1． GUI 编程引言

以前的学习当中，我们都使用的是命令交互方式：

例如:在 DOS 命令行中通过 javac java 命令启动程序.

软件的交互的方式：

1．命令交互方式

图书管理系统

2．图形交互方式

Java 提供了专业的 API 用于开发图形用户界面

GUI--> Graphic User Interface

将要了解 GUI API 的框架结构,以及 GUI 组件以及组件之间的关系,容器和布局管理器,颜色,字体等.

2． GUI 的分类

2.1. AWT

Java1.0 版本的图形用户界面库，设计目标是帮助程序员编写在所有平台上都能良好表现的 GUI 程序。为了实现这个目标 Java1.0 提供了抽象窗口工具集（AWT），但是这个目标并没有达到。AWT 在所有的系统上表现都不好。因为：最初版本的 AWT 是在一个月内构思，设计和实现的（Think in Java ）。

Abstract Window Toolkit 抽象窗口工具集

Java 将图形用户界面相关的类捆绑在了一起,放在了一个称之为抽象窗口工具集的库中.AWT 适合开发简单的图形用户界面,并不适合开发复杂的 GUI 项目.

位于: java.awt.* 中,定义了很多的组件类，开发者可以直接创建对象加以使用

缺点：所有的图形界面都依赖于底层的操作系统,容易发生于特定平台相关的故障.

AWT 调用本地系统资源生成图形化界面, 依赖本地平台. 1.0

2.2. Swing

SUN 公司对 AWT 进行了升级，基于 AWT，推出了一种更稳定,更通用和更灵活的库.称之为 Swing 组件库(Swing component)。

既然都是用于GUI设计的组件库,那么为了区分Swing组件类和对应的AWT组件类,Swing组件类都已字母 J 为前缀.位于:javax.swing.* 中,提供了和 AWT 中等同的所有的组件类，但是类名的前面多加了一个 J.

SWING 可以跨平台. 1.2

我们主要学习 Swing GUI 组件.

3. Java GUI API

GUI API 包含的类分为三个部分:组件类(component class) 容器类(container class),和辅助类(helper class)

1. 组件类是用来创建用户图形界面的,例如 JButton,JLabel,JTextField.
2. 容器类是用来包含其他组件的,例如 JFrame,JPanel
3. 辅助类是用来支持 GUI 组件的,例如 Color,Font

3.1.组件类

在图形用户界面程序中当我们想要创建按钮、复选框和滚动条等这些可以显示在屏幕上的对象，该如何创建。其实这些都属于一类叫做组件类。

AWT 中的组件根类

```
类 Component
       java.lang.Object
                java.awt.Component
```

Swing 中的组件根类

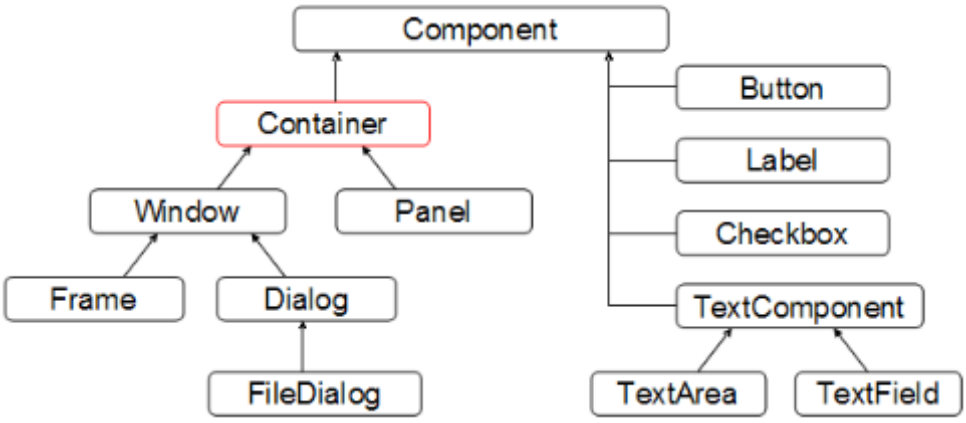
```
javax.swing
类 JComponent
       java.lang.Object
                java.awt.Component
                     java.awt.Container
                          javax.swing.JComponent
```

组件类的实例可以显示在屏幕上.Component 类是包括容器类的所有用户界面类的根类是 java.awt 中的类,对应的 Swing 中的是 Jcomponent.了解了 Component 和 JComponent 都是抽象类.所以不能使用 new 关键字创建对象.所以需要使用它们的具体的实现类来创建对象.

在 AWT 中典型图形用户界面中的按钮(Button)、复选框(Checkbox)和滚动条(Scrollbar)都是组件类,都是 Component 类的子类.

在 Swing 中的 GUI 组件,有对应的 JButton,JCheckBox,JscrollBar

继承关系图(AWT)



3.2. 容器类

容器(Container)，是一个特殊的组件，该组件可以通过 add()添加其他组件。

容器类适用于盛装其他 GUI 组件的 GUI 组件.例如 Panel Frame Dialog 都是 AWT 组件的容器类.对应的 Swing 组件的容器类是 JPanel JFrame JDialog

3.3. GUI 辅助类

用来描述 GUI 组件的属性,例如图像的颜色,字体等. 注意:辅助类是在 java.awt 中的

3.4. GUI 运行原理

在 JDK 的 bin 目录中有 javaw.exe .javaw.exe 是 java 在 window 中专门用于执行 GUI 程序.

4. 体验 GUI

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class Demo {
    public static void main(String[] args) {
        // 创建JFrame
        JFrame frame = new JFrame("hello,world");
        // 设置尺寸
        frame.setSize(200, 100);
```

```
        // JFrame在屏幕居中
        frame.setLocationRelativeTo(null);
        // JFrame关闭时的操作
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 显示JFrame
        frame.setVisible(true);

    }
}
```

5. Jframe 框架

JFrame(框架)是一个容器
创建一个用户界面需要创建一个 JFrame 来存放用户界面组件.例如存放按钮，文本框。

```
javax.swing
类 JFrame
java.lang.Object
  java.awt.Component
     java.awt.Container
        java.awt.Window
           java.awt.Frame
              javax.swing.JFrame
```

5.1. 创建一个框架

```
public class Demo1 {
    public static void main(String[] args) {
        // 创建JFrame
        JFrame frame = new JFrame("我的frame");
        // 显示JFrame
        frame.setVisible(true);

    }
}
```

注意:需要调用 setVisible(true)方法后才会显示框架
运行程序会在窗口的左上角显示一个窗口,但是只能显示标题栏,而且关闭 JFrame 时,程序程序没有停止.

所以需要完成如下需求：
1：设置 Jframe 的宽度和高度
2：让 Jframe 显示在屏幕中间
3：关闭 Jframe 时,程序会停止

5.1.1. 设置 JFrame 的宽度和高度

```
java.awt.Window.setSize(int width, int height)
```

查找 API 文档,查看 setSize 方法,可以指定框架的宽度和高度.参数类型是 int,注意是以像素为单位,普通的笔记本的屏幕分辨率为 1280*800 或者 1366*768.注意:分辨率表示每平方英寸的像素数.屏幕分辨率越高,屏幕的像素越多.所以分辨率越高看到的细节就越多.

setSize 方法被定义在 java.awt.Component 类中,被 Component 的子类 java.awt.Window 重写.而 Jframe 以继承了 Windiws 类所以也具备该方法.

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
```

5.1.2. JFrame 显示在屏幕中间

setLocationRelativeTo()方法

```
java.awt.Window.setLocationRelativeTo(Component c)
 设置窗口相对于指定组件的位置。
 如果 c 为 null，则此窗口将置于屏幕的中央。
```

setLocationRelativeTo(null)方法可以在屏幕上居中显示框架.

如果不想要设置在中间,可以使用

```
setLocation(200, 100);
```

5.1.3. 关闭 JFrame 程序停止

setDefaultCloseOperation(int operation)方法

```
javax.swing.JFrame.setDefaultCloseOperation(int operation)
```

该方法告诉程序,当框架关闭时结束程序.方法的参数是 JFrame 的常量 EXIT_ON_CLOSE
添加完毕

```
public class Demo1 {
```

```
    public static void main(String[] args) {
        // 创建JFrame
        JFrame frame = new JFrame("我的frame");
        // 设置尺寸
        frame.setSize(200, 100);
        // JFrame在屏幕居中
        frame.setLocationRelativeTo(null);
        // JFrame关闭时的操作
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 显示JFrame
        frame.setVisible(true);

    }
}
```

5.2. 框架中添加组件

上述案例中的框架是空的.可以通过 add 方法在框架中添加组件

```
java.awt.Container.add(Component comp)
```

代码:

```
public static void main(String[] args) {
        // 创建JFrame
        JFrame frame = new JFrame("我的frame");

        // 创建按钮
        JButton button = new JButton("OK");
        // 向frame中添加一个按钮
        frame.add(button);

        // 设置尺寸
        frame.setSize(200, 100);
        // JFrame在屏幕居中
        frame.setLocationRelativeTo(null);
        // JFrame关闭时的操作
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 显示JFrame
        frame.setVisible(true);

    }
```

运行程序,会显示上图所示窗口.调整窗口的大小,按钮都是显示在窗口的中央,并且占满整个框架.这是因为组件(本例就是按钮)是被布局管理器放到框架中的.默认布局管理器就是将按钮放到中央.

备注: 可以通过 f.setVisible(false);　隐藏窗体 f.dispose();关闭窗口
设置图片:

```
setIconImage(Toolkit.getDefaultToolkit().createImage("png-0015.png"));
```

6. JoptionPane 对话框

显示一个带有 OK 按钮的模态对话框。

下面是几个使用 showMessageDialog 的例子:

Java 代码

```
JOptionPane.showMessageDialog(null, "错误信息提示", "标题",
JOptionPane.INFORMATION_MESSAGE);
```

效果如下:

Java 代码

```
1.  JOptionPane.showMessageDialog(jPanel, "提示消息", "标题",JOptionPane.WARNING_
    MESSAGE);
```

效果如下:

Java 代码

```
1.  JOptionPane.showMessageDialog(null, "提示消息.", "标题",JOptionPane.ERROR_MES
    SAGE);
```

Java 代码

```
1.  JOptionPane.showMessageDialog(null, "提示消息.", "标题",JOptionPane.PLAIN_MES
    SAGE);
```

1.2 showOptionDialog

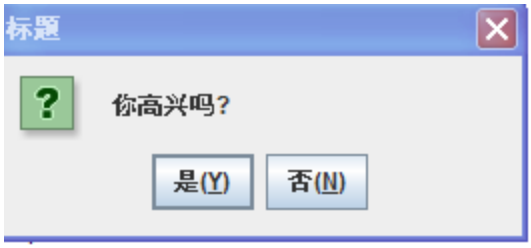
这个函数可以改变显示在按钮上的文字。你还可以执行更多的个性化操作。

常规的消息框：

Java 代码

```
1.  int n = JOptionPane.showConfirmDialog(null, "你高兴吗?", "标题",JOptionPane.Y
    ES_NO_OPTION);//i=0/1
```

效果如下：



输入框 ：

String inputValue = JOptionPane.showInputDialog("请输入你给我金额");

7． 面板 (Panel)

面板也是一个容器的组件，可以在上面添加

注意：面板不能单独使用，必须在顶层窗口中使用。

8． 常见组件

一些常用的组件例如：
JLabel,
JButton ,
JTextField
JPassworldField
JRadioButton
JCheckBox
JTextArea
JList
JMenuBar
JMenu
JMenultem

8.1. Button 按钮

Java 中的 Swing 提供了常规按钮,单选按钮,复选按钮和菜单按钮

8.1.1. JButtion 普通按钮

按钮 Button 是点击时触发动作事件的组件.

8.1.2. JRadioButton 单选按钮

单选按钮,可以让用户从一组选项中选择一个单一条目.例如性别.
使用单选按钮时注意将,单选按钮放在一组,需要使用 java.swing.ButtonGroup. 的 add 方法,添加到一个组中,位于同一个组的单选按钮就是互斥的. 如果没有将单选按钮放在一个组中,就是独立的..我们让然需要把按钮添加在容器中.因为 ButtonGroup 添加到容器中..

8.1.3. JCheckBox 复选框

多选

8.2. JLabel 标签

8.3. JTextField 文本域

8.4. JTextArea 文本区域

8.5. JComboBox 组合框

8.6. JList 列表框

例如:
```
String[] data = { "one", "two", "three" };
        JList list = new JList(data);
        p1.add(list);
```

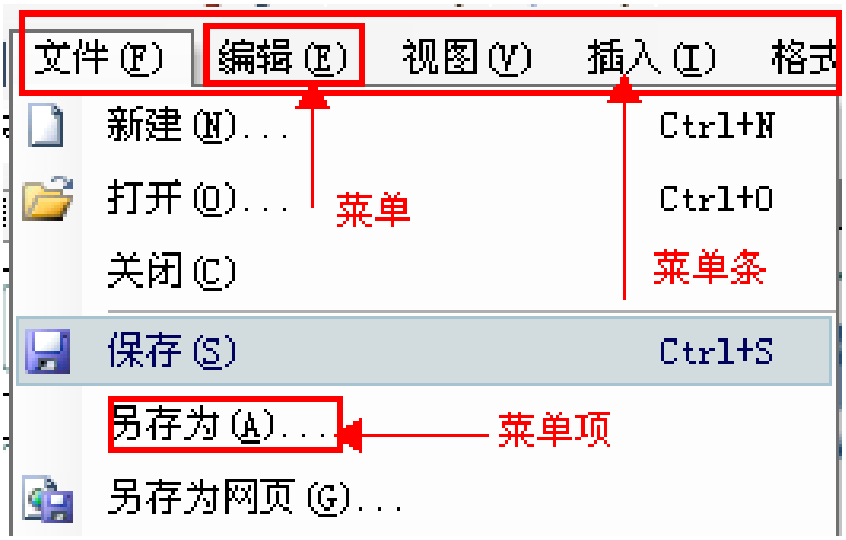
8.7. JmenuBar 菜单条

JMenu 菜单

JMenuItem 菜单项

菜单条(MenuBar)中包含菜单(Menu),菜单中中包含菜单项(MenuItem)

注意添加的顺序。例如：记事本的菜单条中包含文件、编辑、格式、查看、帮助菜单。其中文件菜单中包含新建、打开、保存、另存为等菜单项



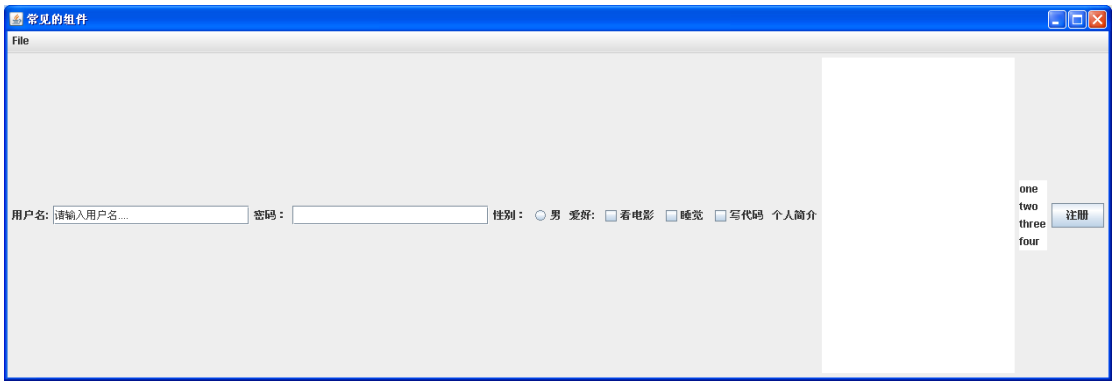
案例综合。

```java
public class CommonComponent extends JFrame {
    public CommonComponent() {

        // 面板
        JPanel p1 = new JPanel();
        add(p1);

        // 标签
        JLabel name = new JLabel("用户名:");
        p1.add(name);

        // 文本域
        JTextField field = new JTextField(8);
        p1.add(field);

        // 标签
        JLabel passwd = new JLabel("密码");
        p1.add(passwd);
        // 密码域
        JPasswordField pass = new JPasswordField(8);
        p1.add(pass);

        // 单选按钮
        JLabel gender = new JLabel("性别");
        p1.add(gender);
        JRadioButton male = new JRadioButton("男");
        JRadioButton female = new JRadioButton("女");
        // 单选按钮组,同一个单选按钮组的互斥.
        ButtonGroup group = new ButtonGroup();
        group.add(male);
        group.add(female);
        // 注意,单选按钮组不能添加进容器
        p1.add(male);
        p1.add(female);

        // 复选框
        JLabel like = new JLabel("爱好:");
        p1.add(like);
        JCheckBox eat = new JCheckBox("吃饭");
        JCheckBox movie = new JCheckBox("看电影");
        JCheckBox sleep = new JCheckBox("睡觉");
        p1.add(eat);
        p1.add(movie);
        p1.add(sleep);
```

```
        // 文本域
        JLabel info = new JLabel("个人简介");
        p1.add(info);
        JTextArea area = new JTextArea(20, 20);
        p1.add(area);

        // 列表
        String[] data = { "one", "two", "three" };
        JList list = new JList(data);
        p1.add(list);

        // 普通按钮
        JButton button = new JButton("注册");
        p1.add(button);

        // 菜单条
        JMenuBar bar = new JMenuBar();
        // 菜单
        JMenu menu = new JMenu("文件");
        // 菜单选项
        JMenuItem myNew = new JMenuItem("新建");
        JMenuItem myOpen = new JMenuItem("打开");
        bar.add(menu);
        menu.add(myNew);
        menu.add(myOpen);
        add(bar, BorderLayout.NORTH);

    }

    public static void main(String[] args) {
        CommonComponent frame = new CommonComponent();
        frame.setTitle("常用组件");
        frame.setSize(400, 400);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 自适应
        frame.pack();
        frame.setVisible(true);
    }
}
```

样式如下：

因为面板默认是流式布局.



9. 布局管理器

Java 的 GUI 组件都放置在容器中,他们的位置是由容器的布局管理器来管理的.在前面的程序中,并没有指定将 OK 按钮放置在框架的什么位置,但是,Java 知道应该把它放置在哪里,因为在后台工作的布局管理器能够将组件放到正确的位置.布局管理器是使用布局管理器类创建的.

我们可以使用 setLayout() 方法在容器中设置布局管理器.

我们将要了解 FlowLayout GridLayout BorderLayout

9.1. FlowLayout 流式布局

FlowLayout(流式布局)是最简单布局管理器. Jpanel 容器默认的布局管理器

流式布局,按照组件添加的顺序,从左到到右将组件排列在容器中.当放满一行,就开始新的一行.在 FlowLayout 有 3 个常量 FlowLayout 可以指定组件的对齐方式.

LEFT 每一行组件都应该是左对齐的

RIGHT 每一行组件都应该是右对齐的

CENTER 每一行组件都应该是居中的

还可以指定组件之间的以像素为单位的间隔.

```
int getHgap()
        获取组件之间以及组件与 Container 的边之间的水平间隙。
int getVgap()
        获取组件之间以及组件与 Container 的边之间的垂直间隙。
void setHgap(int hgap)
        设置组件之间以及组件与 Container 的边之间的水平间隙。
void setVgap(int vgap)
        设置组件之间以及组件与 Container 的边之间的垂直间隙。
```

这个布局管理器的对其方式默认值是 CENTER

这个布局管理器的水平间隔默认值是 5 个像素

这个布局管理器的垂直间隔默认是是 5 个像素
创建该布局管理器

FlowLayout()
构造一个新的 FlowLayout，它是居中对齐的，默认的水平和垂直间隙是 5 个单位。
FlowLayout(int align)
构造一个新的 FlowLayout，它具有指定的对齐方式，默认的水平和垂直间隙是 5 个单位。
FlowLayout(int align, int hgap, int vgap)
创建一个新的流布局管理器，它具有指定的对齐方式以及指定的水平和垂直间隙。

案例:创建框架,使用流失布局管理器.向该框架添加三个标签和文本域.

```
public class ShowFlowLayout extends JFrame {

    public ShowFlowLayout() {
        super.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));

        add(new JLabel("姓名:"));
        add(new JTextField(8));
        add(new JLabel("邮箱:"));
        add(new JTextField(8));
        add(new JLabel("电话:"));
        add(new JTextField(8));

    }

    public static void main(String[] args) {
        ShowFlowLayout frame = new ShowFlowLayout();
        frame.setTitle("FlowLayout");
        frame.setSize(500, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

该案例在本类 main 方法中创建了一个本类对象. 该类的构造函数中创建并且添加组件.该案例使用了 FlowLayout 管理器在框架放置组件.如果改变框架的大小.组建会自动的重新排列,以适应框架.

例如:

如果将同一个按钮在框架中添加 10 次,那么该框架只会出现一次,将一个按钮向容器中添加多以和一次是一样的.

9.2. GridLayout 网格布局

GridLyaout 是以网格形式管理组件的.组件按照他们添加的顺序从左到右排列,显示第一行,接着是第二行,一次类推.
Gridlayout 可以指定网格中的行数和列数
规则如下:
行数和列数可以是 0 但是不能两者都为 0.
　　如果一个为 0 另外一个不为 0,那么不为 0 的行或列的大小就是固定的,为 0 的行或者列有布局管理器动态决定.
例如:如果指定一个网格有 0 行 3 列 10 个组件,GirdLayout 会创建 3 个固定的列和行,最后一行只有一个组件.如果指定一个网格有 3 行 0 列 10 个组件,GridLayout 就会创建 3 行 4 列,最后一行包含 2 个组件.
　　如果行数和列数都不为 0,那么以行数为依据.所以行数是固定的,布局管理器会动态的计算列数.例如,如果指定一个网格有 3 行 3 列 10 个组件,GridLayout 会创建 3 个固定的行和 4 列,最后一行包含 2 个组件.

构造方法

```
GridLayout()
        创建具有默认值的网格布局，即每个组件占据一行一列。
GridLayout(int rows, int cols)
        创建具有指定行数和列数的网格布局。
GridLayout(int rows, int cols, int hgap, int vgap)
        创建具有指定行数和列数,水平间隔,垂直间隔的网格布局。
```

方法:

```
int getRows()
        获取此布局中的行数。 默认值是 1
int getColumns()
```

获取此布局中的列数。 默认值是 1

int getHgap()

获取组件之间的水平间距。 默认值是 0

int getVgap()

获取组件之间的垂直间距。默认值是 0

设置

void setRows(int rows)

将此布局中的行数设置为指定值。默认值是 1

void setColumns(int cols)

将此布局中的列数设置为指定值。 默认值是 1

void setHgap(int hgap)

将组件之间的水平间距设置为指定值。 默认值是 0

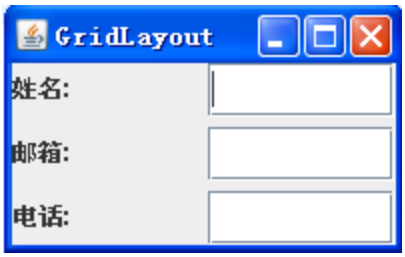
void setVgap(int vgap)

将组件之间的垂直间距设置为指定值。默认值是 0

案例:该案例依然添加 3 个标签和 3 个文本域,只不过布局管理器是 GrigLayout

```
public class ShowGridLayout extends JFrame {
    public ShowGridLayout() {
        setLayout(new GridLayout(3, 2, 5, 5));

        add(new JLabel("姓名:"));
        add(new JTextField(8));
        add(new JLabel("邮箱:"));
        add(new JTextField(8));
        add(new JLabel("电话:"));
        add(new JTextField(8));
    }

    public static void main(String[] args) {
        ShowGridLayout frame = new ShowGridLayout();
        frame.setTitle("GridLayout");
        frame.setSize(200, 125);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

例如：



如果使用 setLayout(new GridLayout(3,10))替换 setlayout 语句,还是会得到 3 行 2 列,因为行的参数非零,所以列的参数会被忽略.列的实际参数是由布局管理器计算出来的.

9.3. BorderLayout 边框布局

边框布局,JFrame 容器默认的布局管理器是边框布局.该管理器将容器非为东西南北中 5 个区域.我们使用 add(Component,index)方法可以将组件添加进到 BorderLayout 中,index 是一个常量,有 5 个值

EAST 东区域的布局约束（容器右边）。 WEST 西区域的布局约束（容器左边）。 SOUTH 南区域的布局约束（容器底部）。 NORTH 北区域的布局约束（容器顶部）。 CENTER 中间区域的布局约束（容器中央）。

构造函数

BorderLayout() 构造一个组件之间没有间距的新边框布局。 BorderLayout(int hgap, int vgap) 构造一个具有指定组件间距的边框布局。

方法：

int getHgap() 返回组件之间的水平间距。 int getVgap() 返回组件之间的垂直间距。 void setHgap(int hgap) 设置组件之间的水平间距。 默认值是 0

```
void setVgap(int vgap)
        设置组件之间的垂直间距。默认值是 0
```

组件会根据他们最合适的尺寸和在容器中的位置来放置,南北组件可以水平拉伸,东西组件可以垂直拉伸,中央组件既可以水平拉伸也可以垂直拉伸.

案例:演示边框布局管理器

```
public class ShowBorderLayout extends JFrame {
    public ShowBorderLayout() {
        setLayout(new BorderLayout(5, 10));

        add(new JButton("东"), BorderLayout.WEST);
        add(new JButton("西"), BorderLayout.EAST);
        add(new JButton("南"), BorderLayout.SOUTH);
        add(new JButton("北"), BorderLayout.NORTH);
        add(new JButton("中"), BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        ShowBorderLayout frame = new ShowBorderLayout();
        frame.setTitle("BorderLayout");
        frame.setSize(300, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

注意: 如果布局管理器为 Border 管理器,调用 add 方法,没有说明组件的位置(东西南北中)默认是 center.

所以

add(new JButton("ok"))和 add(new JButton("ok"), BorderLayout.*CENTER*)
效果是一样的.
如果在 Border 容器中添加 2 个组件,都没有指定位置,那么只会显示最后一个组件
add(new JButton("ok1"))和 add(new JButton("ok2")) 只会显示 ok2

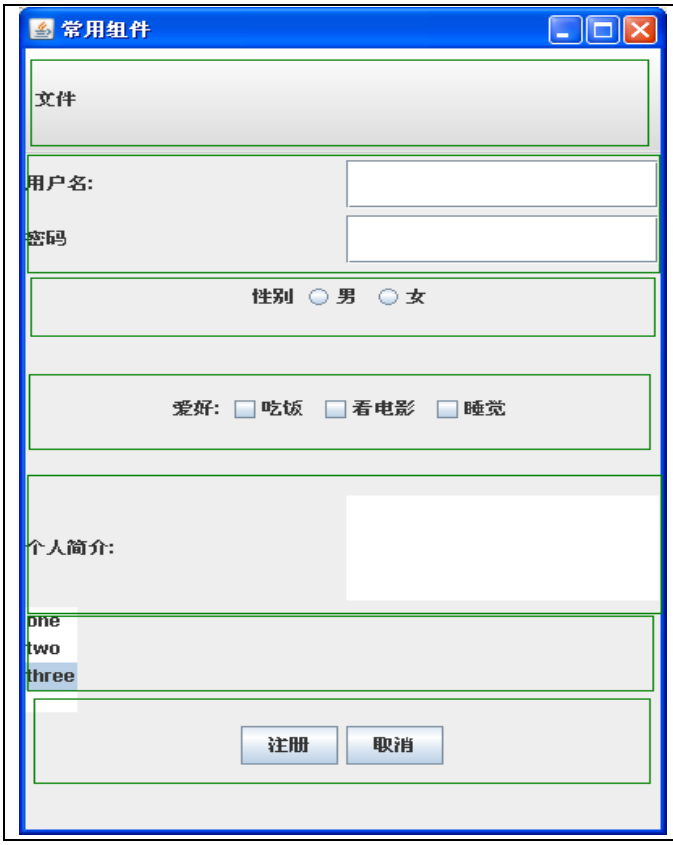
使用布局管理器
代码:

```
public class CommonComponent extends JFrame {
    public CommonComponent() {
        // 菜单条
        JMenuBar bar = new JMenuBar();
        // 菜单
        JMenu menu = new JMenu("文件");
        // 菜单选项
        JMenuItem myNew = new JMenuItem("新建");
        JMenuItem myOpen = new JMenuItem("打开");
        bar.add(menu);
        menu.add(myNew);
        menu.add(myOpen);
        add(bar, BorderLayout.NORTH);

        // 面板
        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(2, 2, 5, 5));
        add(p1);

        // 标签
        JLabel name = new JLabel("用户名:");
        p1.add(name);

        // 文本域
        JTextField field = new JTextField(8);
        p1.add(field);

        // 标签
        JLabel passwd = new JLabel("密码");
        p1.add(passwd);
        // 密码域
        JPasswordField pass = new JPasswordField(8);
        p1.add(pass);

        JPanel p2 = new JPanel();
        // 单选按钮
        JLabel gender = new JLabel("性别");
```

```
        p2.add(gender);
        JRadioButton male = new JRadioButton("男");
        JRadioButton female = new JRadioButton("女");
        // 单选按钮组,同一个单选按钮组的互斥.
        ButtonGroup group = new ButtonGroup();
        group.add(male);
        group.add(female);
        // 注意,单选按钮组不能添加进容器
        p2.add(male);
        p2.add(female);

        JPanel p3 = new JPanel();
        // 复选框
        JLabel like = new JLabel("爱好:");
        p3.add(like);
        JCheckBox eat = new JCheckBox("吃饭");
        JCheckBox movie = new JCheckBox("看电影");
        JCheckBox sleep = new JCheckBox("睡觉");
        p3.add(eat);
        p3.add(movie);
        p3.add(sleep);

        JPanel p4 = new JPanel(new GridLayout(1, 0, 5, 5));
        // 文本域
        JLabel info = new JLabel("个人简介:");
        p4.add(info);
        JTextArea area = new JTextArea(50, 10);
        p4.add(area);

        JPanel p5 = new JPanel(new BorderLayout());
        // 列表
        String[] data = { "one", "two", "three" };
        JList list = new JList(data);
        p5.add(list, BorderLayout.WEST);

        JPanel p6 = new JPanel();
        // 普通按钮
        JButton button = new JButton("注册");
        p6.add(button);
        JButton button2 = new JButton("取消");
        p6.add(button2);

        setLayout(new GridLayout(7, 1, 5, 5));
        add(p1);
```

```
        add(p2);
        add(p3);
        add(p4);
        add(p5);
        add(p6);

    }

    public static void main(String[] args) {
        CommonComponent frame = new CommonComponent();
        frame.setTitle("常用组件");
        frame.setSize(400, 600);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 自适应
        // frame.pack();
        frame.setVisible(true);
    }
}
```

格式如下：



需求:需要在一个 Jframe 框架中放置 10 个按钮和一个文本域.

要求按钮以网格形式放置,文本域单独一行.

如果将这些组件放在一个单独的容器中,很难达要去的效果.我们使用 Java 图形用户界面进行程序设计时,可以将一个窗口分成几个面板 Jpanel.面板的作用就是分组放置用户界面的子容器.这里可以将按钮添加到一个面板中,然后经面板添加到框架中.

在 Swing 中的面板是 JPanel 可以使用 new Jpanel () 创建一个默认是 FlowLayout 管理器的面板,也可以使用 newJPanel(LayoutManager) 创建一个带特定布局管理器的面板.使用 add 方法将组件添加进面板中.

例如:

```
public class ShowJPanel extends JFrame {
    ShowJPanel() {
        JButton button = new JButton("ok");
        JPanel jPanel = new JPanel();
        jPanel.add(button);
        add(jPanel);
    }

    public static void main(String[] args) {
        ShowJPanel frame = new ShowJPanel();
        frame.setTitle("面板");
        frame.setSize(200, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

图示:

进按钮添加进面板面板添加进容器

进按钮直接添加进框架.

案例:

```
public class TestPanel extends JFrame {
    public TestPanel() {
        // 创建面板,默认是流式布局
        JPanel p1 = new JPanel();
        // 指定为网格布局,4行3列
        p1.setLayout(new GridLayout(4, 3));
        for (int i = 1; i <=9; i++) {
            p1.add(new JButton("" + i));
        }
        p1.add(new JButton("0"));
        p1.add(new JButton("OK"));
        p1.add(new JButton("EXIT"));

        // 创建面板,指定边框布局
        JPanel p2 = new JPanel(new BorderLayout());
        // 面板添加文本域,边框北部
        p2.add(new JTextField("我在这里啊!!!"), BorderLayout.NORTH);
        // 面板添加其他面板,边框中部.
        p2.add(p1, BorderLayout.CENTER);
```

```
        // 框架添加面板,框架的布局默认就是边框布局,面板指定位于框架西部
        add(p2, BorderLayout.EAST);
        // 框架添加按钮,位于框架总部.
        add(new JButton("哈哈"), BorderLayout.CENTER);

    }

    public static void main(String[] args) {
        TestPanel frame = new TestPanel();
        frame.setTitle("JPanel");
        frame.setSize(400, 260);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

该案例中创建了两个面板(Jpanel)，面板的默认布局是流式布局.,可以使用setLayout 更改面板的布局(setLayout 是java.awt.Container 定义的，Jpanel是Container 的子类).

该案例中

面板 p1 布局指定为 GridLayout(网格布局)将 12 个按钮,添加进面板 p1 中.

面板 p2 布局指定为 BorderLayout,将 p1 面板添加进来，p1 位于该面板的中部.再添加一个文本域,位于面板 p2 北部.

将面板 p2 添加进框架(框架布局默认为 BorderLayout),位于框架的西部,在框架的中部添加一个按钮.

效果:

注意：面板容器是一个轻量级的容器，该容器不能单独的使用，必须依赖与外层的顶层容器(JFrame)

10.Java 事件监听机制

在上述的程序中,其中菜单条,菜单项，按钮等都是对象,当我们单击对象时,应该能够完成一些任务.例如在程序中通过鼠标操作时,单击,双击,鼠标移入,鼠标移出.能够执行一些任务,在 Java 中我们可以使用事件监听机制,在 Java 的事件监听机制中 ,当事件发生时(点击按钮,移动鼠标等，关闭窗口)会被一类对象发现并处理.

10.1. 事件和事件源

在运行 java 图形用户界面程序时,用户与程序交互,用户执行了某些操作时，会发生一些事情， 那么事件(event)可以定义为程序发生了某些事情的信号.典型用户操作就是用户移动鼠标,点击按钮,敲击键盘等.程序可以选择相应事件或者忽略事件。

能够创建一个事件并触发该事件的组件称为源对象。例如由于按钮能够点击， 那么按钮就是一个源对象,按钮被点击就是一个事件。

一个事件是事件类的实例对象。事件类的根类是 java.util.EventObject。

事件对象包含事件相关的属性，可以使用 EventObject 类中的实例方法 getSource 获得事件的源对象。

EventObject 类的子类可以描述特定类型的事件

例如：

用户动作	源对象	触发的事件类型
点击按钮	JButton	ActionEvent
文本域按回车	JTextField	ActionEvent
窗口打开，关闭，最小化，关闭	Window	WindowEvent
单击，双击，移动，鼠标	Component	MouseEvent
点击单选框	JradioButton	ItemEvent ActionEvent
点击复选框	JcheckBox	ItemEvent ActionEvent

10.2. 监听器

当源对象触发了一个事件,谁来处理这个事件？在 Java 中对此感兴趣的对象会处理它。对此感兴趣的对象称之为监听器（Listener）。

举例来说当我们点击一个按钮，想要按钮执行一些动作，需要一个对象来监控按钮，当点击按钮的事件发生时，该对象就会监听到按钮事件。进而可以执行一些动作。

例如：

Java 中,对象表示的每个事件都是由 java.util 中 EventObject 类的子类,

例如： MouseEvent： 表示鼠标的动作,例如移动光标,单击,双击

KeyEvent： 表示键盘上的按键.

ActionEvent 表示用户采取的用户界面操作,例如点击屏幕上的按钮.

10.3. 事件处理

Java 对组件都有对应的事件监听器，和添加事件监听器方法。
例如： 按钮： 源对象，会触发
体验：当用户点击按钮时，按钮上的信息会发生变化

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ShowButton extends JFrame {
    public ShowButton() {
        JButton button = new JButton("点我");
        add(button);

        // 添加鼠标监听事件
        button.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("按钮被点击");
                Object source = e.getSource();
                JButton button = (JButton) source;
                String text = button.getText();
                if ("按钮被点击".equals(text)) {
                    button.setText("点我");
                } else {
                    button.setText("按钮被点击");
                }
            }
        });
    }

    public static void main(String[] args) {
        ShowButton frame = new ShowButton();
        frame.setTitle("我的框架");
        frame.setSize(400, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

该案例中：构造方法创建了用户界面。只有一个 JFrame 框架和一个按钮。
按钮时事件的源。创建了一个监听器并注册到了按钮。通过匿名内部类的形式创建了监听器类

10.4. 窗口事件

Window 类

```
java.awt
类 Window
java.lang.Object
     java.awt.Component
           java.awt.Container
                java.awt.Window
```

Window 对象是一个顶层窗口。窗口（Window）对应的事件叫做窗口事件（WindowEvent），任何窗口（Window）以及窗口的子类都可能触发窗口事件： 打开窗口，正在关闭窗口，激活窗口，变成非活动窗口，最小化窗口和还原窗口。

Window 添加窗口事件（WindowEvent）监听器的方法

```
void addWindowListener(WindowListener l)
         添加指定的窗口侦听器，以从此窗口接收窗口事件
```

监听 Window 窗口事件（WindowEvent）的监听器：WindowListener

```
java.awt.event
          接口 WindowListener
    用于接收窗口事件的侦听器接口。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时，将调用该侦听器对象中的相关方法
```

WindowListener 接口中定义的方法

```
抽象方法：
void windowActivated(WindowEvent e)
         激活窗口
 void windowClosed(WindowEvent e)
         关闭窗口
 void windowClosing(WindowEvent e)
         正在关闭窗口
 void windowDeactivated(WindowEvent e)
         变为非活动窗口
 void windowDeiconified(WindowEvent e)
         还原窗口
 void windowIconified(WindowEvent e)
         最小化窗口
 void windowOpened(WindowEvent e)
          打开窗口
```

案例： 框架的事件监听处理

```
javax.swing
类 JFrame
java.lang.Object
      java.awt.Component
            java.awt.Container
                  java.awt.Window
                        java.awt.Frame
                              javax.swing.JFrame
```

创建 JFrame，Jfame 是一个框架，属于窗体（Window）体系中的一员，也可以实现窗口的最大化，最小化，关闭，点击窗体，等一系列的操作。那么在用户触发这些事件发生时能够做一些工作，就需要注册事件监听器。

JFrame 是通过 addWindowListener 方法就注册窗体事件监听器，该方法需要接受一个监听器（WindowListener）对象。查找 API 文档，发现 WindowListener 是一个接口，我们需要窗口监听器(WindowListener)的实例对象，所以需要实现该接口，重写 WindowListener 接口的抽象方法。然后创建该实现类对象，作为参数传递给 addWindowListener .

例如： 当像激活窗口这样的窗口事件发生时，windowActivated 方法就会触发。

代码如下：

```
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.JFrame;

public class WindowEventDemo extends JFrame {

    WindowEventDemo() {
        // this对象时JFrame的子类，而JFrame 类是Window体系中的一员所以具备添加窗口事件的方法
        this.addWindowListener(new MyWindow());
    }

    public static void main(String[] args) {
        WindowEventDemo frame = new WindowEventDemo();
        frame.setTitle("我的框架");
        frame.setSize(400, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
```

```
    }
}

// 实现WindowListener
class MyWindow implements WindowListener {
    // 激活窗口
    public void windowActivated(WindowEvent e) {
        System.out.println("激活窗口");
    }

    // 关闭窗口
    public void windowClosed(WindowEvent e) {
        System.out.println("关闭窗口");
    }

    // 正在关闭窗口
    public void windowClosing(WindowEvent e) {
        System.out.println("正在关闭窗口");
    }

    // 变为非活动窗口
    public void windowDeactivated(WindowEvent e) {
        System.out.println("变为非活动窗口");
    }

    // 还原窗口
    public void windowDeiconified(WindowEvent e) {
        System.out.println("还原窗口");
    }

    // 最小化窗口
    public void windowIconified(WindowEvent e) {
        System.out.println(" 最小化窗口");
    }

    // 打开窗口
    public void windowOpened(WindowEvent e) {
        System.out.println("打开窗口");
    }

}
```

总结：Window 类或者 Window 类的任何子类都可能会触发 WindowEvent。因为 JFrame 是 Window 的子类。所以也可以触发 WindowEvent。

10.5. 监听器接口适配器

因为 WindowListener 接口中的方法都是抽象的，所以即使程序中不想关注某些事件，但是还是要实现所以的方法，比较麻烦，为了方便起见，Java 提供了一个针对 WindowListener 接口的实现类，该类中把 WindowListener 接口中的方法全部实现，只不过方法体都为空。例如： 加入我们只对激活窗口感兴趣，那么只需要继承该实现类重写激活窗口的方法即可。简化了代码。

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class AdapterDemo extends JFrame {
    AdapterDemo() {
        addWindowListener(new MyAdapter());
    }

    public static void main(String[] args) {
        AdapterDemo frame = new AdapterDemo();
        frame.setTitle("我的框架");
        frame.setSize(400, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    class MyAdapter extends WindowAdapter {
        public void windowActivated(WindowEvent e) {
            System.out.println("windowActivated....");
        }
    }

}
```

那么这个 WindowAdapter 类就叫做适配器类，是为了简化代码的书写而出现的。

适配器	接口
WindowAdapter	WindowListener
MouserAdapter	MouserListener
KeyAdapter	KeyListener

10.6. 鼠标键盘事件

当在一个组件上按下，释放，点击，移动，或者拖动鼠标时，就会产生鼠标事件。MouseEvent 对象捕获这个事件。

MouseEvent

```
java.awt.event.MouseEvent
```

Java 对鼠标事件提供了 MouseListener 监听器接口，可以监听鼠标的按下，释放，输入，退出和点击动作。

MouseListener

用于接收组件上“感兴趣”的鼠标事件（按下、释放、单击、进入或离开）的侦听器接口

方法：

void mouseClicked(MouseEvent e)

鼠标按键在组件上单击（按下并释放）时调用。

void mouseEntered(MouseEvent e)

鼠标进入到组件上时调用。

void mouseExited(MouseEvent e)

鼠标离开组件时调用。

void mousePressed(MouseEvent e)

鼠标按键在组件上按下时调用。

void mouseReleased(MouseEvent e)

鼠标按钮在组件上释放时调用。

按键事件可以利用键盘来控制和执行一些动作，如果按下、释放一个键就会触发按键事件。KeyEvent 对象可以捕获按键的按下放开和敲击。KeyEvent 提供了 getkeyChar 来获取按键对应的字符。

java.awt.event.KeyEvent

```
char getKeyChar()
```

返回与此事件中的键关联的字符。

Java 提供了 KeyListener 监听器接口来监听按键事件。

KeyListener 接口

用于接收键盘事件（击键）的侦听器接口。

```
void keyPressed(KeyEvent e)
```

按下某个键时调用此方法。

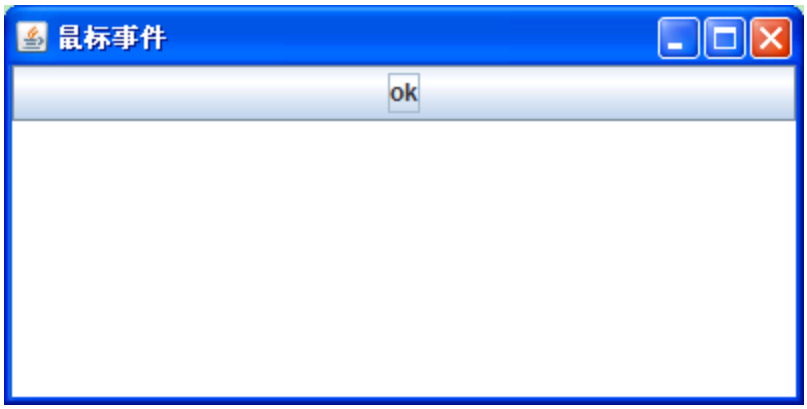
```
void keyReleased(KeyEvent e)
```

释放某个键时调用此方法。

```
void keyTyped(KeyEvent e)
```

键入某个键时调用此方法。

案例：



演示鼠标事件和键盘事件，当在文本域中输入 q 时程序会退出。

```
public class MouseEventDemo extends JFrame {
    MouseEventDemo() {
        JButton button = new JButton("ok");
        JTextArea text = new JTextArea();
        add(button, BorderLayout.NORTH);
        add(text, BorderLayout.CENTER);

        button.addMouseListener(new MouseListener() {

            // 鼠标按钮在组件上释放时调用。
            public void mouseReleased(MouseEvent e) {
                System.out.println("鼠标释放");

            }

            // 鼠标按键在组件上按下时调用。
            public void mousePressed(MouseEvent e) {
                System.out.println("鼠标按下组件");

            }

            // 鼠标离开组件时调用。
            public void mouseExited(MouseEvent e) {
                System.out.println("鼠标离开组件");

            }

            // 鼠标进入到组件上时调用。
            public void mouseEntered(MouseEvent e) {
                // 鼠标进入
```

```
                System.out.println("鼠标进入组件");

            }

            // 鼠标按键在组件上单击（按下并释放）时调用。
            public void mouseClicked(MouseEvent e) {
                System.out.println("鼠标单击组件");

            }
        });
        text.addKeyListener(new KeyListener() {

            // 键入某个键时调用此方法。
            public void keyTyped(KeyEvent e) {
                System.out.println("键入某个键");
                if (e.getKeyChar() == 'q') {
                    System.exit(0);
                }
            }
            // 释放某个键时调用此方法。
            public void keyReleased(KeyEvent e) {
                System.out.println("按键释放");

            }

            // 按下某个键时调用此方法。
            public void keyPressed(KeyEvent e) {
                System.out.println("键盘按下");
            }
        });
    }

    public static void main(String[] args) {
        MouseEventDemo frame = new MouseEventDemo();
        frame.setTitle("鼠标事件");
        frame.setSize(400, 200);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

10.7. 事件监听机制小结

一：确定事件源（容器或组件）

二：注册监听器

通过事件源对象的 addXXXListener()方法将监听器对象注册到该事件源上。

三：监听器对象

注册监听器时，需要指定监听器对象。

以参数的形式进监听器对象传递给 addXXXListener()

监听器对象是 XXXListener 的子类对象或者 XXXAdapter 的子类对象。

监听器对象一般用匿名内部类来表示。（简化书写）

在覆盖方法的时候，方法的参数一般是 XXXEvent 类型的变量接收。

事件触发后会把事件打包成对象传递给该变量。(其中包括事件源对象。通过 getSource() 或者 getComponent()获取。)

四：常见的事件监听器

WindowListener	主要用于监听窗口
ActionListener	主要用于用监听组件对象的单击动作
MouseListener	鼠标监听器
KeyListener	监听键盘

…….

五：常见的事件适配器

WindowAdapter

MouseAdapter

KeyAdapter

11. 作业

完成如下图像程序，在文本框中输入路径，显示该路径下的子文件和子目录。

设计简单的记事本程序。