第七章 图形用户界面

7.1 绘图

1.图形环境和图形对象

（1）坐标：GUI 组件左上角默认为（0,0），从左上角到右下角，水平坐标 x 和垂直坐标 y 增加，坐标的单位是像素。

（2）Graphics 对象：专门管理图形环境。Graphics 是一个抽象类，提供了一个与平台无关的绘图接口，这样编写的程序可以符合 java 平台要求。各平台上实现的 JAVA 系统将创建一个子类，来实现绘图功能，但是这个子类对程序员来说是**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（名字随意）。

2.颜色和字体

（1）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 RGB 三色分量（0~255）创建颜色
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 类方法，用于设置组件的颜色
Graphics: public Color getColor()	Graphics 类方法，用于获得组件的颜色

（2）字体

Font 类，以及 Graphics 类中雨字体有关的方法

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**加粗字体风格**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、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

3.使用 Graphics 类绘图

Graphics 可以绘制字符串或者各种图形。Graphics 类常用方法可以参见 http://blog.csdn.net/zhliro/article/details/45564251。

例：使用 Graphics 类绘图

代码太长，不再累述。

结果见图 1.

图 1

4.使用 Graphics 2D 类绘图

Java2D API：提供了高级的二维图形功能。它可以轻松完成以下功能：

（1）绘制任何宽度的直线；

（2）用渐变颜色和纹理来填充图形；

（3）平移、旋转、伸缩、切变二维图形，对图像进行模糊、锐化等操作；

（4）构建重叠的文本和图形；

（5）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。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

7.2 Swing 基础

1.JFC 与 Swing

JFC（Java Foundation Classes），Java 基础类库，它是关于 GUI 组建和服务的完整集合。它作为 JAVA SE 的一个有机部分，主要包含 5 个内容：

（1）AWT

（2）Java2D

（3）Accessibility

（4）Drag & Drop

（5）Swing

如上所述，Swing 是 JFC 的一部分，它主要是提供按钮、窗口、表格所有组建，它是纯 java 组建。

2.Swing 与 AWT 组件

在 java.awt 包中，包括 button、checkbox、scrollbar 等，都是 Component 类的子类，大部分都含有 native code，所以随操作系统平台的不同会显示出不同的样子，而且不能更改，是重量级组件。

Swing 名称都是在 AWT 组件名称前加 J，都是 JComponent 类的子类，它是完全由 java 编写，外观和功能不依赖于任何宿主平台的窗口系统提供的代码，是轻量级组件，可以提供更丰富的视觉感受，但是一些顶层还是重量级的。

3.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 容器中（通常是 Japplet），这通常在 init 方法中完成；在 application 中有，要将 Swing 组件加载到 application 的顶级容器中。

例：在 Applet 中应用 Swing

代码：

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

```
public class SwingApplet extends Japplet{
    public void init(){
        //重写 Applet 的 init 方法，游览器加载
        //此 applet 时会自动执行
        Container contentPane = getContentPane();
        //获得浏览器提供的顶层容器的内容面板
        contentPane.setLayout(new GridLayout(2,1));//设置内容面板的板式
        JButton button = new JButton("Click me");//创建一个按钮
        final JLabel label = new JLabel();//创建一个标签
        contentPane.add(button);//把按钮加到内容面板上
        contentPane.add(label);//把标签加到内容面板上去
        button.addActionListener(
        //为按钮创建事件监听器，使其能够对 click 事件作出反应
        new ActionListener(){
            public void actionPerformed(ActionEvent event){
                String information = JOptionPane.showInputDialog("请输入一串字符");
                label.setText(information);
            }
        });
    }
}
```

例：在 Application 中应用 Swing

```
//SwingApplication.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingApplication{
    public static void main(String[] args){
        JFrame f = new JFrame("Simple Swing Application");
        //创建一个框架 f 作为顶层容器
        Container contentPane = f.getContentPane();//获得内容面板
        contentPane.setLayout(new GridLayout(2,1));//设置布局
        JButton.button = new JButton("Click me");
        final JLabel label = new JLabel();//创建一个标签
        contentPane.add(button);//添加按钮
        contentPane.add(label);
        button.addActionListener(
        new ActionListener(){
            public void actionPerformed(ActionEvent event){
                String information = JOptionPane.showInputDialog("Input");
                label.setText(information);
            }
        });
        f.setSize(200,100);
```

```
            f.setVisible(true);
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

7.3 Swing 的层次

Swing 组件通常被归为三个层次：顶层容器、中间层容器和原子组件。在介绍之前，先看一下绝大多数 Swing 组件的继承层次：

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

JComponent 类是除了顶层容器以外所有 Swing 组件的超类，根据继承关系，可以在每个超类中找到大多数 GUI 组件常用的操作。

Component 类中包含了 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 类是容纳相关组件，包括 add 方法，用来增加组件，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，课定制观感，有快捷键，一般的事件处理功能。

1.Swing 的组件和容器层次

（1）顶层容器。它三个类：JFrame 实现单个主窗口，JDialog 实现一个二级窗口（对话框），JApplet 在浏览器中实现 applet 显示区域。这些都是重量级组件，从继承结构看，它们分别从原来 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示出来。

（2）中间层容器，它存在的目的是为了容纳别的组件，包括两个雷：一般用途的类和特殊用途的类。一般用途的有 JPanel、JScrollPane、JSplitPane、JTabbedPane、JToolBar；特殊用途的类有 JInternalFrame、JRootPane 两类。

（3）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，它的基本功能就是和用户交互信息。根据功能不同，可以分为三类：显示不可编辑信息的 JLabel、JProgressBar、JToolTip；用控制功能、可以用来输入信息的 JButon、JCheckBox、JRadioButton、JComboBox、JList、JMenu、JSlider、JSpinner、JTexComponet 等；还有能提供格式化的信息并允许用户选择的 JColorChooser、JfileChooser、JTable、JTree。

7.4 布局管理

在窗口放置不同的组件，需要靠布局管理器来对其自动放置合适的位置。

1.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的进行布局，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，不用担心版面因此混乱。

在 Java 中常用的布局管理器类有很多，经常用到的有以下几种：（1）BorderLayout（2）FlowLayout（3）FridLayout（4）CardLayout（5）GridBagLayout（6）BoxLayout（7）SpringLayout（8）内容面板（content pane）默认使用的就是 BorderLayout，它可以将组建放置到 5 个区域：东、西、南、北、中。

7.5 内部类

写在别的类体或方法定义中定义的类，它可以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它类来说，它能够隐藏，可以隐藏一些实现细节，内部类还可以非常方便地编写事件驱动程序。

它的声明方式：

（1）命名的内部类：可在类的内部多次使用

（2）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）

如果我们想让这个内部类在外面完全看不见，那么我们可以定义一个接口，或者一个抽象类，然后让这个内部接口实现接口、继承抽象类，可以方便实现隐藏实现细节，外部能得到的仅仅是指向超类或者接口的一个引用。

一个内部类可以定义在一个方法里面、任意一个大括号甚至参数列表里面。这样操作可以实现某个接口，产生并返回一个引用；也可以应用在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：匿名的内部类

代码：

```
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();
    }
}
```

代码分析：

如果我们这个内部类只需要用一次，那么它其实可以练名字都没有。比如说就在 cont 方法里面 return 语句中，我们就用这个方法来匿名一个类。这其实是以 conents 为超类，派生出一个非抽象的类，在这个类中覆盖了 contents 里面的定义。

7.6 事件处理的基本概念

1.基本知识

GUI 程序都是事件驱动的。所谓事件驱动，就是我们看到的大多数程序，在打开的时候会有一些组件，当你操作的时候就会有响应，做出反应。

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，关闭窗口等。

在 Swing 中是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相关信息，对应的程序段可以进行相应的处理。

事件处理的几个要素包括:

（1）事件源。所谓事件源，就是要与用户进行交互的 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，按钮就是这个事件源。事件源提供注册监听器和取消的方法，如果事件发生，已注册的监听器就会被通知，一个事件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响应。

（2）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做出响应。事件监听器是一个对象，通过事件源的 add***Listener 方法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，一个事件监听器可以包含对多种具体事件的专用处理方法。

（3）事件对象。封装了有关已发生的事件的信息。

我们只需要关注两件事，一是为事件源注册一个事件监听器，二是实现实践方法。

2.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的五个方法。

事件监听器适配器类：有时候我们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。你关注哪个方法体的具体实现，你就直接覆盖即可。

3.事件处理的方法

（1）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，对于我们不需要的方法，也要列出来，其方法体使用一堆空的花括号。

（2）继承事件监听器适配器类。只需要重写我们感兴趣的事件。

（3）使用匿名内部类。特别适用于已经继承某个父类。

（4）lambda 表达式。对于只有一个抽象方法的函数式监听器接口，也可以使用本方法。

7.7 事件派发机制

在 Swing 中，这些都不是线程安全的，也就是说不能有多个程序段并发对其操作。在 java 中有一个事件派发线程，所有事件都由其派发。

1.事件派发机制——事件派发线程

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。，它可以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去除事件并处理，**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里面执行的，比如 ActionListener 的 actionPerformed 方法。

2.事件派发机制——由事件派发线程启动 GUI

可以调用 invokeLater 或者 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，事件派发线程就会启动这段代码执行。**注意，invokeLate 是异步执行，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，调用时应避免死锁**。

例：实现 CardLayout

代码：

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

public class CardLayoutDemo implements ItemListener{
    //监听器，简单例子注册自己作为事件监听器
    JPanel cards;
    //准备好两个字符串，分别表示两个卡片的标题
    final static String BUTTONPANEL ="JPanel with JButtons";
    final static String TEXTPANEL = "Jp with JTEXT";

    public void addComponentToPane(Container pane){//参数是一个容器
        //将原子组件 JComboBox 放进 JPanel,JPanel 是中间容器
        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();//card1 放置的内容，都有自己的中间容器
        card.add(new JButton("Button 1"));
        card.add(new JButton("Button 2"));
        card.add(new JButton("Button 3"));

        Jpanel card2 = new JPanel();//card2 放置的内容，都有自己的中间容器
        card2.add(new JTextField("TextField", 20));

        cards = new JPanel(new CardLayout());//将这两个 card 都在放到另外一个中间容器
        cards.add(card1,BUTTONPANEL);
        cards.add(card2,TEXTPANEL);

        pane.add(comboBoxPane, BorderLayout.PAGE_START);//
        pane.add(cards, BorderLayout.CENTER);//把 comboboxpane 和 cards 两个
        //中间容器都添加到 pane 这个容器上去
    }
```

```
    public void itemStateChanged(ItemEvent evt){
        CardLayout cl =(CardLayout)(cards.getLayout());//强制转换类型
        cl.show(cards,(String)evt.getItem());//显示 cards 中的某一张，取决于 evt
    }

    private static void createAndShowGUI(){
        //构造主窗口 JFrame，设置关闭窗口按钮
        JFrame frame = new JFrame("CardLayoutDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //构造对象，组件加到顶层容器
        CardLayoutDemo demo = new CardLayoutDemo();
        demo.addComponentToPane(frame.getContentPane());

        frame.pack();//设置大小正好容纳组件
        frame.setVisible(true);//让它显示
    }

    public static void main(String[] args){
        //new 一个实现了 Runnable 接口的匿名内部类的对象
        //而不是直接 new 了一个 runnable
        javax.swing.SwingUtilities.invokeLater(new Runnable(){
            public void run(){//包含我想执行的操作
                createAndShowGUI();
            }
        });
    }
}
```

7.8 顶层容器

Swing 这个结构是以顶层容器为根的树状结构，每个组件只能放在一个容器上。Swing 有三个顶层容器类，分别是 JFrame、JApplet、JDialog，他们都是重量级组件。每个顶层容器都有一个内容面板，通常直接或者间接的容纳别的可视组件。可以由选择为顶层容器添加菜单，菜单位于顶层容器上，但是在内容面板之外的。

（1）JFrame 的继承结构：

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

（2）JApplet 的继承结构：

```
java.lang.Object
  - java.awt.Component
```

-java.awt.Container
-java.awt.Panel
-java.awt.Applet
-javax.swing.JApplet

（3）JDialog 的继承结构：

java.lang.Object
- java.awt.Component
-java.awt.Container
-java.awt.Window
-java.awt.Dialog
-javax.swing.J Dialog

只有 JApplet 是浏览器自动生成，其它都需要我们构造对象，用构造方法去初始化。

例：FrameDemo.java

代码：

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrameDemo{
    public static void main(String[] s){
        JFrame frame = new JFrame("FrameDemo");//构造一个 JFrame 对象
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//关闭按钮功能
        JLabel emptyLabel = new JLabel("");//生成一个空的 label
        emptyLabel.setPreferredSize(new Dimension(175,100));//设置尺寸
        frame.getContentPane().add(emptyLabel, BorderLayout.CENTER);
//获得一个内容面板，然后把这个 label 按照后面的格式要求放上去
        frame.pack();
        frame.setVisible(true);
    }
}
```

例：JOptionPaneDemo.java

通过静态方法 show***Dialog，可以产生四种简单的对话框。他们的方法参数中绝大部分（除了输入对话框可以不指定父窗口）都需要提供一个父窗口组件 ParentComponent，只有关闭这些简单的对话框后，才可以返回到其父窗口，也就是说，他们绝大部分都是模态的。

7.9 中间层容器

1.JRootPane

它是唯一一个可以从顶层容器继承过来的，它的层次结构见图 2 所示。

图 2

从图中可以看出，JRootPane 上面包括了有 glassPane 和 layeredPane。

（1）glassPane

默认状态下是隐藏的，可以使用 glassPane 截获所有要到达 JRootPane 别的部分的事件。

（2）layeredPane

分层的每个层都有一个代表层深度的整数值（Z-order），深度值高的组件将覆盖在深度值低的组件上门。它又包括：

contentPane

一般将所有的组件添加在上面

JMenuBar（可选），若没有，则被 contentPane 充满顶层容器。

2.JPanel

在默认下，除了背景色什么都没有，可以本容易设置边框和绘制特性，有效利用其可以使版面管理更容易，可以使用布局管理器可以管理容纳组件的位置和大小。

3.JScrollPane

容器有滚动条，通过拖动滑块就可以看到更多的内容。它由就各部分组成，包括一个中心显示地带、四个角和四条边。

图 3

4. JSplitPane

可以将窗口分为两个两个部分，两个部分操作都是相互联系的。这种分割可以水平、垂直，也可以动态拖拽（拖动分界线时两边组件是否会随着拖拽动态改变大小还是在拖拽结束后才改动）。我们通常先把组件放到 Scroll Pane 中，再把 Scroll Pane 放到 Split Pane 中。这样在每部分窗口中，都可以拖动滚动条看到组件的全部内容。

5. JTabbedPane

如果一个窗口的功能有几项，可以给每项设置一个标签，每个标签下面包含为完成此功能专用的若干组件。

6.JToolBar

将一些常用的功能以工具栏的方式呈现。

7. JInternalFrame（轻量级组件，只能是中间容器）

打开多个子窗口，每个子窗口（文档）各自占用一个新窗口。

7.10 原子组件

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，它的基本功能就是和用户交互信息。根据功能不同，可以分为三类：显示不可编辑信息的 JLabel、JProgressBar、JToolTip；用控制功能、可以用来输入信息的 JButon、JCheckBox、JRadioButton、JComboBox、JList、JMenu、JSlider、JSpinner、JTexComponet 等；还有能提供格式化的信息并允许用户选择的 JColorChooser、JfileChooser、JTable、JTree。

1.显示不可编辑信息的原子组件

（1）JLabel 可以显示文字和图像，并且能够指定两者的位置，但不能修改内容。

（2）JProgressBar：在一些软件运行时候的进度条。

（3）JToolTip：使用 setToolTipText()方法为组件设置提示信息。有的组件例如 JTabbedPane 由多个部分组成，需要鼠标在不同部分停留时显示不同的提示信息，这时候可以在其 addTab()方法中设置提示信息参数，也可以通过 setTooltipTextAt 方法进行设置。

2.具有控制功能，可以输入信息的原子组件

（1）Abstract 按钮

它是众多按钮类的超类，它是一个抽象类。继承它的类如图 4 所示。

图 4

JButton：普通按钮

JToggleButton：表示有两个选择状态的按钮，其包括 CheckBox（多选按钮）和 JradioButton（单选按钮）。

JMenuItem：在菜单中使用，它包括 JCheckBoxMenuItem（多选按钮）、JRadioButtonMenuItem（单选按钮）和 JMenu（一般菜单项）。

（2）JList

JList 可以选择一个到多个选项。它由几种各有特色的构造方法（选项是否可以添加、删除），也提供了很多 API 可以设置各选项的显示方式；考虑到其含有较多选项，经常把它放在一个 JScrollPane 对象里面。

JList 的事件处理一般可以分为两类：

取得用户选取的项目，其事件监听器是 ListSelectionListener；

对鼠标事件作出响应，其事件监听器是 MouseListener。

（3）JComboBox

在许多选项中选择一个。其有两种方式，可以用下拉列表方式，也可以用更紧凑的方式：

默认情况下是不可编辑的模式，其特色是包括一个按钮和一个下拉列表，用户只能在下拉列表的内容选择一个；

另一种是可编辑，多了一个文本区域，用户可以在文本区域填入列表不存在的内容。

（4）连续数值选择——JSlider 和 JSpinner

JSlider 占空间大，可以设置它的最大、最小、初始值，还可以设置其方向，还可以为其标上刻度或者文本。在 JSlider 上移动滑动杆，会产生 ChangeEvent 事件。如图 5 上侧图片所示。

JSpinner 占空间小，类似于可编辑的 JComboBox，是一种复合组件，由三个部分组成：向上按钮、向下按钮和一个文本编辑区，它可以通过按钮拉来选择待选项，也可以直接在文本编辑区输入。但是 JComboBox 不同的是，它的待选项不会显示出来。其如图 5 的下图所示。

图 5

（5）文本组件

都继承自 JTextComponent 抽象类，可分为三类：

①JTextField/JPasswordField/JForattedTextField

只能显示和编辑一行文本，像按钮一样，它们可以产生 ActionEvent 事件，通常用来接受少量用户输入信息并在输入结束进行一些事件处理。

②JTextArea

可以现实和编辑多行文本，但是这些文本只能是单一风格，通常让用户输入任意长度的无格式文本或者显示无格式的帮助信息。

③JEditorPane/JTextPane

可以显示和编辑多行多种式样的文本，嵌入图像或者别的组件。

3. 提供格式化的信息并允许用户选择的组件

（1）JColorChooser 颜色选择对话框

可以让用户选择所需要的颜色。通常使用这个类的静态方法 showDialog()来输出标准的颜色选择对话框，其返回值就是选择的颜色。

也可以通过静态方法 createDialog()方式输出个性化的颜色选择对话框，例如为其添加菜单、定义其事件处理程序，这个方法的返回值就是一个对话框。

（2）JFileChooser 文件选择对话框

它的作用是让用户选择一个已有的文件或者新建一个文件。

可以使用 JFileChooser 的 showDialog()、showOpenDialog()或者 showSaveDialog()方法来打开文件对话框，但是它仅仅返回用户选择的按钮（确认还是取消）和文件名（如果确认的话），接下来的要实现的例如存盘或者打开功能还需要程序员自己编写。

这个类提供了专门方法用于设置可选择文件的类型，还可以指定每类文件使用的类型图标。

（3）JTable

用表格展示结构化的数据。它可以为表格设置显示外观（是否有滚动条、调整某一列宽其他列宽变化情形）、显示模式（根据数据类型有不同的排列显示方式、为某一字段添加组合框 JComboBox）、选择模式（单选、多选、连续选、任意选等）。

它的事件都是针对表格内容的操作处理，称之为 TableModelEvent 事件，可以通过 addTableModelListener 方法为表格添加此种事件监听器。

（4）JTree

用来产生树状结构来直观地表现层次关系，有根节点、树枝节点、树叶节点。

它的构造方法有很多中，参数可以是一个 Hashtable，也可以是 TreeNode 或 TreeModel 对象。还可以使用 JComponent 提供的 putClientProperty 方法来设置 JTree 的外观，也可以使用 TreeCellRenderer 来个性化各类节点的显示样式。

7.11 其它 Swing 特性

1.Action 对象

（1）功能

用 action 封装不同组件的相同功能，也可以封装其它的一些属性。Action 接口是对 ActionListener 接口的一个有用扩展，它的继承关系如下：

public interface Action extends ActionListener

在很多既有菜单又有工具栏的应用程序中，可以通过 Action 接口封装事件相应代码和相关设置，并添加到不同的事件源中。

还可以通过它对不同组件的显示文字、图标、快捷键、提示文字、是否可用等属性进行统一的设置。

（2）创建 Action 对象

AbstractAction 类实现了 Action 接口中除了 actionPerformed 方法以外的其它方法，而且还提供了一些获取和设置 Action 域属性的方法。

因此，首先有需要创建一个继承抽象来 AbstractAction 类的子类，然后再实例化这个子类。在子类中我们需要设置需要的属性值、定义 actionPerformed 方法。

（3）使用 Action 对象

通过 GUI 组件的 setAction 方法将 Action 对象关联组件。每个具有 addActionListener 方法的组件也都具有 setAction 方法。Action 是一个事件监听器，如果需要添加多个监听器，应使用 addActionListener 方法。一个 GUI 组件可以调用 setAction 不止一次，但组件和前一个 Action 对象之间的关联会被删除。

通过 setAction 方法把 Action 对象关联到某 GUI 组件后，会有以下效果：**此组件的属性会被设置为符合这个 Action 对象的属性**；**这个 Action 对象会被注册为此组件的一个事件监听器对象**；**如果改变了 Action 对象的属性或方法，那和它关联的组件的属性或方法也会自动变更**。

2.边框

每个继承自 JComponent 的 Swing 组件都可以有边框。

使用组件的 setBorder 方法为组件添加边框，需要提供一个 Border 类型的对象。我们可以使用 BorderFactory 类提供的很多静态方法产生一个常用的 Border 对象；如果不满足要求，可以直接用 javax.swing.border 里面的 API 来定义自己的边框。

3.设置组件观感

在产生任何可视组件以前需要使用 UIManager 类所提供的 setLookAndFeel()静态方法设置好它们的观感。

java 提供跨平台的观感。可以利用 UIManager 类提供的 getCrossPlatformLookAndFeelClassName()静态方法获得类名。

程序所处系统的观感。可以利用 UIManager 类提供的 getSystemLookAndFeel()静态方法获得目前操作平台的 Look and Feel 类名称字符串。

4.设置顶层容器的观感

JFrame 和 JDialog 是重量级组件，依赖于操作系统，当使用的操作系统不同时，所显示的顶层容器就会不同，针对这两个顶层容器，有一个静态方法专门为其设置观感。

static void setDefalutLookAndFeelDecorated(boolean)

说明：

（1）参数是 true，使用默认外观。

（2）参数是 false，使用操作系统外观。

5.桌面 API

从 Java 6 开始，对于特定的文件类型，Java 程序可以和关联该文件类型的主机应用程序进行交互。这种交互是通过 java.awt.DeskTop 类进行的，因此该类的 API 叫做桌面 API。

桌面 API 运行 Java 应用程序完成三件事情：

（1）启用主机平台上默认的浏览器打开 URL，这个功能由 DeskTop 的 browse 方法完成；

（2）启用主机平台默认的邮件客户端，此功能由 DeskTop 的 mail 方法完成；

（3）对特定的文件，启用主机平台上与之相关联的应用程序，对其进行打开、编辑、打印操作，这些功能分别由 DeskTop 的 open、edit、print 方法完成。