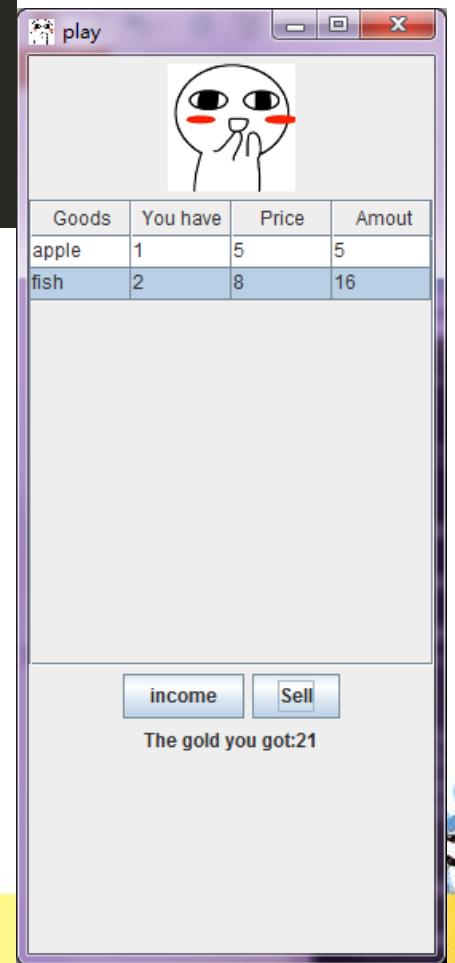
jt=new JTable(a,columnName);
jt.setRowHeight(20);
JScrollPane tbPane=new JScrollPane(jt);
add(tbPane,BorderLayout.CENTER);
```



```
cal.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        for(int i=0;i<2;i++){
            a[i][3] = "" + Integer.parseInt(a[i][1].toString()) * Integer.parseInt(a[i][2]);
            jt.repaint();
        }
    });
});
```



```
sell.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        int sum=0;
        for(int i=0;i<2;i++){
            sum=sum+Integer.parseInt(a[i][3].toString());
        }
        sellMessage.setText("The gold you got:"+sum);
    }
});
```



扩展：更丰富的表格功能

- 通过表格模型TableModel（接口）来创建表格，一般情况可以利用已经实现该接口的实现类DefaultTableModel来创建。

```
DefaultTableModel dm=new DefaultTableModel(int row,int column);  
JTable jt=new JTable(dm);
```

- 除此之外，还有 TableColumnModel等辅助类和接口，提供了一系列的方法来辅助操作，如增加、删除、移动数据行、列，设置单元格值等。
- Swing使用的单元格绘制器都有一致的编程模型TableCellRenderer，Swing为该接口提供了一个实现类：DefaultTableCellRenderer，该绘制器可以绘制如下三种类型的单元格值（根据其TableModel中的getColumnClass方法来获取其单元格类型）
 - Icon：绘制图标对象
 - Boolean：绘制复选按钮
 - Object：绘制对象的toString值。

- 需要注意的是，getColumnClass方法总返回Object，这将导致默认的单元格绘制器会把Icon和Boolean都绘制为字符串，为了实现图标和复选框绘制，需要扩展DefaultTableModel类并重写 getColumnClass方法。

```
class ExtendedTableModel extends DefaultTableModel{  
    public ExtendedTableModel(String[] columnName, Object[][] a){  
        super(a, columnName);  
    }  
    public Class getColumnClass(int c){  
        return getValueAt(0, c).getClass();  
    }  
}
```

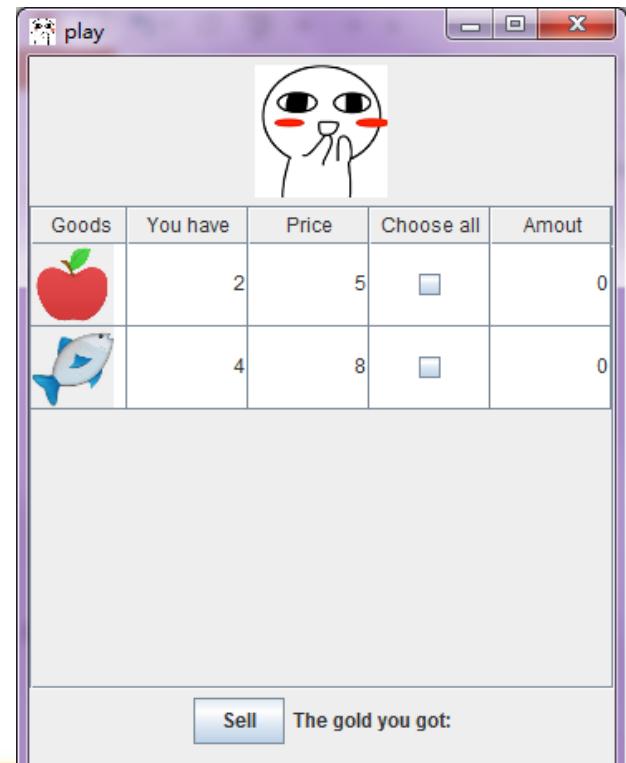
```
ExtendedTableModel etm=new ExtendedTableModel(columnName, a);  
jt=new JTable(etm);
```

- 如果希望程序采用自己定制的单元格绘制器，则同样需要让自定义绘制器实现TableCellRenderer接口，并实现getTableCellRendererComponent方法，该方法返回的Component将会作为指定单元格绘制的组件。

```
class GenderTableCellRenderer extends JPanel implements TableCellRenderer{  
    private String cellValue;  
    public Component getTableCellRendererComponent(  
        JTable jt, Object a, boolean isSelected, boolean hasFocus, int row, int column){  
        cellValue=(String)a;  
        return this;  
    }  
    public void paint(Graphics g){  
        if(cellValue.equalsIgnoreCase("apple")){  
            g.drawImage(new ImageIcon("apple.png").getImage(), 0, 0, null);  
        }else{  
            g.drawImage(new ImageIcon("fish.png").getImage(), 0, 0, null);  
        }  
    }  
}
```

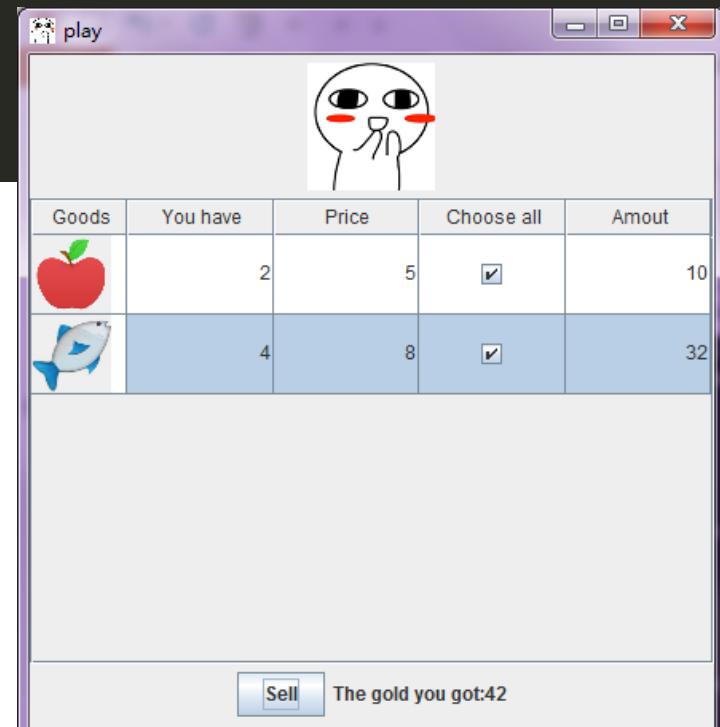
- 实现了单元格绘制器后，还要将其“绑定”到制定的表格上：

```
TableColumn firstCol=jt.getColumnModel().getColumn(0);  
firstCol.setCellRenderer(new GenderTableCellRenderer());
```



```

etm.addTableModelListener(new TableModelListener(){
    public void tableChanged(TableModelEvent e){
        int row=e.getFirstRow();
        int col=e.getColumn();
        if(col==3){
            boolean b=Boolean.parseBoolean(jt.getValueAt(row,3).toString());
            if(b==true){
                etm.setValueAt(Integer.parseInt(a[row][1].toString())*Integer.parseInt(a[row][2].toString()),row,4);
            }
            else if(b==false){
                etm.setValueAt("0",row,4);
            }
        }
    }
});
```



◆树组件JTree

- JTree用来显示一个层次关系分明的一组数据，用树状图表示能给用户一个直观而易用的感觉。JTree可以通过树形结构分层组织数据来让用户清楚的了解各个节点之间的关系。
- 要构造一个树组件必须事先创建出称为节点的对象。节点是树中最基本的对象，表示在给定层次结构中的数据项。树中只有一个根节点，除此以外的节点分为两类：
 - 叶节点（即没有子节点的节点）
 - 分支节点（即有子节点的节点）
- 每个节点关联着一个描述该节点的文本标签和图像图标，其中：
 - 文本标签是节点的字符串表示
 - 图标指明该节点是否为叶节点

- 实现
 - javax.swing.tree包中提供了一个MutableTreeNode接口来实现创建并成为树节点的对象。
 - DefaultMutableTreeNode类是实现了MutableTreeNode接口的类，可以使用这个类为要创建的树准备节点，其常用构造方法如下：

DefaultMutableTreeNode(Object userObj)	创建没有父节点和子节点、但允许有子节点的树节点，并使用指定的用户对象对它进行初始化。
DefaultMutableTreeNode(Object userObj, boolean child)	创建没有父节点和子节点的树节点，使用指定的用户对象对它进行初始化，仅在指定时才允许有子节点。

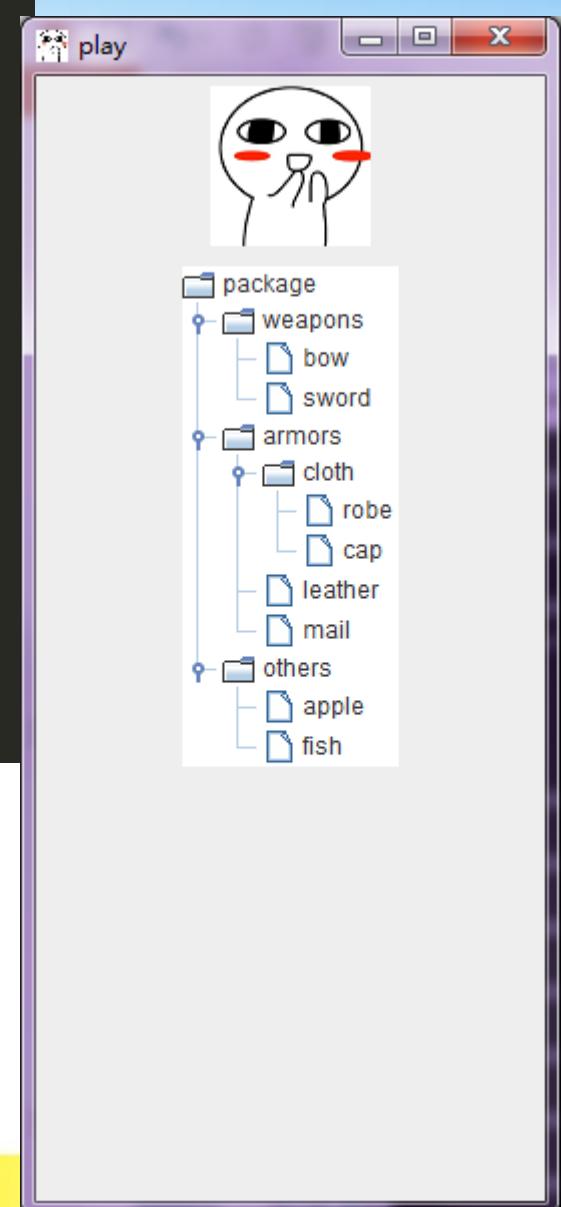
- 注册监听器：
 - addTreeSelectionListener(TreeSelectionListener e)
 - 当鼠标单击树上的节点时，系统将通知监听器自动调用TreeSelectionListener接口中的方法valueChanged实施相应操作。

- 系统提供的树组件操作方法

JTree(TreeNode root)	返回一个 JTree，指定的 TreeNode 作为其根，它显示根节点。
getParent()	返回此节点的父节点，如果此节点没有父节点，则返回 null。
isLeaf()	如果此节点没有子节点，则返回 true。
getChildCount()	返回此节点的子节点数。
setAllowsChildren(boolean b)	确定是否允许此节点拥有子节点。
getUserObject()	返回此节点的用户对象。

```
DefaultMutableTreeNode root=new DefaultMutableTreeNode("package");
DefaultMutableTreeNode n1=new DefaultMutableTreeNode("weapons");
DefaultMutableTreeNode n11=new DefaultMutableTreeNode("bow");
DefaultMutableTreeNode n12=new DefaultMutableTreeNode("sword");
DefaultMutableTreeNode n2=new DefaultMutableTreeNode("armors");
DefaultMutableTreeNode n21=new DefaultMutableTreeNode("cloth");
DefaultMutableTreeNode n211=new DefaultMutableTreeNode("robe");
DefaultMutableTreeNode n212=new DefaultMutableTreeNode("cap");
DefaultMutableTreeNode n22=new DefaultMutableTreeNode("leather");
DefaultMutableTreeNode n23=new DefaultMutableTreeNode("mail");
DefaultMutableTreeNode n3=new DefaultMutableTreeNode("others");
DefaultMutableTreeNode n31=new DefaultMutableTreeNode("apple");
DefaultMutableTreeNode n32=new DefaultMutableTreeNode("fish");
root.add(n1);root.add(n2);root.add(n3);
n1.add(n11);n1.add(n12);
n2.add(n21);n2.add(n22);n2.add(n23);
n3.add(n31);n3.add(n32);
n21.add(n211);n21.add(n212);

jt=new JTree(root);
```

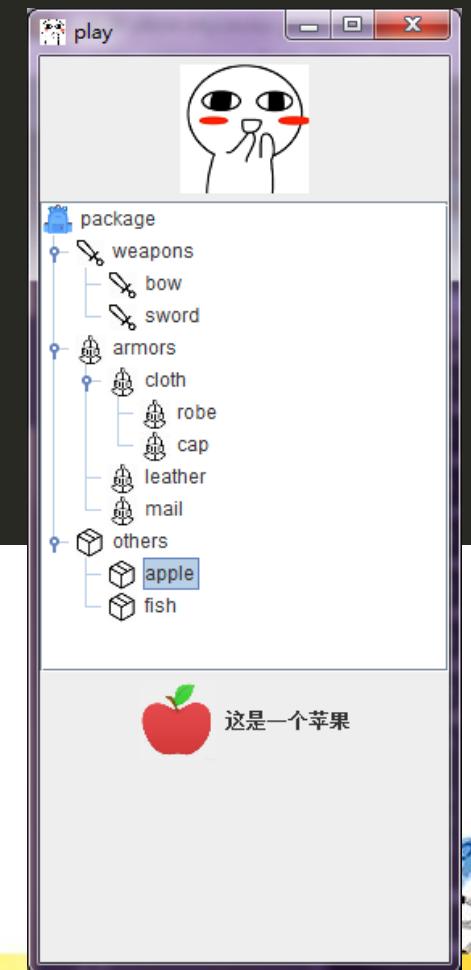


扩展：改变节点外观

- 可使用DefaultTreeCellRenderer来改变节点的外观（字体、颜色、图标）；
 - 这种情况是改变整棵树所有节点的外观
- 可为JTree指定DefaultTreeCellRenderer的子类作为其节点绘制器；
 - 是较为通用的方式，提供很多方法来修改节点外观
- 为JTree指定一个实现TreeCellRenderer借口的节点绘制器
 - 最复杂最灵活的方式

```
class MyRenderer extends DefaultTreeCellRenderer{
    ImageIcon icon;
    String nodeName;
    public Component getTreeCellRendererComponent(
        JTree jt, Object node, boolean isSelected, boolean expanded, boolean leaf, int row, boolean hasFocus){
        super.getTreeCellRendererComponent(jt, node, isSelected, expanded, leaf, row, hasFocus);
        DefaultMutableTreeNode dn=(DefaultMutableTreeNode)node;
        nodeName=dn.toString();
        if(dn.isRoot()==true){
            icon=new ImageIcon(nodeName+".png");
        }else{
            DefaultMutableTreeNode temp=dn;
            while(((DefaultMutableTreeNode)temp.getParent()).isRoot()!=true){
                temp=(DefaultMutableTreeNode)temp.getParent();
            }
            String s=temp.toString();
            icon=new ImageIcon(s+".png");
        }
        this.setIcon(icon);
        return this;
    }
}

jt.setCellRenderer(new MyRenderer());
```



1.3 在AWT中绘图

◆在Component类中有三种和绘图有关的方法：

- paint(Graphics g) 绘制组件外观
- update(Graphics g) 调用paint()方法，刷新组件外观
- repaint() 调用update()方法，刷新组件外观
- Container中的update()方法先以组件的背景色填充整个组件区域，然后调用paint()方法；
- 普通组件的update()方法直接调用paint()方法。

组件第一次显示
时系统自动调用



◆ 使用 Graphics 类

- Graphics 是一个抽象的画笔对象，提供了多个方法用于绘制几何图形和位图。

draw3DRect(int x, int y, int w, int h, boolean raised)	绘制以(x,y)为起点宽高为w和h的3D矩形，当 raised 为 true 时 3D 效果为凸
fill3DRect(int x, int y, int w, int h, boolean raised)	填充以(x,y)为起点宽高为w和h的3D矩形，当 raised 为 true 时 3D 效果为凸
drawLine(int x1, int y1, int x2, int y2)	画点(x1,y1)和点(x2,y2)之间的直线
drawRect(int x, int y, int w, int h)	绘制以(x,y)为起点宽高为w和h的矩形
fillRect(int x, int y, int w, int h)	填充以(x,y)为起点宽高为w和h的矩形
clearRect(int x, int y, int w, int h)	以背景色填充以(x,y)为起点宽高为w和h的矩形
drawRoundRect(int x, int y, int w, int h, int arcW, int arcH)	绘制以(x,y)为起点宽高为w和h的圆角矩形，四个圆角的宽度为 arcW，高度为 arcH
fillRoundRect(int x, int y, int w, int h, int arcW, int arcH)	填充以(x,y)为起点宽高为w和h的圆角矩形，四个圆角的宽度为 arcW，高度为 arcH

<code>drawOval(int x, int y, int w, int h)</code>	以(x,y)为起点、按宽高为w和h的外接矩形绘制椭圆
<code>fillOval(int x, int y, int w, int h)</code>	以(x,y)为起点、按宽高为w和h的外接矩形填充椭圆
<code>drawArc(int x, int y, int h, int sAng, int arcAng)</code>	以sAng为起始角，arcAng为张角画圆弧或扇形，逆时针为正
<code>fillArc(int x, int y, int h, int sAng, int arcAng)</code>	以sAng为起始角，arcAng为张角填充圆弧或扇形，顺时针为负
<code>drawPolygon(int x[], int y[], int points)</code>	按照x[]和y[]数组为定点绘制封闭的points边形
<code>fillPolygon(int x[], int y[], int points)</code>	按照x[]和y[]数组为定点填充封闭的points边形

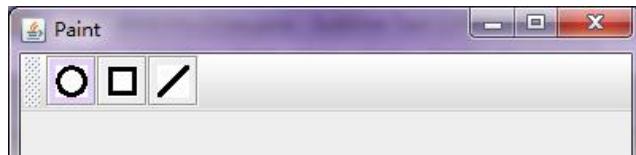
- 可以通过Color类来定义图形颜色

<code>Color(int r, int g, int b)</code>	创建具有指定红绿蓝值的不透明sRGB颜色，值范围为0-255
<code>Color(float r, float g, float b)</code>	创建具有指定红绿蓝值的不透明sRGB颜色，值范围为0.0-1.0
<code>Color(int rgb)</code>	创建具有指定组合的RGB值的不透明的sRGB 颜色，此 sRGB 值的16-23位表示红色分量，8-15位表示绿色分量，0-7位表示蓝色分量

- 此外还有系统定义的13中颜色常量：
 - WHITE / RED / GREEN / BLUE / GRAY / YELLOW / CYAN / BLACK / MAGENTA / PINK / DARKGRAY / LIGHTGRAY / ORANGE

◆简单的绘图小程序

创建工具栏，并添加三个图标——椭圆、矩形、直线

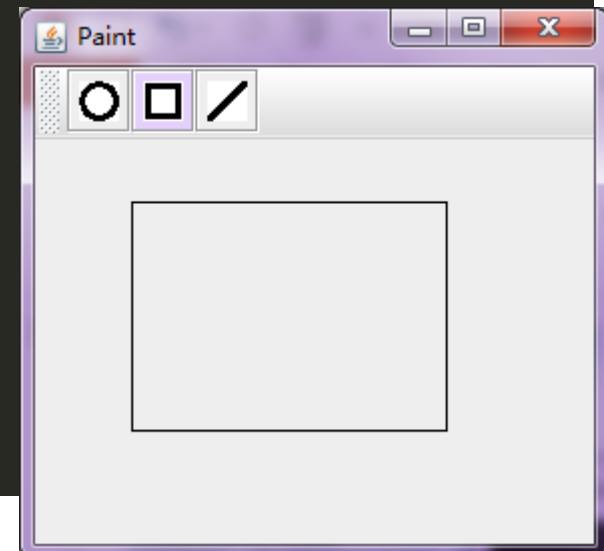


```
JToolBar jtb=new JToolBar();
String operation="";
Action Oval=new AbstractAction("",new ImageIcon("oval.png")){
    public void actionPerformed(ActionEvent e){
        operation="oval";
        ovalBut.setBackground(new Color(229,208,252));
        recBut.setBackground(null);
        lineBut.setBackground(null);
    }
};
```

```
JButton ovalBut=new JButton(Oval);
```

对鼠标点击和拖拽事件进行处理

```
class MyCanvas extends Canvas implements MouseListener,MouseMotionListener{  
    boolean released=false;  
    public MyCanvas(){  
        setPreferredSize(new Dimension(WIDTH,HEIGHT));  
        addMouseListener(this);  
        addMouseMotionListener(this);  
    }  
    public void paint(Graphics g){  
        g.setColor(foreColor);  
        switch(operation){  
            case "oval":g.drawOval(sx,sy,dx-sx,dy-sy);break;  
            case "rec":g.drawRect(sx,sy,dx-sx,dy-sy);break;  
            case "line":g.drawLine(sx,sy,dx,dy);break;  
        }  
    }  
    public void mousePressed(MouseEvent e){  
        sx=e.getX();  
        sy=e.getY();  
    }  
    public void mouseDragged(MouseEvent e){  
        dx=e.getX();  
        dy=e.getY();  
        repaint();  
    }  
}
```

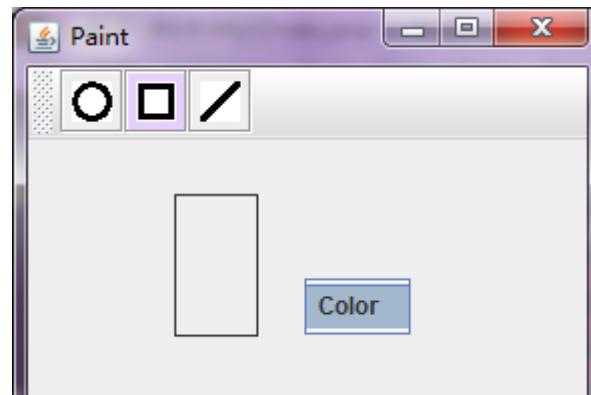


添加右键弹出菜单，选择颜色

```
JPopupMenu pop=new JPopupMenu();
JMenuItem chooseColor=new JMenuItem("Color");

chooseColor.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent ae){
        foreColor = JColorChooser.showDialog(f,"Choose a color",foreColor);
    }
});
pop.add(chooseColor);

public void mouseReleased(MouseEvent e){
    if(e.isPopupTrigger()){
        pop.show(this,e.getX(),e.getY());
    }
}
```



◆ 图像数据缓冲区 BufferedImage

- 可创建指定大小、指定图像类型的对象
- 当需要向GUI组件上绘制图形时，不是直接绘制到GUI组件上，而是先绘制到缓冲区中，然后在调用drawImage方法一次性绘制

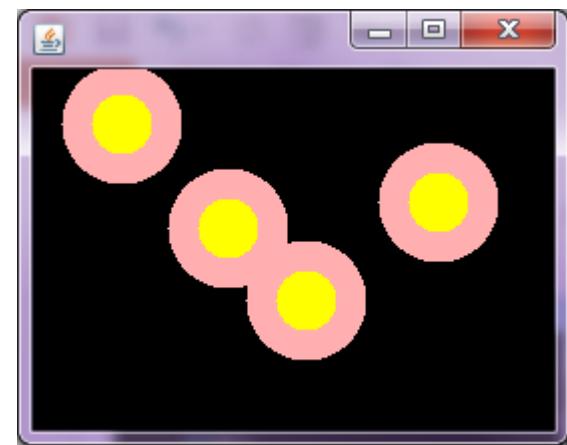
```
BufferedImage bi = new BufferedImage(100,100,BufferedImage.TYPE_INT_RGB);
```

- BufferedImage提供了getGraphics方法返回该对象的Graphics对象，从而允许通过该Graphics对象向Image中添加图形
- 在上一个例子中使用BufferedImage：

```
BufferedImage image=new BufferedImage(260,180,BufferedImage.TYPE_INT_RGB);
Graphics g=image.getGraphics();

public void mouseClicked(MouseEvent e){
    clicked=true;
    x=e.getX();
    y=e.getY();
    g.setColor(Color.pink);
    g.fillOval(x-30,y-30,60,60);
    g.setColor(Color.yellow);
    g.fillOval(x-15,y-15,30,30);
    g.drawImage(image,0,0,null);
    mc.repaint();
}

class MyCanvas extends Canvas{
    public void paint(Graphics g) {
        g.drawImage(image,0,0,null);
    }
}
```

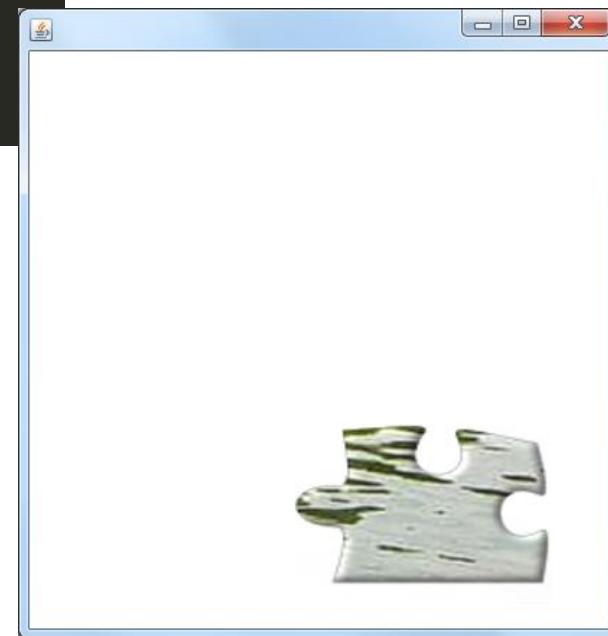


◆ 使用ImageIO输出图像

- ImageIO包含两个静态方法：read()和write()，完成对于位图文件的读写

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
public class MouseDragImg extends Frame implements MouseMotionListener,MouseListener {
    Image img;
    int x=189,y=136,posX=189,posY=136,dx,dy;
    public void init(){
        try{
            img=javax.imageio.ImageIO.read(new File("1.png"));
        }catch(Exception e){
        }
        addMouseListener(this); addMouseMotionListener(this);
        setVisible(true);
    }
    public void mousePressed(MouseEvent e){dx=e.getX()-posX; dy=e.getY()-posY;}
    public void mouseDragged(MouseEvent e){
        x=e.getX()-dx; y=e.getY()-dy;
        if(dx>0&&dx<189&&dy>0&&dy<136){
            Graphics g=getGraphics();
            update(g);
        }
    }
}
```

```
public void mouseMoved(MouseEvent e){}
public void mouseEntered(MouseEvent e){}
public void mouseExited(MouseEvent e){}
public void mouseReleased(MouseEvent e){}
public void mouseClicked(MouseEvent e){}
public void paint(Graphics g) {
    g.drawImage(img,x,y,189,136,this);
    posX=x; posY=y;
}
public static void main(String[] args){
    new MouseDragImg().init();
}
```



1.4 其他扩展

◆键盘事件范例：

- 组件使用addKeyListener()方法监听键盘事件。事件处理器：
 - public void keyPressed(KeyEvent e): 按下键盘上的某个键
 - public void keyTyped(KeyEvent e): 键被按下又释放（仅响应文字输入键，如字母、数字和标点符号等，它不会响应CTRL/ENTER/F1等功能键。）
 - public void KeyReleased(KeyEvent e): 释放键盘上的某个键
- KeyEvent类提供的主要方法：
 - getKeyChar(): 判断被按下的键的字符；
 - getKeyCode(): 判断哪个键被按下、点击或释放并获取其键码值。

- Java语言的部分键码表

键码	键	键码	键
VK_CANCEL/CLEAR/COMMA	取消/清除/逗号	VK_F1~VK_F2	F1~F12
VK_UNMPAD0~VK_UNMPAD9	小键盘0~9	VK_0~VK_9	0~9
VK_LEFT/RIGHT/UP/DOWN	左右上下键	VK_A~VK_Z	a~z
VK_KP_LEFT/RIGHT/UP/DOWN	小键盘左右上下键	VK_HOME/END	HOME/END
VK_BACK_SLASH/QUOTE	“\” / 单引号	VK_CAPS_LOCK	大写锁定
VK_ALT/CONTROL/SHIFT/ESCAPE	Alt/Ctrl/Shift/Esc	VK_NUM_LOCK	数字锁定
VK_SEMICOLON/PERIOD/SLASH	分号/点/ “/”	VK_SPACE/TAB	空格/制表符
VK_OPEN/CLOSE_BRACKET	[/]	VK_QUOTE	单引号
VK_INSERT/DELETE/ENTER/PAUSE	插入/删除/回车/暂停	VK_PAGE_UP/DOWN	前后翻页

- 注册键盘事件监听器

```
public KeyPanel(){  
    addKeyListener(this);  
}
```

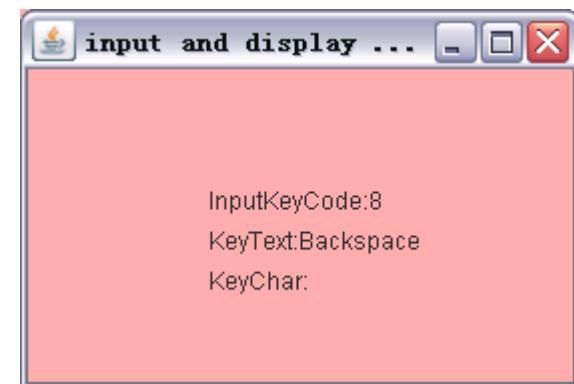
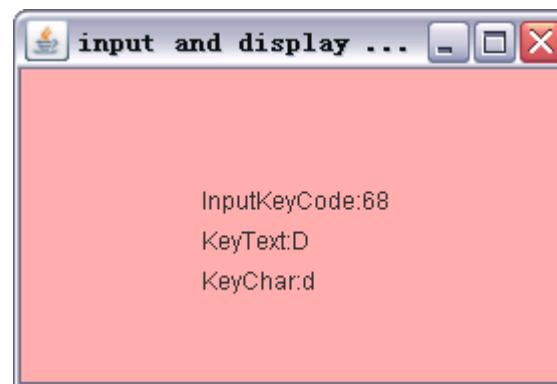
- 允许面板获得焦点

```
public boolean isFocusable(){  
    return true;  
}
```

- 对键盘事件进行响应（即实现接口方法）

```
public void keyPressed(KeyEvent e){  
    KeyInputCode=e.getKeyCode();  
}  
public void keyReleased(KeyEvent e){  
    KeyInputCode = e.getKeyCode();  
    KeyText = e.getKeyText(KeyInputCode);  
    if(!isChar)  
        KeyChar=' ';  
    isChar=false;  
    repaint();  
}  
public void keyTyped(KeyEvent e){  
    KeyChar = e.getKeyChar();  
    isChar = true;  
}
```

```
public void paintComponent(Graphics g){  
    super.paintComponent(g);  
    if(KeyInputCode==1)  
        g.drawString("InputKeyCode:",90,70);  
    else  
        g.drawString("InputKeyCode:"+KeyInputCode,90,70);  
    g.drawString("KeyText:"+KeyText,90,90);  
    g.drawString("KeyChar:"+KeyChar,90,110);  
}
```



- 复合键盘事件的使用

- KeyEvent类对象可调用getModifiers()方法返回以下整数值：
 - CTRL_MASK
 - ALT_MASK
 - SHIFT_MASK
- 以上是InputEvent类的类常量，根据返回值对复合键事件做处理。
 - 当使用Ctrl+C复合键时，下面的逻辑表达式为真：

```
e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_C
```

- 示例：利用对组合键事件的响应来处理复制、剪切与粘贴
 - 对复制、剪切、粘贴的处理

```
public void keyTyped(KeyEvent e){  
    JTextArea te=(JTextArea)e.getSource();  
    if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_X){te.cut();}  
    else if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_C){te.copy();}  
    else if(e.getModifiers()==InputEvent.CTRL_MASK&&e.getKeyCode()==KeyEvent.VK_P){te.paste();}  
}
```

◆更灵活的布局方式： GridBagConstraints

- 布局策略：采用网格的形式来布置组件，允许指定的组件跨多行或多列，同时允许组件部分重叠。组件的大小、跨越性等由 GridBagConstraints对象来设定。
- 步骤：
 1. 创建GridBagLayout对象，并指定GUI容器使用该布局管理器；

```
GridBagLayout gb=new GridBagLayout();
setLayout(gb);
```

2. 创建 GridBagConstraints对象，并设置该对象的相关属性（用于设置受该对象控制的组件的大小、跨越性等）；

```
gbc.gridx=2;
gbc.gridwidth=2;
```

3. 建立 GridBagConstraints 对象和受控组件之间的关联：

gb.setConstraints(c,gbc);

4. 将该组件添加到容器中：

add(c);

如果需要添加多个组件，则需要重复上述步骤2-4。

GridBagConstraints 对象可以重复使用，实际上创建一个就行了。

- GridBagConstraints 的属性

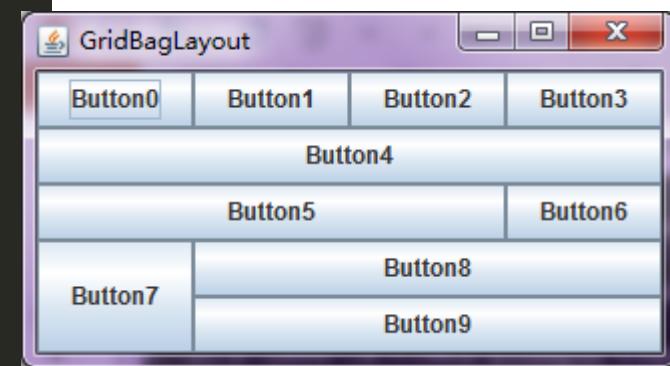
gridx gridy	指定组件的左上角所在的行和列	第一行或第一列均用0表示； 值为 GridBagConstraints.RELATIVE 时表明紧跟在上一个组件之后
gridwidth gridheight	指定组件显示区域所占行数和列数	整数，默认值为1。值为 REMAINDER 表示组件为当前行、列的最后一个组件， RELATIVE 表示为当前行、列的倒数第二个组件

fill	当组件比所在网格小时的填充方式	NONE, HORIZONTAL, VERTICAL, BOTH
ipadx ipady	组件横向、纵向内部填充的大小	组件横向、纵向大小为最小宽度、高度加上ipadx*2、 ipady*2
insets	组件边界和网格边缘间的距离	默认为Insets(0,0,0,0)， 四个参数分别代表上下左右
anchor	组件在显示区域中的定位	CENTER(默认值), NORTH, WEST, EAST, SOUTH, SOUTHEAST, SOUTHWEST, NORTHWEST, NORTHEAST
weightx weighty	指定分配给组件的额外水平空间和额外垂直空间	默认值为0，即不占用多余空间。假设某容器水平线上有3个组件，水平增加比例分别为1, 2, 3，容器增加60像素宽度的时候，它们分别增加10、20、30。

```

JFrame f = new JFrame("GridBagLayout");
GridBagLayout gb = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
private JButton[] bs=new JButton[10];
public void init(){
    f.setLayout(gb);
    for(int i=0;i<10;i++){
        bs[i]=new JButton("Button"+i);
    }
    gbc.fill=GridBagConstraints.BOTH;
    gbc.weightx=1;
    addButton(bs[0]);addButton(bs[1]);addButton(bs[2]);
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    addButton(bs[3]);addButton(bs[4]);
    gbc.gridwidth=GridBagConstraints.RELATIVE;
    addButton(bs[5]);
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    addButton(bs[6]);
    gbc.gridwidth=1;gbc.gridheight=2;
    gbc.weighty=1;
    addButton(bs[7]);
    gbc.weighty=0;
    gbc.gridwidth=GridBagConstraints.REMAINDER;
    gbc.gridheight=1;
    addButton(bs[8]);addButton(bs[9]);
    f.pack();
    f.setVisible(true);
}
private void addButton(JButton jb){
    gb.setConstraints(jb,gbc);
    f.add(jb);
}

```



总结

