Java语言程序设计（第2版）

第7章 图形用户界面（一）

郑　莉

导学

本章主要内容

- 绘图
- Swing基础
- Swing的层次
- 事件处理
- Swing组件
- 其它Swing特性
- 桌面API简介

（说明：本章部分例题引用自The Java Tutorials http://docs.oracle.com/javase/tutorial/）

绘图

绘图

- 图形环境和图形对象
- 颜色和字体
- 使用Graphics类绘图
- 使用Graphics2D类绘图

图形环境和图形对象

- 坐标
  - GUI组件的左上角坐标默认为（0，0）。
  - 从左上角到右下角，水平坐标x和垂直坐标y增加。
  - 坐标的单位是象素。
- Graphics对象
  - 专门管理图形环境。Graphics类是一个抽象类。
  - 抽象类Graphics提供了一个与平台无关的绘图接口。
  - 各平台上实现的Java系统将创建Graphics类的一个子类，来实现绘图功能，但是这个子类对程序员是透明的。
  - 在执行paint方法时，系统会传递一个指向特定平台的Graphics子类的图形对象g。

颜色

- Java中有关颜色的类是Color类，它在java.awt包中，声明了用于操作颜色的方法和常量

颜色

- Color类，以及Graphics类中与颜色有关的方法

名称	描述
public final static Color GREEN	常量 绿色
public final static Color RED	常量 红色
public Color(int r,int g,int b)	通过指定红、蓝、绿颜色分量（0～255），创建一种颜色
public int getRed()	返回某颜色对象的红色分量值(0～255)
Graphics：public void setColor(Color c)	Graphics类的方法，用于设置组件的颜色
Graphics：public Color getColor()	Graphics类的方法，用于获得组件的颜色

字体

- Font类——有关字体控制，在java.awt包中

字体

- Font类，以及Graphics类中与字体有关的方法

名称	描述
public final static int PLAIN	一个代表普通字体风格的常量
public final static int BOLD	一个代表黑体字体风格的常量
public final static int ITALIC	一个代表斜体字体风格的常量
public Font(String name,int style,int size)	利用指定的字体、风格和大小创建一个Font对象
public int getStyle()	返回一个表示当前字体风格的整数值
public Boolean isPlain()	测试一个字体是否是普通字体风格
Graphics：public Font getFont()	获得当前字体
Graphics：public void setFont(Font f)	设置当前字体为f指定的字体、风格和大小

Graphics类

- Graphics类对象可以绘制文本、线条、矩形、多边形、椭圆、弧等多种图形——这一行文字不显示

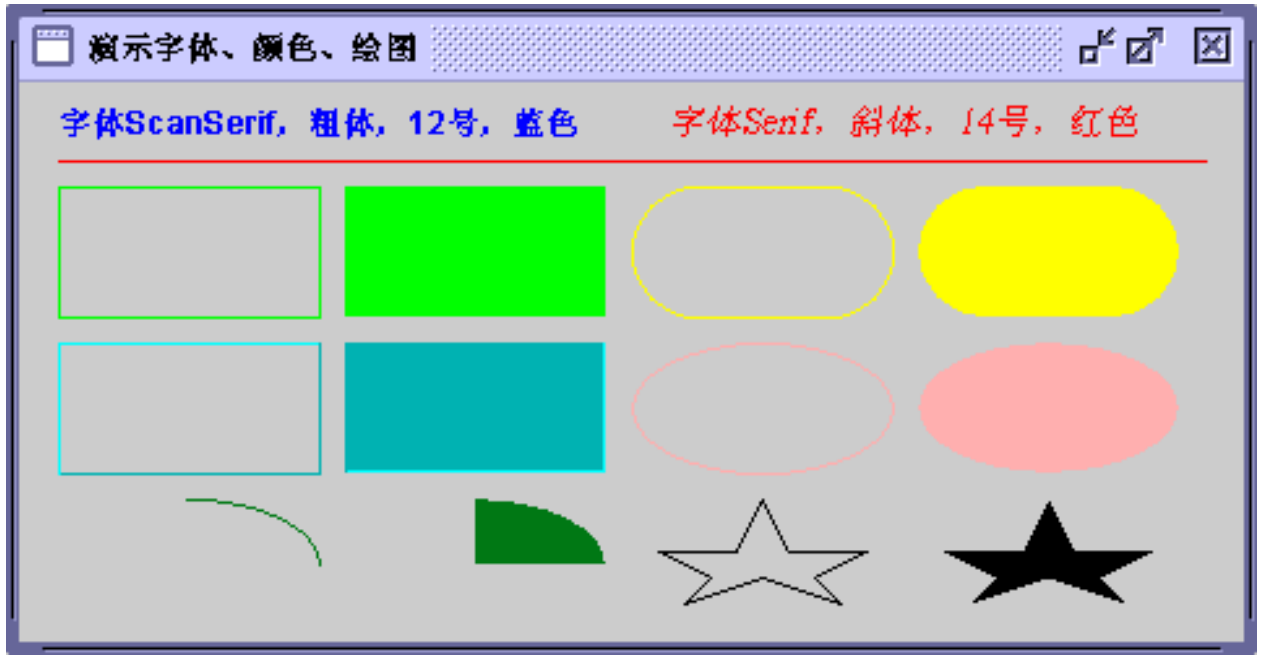
Graphics类常用方法

名称	描述
public void drawString(String str, int x, int y)	绘制字符串，左上角的坐标是（x,y）
public void drawLine(int x1, int y1, int x2, int y2)	在(x1,y1)与(x2,y2)两点之间绘制一条线段
public void drawRect(int x, int y, int width, int height)	用指定的width和height绘制一个矩形，该矩形的左上角坐标为(x,y)
public void fillRect(int x, int y, int width, int height)	用指定的width和height绘制一个实心矩形，该矩形的左上角坐标为(x,y)

public void clearRect(int x, int y, int width, int height)	用指定的width和height，以当前背景色绘制一个实心矩形。该矩形的左上角坐标为（x,y）
public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	用指定的width和height绘制一个圆角矩形，圆角是一个椭圆的1/4弧，此椭圆由arcWidth、arcHeight确定两轴长。其外切矩形左上角坐标为（x,y）
public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	用当前色绘制实心圆角矩形，各参数含义同drawRoundRect。
public void draw3DRect(int x, int y, int width, int height, boolean b)	用指定的width和height绘制三维矩形，该矩形左上角坐标是(x,y)，b为true时，该矩形为突出的，b为false时，该矩形为凹陷的。
public void fill3DRect(int x, int y, int width, int height, boolean b)	用当前色绘制实心三维矩形，各参数含义同draw3DRect。

public void drawPolygon(int[] xPoints, int [] yPoints, int nPoints)	用xPoints，yPoints数组指定的点的坐标依次相连绘制多边形，共选用前nPoints个点。
public void fillPolygon(int[] xPoints, int [] yPoints, int nPoints)	绘制实心多边形，各参数含义同drawPolygon。
public void drawOval(int x, int y, int width, int height)	用指定的width和height，以当前色绘制一个椭圆，外切矩形的左上角坐标是(x,y)。
public void fillOval(int x, int y, int width, int height)	绘制实心椭圆，各参数含义同drawOval。
public void drawArc(int x, int y,int width, int height, int startAngle, int arcAngle)	绘制指定width和height的椭圆，外切矩形左上角坐标是(x,y)，但只截取从startAngle开始，并扫过arcAngle度数的弧线。
public void fillArc(int x, int y,int width, int height, int startAngle, int arcAngle)	绘制一条实心弧线（即扇形），各参数含义同drawArc

例：使用Graphics类绘图



例：使用Graphics类绘图

```
import java.awt.*;
import javax.swing.*;

public class GraphicsTester  extends JFrame {
   public GraphicsTester ()
   {  super(  “演示字体、颜色、绘图”  );
      setVisible( true ); //显示窗口
      setSize( 480, 250 ); //设置窗口大小
   }
   public void paint( Graphics g )  {
      super.paint( g );
      g.setFont( new Font( "SansSerif", Font.BOLD, 12 ) );
      g.setColor(Color.blue); //设置颜色
      g.drawString("字体ScanSerif，粗体，12号，蓝色",20,50);

      g.setFont( new Font( "Serif", Font.ITALIC, 14 ) );
      g.setColor(new Color(255,0,0));
      g.drawString( " 字体Serif，斜体，14号，红色", 250, 50 );

      g.drawLine(20,60,460,60); //绘制直线
```

运行结果

Java2D API

- 提供了高级的二维图形功能。

Java2D API

- 分布在java.awt、java.awt.image、java.awt.color、java.awt.font、java.awt.geom、java.awt.print和java.awt.image.renderable包中。
- 它能轻松使你完成以下功能：
  - 可以很容易绘制各种形状；
  - 可以控制笔画：粗细、端头样式、虚线；
  - 可以用单色、渐变色和纹理填充形状；
  - 平移、旋转、伸缩、切变二维图形，对图像进行模糊、锐化等操作；
  - 构建重叠的文本和图形；
  - 可以对形状进行剪切，将其限制在任意区域内；
  - ……

Graphics2D类

- 是Graphics类的抽象子类；
- 要使用Java2D API，就必须建立该类的对象。

Graphics2D类

- Graphics类的抽象子类。
- 传递给paint方法的对象是Graphics2D的一个子类实例，被向上转型为Graphics类的实例。要访问Graphics2D功能，必须将传递给paint方法的Graphics引用强制转换为Graphics2D引用：

```
Graphics2D g2d=(Graphics2D)g
```

例：使用Graphics2D类绘图

使用Java2D使文字出现渐变色效果

```
import java.awt.*;
import javax.swing.*;
public class Graphics2DTester extends JApplet{
    public void paint(Graphics g) {
      super.paint(g);
      Graphics2D g2d=(Graphics2D)g;
      g2d.setPaint(new GradientPaint(0,0,Color.red,180,45,Color.yellow));
      g2d.drawString("This is a Java Applet!",25,25);
    }
}
```

运行结果：

结束语

Swing基础

前面介绍了如何在屏幕上绘制普通的图形，但如果需要绘制一个按钮，并使其可以对点击事件作出响应，就需要使用java Swing提供的组件

其实前面我们已经用到的JFrame、JApplet都是Swing组件，它们分别代表窗口组件和Applet容器组件

JFC与Swing

- JFC（Java Foundation Classes）
  - 是关于GUI 组件和服务的完整集合。
  - 作为JAVA SE 的一个有机部分，主要包含5 个部分
    - AWT
    - Java2D
    - Accessibility
    - Drag & Drop
    - Swing
- Swing
  - JFC 的一部分。
  - 提供按钮、窗口、表格等所有的组件。
  - 纯Java组件（完全用Java写的）。

AWT组件

- 在java.awt包里，包括Button、Checkbox、Scrollbar等，都是Component类的子类。
- 大部分含有native code，所以随操作系统平台的不同会显示出不同的样子，而不能进行更改，是重量级组件。

Swing组件

- 其名称都是在原来AWT组件名称前加上J，例如JButton、JCheckBox、JScrollbar等，都是JComponent类的子类。
- 架构在 AWT 之上，是AWT的扩展而不是取代。
- 完全是由java语言编写的，其外观和功能不依赖于任何由宿主平台的窗口系统所提供的代码，是轻量级组件。
- 可提供更丰富的视觉感受。

在Applet和Application中应用Swing

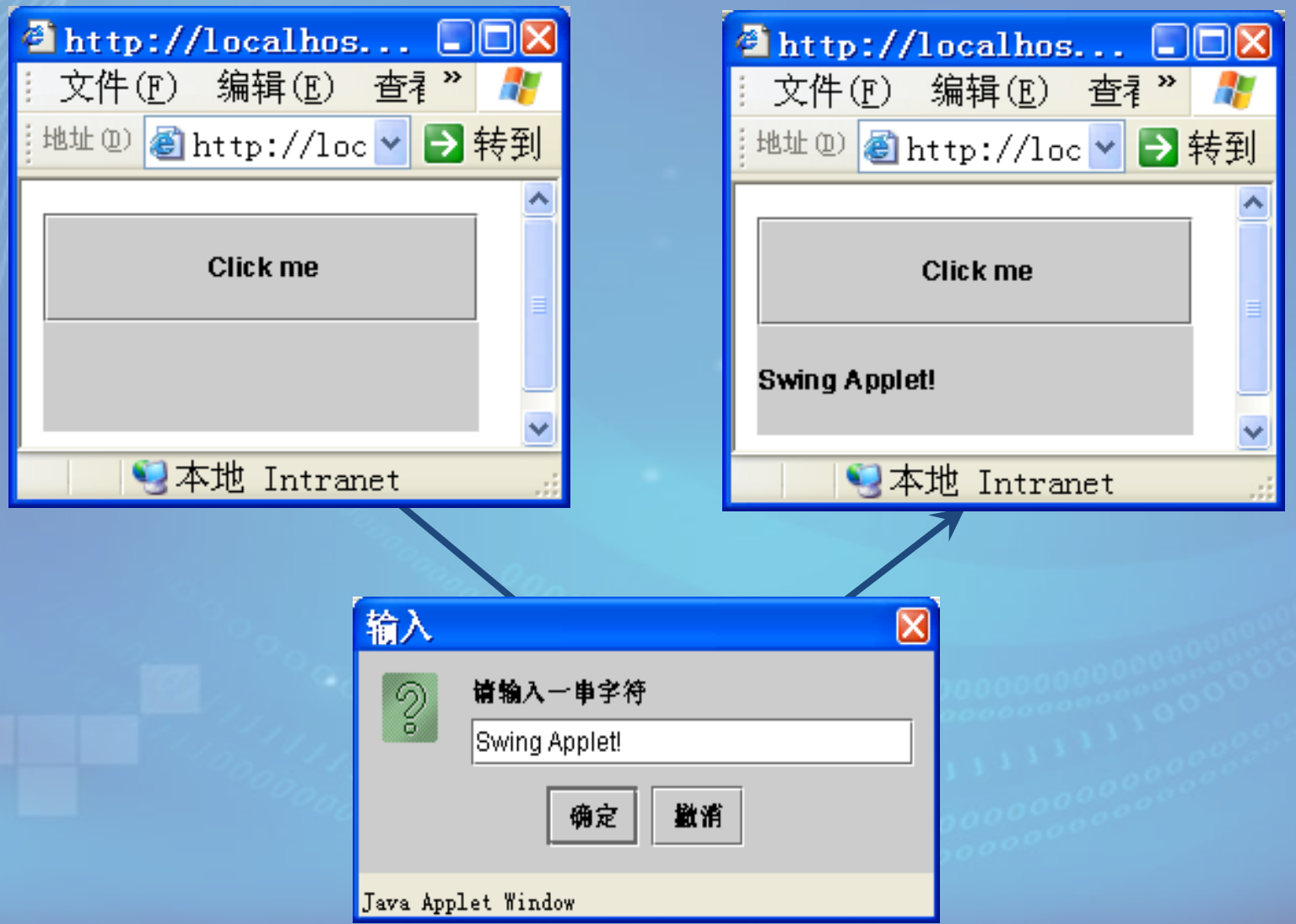
- 在Applet中应用Swing，就是要将Swing组件加载到Applet容器上（通常是JApplet），这通常在init方法中完成；
- 在Application中应用Swing，也是要将Swing组件加载到这个Application的顶层容器（通常是JFrame）中 。

例：在Applet中应用Swing

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingApplet extends JApplet{
   public void init() {
      Container contentPane=getContentPane();
      contentPane.setLayout(new GridLayout(2,1));
      JButton button=new JButton("Click me");
      final JLabel label=new JLabel();
      contentPane.add(button);
      contentPane.add(label);
      button.addActionListener( new ActionListener(){
         public void actionPerformed(ActionEvent event) {
           String information=JOptionPane.showInputDialog("请输入一串字符");
           label.setText(information);
          } } );//创建监听器语句结束
  }//init方法结束
}
```

例：在Applet中应用Swing (续)

运行结果

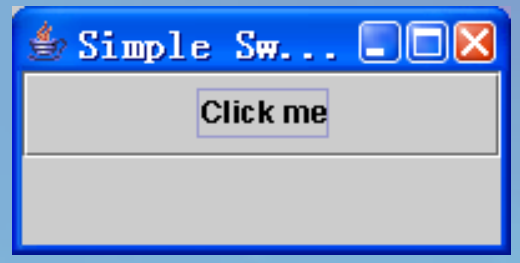


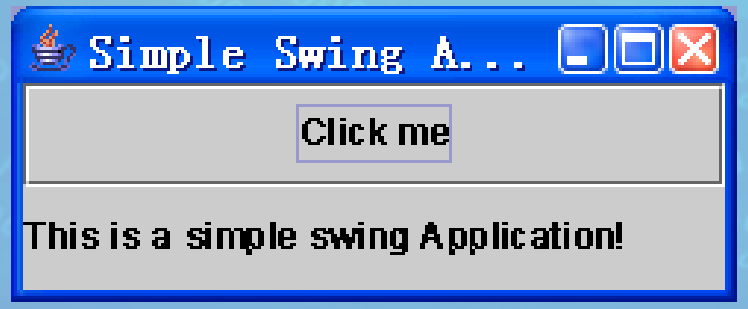
例：在Application中应用Swing

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class SwingApplication {
    public static void main(String[] args){
        JFrame f=new JFrame( “Simple Swing Application”  );
        Container contentPane=f.getContentPane();
        contentPane.setLayout(new GridLayout(2,1));
        JButton button=new JButton("Click me");
         final JLabel label=new JLabel();
         contentPane.add(button);//添加按钮
         contentPane.add(label);//添加标签
        button.addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent event) {
                String information=JOptionPane.showInputDialog("请输入一串字符");
                label.setText(information);
            } } );
        f.setSize(200,100);//设置大小
        f.show();//显示
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

例：在Application中应用Swing (续)

- 运行结果

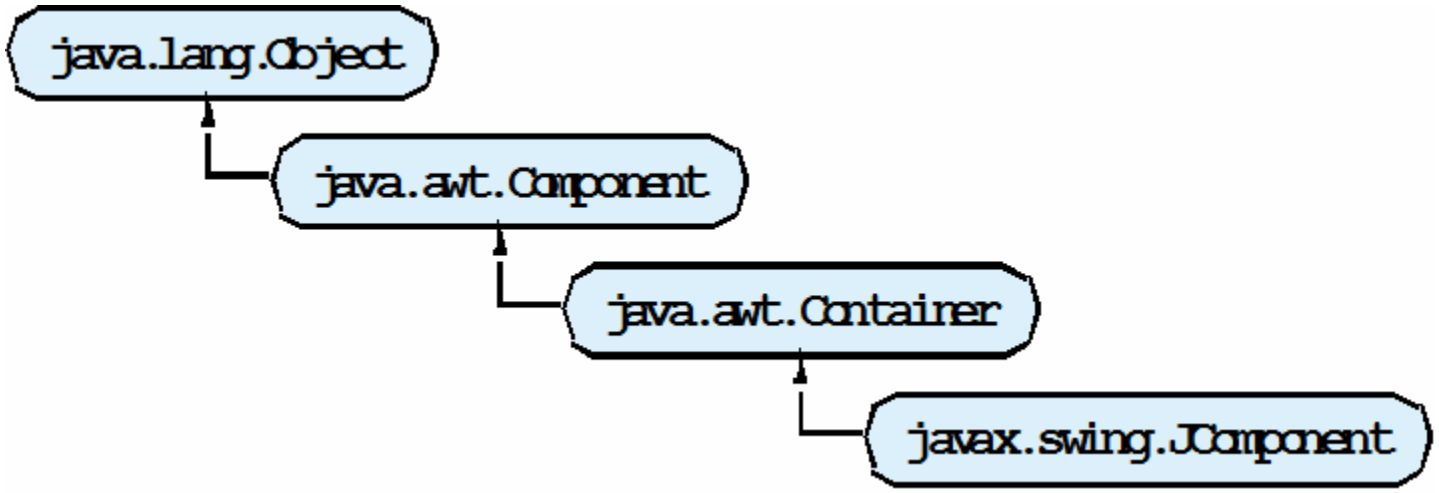




结束语

Swing的层次

多数Swing组件的继承层次



JComponent类是除了顶层容器以外所有Swing组件的超类

- Component 类
  - 包含paint、repaint方法，可以在屏幕上绘制组件。
  - 大多数GUI组件直接或间接扩展Component。
- Container 类
  - 容纳相关组件。
  - 包括add方法，用来添加组件。
  - 包括setLayout方法，用来设置布局，帮助Container对象对其中的组件进行定位和设置组件大小。
- JComponent 类——多数Swing组件的超类
  - 可定制的观感，即可根据需求定制观感。
  - 快捷键 (通过键盘直接访问GUI组件)。
  - 一般的事件处理功能。

Swing的组件和容器层次

- 顶层容器
- 中间层容器
- 原子组件

顶层容器

- JFrame
- JDialog
- JApplet

顶层容器

- Swing的三个顶层容器的类
  - JFrame 实现单个主窗口。
  - JDialog 实现一个二级窗口(对话框)。
  - JApplet 在浏览器窗口中实现一个applet显示区域。
- 必须和操作系统打交道，所以都是重量级组件。
- 从继承结构上来看，它们分别是从原来AWT组件的Frame、Dialog和Applet类继承而来。
- 每个使用Swing组件的Java程序都必须至少有一个顶层容器，别的组件都必须放在这个顶层容器上才能显现出来。

中间层容器

- 是为了容纳别的组件。

中间层容器

- 分为两类
  - 一般用途的
    - JPanel
    - JScrollPane
    - JSplitPane
    - JTabbedPane
    - JToolBar
  - 特殊用途的
    - JInternalFrame
    - JRootPane
- 可以直接从顶层容器中获得一个JRootPane对象来直接使用，而别的中间容器使用的时候需要新建一个对象。

原子组件

- 通常是在图形用户界面中和用户进行交互的组件。

原子组件

- 显示不可编辑信息的
  - 例如：JLabel、JProgressBar、JToolTip
- 有控制功能、可以用来输入信息的
  - 例如：JButton、JCheckBox、JRadioButton、JComboBox、JList、JMenu、JSlider、JSpinner、JTexComponent等
- 能提供格式化的信息并允许用户选择的
  - 例如：JColorChooser、JFileChooser、JTable、JTree

例：三层容器结构

例：三层容器结构

```
import javax.swing.*;
import java.awt.*;
public class ComponentTester {
  public static void main(String[] args){
    JFrame.setDefaultLookAndFeelDecorated(true);
    JFrame frame=new JFrame("Swing Frame");
    Container contentPane=frame.getContentPane();
    JPanel panel=new JPanel();
     panel.setBorder(BorderFactory.createLineBorder(Color.black,5));
    panel.setLayout(new GridLayout(2,1));
    JLabel label=new JLabel("Label",SwingConstants.CENTER);
    JButton button=new JButton("Button");
    panel.add(label);
    panel.add(button);
    contentPane.add(panel);
    frame.pack();//对组件进行排列
    frame.show();//显示
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

运行结果：

常用Swing组件

Box	BoxLayout	JButton
JCheckBox	JCheckBoxMenuItem	JComboBox
JComponent	JDesktopPane	JDialog
JEditorPane	JFrame	JInternalFrame
JLabele	JLayeredPane	JList
JMenu	JMenuBar	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane
JSeparator	JSlider	JSplitPane
JTabbedPane	JTable	JTextArea
JTextField	JTextPane	JToggleButton
JToolBar	JToolTip	JTree
JViewport	JMenuItem	JOptionPane
JPasswordField	JPopupMenu	JProgressBar
JRadioButton	JWindow	OverlayLayout
ProgressMonitor	ProgressMonitorInputStream	Timer
UIDefaults	UIManager	

结束语

布局管理

如何将下级组件有秩序地摆在上一级容器中？

- 在程序中具体指定每个组件的位置
- 使用布局管理器（Interface LayoutManager）

布局管理器

- 调用容器对象的setLayout方法，并以布局管理器对象为参数，例如：

```
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

- 使用布局管理器可以更容易地进行布局，而且当改变窗口大小时，它还会自动更新版面来配合窗口的大小，不需要担心版面会因此混乱。

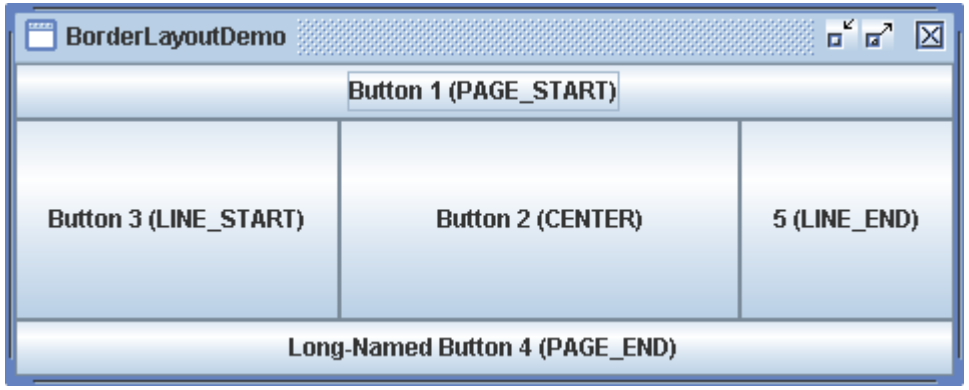
常用的布局管理器类

- 在Java中有很多实现LayoutManager接口的类，经常用到的有以下几个
  - BorderLayout
  - FlowLayout
  - GridLayout
  - CardLayout
  - GridBagLayout
  - BoxLayout
  - SpringLayout
  - 内容面板（content pane）默认使用的就是BorderLayout，它可以将组件放置到五个区域：东、西、南、北、中。

布局示例

布局示例： BorderLayout

- 将组件放置到五个区域：东、西、南、北、中

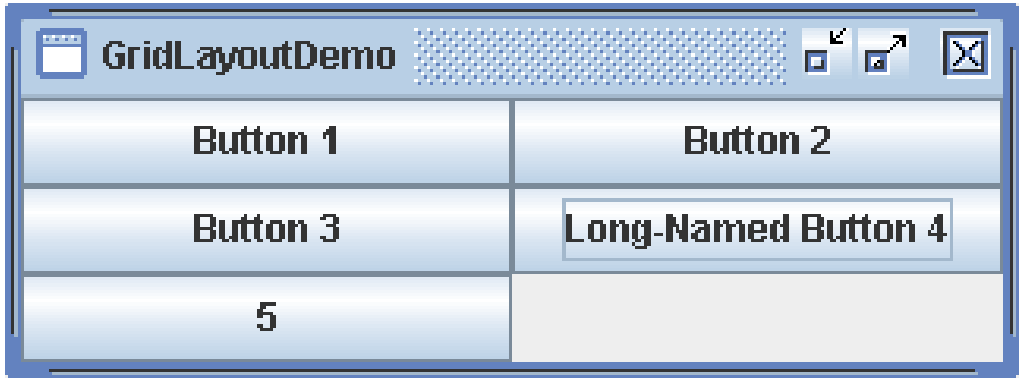


布局示例：FlowLayout

- 是JPanel默认使用的布局管理器，它只是简单地把组件放在一行，如果容器不是足够宽来容纳所有组件，就会自动开始新的一行

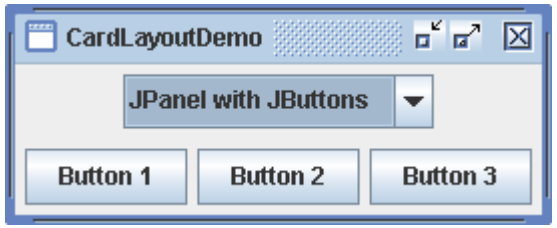
布局示例：GridLayout

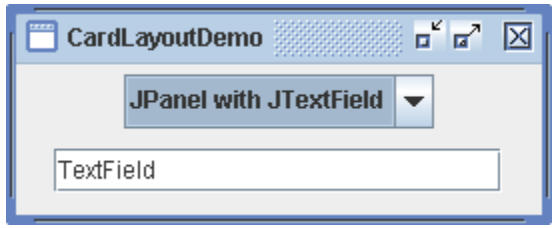
- 按照指定的行数和列数将界面分为等大的若干块，组件被等大地按加载顺序放置其中。



布局示例：CardLayout

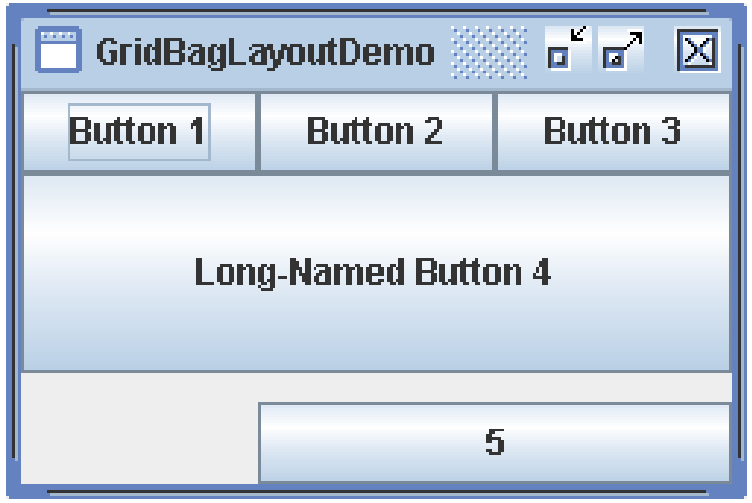
- 可以实现在一个区域出现不同的组件布局，就像在一套卡片中选取其中的任意一张一样。它经常由一个复选框控制这个区域显示哪一组组件，可通过组合框像选择卡片一样选择某一种布局





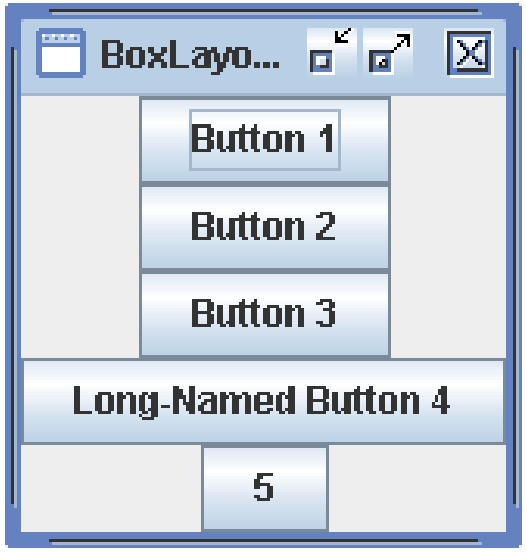
布局示例：GridBagLayout

- 把组件放置在网格中，这一点类似于GridLayout，但它的优点在于不仅能设置组件摆放的位置，还能设置该组件占多少行/列。这是一种非常灵活的布局管理器



布局示例：BoxLayout

- 将组件放在单一的行或列中，和FlowLayout不同的是，它可以考虑到组件的对齐方式，最大、最小、优选尺寸



布局示例：SpringLayout

- 是一种灵活的布局管理器。它能够精确指定组件之间的间距。组件之间的距离通过Spring类的对象来表示，每个spring有四个属性，最小值，最大值，优选值和实际值。每个组件的spring对象集合在一起就构成了SpringLayout.Constraints对象

结束语

- 要说明代码实例在讲了事件处理以后给出

内部类

在另一个类或方法的定义中定义的匿名类

内部类

- 在另一个类或方法的定义中定义的类
- 可访问其外部类中的所有数据成员和方法成员
- 可对逻辑上相互联系的类进行分组
- 对于同一个包中的其他类来说，能够隐藏
- 可非常方便地编写事件驱动程序
- 声明方式
  - 命名的内部类：可在类的内部多次使用
  - 匿名内部类：可在new关键字后声明内部类，并立即创建一个对象
- 假设外层类名为Myclass，则该类的内部类名为
  - Myclass$c1.class (c1为命名的内部类名)
  - Myclass$1.class (表示类中声明的第一个匿名内部类)

例：内部类

例：内部类

```
public class Parcel1 {
    class Contents { //内部类
        private int i = 11;
        public int value() { return i; }
    }
    class Destination  { //内部类
        private String label;
        Destination(String whereTo)  {   label = whereTo;    }
        String readLabel() { return label; }
    }
    public void ship(String dest) {
        //生成两个内部类对象，并调用了内部类中声明的一个方法
        Contents c = new Contents();
        Destination d = new Destination(dest);
        System.out.println(d.readLabel());
    }
    public static void main(String[] args)
    {
        Parcel1 p = new Parcel1();
        p.ship("Tanzania");
    }
}
```

注：本例题引自《Java编程思想》

例：外部类方法返回内部类的引用

例：外部类方法返回内部类的引用

```
public class Parcel2 {
  class Contents  {
      private int i = 11;
      public int value() { return i; }
   }
   class Destination {
       private String label;
       Destination(String whereTo) {  label = whereTo;  }
       String readLabel() { return label; }
    }
   public Destination to(String s)     //to()方法返回内部类Destination的引用
      { return new Destination(s);  }
   public Contents cont() { return new Contents();  }  //cont()方法返回内部类Contents的引用
```

注：本例题引自《Java编程思想》

例：外部类方法返回内部类的引用

```
    public void ship(String dest) {
        Contents c = cont();
        Destination d = to(dest);
        System.out.println(d.readLabel());
    }
    public static void main(String[] args) {
        Parcel2 p = new Parcel2();
        p.ship("Tanzania");
        Parcel2 q = new Parcel2();
        Parcel2.Contents c = q.cont();
        Parcel2.Destination d =q.to("Borneo");
    }
}
```

内部类实现接口、继承抽象类

- 可以完全不被看到，而且不能被调用。
- 可以方便实现“隐藏实现细则”，外部能得到的仅仅是指向超类或者接口的一个引用。

例：内部类实现接口、继承抽象类

首先假设有如下抽象类和接口

```
abstract class Contents {
    abstract public int value();
}
interface Destination {
    String readLabel();
}
```

例：内部类实现接口、继承抽象类

```
public class Parcel3 {
    private class PContents extends Contents {
        private int i = 11;
        public int value() { return i; }
    }
    protected class PDestination implements Destination {
        private String label;
        private PDestination(String whereTo) { label = whereTo;}
        public String readLabel() { return label; }
     }
    public Destination dest(String s) { return new PDestination(s); }
    public Contents cont() { return new PContents(); }
}
```

注：本例题引自《Java编程思想》

例：内部类实现接口、继承抽象类

```
class Test {
    public static void main(String[] args) {
        Parcel3 p = new Parcel3();
        Contents c = p.cont();
        Destination d = p.dest("Tanzania");
    }
}
```

- 说明
  - 内部类PContents实现了抽象了Contents
  - 内部类PDenstination实现了接口Destination
  - 外部类Test中不能声明对private的内部类的引用

局部作用域中的内部类

- 实现某个接口，产生并返回一个引用
- 为解决一个复杂问题，需要建立一个类，而又不想它为外界所用

例：方法中的内部类

```
public class Parcel4 {
    public Destination dest(String s) {
        class PDestination implements Destination {
            private String label;
            private PDestination(String whereTo) {
                label = whereTo;
            }
            public String readLabel() { return label; }
        }
        return new PDestination(s);
    }
    public static void main(String[] args) {
        Parcel4 p = new Parcel4();
        Destination d = p.dest("Tanzania");
    }
}
```

注：本例题引自《Java编程思想》

例：块作用域中的内部类

```
public class Parcel5 {
    private void internalTracking(boolean b) {
        if(b) {
            class TrackingSlip {
                private String id;
                TrackingSlip(String s) { id = s;   }
                String getSlip() { return id; }
             }
            TrackingSlip ts = new TrackingSlip("slip");
            String s = ts.getSlip();
         }
    }
    public void track() { internalTracking(true); }
    public static void main(String[] args)  {
        Parcel5 p = new Parcel5();
        p.track();
    }
}
```

注：本例题引自《Java编程思想》

匿名的内部类

```
public class Parcel6 {
    public Contents cont() {
        return new Contents() {
            private int i = 11;
            public int value() { return i; }
        };
    }
    public static void main(String[] args) {
        Parcel6 p = new Parcel6();
        Contents c = p.cont();
    }
}
```

注：本例题引自《Java编程思想》

结束语

- 说明简单了解即可，主要是为了理解在事件处理时用到匿名内部类

事件处理的基本概念

GUI是由事件驱动的

常见的事件包括

- 移动鼠标
- 单双击鼠标各个按钮
- 单击按钮
- 在文本字段输入
- 在菜单中选择菜单项
- 在组合框中选择、单选和多选
- 拖动滚动条
- 关闭窗口
- ……
- Swing通过事件对象来包装事件，程序可以通过事件对象获得事件的有关信息

事件处理的几个要素

- 事件源
- 事件监听器
- 事件对象

事件处理的几个要素

- 事件源
  - 与用户进行交互的GUI组件，表示事件来自于哪个组件或对象
  - 比如要对按钮被按下这个事件编写处理程序，按钮就是事件源
- 事件监听器
  - 负责监听事件并做出响应
  - 一旦它监视到事件发生，就会自动调用相应的事件处理程序作出响应
- 事件对象
  - 封装了有关已发生的事件的信息
  - 例如按钮被按下就是一个要被处理的事件，当用户按下按钮时，就会产生一个事件对象。事件对象中包含事件的相关信息和事件源

- 程序员应完成的两项任务
  - 为事件源注册一个事件监听器
  - 实现事件处理方法

事件源

- 提供注册监听器或取消监听器的方法

事件源

- 提供注册监听器或取消监听器的方法
- 如有事件发生，已注册的监听器就会被通知
- 一个事件源可以注册多个事件监听器，每个监听器又可以对多种事件进行相应，例如一个JFrame事件源上可以注册
  - 窗口事件监听器，响应：
    - 窗口关闭
    - 窗口最大化
    - 窗口最小化
  - 鼠标事件监听器，响应：
    - 鼠标点击
    - 鼠标移动

事件监听器

- 是一个对象，通过事件源的add×××Listener方法被注册到某个事件源上
- 不同的Swing组件可以注册不同的事件监听器
- 一个事件监听器中可以包含有对多种具体事件的专用处理方法
  - 例如用于处理鼠标事件的监听器接口MouseListener中就包含有对应于鼠标压下、放开、进入、离开、敲击五种事件的相应方法mousePressed、mouseReleased、mouseEntered、mouseExited、mouseClicked，这五种方法都需要一个事件对象作为参数

事件对象

- 常用的事件对象
  - ActionEvent
    - 发生在按下按钮、选择了一个项目、在文本框中按下回车键
  - ItemEvent
    - 发生在具有多个选项的组件上，如JCheckBox、JComboBox
  - ChangeEvent
    - 用在可设定数值的拖曳杆上，例如JSlider、JProgressBar等
  - WindowEvent
    - 用在处理窗口的操作
  - MouseEvent
    - 用于鼠标的操作

常用的Swing事件源可能触发的事件及事件监听器

常用的Swing事件源

事件源	事件对象	事件监听器
JFrame	MouseEvent WindowEvent	MouseEventListener WindowEventListener
AbstractButton (JButton, JToggleButton, JCheckBox, JRadioButton)	ActionEvent ItemEvent	ActionListener ItemListener
JTextField JPasswordField	ActionEvent UndoableEvent	ActionListener UndoableListener
JTextArea	CareEvent InputMethodEvent	CareListener InputMethodEventListener
JTextPane JEditorPane	CareEvent DocumentEvent UndoableEvent HyperlinkEvent	CareListener DocumentListener UndoableListener HyperlinkListener

常用的Swing事件源

JComboBox	ActionEvent ItemEvent	ActionListener ItemListener
JList	ListSelectionEvent ListDataEvent	ListSelectionListener ListDataListener
JFileChooser	ActionEvent	ActionListener
JMenuItem	ActionEvent ChangeEvent ItemEvent MenuKeyEvent MenuDragMouseEvent	ActionListener ChangeListener ItemListener MenuKeyListener MenuDragMouseListener
JMenu	MenuEvent	MenuListener
JPopupMenu	PopupMenuEvent	PopupMenuListener

常用的Swing事件源

JProgressBar	ChangeEvent	ChangeListener
JSlider	ChangeEvent	ChangeListener
JScrollBar	AdjustmentEvent	AdjustmentListener
JTable	ListSelectionEvent TableModelEvent	ListSelectionListener TableModelListener
JTabbedPane	ChangeEvent	ChangeListener
JTree	TreeSelectionEvent TreeExpansionEvent	TreeSelectionListener TreeExpansionListener
JTimer	ActionEvent	ActionListener

接口与适配器

- 事件监听器接口
  - 规定实现各种处理功能接口。
- 事件监听器适配器类
  - 有时我们并不需要对所有事件进行处理，为此Swing提供了一些适配器类×××Adapter。

接口与适配器

- 事件监听器接口
  - 例如MouseListener是一个接口，为了在程序中创建一个鼠标事件监听器的对象，我们需要实现其所有五个方法，在方法体中，我们可以通过鼠标事件对象传递过来的信息（例如点击的次数，坐标），实现各种处理功能。
- 事件监听器适配器类
  - 有时我们并不需要对所有事件进行处理，为此Swing提供了一些适配器类×××Adapter，这些类含有所有×××Listener中方法的默认实现（就是什么也不做），因此我们就只需编写那些需要进行处理的事件的方法。例如，如果只想对鼠标敲击事件进行处理，如果使用MouseAdapter类，则只需要重写mouseClicked方法就可以了 。

事件处理

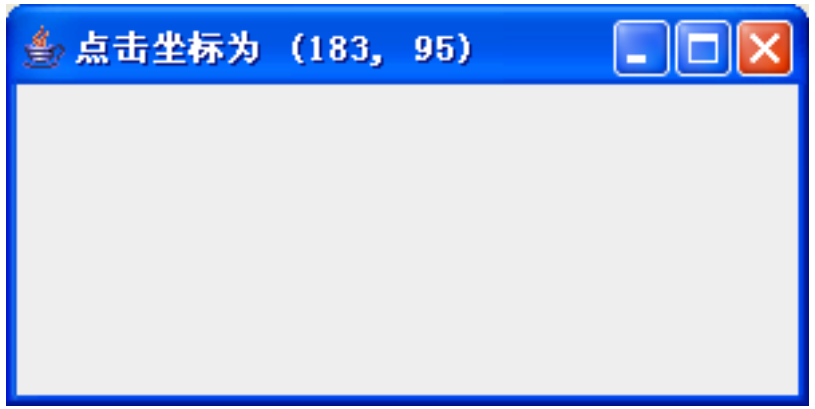
- 实现事件监听器接口
- 继承事件监听器适配器类
- 使用匿名内部类
- lambda表达式

事件处理

- 实现事件监听器接口
  - 这种方法需要实现接口中所有的方法，对我们不需要进行处理的事件方法，也要列出来，其方法体使用一对空的花括号
- 继承事件监听器适配器类
  - 只需要重写我们感兴趣的事件
- 使用匿名内部类
  - 特别适用于已经继承了某个父类（例如Applet程序，主类必须继承JApplet类或Applet类），则根据java语法规则，就不能再继承适配器类的情况，而且使用这种方法程序看起来会比较清楚明了
- lambda表达式
  - 对于只有一个抽象方法的函数式监听器接口，也可以使用lambda表达式。

例：

- 创建一窗口，当鼠标在窗口中点击时，在窗口标题栏中显示点击位置坐标。



- 方法一：实现MouseListener接口。
- 方法二：继承MouseAdapter类。
- 方法三：使用匿名内部类。

例：实现MouseListener接口

```
import java.awt.event.*; //载入MouseListener类所在的包
import javax.swing.*;   //载入JFrame所在的包
public class ImplementMouseListener  implements MouseListener{
    JFrame f;
    public ImplementMouseListener () {
        f=new JFrame(); //新建一窗口
        f.setSize(300,150);
        f.show();
          f.addMouseListener(this); //为窗口增加鼠标事件监听器
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
     public void mousePressed(MouseEvent e){}
     public void mouseReleased(MouseEvent e){}
     public void mouseEntered(MouseEvent e){}
     public void mouseExited(MouseEvent e){}
     public void mouseClicked(MouseEvent e){
        f.setTitle("点击坐标为 ("+e.getX()+", "+e.getY());
     }
    public static void main(String[] args){ new ImplementMouseListener ();}
}
```

例：继承MouseAdapter类

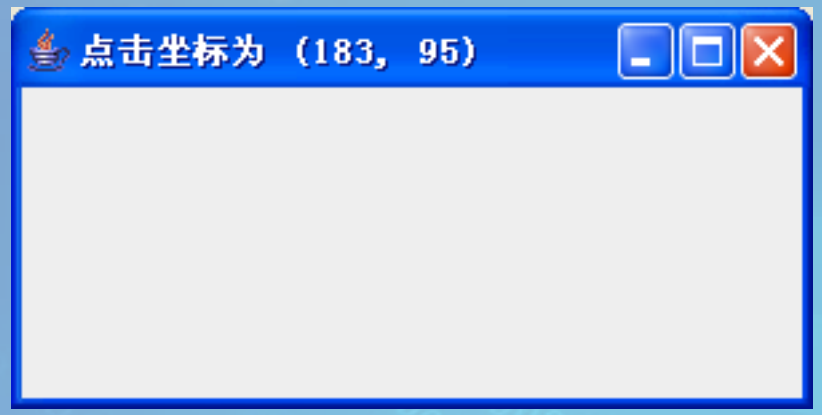
```
import java.awt.event.*; //载入MouseAdapter所在的包
import javax.swing.*;

public class ExtendMouseAdapter   extends MouseAdapter{
    JFrame f;
    public ExtendMouseAdapter () {
        f=new JFrame();
        f.setSize(300,150);
        f.show();
        f.addMouseListener(this);
        f.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
    }
     public void mouseClicked(MouseEvent e){
         f.setTitle("点击坐标为 ("+e.getX()+", "+e.getY()+")");
     }
    public static void main(String[] args){ new ExtendMouseAdapter();}
}
```

例：使用匿名内部类

```
import java.awt.event.*;
import javax.swing.*;
public class UseInnerClass {
    JFrame f;
    public UseInnerClass () {
        f=new JFrame();
        f.setSize(300,150);
        f.show();
        f.addMouseListener(new MouseAdapter(){
            public void mouseClicked(MouseEvent e){
                f.setTitle("点击坐标为 ("+e.getX()+", "+e.getY()+")");
            }
        }); //为窗口添加鼠标事件监听器语句结束
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){ new UseInnerClass (); }
}
```

运行结果

- 采用不同方法的程序，其运行效果都是一样的，当鼠标在窗口中点击的时候，窗口标题栏将出现所点位置的坐标信息



结束语

事件派发机制

事件派发机制——事件派发线程

- Swing中的组件是非线程安全的，在Swing中专门提供了一个事件派发线程（EDT）用于对组件的安全访问。

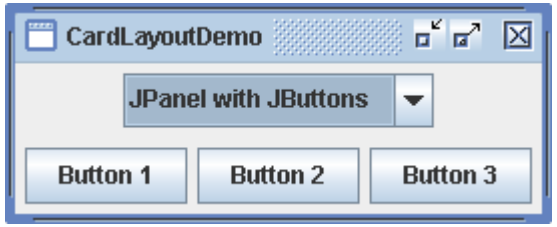
事件派发机制——事件派发线程

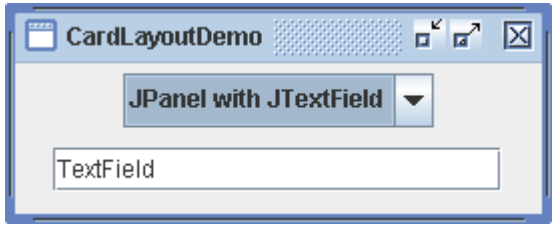
- Swing中的组件是非线程安全的，在Swing中专门提供了一个事件派发线程（EDT）用于对组件的安全访问。
  - 用来执行组件事件处理程序的线程（如按钮的点击事件），依次从系统事件队列取出事件并处理，一定要执行完上一个事件的处理程序后，才会处理下一个事件。
  - 事件监听器的方法都是在事件派发线程中执行的，如ActionListener的actionPerformed方法。

事件派发机制——由事件派发线程启动GUI

- 可以调用invokeLater或invokeAndWait请事件分发线程以运行某段代码
  - 要将这段代码放入一个Runnable对象的run方法中，并将该Runnable对象作为参数传递给invokeLater
- invokeLater是异步的，不用等代码执行完就返回。
- invokeAndWait是同步的，要等代码执行完才返回。调用时要避免死锁。

重温布局示例：CardLayout

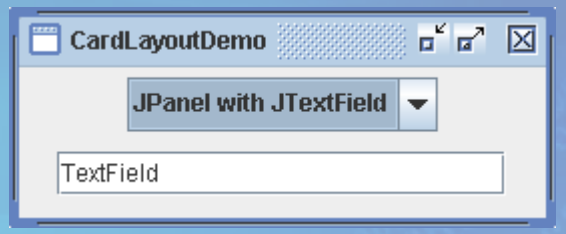




例：CardLayout

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CardLayoutDemo implements ItemListener {
    JPanel cards;
    final static String BUTTONPANEL = "JPanel with JButtons";
    final static String TEXTPANEL = "JPanel with JTextField";

    public void addComponentToPane(Container pane) {
        //将JComboBox放进JPanel
        JPanel comboBoxPane = new JPanel(); //默认使用 FlowLayout
        String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };
        JComboBox cb = new JComboBox(comboBoxItems);
        cb.setEditable(false);
        cb.addItemListener(this);
        comboBoxPane.add(cb);

        JPanel card1 = new JPanel();
```

- 运行结果：



引用自The Java Tutorials http://docs.oracle.com/javase/tutorial/

结束语