第 14 章　GUI 程序设计

本章前面所有的程序，都是在控制台用命令行方式运行。它们被称为控制台程序，也被称为命令行（CLI）程序。这种程序最大的缺点是界面不够美观，而且控制起来不是很方便，所以编制成应用软件的话，不太适合初级用户使用。

GUI（图形用户界面，Graphical User Interface），是用户与计算机之间交互的图形化操作界面，又称为图形用户接口。Windows 中的绝大多数程序都是以这种形式与用户交互的，这种程序也被称为窗口程序或图形程序。而 Unix/Linux 也越来越向这方面靠拢，它们都提供了一种称为 X-Window 的 GUI 接口。

GUI 程序不仅界面美观，而且使用方便，各种软件的操作上也可以统一，非常适合普通用户使用，已经成为绝对的主流，所以编制 GUI 程序是现代程序员的基本工作。

本章就将介绍如何使用 Java 来编制 GUI 程序。Java 和其他语言的一个重大区别就是它提供了编写 GUI 程序所需要的各种类（其他的语言本身不具备这种功能，需要由开发工具来辅助实现），这也使得 Java 无需借助专门的开发工具就可以轻松地写出 GUI 程序。目前在编写 GUI 程序中，使用最为广泛的是 Swing 包中的各种类，这也是本章的重点。

14.1　GUI 程序设计的基本概念

在介绍 GUI 程序设计的基本概念之前，先来看一副图片。图 14.1 是笔者编写的一个小程序：

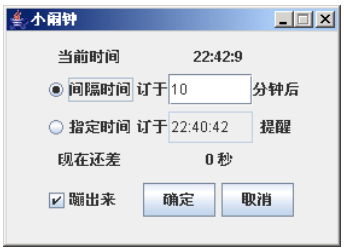

图 14.1　GUI 程序界面

它与 Windows 系统中的窗口程序形式上完全一致。它也拥有标签、按钮、文本框、复选框、单选按钮等一些常用的控件。这些控件，在 Java 中都是由特定的类来生成对象实现的。

用 Java 开发 GUI 程序，基本方法就是创建一些用于交互的控件，按照一定的形式来组装，从而为应用程序提供想要的外观。而后需要为其中的某些控件编写程序，以处理用户的输入。这些控件，也被称为组件。

编写一个一个 GUI 程序，有三个最为重要的部分：组件的创建、布局管理和事件处理。下面先对这三个概念做一个简单的介绍。

14.1.1. 组件

组件（Component），又称为部件或控件，是具有特定功能且不能再分割的一种功能元件（相当于机械中的零件）。常见的组件有：按钮、文本框、表单、滚动条、菜单项、下拉框等等。Java 中的这些部件都以类的形式提供，所以称为组件类。

与组件紧密相关的另一个概念是容器。容器是用来存放组件或容器的一类组件，起到组织、管理的功能。

一般而言，组件具备下述性质：

- 大多从 Component 类派生出来
- 如果是容器，则可以使用 add()方法来添加其他组件
- 由于容器可以存放其他容器，所以容器是有层次的
- 任何一个组件只能放在一个容器中，但一个容器可以存放多个组件
- 与用户交互的组件通常都要响应某个事件

14.1.2 布局管理

创建了组件之后，一个难题是将如何安排这些组件的位置。在其他的语言中，这是一个相当棘手的问题，所以需要借助开发工具来完成这一任务，比如 Delphi、VB 这种 RAD 开发工具。而 Java 则提出了布局管理这一新概念，由系统自动来摆放各个组件，大大降低了布置组件的难度。程序员即便只用纯文本的开发工具，也可以轻松地开发出界面美观的 GUI 程序。

Java 中提供的传统布局有：FlowLayout、GridLayout、BorderLayout、CardLayout 等，在 Swing 中还增加了：BoxLayout、OverlayLayout、ScrollPaneLayout 和 ViewportLayout 等。

一般复杂的程序都会有很多组件，这些组件也可能不会放在同一个布局中，而是分布在各种布局之中。这些布局不再是是一个平面的，而是立体式的。编程时，一般先将界面分层进行布局，再将分好的层次像贴图片一样分层贴上去。

14.1.3 事件处理

用户可以对可视化的组件进行操作以通知程序自己想要做的事情。每对一个组件进行一次操作，就会产生相应的事件。程序员需要编程响应这些事件，也就响应了用户的操作，解决实际的问题。这一过程，就称为事件处理。

14.2 开发工具 Eclipse 简介

本书前面介绍读者使用 UltraEdit 来编辑 Java 源程序，它对于较小的程序而言确实是足够了。但如果程序比较大，要大量使用类库中的各种类，使用 UltraEdit 就显得工作量太大了一些，因为它缺乏一些诸如自动完成之类的辅助功能。本节将简单介绍一个功能强大的集成开发工具——Eclipse。它能够大大降低程序员编程的工作量。从本节起，大多数程序都将使用 Eclipse 来编写。

Eclipse 是开放源代码的项目，它最早是由 IBM 公司开发的。2001 年 11 月，IBM 公司捐出价值 4,000 万美元的源代码组建了 Eclipse 联盟，并由该联盟负责这种工具的后续开发。读者可以到 www.eclipse.org

去免费下载 Eclipse 的最新版本，一般 Eclipse 提供几个下载版本：Release，Stable Build，Integration Build 和 Nightly Build，建议下载 Release 或 Stable 版本。

虽然大多数用户很乐于将 Eclipse 当作 Java IDE 来使用，但 Eclipse 的目标不仅限于此。Eclipse 还包括插件开发环境（Plug-in Development Environment，PDE），这个组件主要针对希望扩展 Eclipse 的软件开发人员，因为它允许他们构建与 Eclipse 环境无缝集成的工具。由于 Eclipse 中的每样东西都是插件，对于给 Eclipse 提供插件，以及给用户提供一致和统一的集成开发环境而言，所有工具开发人员都具有同等的发挥场所。

这种平等和一致性并不仅限于 Java 开发工具。尽管 Eclipse 是使用 Java 语言开发的，但它的用途并不限于 Java 语言；例如，支持诸如 C/C++、COBOL 和 Eiffel 等编程语言的插件已经可用，或预计会推出。Eclipse 框架还可用来作为与软件开发无关的其他应用程序类型的基础，比如内容管理系统。

基于 Eclipse 的应用程序的突出例子是 IBM 的 WebSphere Studio Workbench，它构成了 IBM Java 开发工具系列的基础。例如，WebSphere Studio Application Developer 添加了对 JSP、servlet、EJB、XML、Web 服务和数据库访问的支持。

14.2.1 Eclipse 的安装与配置

要安装 Eclipse，首先必须正确安装 JDK1.4 以上版本。Eclipse 是绿色软件，只需要解压缩就可以运行，第一次运行时，它会自动检测 JDK 的安装路径，并配置好自己的运行环境，只有 Workspace 需要程序员手工配置。如图 14.2 所示：

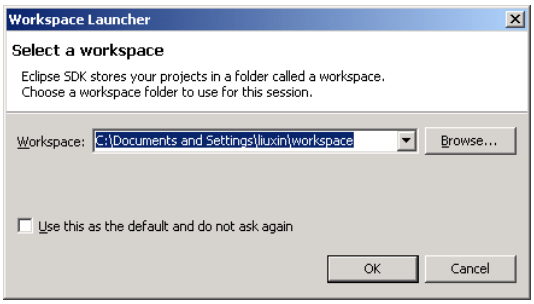

图 14.2　设置 Workspace

这是询问使用者以后用 Eclipse 建立的项目默认存放在哪里，读者可以自行设置成 Java 程序集中存放的位置。比如笔者将其设置为“e:\jdk1.5\exmaple”。如果以后想更改这个设置，也可以在建立项目时更改。

设置完 Workspace 之后，进入到 Eclipse 的主窗口。第一次进来的时候，窗口界面如下所示：

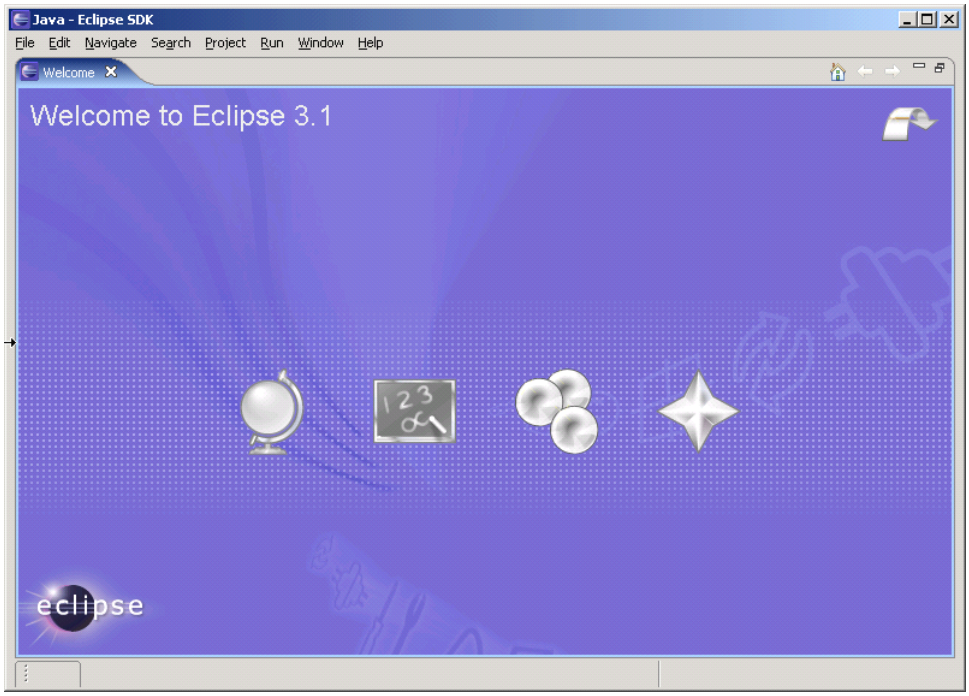

图 14.3　Eclipse 的欢迎界面

如果你不打算看它的帮助，可以单击标签上的“×”，直接关闭掉这个界面，就进入了 Eclipse 的编程界面，如图 14.3 所示：

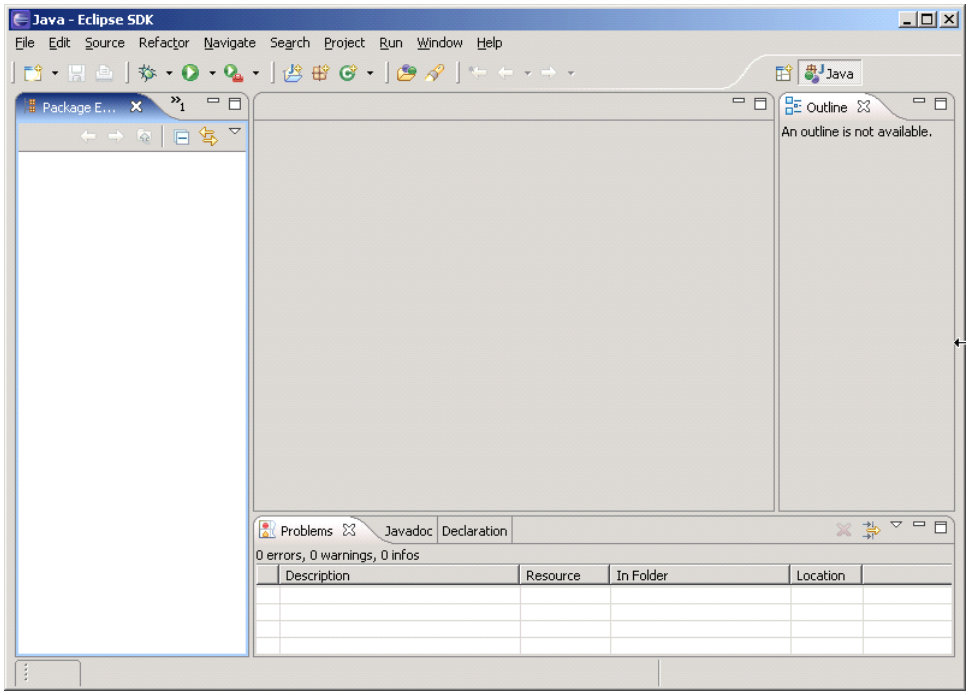

图 14.4　Eclipse 的编程界面

Eclipse 开发环境被称为 Workbench，它主要由三个部分组成：视图（Perspective），编辑窗口（Editor）和观察窗口（View）。图 14.4 中，因为还没有建立项目，所以编辑窗口没有打开。

14.2.2　用 Eclipse 创建一个项目

与 UltraEdit 可以直接编辑并编译运行一个 Java 源文件不同，在 Eclipse 中一个单一的源文件是无法编译运行的，只能先创建一个项目，然后在这个项目中添加源程序文件。

本节用 Eclipse 来创建一个简单的项目，项目名称为“Hello”，它仍然是一个控制台程序，会在屏幕上输出“Hello world”。

（1）创建一个新项目

在菜单中选择“File/New/Project”，出现如图 14.5 所示的窗口：

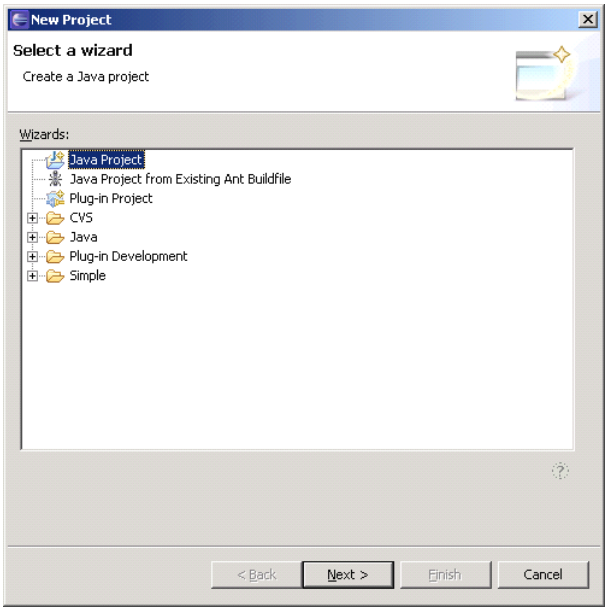

图 14.5　新建项目窗口

（2）输入项目名称和属性

选中其中的“Java Project”，然后单击“Next”按钮，出现如图 14.6 所示窗口：

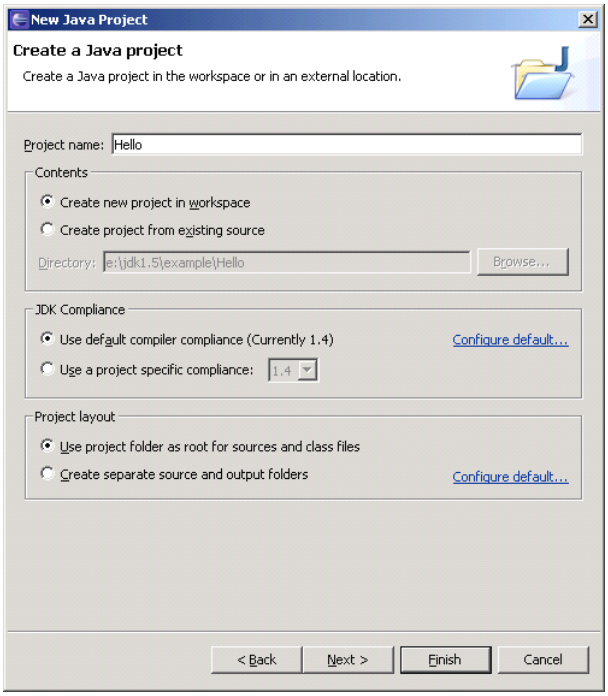

图 14.6　设置项目属性

在“Project name”所对应的编辑框中填入“Hello”，作为项目的名称。Eclipse 会自动在原先设置的 Workspace 目录下新建一个名为“Hello”的目录，作为本项目所有文件的存放地点。

再单击“Next”，出现如图 14.7 所示的窗口：

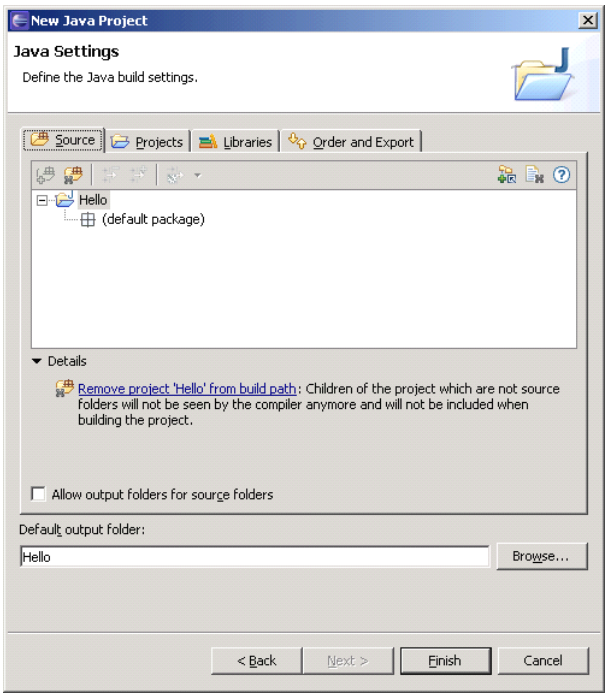

图 14.7　设置项目编译属性

这里可以设置项目编译的属性，一般不用改动它，直接单击“Finish”，就返回到主窗口。在左侧的视图中，出现了项目名称“Hello”，如图 14.8 所示：

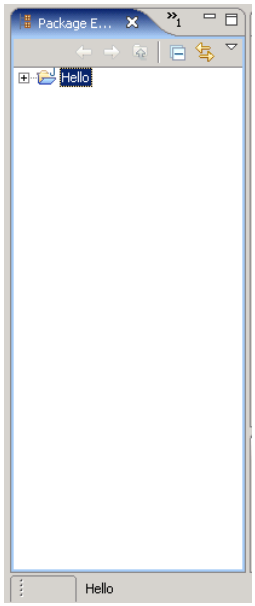

图 14.8　视图窗口

（3）向项目中插入一个新文件

现在右侧的编辑窗口仍然不能使用，因为还没有为项目创建源文件。所以还需要再次选择菜单“File/New/Class”，添加一个类文件，出现如图 14.9 所示窗口：

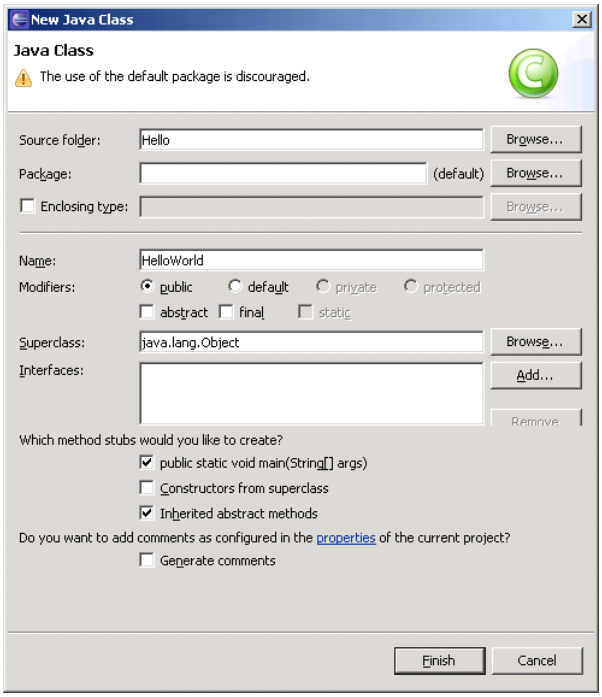

图 14.9　添加类文件

图中，只有 Name 所标识的文本框是必填的，它是类的名称，同时也是源文件的名称，这里写的是“HelloWorld”。在下面的复选框中，有一个“public static void main(String [] args)”选项，由于本类是需要直接运行的，所以选中这个复选框。单击“Finish”，回到主窗口，这次主窗口如图 14.10 所示：

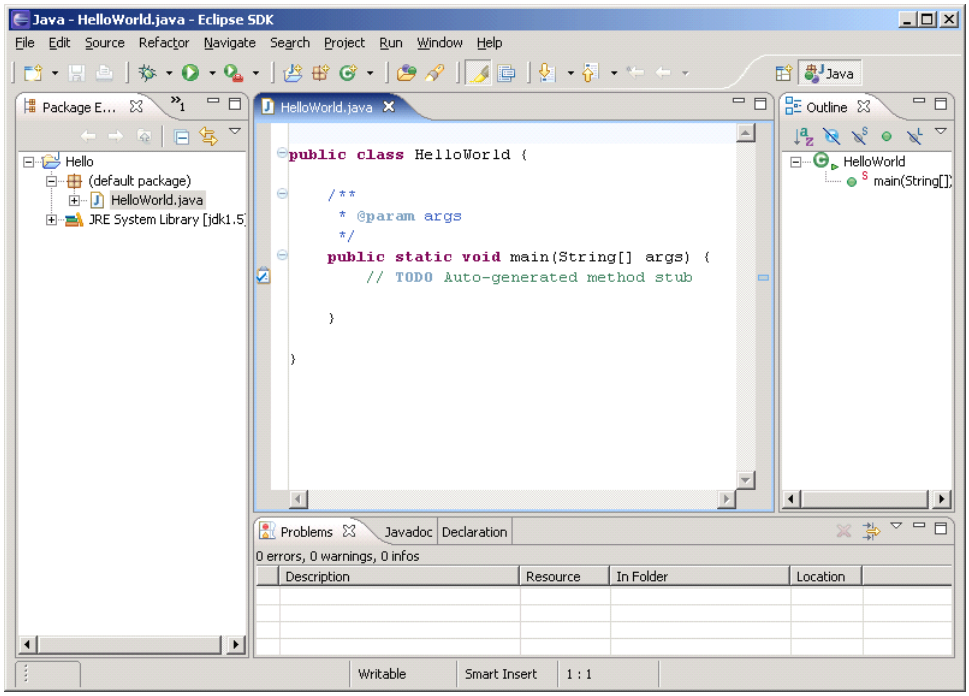

图 14.10　添加类文件之后的窗口

现在编辑窗口已经可用了，其中有 Eclipse 自动为类 HelloWorld 添加的一些代码。在左侧的视图中，也出现了 HelloWorld.java 的名字。接下来，就是向 main()方法中添加程序代码。

（4）Eclipse 的自动完成功能

在代码编辑过程中，Eclipse 提供了相当多的帮助，其中之一就是代码自动完成功能，当程序员输完一个类名，并打上“.”后，将如图 14.11 所示的窗口：

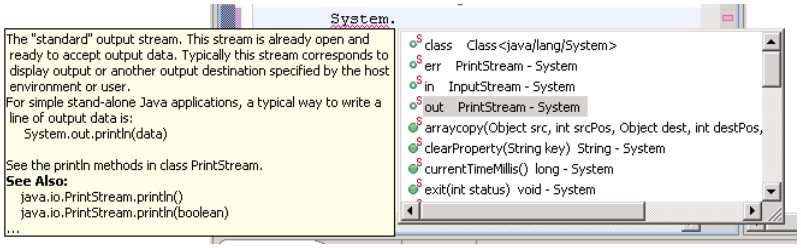

图 14.11　Eclipse 的自动完成功能

图 14.11 的右侧窗口是供程序员选择的当前类中的方法，左侧是对该方法的说明。程序员应该在右侧窗口中选择方法或属性名，以避免输入错误。

与此同时 Eclipse 还会自动查找编译错误，如果本行代码有错，将会在代码的右侧出现一个红色的提示符，并且会在它认为错误的地方画出一条红色的波浪线，这一点也非常有用。

（5）编译运行程序

当全部代码输入完成之后，如果没有编译错误，Eclipse 将自动对其进行编译。这时程序员只需要在编辑窗口按下鼠标右键，将有弹出菜单如图 14.12 所示：

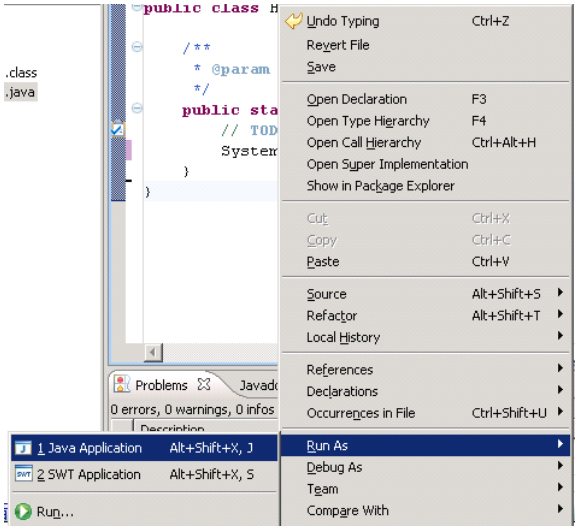

图 14.12　运行菜单

在菜单中选择“Run As/Java Application”，就可以开始运行程序。如果是该程序第一次运行，将出现一个如图 14.13 所示的资源保存窗口：

图 14.13　资源保存窗口

这里可以直接选择“OK”，Eclipse 将加载虚拟机运行程序。对于这种控制台程序，输出的结果将被 Eclipse 截获，显示在观察窗口中。如图 14.14 所示：

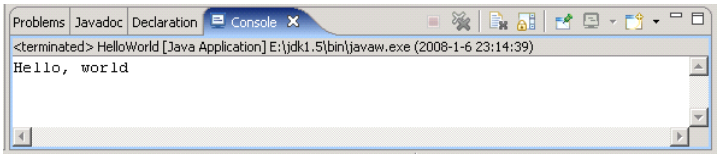

图 14.14　程序输出结果

到这里，一个简单的项目就介绍完了。但是，这只不过是 Eclipse 强大功能的一个小小展示。如果要全部介绍 Eclipse 的功能，足够写出厚厚的一本书，有兴趣的读者可以参考《精通 Eclipse》一书。

14.3 AWT 组件简介

AWT(Abstract Windowing Toolkit)，中文译为抽象窗口工具包，是 Java 提供的用来建立和设置 Java 的图形用户界面的基本工具。AWT 由 Java 中的 java.awt 包提供，里面包含了许多可用来建立与平台无关的图形用户界面(GUI)的类。

AWT 提供了 Java Applet 和 Java Application 中可用的用户图形界面 GUI 中的基本组件。由于 Java 是一种独立于平台的程序设计语言，但 GUI 却往往是依赖于特定平台的，Java 采用了相应的技术使得 AWT 能提供给应用程序独立于机器平台的接口，这保证了同一程序的 GUI 在不同机器上运行具有类似的外观（不一定完全一致）。

Java1.0 的 AWT（旧 AWT）和 Java1.1 以后的 AWT（新 AWT）有着很大的区别，新的 AWT 克服了旧 AWT 的很多缺点，在设计上有较大改进，使用也更方便，这里主要介绍新的 AWT，但在 Java1.1 及以后版本中旧的 AWT 的程序也可运行。

AWT 支持的图形用户界面编程的功能包括：用户界面组件；事件处理模型；图形和图像工具（包括形状、颜色和字体类）；布局管理器，可以进行灵活的窗口布局而与特定窗口的尺寸和屏幕分辨率无关；数据传送类，可以通过本地平台的剪贴板来进行剪切和粘贴。

Java 刚刚公布的时候，AWT 作为 Java 最弱的组件受到不小的批评。最根本的缺点是 AWT 在原生的用户界面之上仅提供了一个非常薄的抽象层。例如，生成一个 AWT 的复选框会导致 AWT 直接调用下层原生例程来生成一个复选框。不幸的是，一个 Windows 平台上的复选框同 MacOS 平台或者各种 UNIX 风格平台上的复选框并不是那么相同。

这种糟糕的设计选择使得那些拥护 Java“一次编写，到处运行”信条的程序员们过得并不舒畅，因为 AWT 并不能保证他们的应用在各种平台上表现得有多相似。一个 AWT 应用可能在 Windows 上表现很好可是到了 Macintosh 上几乎不能使用，或者正好相反。在 90 年代，程序员中流传着一个笑话：Java 的真正信条是“一次编写，到处测试”。导致这种糟糕局面的一个可能原因据说是 AWT 从概念产生到完成实现只用了一个月。

在第二版的 Java 开发包中，AWT 的器件很大程度上被 Swing 工具包替代。Swing 通过自己绘制器件而避免了 AWT 的种种弊端：Swing 调用本地图形子系统中的底层例程，而不是依赖操作系统的高层用户界面模块。

鉴于此，本书也不打算花大量篇幅详细介绍 AWT，而仅仅只用一节来做一个简要的介绍，后面会详细介绍 Swing 的使用。

14.3.1 AWT 组件的层次结构

与其他包相同，AWT 中的类也是按照继承关系来组织的，如图 14.15 所示：

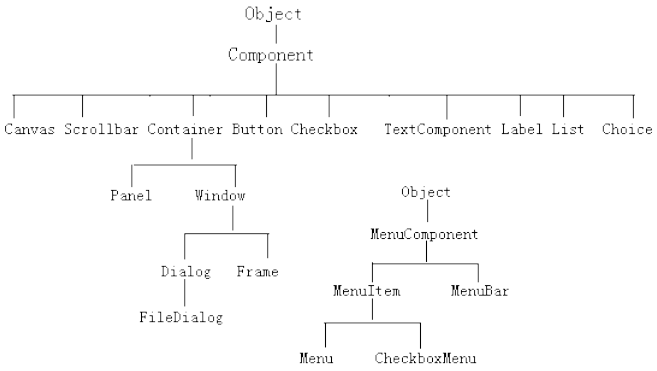

图 14.15　AWT 组件的层次结构

在 AWT 中，所有能在屏幕上显示的组件对应的类，均是抽象类 Component 的直接或间接子类或子孙类。这些类均可继承 Component 类的变量和方法。Container 类是 Component 的子类，它也是一个抽象类，它允许其他的组件加入其中。加入的 Component 也允许是 Container 类型，即允许多层嵌套的层次结构。Container 类在将组件以合适的形式安排在屏幕上时很有用，它有两个子类，Panel 和 Window, 它们不是抽象类。

Window 对应的类为 java.awt.Windows，它可独立于其他 Container 而存在，它有两个子类：Frame 和 Dialog。Frame 是具有标题和可调整大小的窗口(Window)。Dialog 则没有菜单条，虽然它能移动，但不能调整大小。滚动面板(ScrollPane)也是 Window 类的子类，这里就不讨论了。

Panel 对应的类为 java.awt.Panel，它可包含其他 Container 类型的组件，或包含在浏览器窗口中。Panel 标识了一个矩形区域，该区域允许其他组件放入。Panel 必须放在 Window 或其子类中才能显示。

AWT 中很重要的一类组件是菜单，但它不是从 Component 继承而来，而是从 MenuComponent 继承而来，这是因为菜单的外形和使用方法与其他的可视化组件有很大的区别。

14.3.2　AWT 中的基本组件和子包

java.awt 是整个系统中最大的包之一，这里不会详细介绍其中每一个组件，只简单列出其中的一些基本组件，如表 14.1 所示：

表 14.1 AWT中的基本组件介绍

类	作用
AWTEvent	封装AWT事件
AWTEventMulticaster	分配事件到多个事件监听器
BorderLayout	边界布局管理器。边界布局使用了五个方位：North，South，East，West和Center
Button	产生一个下压式按钮控件
Canvas	一个可以用来画各种图形的画布
CardLayout	卡片布局管理器。卡片布局仿效索引卡片。只有顶部的卡片可以看到。
Checkbox	产生一个复选框
CheckboxGroup	产生一个复选框控件组
CheckboxMenuItem	产生一个带开/关的菜单项
Choice	产生一个下拉列表框
Color	用可移植的、跨平台的方式来管理颜色
Component	各种AWT组件的抽象的超类
Container	一个可以用来存放其他组件的组件类的子类
Cursor	封装一个位图光标

Dialog	产生一个对话框窗口
Dimension	确定一个对象的尺寸，宽度存放在变量width中，高度存放在height中
Event	封装事件
EventQueue	给事件排队
FileDialog	产生一个用于选择文件的窗口
FlowLayout	流式布局管理器。流式布局从左到右，从上到下的定位组件
Font	封装字体
FontMetrics	封装各种和字体有关的信息。这些信息有助于在窗口中显示文本
Frame	产生一个具有标题栏、可调整大小的边框以及一个菜单栏的标准窗口
Graphics	封装图形上下文。这个上下文被各种输出方法使用来在一个窗口中输出图形或文本
GraphicsDevice	描述一个图形设备，比如一个屏幕和一个打印机
GraphicsEnvironment	描述各种Font和GraphicsDevice对象的集合
GridBagConstraints	定义各种与GridBagLayout类相关的常量
GridBagLayout	网格包布局管理器。网格包布局通根据GridBagConstraints提供的限制来布置组件
GridLayout	网格布局管理器。网格布局管理器用二维的网格来显示组件
Image	封装一个图形图像
Insets	封装一个容器的边框
Label	产生一个显示字符串的标签
List	产生一个用户可以选择的列表。与标准的窗口列表框相似
MediaTracker	管理媒体对象
Menu	产生一个下拉式菜单
MenuBar	产生一个菜单栏
MenuComponent	一个被各种菜单类所实现的抽象类
MenuItem	产生菜单项
MenuShortcut	封装与菜单项相应的快捷键
Panel	容器类的最简单的具体子类
Point	封装一个笛卡儿坐标描述的点，坐标分别存储在变量x和y中
Polygon	封装一个多边形
PopupMenu	产生一个弹出式菜单
PrintJob	代表一个打印机任务的抽象类
Rectangle	封装一个矩形
Robot	支持自动测试基于AWT的应用程序（Java2,v1.3新增）
Scrollbars	产生一个滚动条控件
ScrollPane	为另一个组件提供水平和/或垂直滚动条的容器
SystemColor	存放窗口，滚动条，文本以及其他GUI小部件的颜色
TextArea	生成多行编辑控件
TextComponent	TextArea和TextField的一个超类
TextField	生成一个单行编辑控件
Toolkit	由AWT实现的抽象类
Window	生成一个无框架，无菜单栏，无标题的窗口

表中所列出的组件，除了事件、字体、色彩、菜单和布局在后继版本中继续沿用外，多数可视化的组件都被 Swing 包中的新组件所取代。

java.awt 的包中，还包含了 11 个子包，简单说明如表 14.2 所示：

表 14.2　AWT中的子包说明

子包名	说明
java.awt.color	该包提供了用于颜色的类。类中一个颜色空间的实现， 该实现基于国际颜色联盟（International Color Consortium，简称ICC）的格式规范（版本3.4）
Java.awt.datatransfer	该包提供了在应用程序之间或之中传送数据的接口和类。 该包定义了一个“可传递”对象的概念，“可传递”对象通过实现Transferable接口来标识自己为可传递。另外，它还提供了一个剪切板机制，剪切板是一个临时含有一个可传递对象的对象， 通常用于复制和粘贴操作。尽管可以在应用程序中创建一个剪切板， 大多数应用程序一般都使用系统剪切板来确

	保数据能够在不同平台的应用程序之间传递。
Java.awt.dnd	提供了一些接口和类用于支持拖放（drag-and-drop）操作， 其定义了拖的源（drag-and-drop）和放的目标（drop-target）以及传递拖放数据的事件， 并对用户执行的操作给出可视的问馈。
java.awt.event	该包提供处理不同种类事件的接口和类，这些事件由AWT组件激发。 事件由事件源激发，事件监听者登记事件源，并接收事件源关于特定类型事件的通知。 Java.awt.event包定义了事件、事件监听者和事件监听者适配器。使用事件监听者适配器，更加容易编写事件监听者。
java.awt.font	该包提供与字体（font）相关的类和接口。
java.awt.geom	该包提供Java 2D类，用于定义和执行与二维几何相关的对象上的操作。
java.awt.im	该包提供一些类和一个输入法框架接口。该框架使得所有的文本编辑组件能够接收日文、中文和韩文的输入法的输入，输入法让用户使用键盘上有限的键输入成千上万个不同的字符，文本编辑组件可以使用java.awt.geom包和java.awt.event中相关类支持不同语言的输入法。同时，框架还支持其他语言的输入法或者其他输入方式，例如手写或语音识别。
java.awt.im.spi	该包提供一些接口，用于支持可以在任何Java运行时环境中使用的输入法的开发， 输入法是一个让用户输入文本的软件组件，通常用于输入日文、中文和韩文。同时， 还可以用于开发其他语言的输入法以及其他方式的输入，例如手写或语音识别。
java.awt.image	该包提供创建和修改图像的类。
java.awt.image.renderable	该包提供一些类和接口，用于生成与表现无关的图像。
java.awt.print	提供一些类和接口，用于普通的打印API，该API包括指定文档类型的能力、页面设置和页面格式控制的机制、管理任务控制对话框的能力

14.3.3 AWT 组件通用属性与方法

由于大多数的组件都是从 Component 继承下来的，所以 Component 的属性和方法就被这些子类所共享。常用的属性简要描述如下：

- Color：java.awt.Color 类可以定义颜色对象。它定义了大量的静态属性，来代表内置的颜色，所有颜色都由红、律、蓝混合而成，亮度均从 0 至 255。要改变一个可视组件的颜色，可以分别使用 setForeground(Color)和 setBackground(Color)方法来设置前景和背景颜色。
- 激活：一个可视组件在被创建时默认为处于激活状态，并具有一定的特征，表明它已经被选定。通过调用 setEnabled(Boolean)方法，可以激活或者停用一个组件。
- 可视性：一些组件在被创建时是自动可见的，而另外一些组件默认地呈现出它们所属的容器的可视性。如，在一个 Frame 组件中添加 Button 组件，如果 Frame 是不可视的，那么 Button 组件也不可见。反之，Button 是可见的。通过 setVisible(Boolean)可以修改组件的可视性。

可视化组件中常用的方法如表 14.3 所示：

表 14.3　可视化组件的常用方法

方法	说明
void add(PopupMenu popup)	为本组件增加指定的弹出式菜单
void addComponentListener(ComponentListener l)	为本组件安装事件监听器
void addFocusListener(FocusListener l)	为本组件安装获得焦点事件监听器
void addKeyListener(KeyListener l)	为本组件安装键盘事件监听器
void addMouseListener(MouseListener l)	为本组件安装鼠标事件监听器
boolean hasFocus()	如果本组件拥有焦点，返回true
boolean isShowing()	测试本组件是否显示在屏幕上
void repaint(int x, int y, int width, int height)	将本组件按照指定的位置和高度、宽度重绘
void setBackground(Color c)	设置组件的背景色
void setBounds(int x, int y, int width, int height)	移动并调整组件的大小
void setCursor(Cursor cursor)	设置光标的外形为指定的图形
void setEnabled(boolean b)	设置本组件是否可用
void setFont(Font f)	设置组件的字体

void setForeground(Color c)	设置组件的前景色
void setLocation(int x, int y)	移动组件到新的位置
void setSize(int width, int height)	重新设置组件的宽度和高度
void setVisible(boolean b)	设置组件是否可见
void transferFocus()	将输入焦点传递给下一个组件
void transferFocusBackward()	将输入焦点传递给上一个组件

这些方法虽然是 AWT 组件使用的，但实际上在 Swing 中，大多数方法仍然可以使用。

14.3.4 使用 AWT 编制 GUI 程序示例

AWT 组件可以使用在 Applet 和 Application 程序中，这里各举一例。如果读者忘了如何让 Applet 运行起来，可以查阅第 1 章。如果您已经在使用 Eclipse 了，则可以在文件窗口中使用右键菜单，其中有“Run/Java Applet”就可以直接运行它，而不必建立 html 文件。

【例 14.1】在 Applet 中使用 AWT 组件

//----------文件名 AWTComponents.java，程序编号 14.1----------

```
import java.applet.Applet;
import java.awt.*;
public class AWTComponents extends Applet {
   Label myLabel;
   List  myList;
   Button myBtn;
   Choice  myChoice;
   TextField myText;
   Panel myPanel;
   Checkbox myChk1,myChk2;
   Scrollbar myScrollbar;
   Container con;
   //Applet 程序的入口
   public void init(){
     //创建容器
     con=new Container();
     //创建标签
     myLabel=new Label("Label-标签");
     //创建一个显示 3 行的列表
     myList=new List(3);
     myList.add("List");
     myList.add("列表");
     myList.add("只显示三行");
     //创建一个 Panel 容器
     myPanel=new Panel();
     myPanel.setBackground(Color.red);
     //创建两个复选框
     myChk1=new Checkbox("Checkbox");
     myChk2=new Checkbox("复选框");
     //创建按钮
     myBtn=new Button("Button-按钮");
     //创建单行文本框
     myText=new TextField("TextField-单行文本框");
```

```
        //创建一个有两行下拉列表框
        myChoice=new Choice();
        myChoice.add("Choice");
        myChoice.add("下拉列表框");
        //创建滚动条
        myScrollbar=new Scrollbar(Scrollbar.HORIZONTAL, 0,10,0,30);
        //设置布局为 3 行 3 列
        con.setLayout(new GridLayout(3,3));
        //将上述可视化组件添加到容器中
        con.add(myLabel);
        con.add(myList);
        con.add(myPanel);
        con.add(myChk1);
        con.add(myChk2);
        con.add(myBtn);
        con.add(myChoice);
        con.add(myText);
        con.add(myScrollbar);
        //将容器加入到 Applet 对象中
        add(con);
    }
}
```

上面这个程序和第一章看的例子有个很大的区别：第一章的例子中，覆盖了 paint()方法，而这里覆盖的是 init()方法。这是因为，paint()方法适合程序在屏幕上画一些图形图像或文字；而如果要添加系统定义好的组件在 applet 容器上面，就需要用到 init()方法。

程序运行情况如图 14.16 所示：

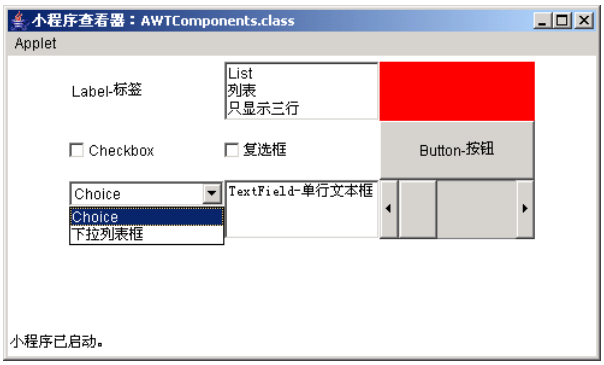

图 14.16　程序运行截图

下面再用 AWT 编写一个标准的窗口，这是一个 Application 程序。

【例 14.2】用 AWT 编写一个简单窗口

//----------文件名 AWTFrame.java，程序编号 14.2----------

```
import java.awt.*;
public class AWTFrame{
  Frame myFrame;     //Frame 是标准窗口
  Label myLabel;     //标签用于显示信息
  //在本类的构造方法中创建需要的组件
  public AWTFrame(){
    //创建窗口，并设置标题栏
```

```
    myFrame=new Frame("AWT 使用示例");
    //创建标签，并显示信息
    myLabel=new Label("世界，你好！");
    //添加到窗口
    myFrame.add(myLabel);
    //设置窗口大小
    myFrame.setSize(200,200);
    //设置窗口布局为流式布局
    myFrame.setLayout(new FlowLayout());
    //设置窗口可见
    myFrame.setVisible(true);
  }
  public static void main(String[] args) {
    new AWTFrame();
  }
}
```

编译并运行这个程序，如图 14.17 所示：

图 14.17　程序运行截图

这个窗口可以进行移动、改变大小、最大化、最小化等等操作，和普通窗口没有什么区别。其中的文字“世界，你好！”还会随着窗口大小的变化而改变位置，始终位于正中间。这正是使用布局的好处。如果没有布局管理，要实现这一功能将不得不编写数行代码。

唯一不足的是，这个窗口不能关闭，这是因为程序并没有提供响应关闭消息的代码。要响应消息，需要用到下一节的知识。

总结上面两个程序，可以看出，编写一个 GUI 程序的基本要素如下：

- 创建需要的组件，并设置属性；
- 设置合适的布局；
- 将组件添加到容器中；
- 为事件编写方法。

14.4　事件处理

用户在 GUI 程序中是通过鼠标或键盘对特定的界面元素（组件）进行控制，这些控制都是通过动作来完成的。而每一个动作都会引发一个系统预先定义好的事件。程序需要有相应的代码来处理这些事件。对于一些基本的事件系统会提供默认的处理代码，只有当程序员对这些默认的处理方式不认可时，才需要为它重新编写代码；而另外一些系统没有提供默认代码的事件，就必须由程序员编写代码来处理

正因为如此，GUI 程序是也被称为事件驱动。事件处理代码是编写一个成功的 GUI 程序的核心。

GUI 程序需要响应的事件大多是由用户触发的，也有少数由系统触发。这些事件以各种各样的方式传递给应用程序，而特定的方法总是依赖于实际的事件。

事件有很多种类型。最常见的事件是那些由鼠标、键盘在各种控件上触发的事件。这些事件在 java 的 java.awt.event 包中提供。尽管它们是在 awt 中提供的，但现在的 Swing 仍然在使用它们，所以本节对它们做详细的介绍。

14.4.1 授权事件模型

从 JDK1.1 起，Java 中处理事件的方法就是基于授权事件模型（delegation event model）的，这种模型定义了标准一致的机制去产生和处理事件。

它的概念十分简单：一个源（source）产生一个事件（event），并把它送到一个或多个的监听器（listeners）那里。在这种方案中，监听器简单地等待，直到它收到一个事件。一旦事件被接受，监听器将处理这些事件，然后返回。这种设计的优点是那些处理事件的应用程序可以明确地和那些用来产生那些事件的用户接口程序分开。一个用户接口元素可以授权一段特定的代码处理一个事件。

在授权事件模型中，监听器为了接受一个事件通知必须注册。这样有一个重要的好处：通知只被发送给那些想接受的它们的监听器那里。这是一种比 Java 1.0 版设计的方法更有效的处理事件的方法。以前，一个事件按照封装的层次被传递直到它被一个组件处理。这需要组件接受那些它们不处理的事件，所以这样浪费了宝贵的时间。而授权事件模型去掉了这个开销。

14.4.2 事件

在授权事件模型中，一个事件是一个描述了事件源的状态改变的对象。它可以作为一个人与图形用户接口相互作用的结果被产生。一些产生事件的活动可以是通过按一个按钮、用键盘输入一个字符、选择列表框中的一项、单击一下鼠标等用户操作产生。

另外，事件也可能不是由于用户操作组件直接发生的。例如，一个事件可能由于在定时器到期，一个计数器超过了一个值，一个软件或硬件错误发生，或者一个操作被完成而产生。

程序员还可以自由地定义一些适用于某个应用程序的事件。

14.4.3 事件源

一个事件源是一个产生事件的对象。当这个对象内部的状态以某种方式改变时，事件就会产生。一个事件源一次可能产生不止一种事件。

一个事件源必须注册监听器以便监听器可以接受关于一个特定事件的通知。每一种事件有它自己的注册方法。下面是一般的形式：

```
public void addTypeListener(TypeListener el)
```

在这里，type 是事件的名称，而 el 是一个事件监听器的对象。例如，要注册一个键盘事件。监听器的方法叫做 addKeyListener()，注册一个鼠标活动监听器的方法被叫做 addMouseMotionListener()，当一个事件发生时，所有被注册的监听器都被通知并收到一个事件对象的拷贝，这被称为多播（multicasting）。不过，无论如何，事件消息只被送给那些注册接受它们的监听器。

一些事件源可能只允许注册一个监听器。这种方法的通用形式如下所示：

```
public void addTypeListener(TypeListener el)
  throws java.util.TooManyListenersException
```

在这里，type 是事件的名称，el 是一个事件监听器的对象。当这样一个事件发生时，被注册的监听器被通知。这就是单播事件。

一个事件源必须也提供一个允许监听器注销一个特定事件的方法。这个方法的通用形式如下所示：

```
public void removeTypeListener(TypeListener el)
```

这里，type 是事件的名字，el 是一个事件监听器对象。例如，为了注销一个键盘监听器，需要调用 removeKeyListener()函数。

这些增加或删除监听器的方法被产生事件的事件源提供。例如，component 类提供了 14.4.7 小节中那些增加或删除键盘和鼠标事件监听器的方法。

14.4.4 事件监听器和适配器

一个事件监听器是一个在事件发生时被通知的对象。它有两个要求。首先，为了可以接受到特殊类型事件的通知它必须在事件源中已经被注册。第二，它必须实现接受和处理通知的方法。

用于接受和处理事件的方法在 java.awt.event 中被定义为一系列的接口。例如，MouseMotionListener 接口定义了两个在鼠标被拖动时接受通知的方法。如果实现这个接口，任何对象都可以接受并处理这些事件的一部分。

最底层的监听器都是以接口形式提供的，而一个接口往往含有多个方法。为了避免重复劳动，系统提供了对这些接口的实现类，这些类就称为适配器。程序员在注册监听器时，可以不必实现接口，而直接继承适配器，然后修改其中不符合要求的方法，这就降低了编程的工作量。

14.4.5 编写事件处理程序的基本方法

一般需要三个步骤来编写事件处理程序：

（1）写一个类，该类要么是要处理事件的适配器类的子类，要么是实现监听器接口，通常为如下形式：

```
class example implements ActionListener{……}
```

（2）实现该接口中的方法，或是重载适配器类中需要改写的方法：

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

（3）需要被监听的组件添加这个事件处理对象：

```
组件名.addActionListener(监听器对象)
```

这三个步骤中，（1）和（3）都是固定的，最具有难度也是最核心的部分是第（2）步。它也是整个 GUI 程序设计的核心。

14.4.6 响应窗口关闭事件处理示例

在 14.3.4 小节中，例 14.2 所举的 AWT 编写的窗口程序是无法关闭的，就是因为没有响应窗口的关闭事件。这里将这个程序改进一下，让其可以在用户单击关闭按钮时立即关闭掉。

【例 14.3】可以关闭的 AWT 窗口

//----------文件名 AWTFrame.java，程序编号 14.3----------

```
import java.awt.*;
import java.awt.event.*; //要加入事件包
//将本类声明为适配器 WindowAdapter 的子类，以便能响应关闭事件
public class AWTFrame  extends WindowAdapter{
```

```
  Frame myFrame;
  Label myLabel;
  public AWTFrame(){
    myFrame=new Frame("AWT 使用示例");
    myLabel=new Label("世界，你好！");
    myFrame.add(myLabel);
    myFrame.setSize(200,200);
    myFrame.setLayout(new FlowLayout());
    //增加监听器，事件响应的对象就是自己
    myFrame.addWindowListener(this);
    myFrame.setVisible(true);
  }
  //覆盖父类中的窗口关闭方法
  public void windowClosing(WindowEvent e){
    //关闭窗口
    myFrame.dispose();
  }
  public static void main(String[] args) {
    new AWTFrame();
  }
}
```

这个程序的运行情况和例 14.2 是一样的，唯一的区别是当用户按下关闭按钮时，窗口将像其他正常程序一样关闭，同时退出虚拟机。

分析程序 14.3 和 14.2，只有几个地方不同。

首先是类 AWTFrame 被声明为 WindowAdapter 的子类。这个类是适配器类，它实现了 WindowListener 接口，该接口是专用于响应各种窗口事件的。

由于窗口事件比较多，所以 WindowAdapter 中实现的方法也有很多。其中响应窗口关闭事件的方法是 windowClosing()，所以在程序中覆盖了此方法，添加上关闭窗口的语句：

```
myFrame.dispose();
```

当然，这里也可以不用上面的方法，而调用 System 类中的一个静态方法：

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

也是退出虚拟机。

在 windowClosing()方法中编写代码，其实就是响应窗口关闭事件。

最后要做的事情是注册（或称安装）监听器，这里的代码如下：

```
myFrame.addWindowListener(this);
```

由于要监听的是窗口事件，所以组件是 myFrame。方法中的参数是 this，这是因为本对象就是 WindowListener 的子类，可以响应窗口事件。

14.4.7 事件监听器接口和适配器类

AWT 中定义了很多事件监听器接口以及相应的适配器类，以适应各种事件。下面列出了一些常用的监听器以及适配器类，如表 14.4 所示：

表 14.4 事件接口和适配器类说明

监听器接口	适配器类	监听器中的方法	监听的事件
ActionListener	无	actionPerformed(ActionEvent e)	监听动作事件

AdjustmentListener	AWTEventMulticaster	adjustmentValueChanged(AdjustmentEvent e)	监听滚动条调整事件
ContainerListener	无	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)	监听容器事件
FocusListener	FocusAdapter	focusGained(FocusEvent e) focusLost(FocusEvent e)	监听输入焦点事件
InputMethodListener	AWTEventMulticaster	caretPositionChanged(InputMethodEvent e) inputMethodTextChanged(InputMethodEvent e)	监听输入法事件
ItemListener	无	itemStateChanged(ItemEvent e)	监听项事件，一般是指下拉框、列表框中的项
KeyListener	KeyAdapter	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)	监听键盘事件
MouseListener	MouseAdapter	mouseClicked(MouseEvent e) mouseEntered(MouseEvent e) mouseExited(MouseEvent e) mousePressed(MouseEvent e) mouseReleased(MouseEvent e)	监听鼠标事件
MouseMotionListener	MouseMotionAdapter	mouseDragged(MouseEvent e) mouseMoved(MouseEvent e)	监听鼠标移动事件
TextListener	无	textValueChanged(TextEvent e)	监听文本事件
WindowListener	WindowAdapter	windowActivated(WindowEvent e) windowClosed(WindowEvent e) windowClosing(WindowEvent e) windowDeactivated(WindowEvent e) windowDeiconnified(WindowEvent e) windowIconified(WindowEvent e) windowOpened(WindowEvent e)	监听窗口事件

14.4.8 作为参数的事件类

监听器中的每个方法都拥有一个形式参数，这些参数都是“***Event”的形式，表明这个参数是一个事件类。当某个事件发生时，系统会自动调用相应的方法，同时会传递一个对应的对象给处理者。这个对象中含有事件发生时的有关信息，处理者可以根据这些信息做出相应的动作。不同监听器中的方法的参数类型可以相同也可以不同，但同一监听器中的方法的参数类型一定相同。常用的事件类如表 14.5 所示：

14.5 java.awt.event中的主要事件类

事件类	说明
ActionEvent	通常在按下一个按钮，双击一个列表项或者选中一个菜单项时发生
AdjustmentEvent	当操作一个滚动条时发生
ComponentEvent	当一个组件隐藏，移动，改变大小或成为可见时发生
ContainerEvent	当一个组件从容器中加入或删除时发生
FocusEvent	当一个组件获得或失去键盘焦点时发生
InputEvent	所有组件的输入事件的抽象超类
ItemEvent	当一个复选框或列表项被单击时发生； 当一个选择框或一个可选择菜单的项被选择或取消时发生
KeyEvent	当输入从键盘获得时发生
MouseEvent	当鼠标被拖动，移动，单击，按下，释放时发生； 或者在鼠标进入或退出一个组件时发生
TextEvent	当文本区和文本域的文本改变时发生
WindowEvent	当一个窗口激活，关闭，失效，恢复，最小化，打开或退出时发生

下面对这些事件类做一个更详细的说明。

1. ActionEvent 类

在一个按钮被按下，列表框中的一项被选择，或者是一个菜单项被选择时都会产生一个 ActionEvent 类型的事件。在 ActionEvent 类中定义了四个用来表示功能修改的整型常量：

ALT_MASK，CTRL_MASK，META_MASK 和 SHIFT_MASK。除此之外，还有一个整型常量 ACTION_PERFORMED 用来标识事件。

ActionEvent 类有两个构造函数：

- ActionEvent(Object src, int type, String cmd)
- ActionEvent(Object src, int type, String cmd, int modifiers)

在这里，src 是一个事件源对象。事件的类型由 type 指定，cmd 是它的命令字符串，modifiers 这个参数显示了在事件发生时，ALT、CTRL、META 或 SHIFT 中的哪一个组合键被按下。

程序可以通过调用 ActionEvent 对象的 getActionCommand()方法来获得命令的名字，下面是这个方法的声明：

```
String getActionCommand()
```

例如，当一个按钮被按下时，一个 ActionEvent 类事件会产生，它的命令名和按钮上的显示的文字相同。

```
int getModifiers()
```

这个方法返回了一个值，它表示了在事件产生时 ALT、CTRL、META 或 SHIFT 这些组合键哪一个被按下。

2. AdjustmentEvent 类

一个 AdjustmentEvent 类的事件由一个滚动条产生。调整事件有五种类型。在 AdjustmentEvent 类中定义了用于标识它们的整型常量。这些常量和意义如下所示：

- BLOCK_DECREMENT：用户点击滚动条内部减少这个值
- BLOCK_INCREMENT：用户点击滚动条内部增加这个值
- TRACK：滑块被拖动
- UNIT_DECREMENT：滚动条左端的按钮被点击减少它的值
- UNIT_INCREMENT：滚动条右端的按钮被点击增加它的值
- ADJUSTMENT_VALUE_CHANGED：表示改变已经发生。

AdjustmentEvent 类有两个构造函数：

- AdjustmentEvent(Adjustable source, int id, int type, int value)
- AdjustmentEvent(Adjustable source, int id, int type, int value, boolean isAdjusting)

在这里，source 是一个产生事件的对象，id 等于 ADJUSTMENT_VALUE_CHANGED 这个常量，事件的类型由 type 指定，value 是与它相关的数据，isAdjusting 描述改变是否已经发生。

getAdjustable()方法返回了产生事件的对象。它的形式如下所示：

```
Adjustable getAdjustable()
```

通过 getAdjustmentType()方法，可以获得调整事件的类型。它返回被 AdjustmentEvent 定义的常量之一。下面是通常的形式：

```
int getAdjustmentType()
```

调整的值可以通过 getValue()方法获得，它的原形如下所示：

```
int getValue()
```

例如，当一个滚动条被调整时，这个方法返回了代表变化后的值。

3. ComponentEvent 类

一个 ComponentEvent 事件通常在一个组件的大小、位置或者是可视性发生了改变时产生。组件的事件类型有四种。ComponentEvent 这个类定义了用于标识它们的整型常量。这些常量和它们的意义如下所示：

- COMPONENT_HIDDEN：组件被隐藏
- COMPONENT_MOVED：组件被移动
- COMPONENT_RESIZED：组件被改变大小
- COMPONENT_SHOWN：组件被显示

ComponentEvent 类只有一个构造函数：

```
ComponentEvent(Component src, int type)
```

在这里，src 是产生事件的对象，Type 指定了事件的类型。

ComponentEvent 类是 ContainerEvent、FocusEvent、KeyEvent、MouseEvent 和 WindowEvent 这几个类的父类。

getComponent()方法返回了产生事件的组件。它的形式如下所示：

```
Component getComponent()
```

4. ContainerEvent 类

一个 ContainerEvent 事件是在容器中被加入或删除一个组件时产生的。容器有两种事件类型。在 ContainerEvent 类中定义了用于标识它们的整型常量：

- COMPONENT_ADDED：在容器中加入一个组件
- COMPONENT_REMOVED：在容器中删除一个组件

ContainerEvent 是 ComponentEvent 类的子类，它有如下所示构造函数：

```
ContainerEvent(Component src,int type,Component comp)
```

在这里，src 是产生事件的容器对象。Type 指定了事件的类型。Comp 指定了从容器中被加入或删除的组件。

你可以通过调用 getContainer()方法来获得产生这个事件的容器的一个引用，它的形式如下所示：

```
Container getContainer()
```

通过调用 getChild()方法可以返回在容器中被加入或删除的组件。它的通常形式如下所示：

```
Component getChild()
```

5. FocusEvent 类

一个 FocusEvent 是在一个组件获得或失去输入焦点时产生。这些事件用 FOCUS_GAINED 和 FOCUS_LOST 这两个整型变量来表示。

FocusEvent 类是 ComponentEvent 类的子类，它有两个构造函数：

- FocusEvent(Component src,int type)
- FocusEvent(Component src,int type,boolean temporaryFlag)

在这里，src 是产生事件的对象。Type 指定了事件的类型。如果焦点事件是暂时的，那么参数 temporaryFlag 被设为 true。否则，它是 false（一个暂时焦点事件被作为另一个用户接口操作的结果产生。例如，假如焦点在一个文本框中，如果用户移动鼠标去调整滚动条，这个焦点就会被暂时失去）。

通过调用 isTemporary()方法可以知道焦点的改变是否是暂时的。它的调用形式如下所示：

```
boolean isTemporary()
```

如果这个改变是暂时的，那么这个方法返回 true，否则返回 false。

6. InputEvent 类

InputEvent 抽象类是 ComponentEvent 类的子类，同时是一个组件输入事件的父类。它的子类包括 KeyEvent 类和 MouseEvent 类。在 InputEvent 类中定义了如下所示的八个整型常量，它们被用来获得任何和这个事件有关的修改符的信息。

- ALT_MASK
- BUTTON2_MASK
- META_MASK
- ALT_GRAPH_MASK
- BUTTON3_MASK
- SHIFT_MASK
- BUTTON1_MASK
- CTRL_MASK

另外它还提供了一些方法用来测试是否在事件发生时相应的修改符被按下。这些方法如下所示：

- boolean isAltDown()
- boolean isAltGraphDown()
- boolean isControlDown()
- boolean isMetaDown()
- boolean isShiftDown()

7. ItemEvent 类

一个 ItemEvent 事件是当一个复选框或者列表框被点击，或者是一个可选择的菜单项被选择或取消选定时产生（复选框和列表框在本书的后面将作论述）。这个事件有两种类型，它们可以用如下所示的整型常量标识：

- DESELECTED：用户取消选定的一项
- SELECTED：用户选择一项

除此之外，ItemEvent 类还定义了一个整型常量 ITEM_STATE_CHANGED，用它来表示一个状态的改变。

ItemEvent 类只有一个构造函数：

```
ItemEvent(ItemSelectable src,int type,Object entry,int state)
```

在这里，src 是一个产生事件的对象，例如，它可能是一个列表或可选择元素。Type 指定了事件的类型，产生该项事件的特殊项在 entry 中被传递，该项当前的状态由 state 表示。

GetItem()方法能被用来获得一个产生事件的项的引用。它的形式如下所示：

```
Object getItem()
```

getItemSelectable()方法被用来获得一个产生事件的 ItemSelectable 对象。如下所示：

```
ItemSelectable getItemSelectable()
```

8. KeyEvent 类

一个 KeyEvent 事件是当键盘输入发生时产生。键盘事件有三种，它们分别用整型常量：KEY_PRESSED，KEY_RELEASED 和 KEY_TYPED 来表示。前两个事件在任何键被按下或释放时发生。而最后一个事件只在输入一个字符时发生。

注意：不是所有被按下的键都产生字符。例如，按下 SHIFT 键就不能产生一个字符。

还有许多别的整型常量在 KeyEvent 类中被定义。例如，从 VK_0 到 VK_9、从 VK_A 到 VK_Z 等定义了与这些数字和字符等价的 ASCII 码。下面还有一些其他的字符：

- VK_ENTER
- VK_ESCAPE
- VK_CANCEL
- VK_UP
- VK_DOWN
- VK_LEFT
- VK_RIGHT
- VK_PAGE_DOWN
- VK_PAGE_UP
- VK_SHIFT
- VK_ALT
- VK_CONTROL

从名字很容易猜出这些虚拟键（virtual key codes）值的意思，并且只要按下对应的键，不必考虑是否同时还按下了 control、shift 或 alt 组合键，对应的虚拟键值都是相同的。

KeyEvent 类是 InputEvent 类的子类，它有两个构造方法：

- KeyEvent(Component src,int type,long when,int modifiers,int code)
- KeyEvent(Component src,int type,long when,int modifiers,int code,char ch)

在这里，src 是产生事件的组件对象。Type 指定了事件的类型。当这个键被按下时，系统时间存放在 when 中。参数 Modifiers 决定了在键盘事件发生时哪一个组合键同时被按下。像 VK_UP 和 VK_A 这样的虚拟键值存储在 code 中。如果与这些虚拟键值相对应的 ASCII 码字符存在，则存放在 ch 中，否则 ch 中的值是 CHAR_UNDEFINED。对于 KEY_TYPED 事件，code 值将是 VK_UNDEFINED。

KeyEvent 类定义了一些方法，但是其中用的最多的是用来返回一个被输入的字符的方法和用来返回键值的方法 getKeyCode()。它们的通常形式如下所示：

- char getKeyChar()
- int getKeyCode()

如果没有合法的字符可以返回，getKeyChar()方法将返回 CHAR_UNDEFINED。同样，在一个 KEY_TYPED 事件发生时，getKeyCode()方法返回的是 VK_UNDEFINED。

9. MouseEvent 类

鼠标事件有 7 种类型。在 MouseEvent 类中定义了如下所示的整型常量来表示它们：

- MOUSE_CLICKED：用户单击鼠标
- MOUSE_DRAGGED：用户拖动鼠标
- MOUSE_ENTERED：鼠标进入一个组件内
- MOUSE_EXITED：鼠标离开一个组件
- MOUSE_MOVED：鼠标移动
- MOUSE_PRESSED：鼠标被按下
- MOUSE_RELEASED：鼠标被释放

MouseEvent 类是 InputEvent 类的子类。它有如下所示的构造函数：

```
MouseEvent(Component src,int type,long when,int modifiers,
           int x,int y,int clicks,boolean triggersPopup)
```

在这里，src 是产生事件的组件对象。Type 指定了事件的类型。鼠标事件发生时的系统时间存储在 when。参数 modifiers 标识了在鼠标事件发生时哪一个组合键被按下。鼠标的坐标存储在 x 和 y 中。点

击的次数存储在 clicks 中。triggersPopup 标志决定了是否由这个事件引发弹出了一个弹出式菜单。

在这个类中用得最多的方法是 getX()和 getY()。它们返回了在事件发生时，对应的鼠标所在坐标点的 X 和 Y 值。方法形式如下所示：

- int getX()
- int getY()

相应的，也可以用 getPoint()方法去获得鼠标的坐标。形式如下所示：

```
Point getPoint()
```

它返回了一个 Point 对象，在这个对象中以整数成员变量的形式包含了 x 和 y 坐标。

translatePoint()方法可以改变事件发生的位置。它的声明如下所示：

```
void translatePoint(int x,int y)
```

在这里，参数 x 和 y 被加到了该事件的坐标中。

getClickCount()方法可以获得这个事件中鼠标的单击次数。它的声明如下所示：

```
int getClickCount()
```

isPopupTrigger()方法可以测试是否这个事件将引起一个弹出式菜单在平台中弹出。如下所示：

```
boolean isPopupTrigger()
```

10. TextEvent 类

当字符被用户输入或程序添加到文本框或文本域时，就会产生文本事件。TextEvent 类定义了 3 个整型常量：

TEXT_FIRST：用于文本事件的 id 范围的起始编号

TEXT_LAST：用于文本事件的 id 范围的结束编号

TEXT_VALUE_CHANGED：标志对象的文本已经被改变

该类只有一个构造方法，如下所示：

```
TextEvent(Object src,int type)
```

在这里，src 是一个产生事件的对象的引用。Type 指定了事件的类型。

TextEvent 类不包括在产生事件的文本组件中现有的字符。相反，程序必须用其他的与文本组件相关的方法来获得这些信息。由于这个原因，这里不需要讨论 TextEvent 类的方法。

11. WindowEvent 类

窗口事件有七种类型。在 WindowEvent 类中定义了用来表示它们的整数常量。这些常量和它们的意义如下所示：

- WINDOW_ACTIVATED：窗口被激活
- WINDOW_CLOSED：窗口已经被关闭
- WINDOW_CLOSING：用户要求窗口被关闭
- WINDOW_DEACTIVATED：窗口变为非激活状态
- WINDOW_DEICONIFIED：窗口被恢复
- WINDOW_ICONIFIED：窗口被最小化
- WINDOW_OPENED：窗口被打开

WindowEvent 类是 ComponentEvent 类的子类。它的构造函数如下所示：

```
WindowEvent(Window src,int type)
```

在这里，src 是一个产生事件的对象的引用。Type 指定了事件的类型。

在这个类中用的最多的方法是 getWindow()。它返回的是产生事件的 Window 对象。一般形式如下所示：

```
Window getWindow()
```

14.4.9 处理多个事件的例子

有了前面两节做基础，本节来介绍如何在一个程序中响应多种事件，这也是通常编程要处理的问题。

【例 14.4】响应单击按钮事件

在本例中，添加一个文本框和一个按钮。用户可以在文本框中输入数据，然后按下按钮，清除掉文本框中的数据。所以程序需要响应用户按下按钮的事件。同时，为了能够关闭程序，还需要响应窗口事件。

//----------文件名 reMulEvent.java，程序编号 14.4----------

```
import java.awt.*;
import java.awt.event.*;
//本类需要响应两种事件，所以需要继承窗口适配器类和动作接口
public class reMulEvent extends WindowAdapter implements ActionListener{
  Frame myFrame;
  Label mylabel;
  TextField myText;
  Button mybtn;
  public reMulEvent() {
    myFrame=new Frame("AWT 窗口演示");
    mylabel=new Label("世界，你好！");
    //创建输入文本框
    myText=new TextField("请在这里填入内容");
    //创建按钮
    mybtn=new Button("清空内容");
    //添加组件到窗口中
    myFrame.add(mylabel);
    myFrame.add(myText);
    myFrame.add(mybtn);
    //设置窗口大小
    myFrame.setSize(200,200);
    //设置流式布局
    myFrame.setLayout(new FlowLayout());
    //让窗口可见
    myFrame.setVisible(true);
    //为组件添加事件监听器，注意它们两个的区别
    //为窗口添加监听器
    myFrame.addWindowListener(this);
    //为按钮添加监听器
    mybtn.addActionListener(this);
  }
  //响应窗口关闭事件
  public void windowClosing(WindowEvent e){
    myFrame.dispose();
  }
  //响应按钮事件
  public void actionPerformed(ActionEvent e){
    //判断事件源是否为按钮
    if (e.getSource()==mybtn)
      //清空文本框
```

```
    myText.setText(null);
  }
  public static void main(String[] args) {
    new reMulEvent();
  }
}
```

程序运行情况如图 14.18 所示：

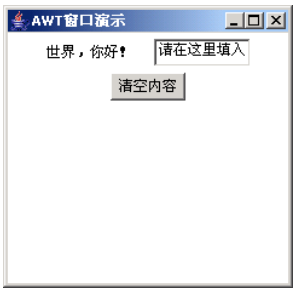

图 14.18　程序启动时截图

当用户按下“清空内容”按钮后，如图 14.19 所示：

图 14.19　按下按钮之后的截图

由于程序中响应了按下按钮事件，并且用语句“myText.setText(null);”清空了文本框中的内容，所以可以看到文本框已经为空。同样，程序还可以正常退出。

程序中，为两个组件添加了监听器，添加的模式差不多，但所用的方法以及对象都有区别：

```
//为窗口添加监听器
myFrame.addWindowListener(this);
//为按钮添加监听器
mybtn.addActionListener(this);
```

要被监听的对象一个是 myFrame，一个是 mybtn；添加的监听器方法一个是 addWindowListener，另一个是 addActionListener。不过用于监听的对象都是本身，所以参数都是 this。

下面再将上面的例子改得更复杂一点，对输入文本框的内容进行实时监控。如果输入的不是数字，则给出提示。这需要监听键盘事件，或者实现 KeyListener 接口，或者继承 KeyAdapter 类。由于本类已经继承了 WindowAdapter，不能再继承其他的类，所以采用内部类来继承。读者可以体会一下内部类在实现多重继承中的作用。

【例 14.5】监听键盘事件

//----------文件名 reMulEvent.java，程序编号 14.5----------

```
import java.awt.*;
import java.awt.event.*;
public class reMulEvent extends WindowAdapter implements ActionListener{
```

```
  Frame myFrame;
  Label mylabel;
  TextField myText;
  Button mybtn;
  // 这里写一个内部类来监听键盘事件，它由 KeyAdapter 派生出来
  public class myKeyAdapter extends KeyAdapter {
    //响应按键
    public void keyPressed(KeyEvent l) {
      //判断按下的键是否为数字键
      if (l.getKeyChar() < '0' || l.getKeyChar() > '9')
        mylabel.setText("你输入了非数字键");
      else
        mylabel.setText("你输入了数字键");
    }
  }
  public reMulEvent() {
    //创建监听器对象
    myKeyAdapter whatKey = new myKeyAdapter();
    myFrame=new Frame("AWT 窗口演示");
    mylabel=new Label("世界，你好！");
    myText=new TextField("请在这里填入内容");
    mybtn=new Button("清空内容");
    myFrame.add(mylabel);
    myFrame.add(myText);
    myFrame.add(mybtn);
    myFrame.setSize(200,200);
    myFrame.setLayout(new FlowLayout());
    myFrame.setVisible(true);
    //为窗口添加监听器
    myFrame.addWindowListener(this);
    //为按钮添加监听器
    mybtn.addActionListener(this);
    //为文本框添加监听器
    myText.addKeyListener(whatKey);
  }
  public void windowClosing(WindowEvent e){
    myFrame.dispose();
  }
  public void actionPerformed(ActionEvent e){
    if (e.getSource()==mybtn)
      myText.setText(null);
  }
  public static void main(String[] args) {
    new reMulEvent();
  }
}
```

当用户输入数字键时，程序运行情况如图 14.20 所示：

图 14.20　输入数字键截图

当用户输入非数字键时，程序运行情况如图 14.21 所示：

图 14.21　输入非数字键截图

这个类中增加了一个内部类：

```
public class myKeyAdapter extends KeyAdapter
```

它从 KeyAdapter 派生而来，所以可以接收键盘事件。然后在 main()方法中，再创建这个类的对象：

```
myKeyAdapter whatKey = new myKeyAdapter();
```

随后，为文本框添加监听器：

```
myText.addKeyListener(whatKey);
```

注意它的参数是 whatKey，而非 this。这是因为 whatKey 才能接收到键盘事件。本程序很好地体现了内部类的作用。当然，根据 4.13 节的介绍，这里的内部类也可以写成匿名内部类的形式。如果方法体内部的语句不多，写成匿名内部类比较简洁；反之，则写成本例的形式更清晰一些。

14.5　Swing 组件的特性

Swing 包是从 JDK1.2 起加入的界面设计接口。Swing 对 AWT 进行了大量的扩充，提供了一套非常漂亮的外观组件，也提供了不少新的功能。除了 AWT 中的组件如按钮、复选框和标签外，Swing 还包括许多新的组件，如选项板、滚动窗口、树、表格。许多一些开发人员已经熟悉的组件，如按钮，在 Swing 都增加了新功能。而且，按钮的状态改变时按钮的图标也可以随之改变。

与 AWT 组件不同，Swing 组件实现不包括任何与平台相关的代码。Swing 组件是纯 Java 代码，因此与平台无关。一般用轻量级（lightweight）这个术语描述这类组件。目前大多数的 GUI 程序都由 Swing 组件构成。

14.5.1　Swing 组件的优势

相比 AWT，Swing 组件的优势十分明显：

- 它是 100%的纯 Java 实现，没有本地代码，是一种轻量级组件，不依赖本机操作系统的支持。在不同的平台上表现一致，而且可以提供本地系统不支持的特性。

- 可插入的外观感觉(Pluggable Look and Feel,PL&F)。程序员可以在程序中加入相应的开关，使得用户可以根据喜好选择不同的界面，甚至可以设计自己的外观和感觉。
- 组件多样化。Swing 是在 AWT 基础上的扩展，以“J”开头，增加了一些更为高级的组件集合，如表格(Jtable)、树(Jtree)等。
- 采用 MVC（Model-View-Control）体系结构。它设置了三个通讯对象：模型、视图和控件。并将模型和视图分离开来，可以方便用户直接通过模型管理数据。
- 可存取性支持。所有 Swing 组件都实现了可存取性接口，使得辅助功能如屏幕阅读器、语音识别系统等，能十分方便地从 Swing 组件中得到信息。
- 支持键盘操作。通过使用 Jcomponent 类的 registerKeyboardAction 方法，能使用户通过键盘操作代替鼠标驱动 Swing 组件。
- 更漂亮的外表。许多 Swing 组件，如按钮、标签、菜单都可以添加图标，增强外观效果。

正因为有上面这些优势，所以 Swing 组件一经推出，就迅速取代了 AWT 成为了编写 GUI 程序的首选。

14.5.2 Swing 组件的体系结构

Swing 包中的一个小子集对应了基本的 AWT 组件，但不是基于对等模式。大部分的 Swing 组件没有使用 AWT 组件，所以原先的 AWT 集合是被 Swing 扩充而不是代替。程序员可以在程序中同时使用 Swing 和 AWT 组件。图 14.22 是二者的关系：

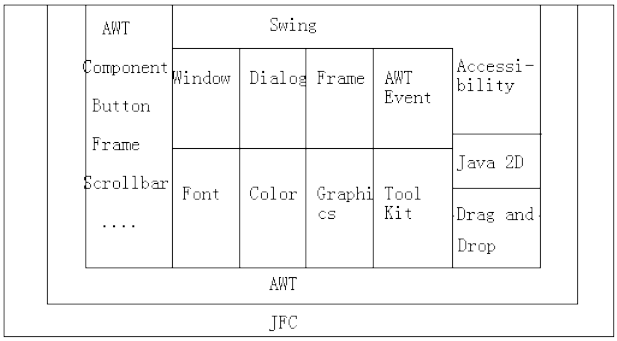

图 14.22 Swing 组件的体系结构图

Swing 的类库由 13 个包构成。其中 javax.swing 包含了大多数的组件、接口，而 javax.swing.event 则定义了 Swing 特有的事件类型和适配器。读者在 JDK API 手册中可以看到 Swing 包中所有的类及其关系图。

14.5.3 使用 Swing 组件编写 GUI 的层次结构

用 Swing 组件编写 GUI 程序与用 AWT 编写并没有太大的差别，层次结构图如 14.23 所示：

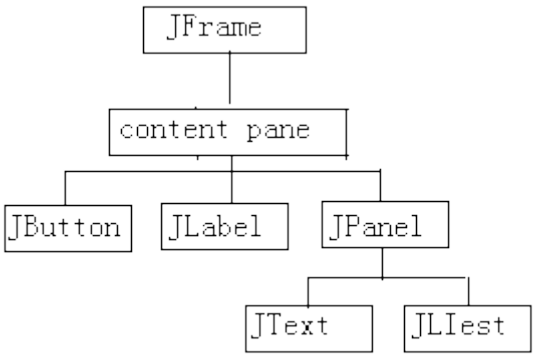

图 14.23　编写 GUI 程序的层次结构图

注意：JButton 不是直接放在 JFrame 上的。和 AWT 不同，系统规定 Swing 的普通组件不可直接放在顶层容器 JFrame 中。这是因为顶层容器是 AWT 组件 Frame 的直接子类，它是一个重量级组件，而一般 Swing 组件是轻量级的，可能会导致系统绘图出错。因此，所有的基本组件都应放在内容面板中。

Swing 组件可以分成下面几类：

- 顶层容器：只有 JFrame、JApplet、JDialog 和 JWindow
- 普通容器：也叫中间容器，包括 JPanel、JScrollPane、JSplitPane、JTabbedPane 等等
- 特殊容器：是起特殊作用的中间容器，包括 JInternalFrame、JLayeredPane、JRootPane 和 JToolBar
- 基本组件：如 JButton、JComboBox、JList、JMenu 等

为了与 AWT 区别，绝大多数 Swing 组件都以 J 开头。下面的章节将详细介绍这些组件的使用。

14.6　顶层容器

Swing 有四种顶层容器：JFrame、JDialog、JApplet 和 JWindow，它们都是 AWT 中 Window 的子类，因此 Swing 的顶层容器是具有对等类的重量级组件。

14.6.1　框架类（JFrame）使用示例

JFrame 是 Swing 中的顶层窗口，它是 Frame 的子类。JFrame 是极少数几个不绘制在画布上的 Swing 组件之一。因此，它的各个组成部件（按钮、标题栏、图标等）是由用户的窗口系统绘制，而不是由 Swing 绘制。

尽管 JFrame 继承了 Frame 中所有的非私有方法，但其中有些方法使用起来会出错。比如 add、setLayout 等。这是因为要添加的组件不能直接加在 JFrame 上，所以这些方法只能是通过内容面板来使用，通常像下面这个样子：

```
jframe.getContentPane().add(child);
```

其中的 getContentPane()是获取 JFrame 自带的内容面板，然后将组件 child 加在这个容器上面。JFrame 拥有多种面板，getContentPane()获取的是内容面板，另外它还有透明面板、根面板等。但通常用户只需要使用内容面板。

注意：Sun 公司似乎如何向 JFrame 添加组件在这个事情上摇摆不定。在 JDK1.3 中，可以直接将组件加在顶层容器上，只是有时候会出现绘制错误；而到了 JDK1.4 中，这么做一定会引发一个异常；到 JDK1.5 中，又允许直接使用 JFrame 的 add()方法，系统会自动将其加在内容面板上面。不过为了保险起见，读者最好还是显示地使用上面的方法。

JFrame 和 Frame 还有一个不同，就是它会响应窗口关闭事件，下面的例子简单演示了如何使用 JFrame 创建一个空白的窗口。

注意：Swing 组件位于 javax.swing 包中，需要引入才能使用。

【例 14.6】JFrame 简单使用示例

//----------文件名 demoJFrame.java，程序编号 14.6----------

```
//注意引入的 javax.swing 包
import javax.swing.*;
public class demoJFrame {
  JFrame mainJFrame;
  public demoJFrame() {
    //创建 JFrame 窗口
    mainJFrame=new JFrame("JFrame 演示窗口");
    //设置大小
    mainJFrame.setSize(200,200);
    //设置为可见
    mainJFrame.setVisible(true);
  }
  public static void main(String[] args) {
    new demoJFrame();
  }
}
```

这个程序和前面写的 AWT 窗口看上去没有什么不同，运行截图如 14.24 所示：

图 14.24　程序运行截图

这个窗口也是一个标准的应用程序窗口，可移动、可调整大小。与 AWT 窗口不同的是，它是可以可以关闭窗口的。当用户单击关闭窗口按钮，窗口会消失掉。不过，尽管窗口不见了，但是 JVM 并没有退出运行，用户可以通过任务管理器看到仍然有一个名为 java.exe 的进程在运行，需要通过任务管理器来关闭它。

要让 JVM 随着窗口的关闭（实际上，这里说关闭似乎并不准确，更为准确的说法是隐藏）而退出运行，可以使用两种方法：一是像 Frame 一样，写窗口响应的代码，用 System.exit()方法退出；另一种是直接使用 JFrame.setDefaultCloseOperation()，修改窗口关闭时的默认动作，指定要退出 JVM，这种方法更简单，本书的多数程序都使用这种方法。

注意：用 Swing 写 GUI 程序一定要记得采用上面两种方法之一来退出 JVM。否则每启动一个程序就会有一个 java.exe 进程驻留在内存中，很消耗系统的内存资源，而且这一问题并不容易发现。

下面的程序改正了上面的这个小 bug，并且增加了一个标签在窗口上，这就需要使用内容面板和布局。

【例 14.7】改正后的 JFrame 使用示例

//----------文件名 demoJFrame.java，程序编号 14.7----------

```
import javax.swing.*;
import java.awt.*;  //布局在此包中
public class demoJFrame {
  JFrame mainJFrame;
  JLabel myJLabel;

  public demoJFrame() {
    mainJFrame=new JFrame("JFrame 演示窗口");
    myJLabel=new JLabel("世界，你好");
    //设置流式布局
    mainJFrame.getContentPane().setLayout(new FlowLayout());
    //注意这种添加组件的方法，需要把组件加在内容面板上
    mainJFrame.getContentPane().add(myJLabel);
    mainJFrame.setSize(200,200);
    mainJFrame.setVisible(true);
    //设置关闭窗口时退出 JVM
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

  public static void main(String[] args) {
    new demoJFrame();
  }
}
```

程序运行截图如 14.25 所示：

图 14.25　程序运行截图

这个窗口在关闭时，会退出 JVM。

14.6.2　小应用程序（JApplet）使用示例

JApplet 类是 Applet 的子类，相对于 Applet，它增加了很多功能，比如它像 JFrame 一样拥有内容容器、透明容器和根容器。但是它与 JFrame 的限制也是一样的，需要将组件添加的内容面板上。

【例 14.8】JApplet 使用示例

//----------文件名 demoJApplet.java，程序编号 14.8----------

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

```
//本类必须是 JApplet 的子类
public class demoJApplet extends JApplet {
  JLabel myJLabel;
  //覆盖 paint 方法，在屏幕上画出图形和文字
  public void paint(Graphics g){
    g.drawRect(100,100,200,200);
    g.drawString("这是画出来的文字",50,50);
  }
  //用它来安放 Swing 组件
  public void init(){
    getContentPane().setLayout(new FlowLayout());
    myJLabel=new JLabel("这是标签上的文字");
    getContentPane().add(myJLabel);
  }
}
```

程序运行截图如 14.26 所示：

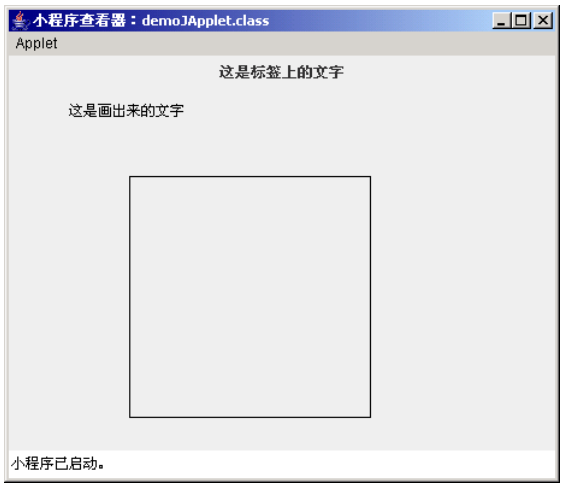

图 14.26　程序运行截图

在这个程序中混合使用了 paint 方法和 init 方法来分别画出文字图形，这个截图看上去还没有什么问题。但实际上这么做有很大的隐患。系统提供的 Swing 组件和用户自己绘制的图形文字并不能很好地共存，在重绘的时候经常会出现问题。因此建议读者只使用其中的一种方式。

14.6.3　对话框（JDialog）使用示例

JDialog 是 Dialog 的子类，是对话框窗口。对话框和普通窗口的形式上有一点区别：它不能最大和最小化，也不能改变窗口的大小。除此之外，和普通窗口都是一样的。Java 规定，JDialog 不能单独存在，需要依附于某个 JFrame 或 JApplet 才行。

【例 14.9】JDialog 简单使用示例

//----------文件名 demoJDialog.java，程序编号 14.9----------

```
import java.awt.FlowLayout;
import javax.swing.*;
public class demoJDialog {
  JFrame mainJFrame;
```

```
  JLabel myJLabel;
  JDialog subDialog;
  public demoJDialog() {
    mainJFrame=new JFrame("带对话框的 JFrame 演示窗口");
    myJLabel=new JLabel("世界，你好");
    //创建对话框
    subDialog=new JDialog(mainJFrame,"我是对话框");
    //为对话框设置布局
    subDialog.getContentPane().setLayout(new FlowLayout());
    //添加标签到对话框
    subDialog.getContentPane().add(myJLabel);
    mainJFrame.setSize(200,200);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //设置对话框的大小
    subDialog.setSize(200,200);
   //将对话框设置为可见
    subDialog.setVisible(true);
  }
  public static void main(String[] args) {
    new demoJDialog();
  }
}
```

程序运行截图如 14.27 所示：

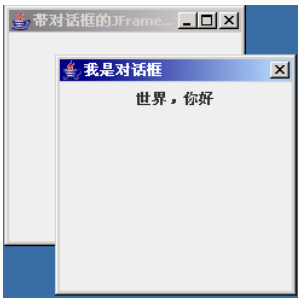

图 14.27　程序运行截图

图中，前面处于激活状态的窗口就是对话框。从程序中可以看出，设置对话框的各种属性以及添加组件的方法和 JFrame 是完全一样的。

不过创建对话框时，使用的是下面的语句：

```
subDialog=new JDialog(mainJFrame,"我是对话框");
```

它的第一个参数是指定本对话框所属的主窗口，第二个参数是标题栏。

对话框有两种形式：模态（modal）和非模态（non-modal），模态对话框是指如果不关闭对话框，无法对主窗口进行操作。非模态对话框则反之。使用哪一种对话框必须在创建时就必须指定，默认的是非模态，上面的例子中，窗口是非模态的。如果想创建模态对话框，需要使用下面的方法：

```
subDialog=new JDialog(mainJFrame, "我是对话框", true);
```

第三个参数为 true，就表示对话框是模态的。

对话框的最常见用途是给用户一些简单提示，在这种情况下如果还要象 JFrame 一样添加标签、按钮，就过于麻烦了，所以 JDK 给程序员提供了一些已经制作好的对话框，可以直接调用，它们都是模

态对话框的，下面是一个简单的例子。

【例 14.10】使用系统预定义的对话框

//----------文件名 showTriDialog.java，程序编号 14.10----------

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

public class showTriDialog implements ActionListener{
  JFrame mainJframe;
  JButton btnMessage,btnComfirm,btnInput;
  Container con;

  public showTriDialog() {
    mainJframe=new JFrame("对话框使用范例");
    btnMessage=new JButton("显示消息对话框");
    btnComfirm=new JButton("显示确认话框");
    btnInput=new JButton("显示输入话框");
    //获取内容面板的引用
    con=mainJframe.getContentPane();
    //向内容面板中添加组件
    con.add(btnComfirm);
    con.add(btnMessage);
    con.add(btnInput);
    //设置流式布局
    con.setLayout(new FlowLayout());
    //3 个按钮共用一个监听器
    btnComfirm.addActionListener(this);
    btnMessage.addActionListener(this);
    btnInput.addActionListener(this);
    mainJframe.setSize(200,200);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  //处理按钮事件
  public void actionPerformed(ActionEvent e){
      //判断事件源是哪一个按钮，并显示相应的对话框
      if(e.getSource()==btnComfirm)
         //显示有三个按钮的对话框
         JOptionPane.showConfirmDialog(mainJframe,
                   "这是一个有三个按钮的确认框，\n 按任意按钮返回");
      else if(e.getSource()==btnMessage)
         //显示只有一个确定按钮的消息对话框
         JOptionPane.showMessageDialog(mainJframe,"这是一个简单的消息框");
      else
         //显示一个有输入框的对话框
         JOptionPane.showInputDialog(mainJframe,"这是一个可供用户输入简单信息的对话框");
  }
  public static void main(String[] args) {
    new showTriDialog();
```

```
    }
}
```

程序启动时运行截图如 14.28 所示：

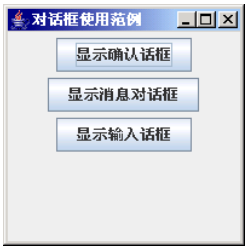

图 14.28　程序启动时截图

当单击“显示确认对话框”之后，运行截图如 14.29 所示：

图 14.29　确认对话框

当单击“显示消息对话框”之后，运行截图如 14.30 所示：

图 14.30　消息对话框

当单击“显示输入对话框”之后，运行截图如 14.31 所示：

图 14.31　输入对话框

这个程序虽然简单，但仍然有一些地方值得研究。

首先，主窗口中的三个按钮是用同一个对象的同一个方法在监听，为了区分事件由哪个按钮引发，需要用到形如：

```
if(e.getSource()==btnComfirm)
```

的方法。这里的 getSource()就是获取事件源的名称。

其次，要获取内容面板需要用到 mainJframe.getContentPane()方法，这里要添加多个按钮，所以这条语句要写多次，这比较麻烦。程序中采用了一个更常用的方法，先声明一个 Container 类型的变量，而后用：

```
con=mainJframe.getContentPane();
```

获取对内容面板的一个引用，以后就可以直接用 con 作 add()操作。

第三个值得注意的地方是 JOptionPane 类，这个类提供了各种静态方法以显示对话框，这里只用到

了其中的三种对话框，每一种对话框在显示时，还可以指定一些参数，程序员可以通过这些参数对对话框进行定制。更为详细的说明请参阅 API 手册中 JOptionPane 类的说明。

多数对话框可以返回用户的输入值，这里的程序没有获取对话框的返回值。不过本章后面的程序实例将演示如何获取这些返回值。

14.7 中间容器

一般来讲，顶层容器是重量级容器，而一些经常用来添加组件的轻量级容器就被称为中间容器。在 Swing 包中，中间容器的共同特点是不能独立存在，必须包含在其他顶层或中间容器中。本节就介绍这些常用的中间容器。

14.7.1 面板（JPanel）使用示例

JPanel 被称为面板，是最常用的的中间容器。一般情况下，先将各种基本组件添加到其中，然后再将 JPanel 添加到其他容器（特别是顶层容器）中。JPanel 的默认布局是流式布局，先看下面这个简单例子。

【例 14.12】面板使用简单示例

//----------文件名 demoJPanel.java，程序编号 14.12----------

```
import java.awt.FlowLayout;
import javax.swing.*;
  public class demoJPanel {
    JPanel myJPanel;
    JFrame mainJFrame;
    JLabel myJlabel;
    public demoJPanel(){
      mainJFrame=new JFrame("JPanel 使用示例");
      myJlabel=new JLabel("本标签放在 JPanel 上");
      myJPanel=new JPanel();
      //设置本面板的布局
      myJPanel.setLayout(new FlowLayout());
      //将标签添加到面板上
      myJPanel.add(myJlabel);
      //可以直接将面板添加到窗口中
      mainJFrame.add(myJPanel);
      mainJFrame.setSize(200,200);
      mainJFrame.setVisible(true);
      mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args){
      new demoJPanel();
    }
}
```

程序运行截图如 14.32 所示：

图 14.32　程序运行截图

从图片上看，完全看不到 JPanel 的踪迹，中间容器大多数如此，它们仅仅起到一个容纳以及方便布局的作用，本身是不会显示出来的。程序中还有一个地方值得注意，JPanel 可以直接添加到窗口上，因为它本身就是中间容器。

JPanel 一个作用是方便布局，这将在本章后面的实例中介绍。另外它还有一个很重要的作用，就是当作画布来用。

【例 14.13】在面板上画出简单图形示例 1

//----------文件名 painting_1.java，程序编号 14.13----------

```
import java.awt.*;
import javax.swing.*;
public class painting_1{
  MyCanvas palette;
  JFrame mainJFrame;
  //用一个内部类继承并扩展 JPanel 类
  public class MyCanvas extends JPanel{
    //当控件发生变化时，系统会调用此方法，所以需要重载此方法
    public void paintComponent(Graphics g){
      //设置画笔颜色为红色
      g.setColor(Color.red);
      //画一个直径 100 的圆
      g.drawArc(10,10,100,100,0,360);
      //画一个正方形
      g.drawRect(50,50,100,100);
      //将文字画出来
      g.drawString("测试",30,30);
    }
  }
  public painting_1(){
    mainJFrame=new JFrame("用 JPanel 画图示例");
    palette=new MyCanvas();
    mainJFrame.add(palette);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args){
     new painting_1();
   }
}
```

程序运行截图如 14.33 所示：

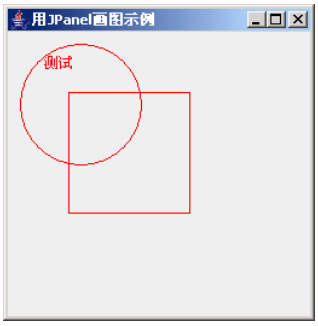

图 14.33　程序运行截图

当这个窗口改变时（包括最大化、最小化和调整窗口大小），窗口中的图形和文字仍然可以正常显示。这看上去似乎是理所应当的。但实际上这要归功于 paintComponent()方法。本程序中所有绘制图形的语句都写在这个方法中。该方法会在 JPanel 发生改变时被系统自动调用，其中的绘图语句自然会被执行。这一点很像 Applet 中的 paint()方法，包括它的参数 Graphics 的使用。

下面的程序中，将要绘制更为复杂的图形，不过这次使用的方法不同，它是由用户单击按钮之后才会画图。

【例 14.14】在面板上画出简单图形示例 2

//----------文件名 painting_2.java，程序编号 14.14----------

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.Math;
public class painting_2 {
  JPanel palette;
  JFrame mainJFrame;
  JButton btn;
  Container con;
  dealPushBtn handleBtn;
  //定义一个内部类来响应按钮事件
  public class dealPushBtn implements ActionListener{
    //当用户按下按钮时，在画布上画出图形
    public void actionPerformed(ActionEvent e){
      Graphics g;
      final int orign_x=100,orign_y=100, radio=50; //圆的初始位置和半径
      final double scope=30.0/180.0*Math.PI;//每次转动的角度
      double angle=0.0;
      int x=orign_x+radio,y=orign_y;
      //注意画笔的获取方式，这是获取面板的画笔
      g=palette.getGraphics();
      //设置画笔颜色为绿色
      g.setColor(Color.green);
      //按照顺时针方向画出 12 个同样大小的圆形
      for(int i=0;i<12;i++){
        g.drawArc(x,y,radio*2,radio*2,0,360);
        //计算下一个圆的位置
        angle += scope;
        x=(int)(radio*Math.cos(angle)+orign_x);
```

```
        y=(int)(radio*Math.sin(angle)+orign_y);
      }
  }
}
  public painting_2(){
    mainJFrame=new JFrame("用 JPanel 画图示例");
    handleBtn=new dealPushBtn();
    btn=new JButton("画图");
    btn.addActionListener(handleBtn);
    palette=new JPanel();
    palette.add(btn);
    mainJFrame.add(palette);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    palette.setLayout(new FlowLayout());
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
  }
  public static void main(String[] args){
    new painting_2();
  }
}
```

程序启动时截图如 14.34 所示：

图 14.34　程序启动时截图

当单击“画图”按钮之后，程序的截图如 14.35 所示：

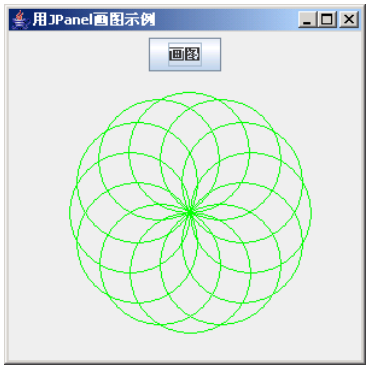

图 14.35　按下“画图”按钮之后截图

这个图形比前面的例题看上去要复杂一些，其实也不过是将 12 个同样大小的圆按照等距离绘制在面板上。如何画这幅图在这里并不是重点，读者应该要注意到上面两个程序之间的差别：同样是利用 Graphics 来画图，在程序 14.13 中，可以直接使用参数，这个参数是由系统调用时传递进来的；在程序 14.14 中，没有这个参数可以使用，所以必须用“palette.getGraphics()”的形式来获取画笔。大多数的组件都有这个方法，一旦获取了某组件的画笔，所绘制的图形就在该组件上面。

两个程序的另外一个差别是：当程序 14.14 的窗口改变时（例如改变大小、最大化、最小化），上面的图形就会不见了，当然按钮还是在上面。这是因为按钮是用 add()方法添加上去的，面板知道自己在什么位置上有个什么样的组件，一旦重绘事件发生，就会通知该组件重新绘制自己；而这幅图片是程序另外画上去的，面板并不知道它的存在，自然也不会重绘。要想取得像程序 14.13 那样的效果，还需要响应 paintComponent 事件。对于程序的改进，留给读者自行练习。

14.7.2 滚动面板（JScrollPane）使用示例

如果组件尺寸较大，在窗口的可视区域内不能全部显示，就需要一个滚动窗口。而 JScrollPane 就提供了水平和垂直滚动条，配合 ScrollPaneLayout 设定布局，就可以构成需要的滚动窗口。

JScrollPane 提供了 4 个构造方法：

- JScrollPane()：创建一个空面板，水平和垂直滚动条在需要的时候出现
- JScrollPane(int vsbPolicy, int hsbPolicy)：创建一个空面板，两个参数分别指定垂直和水平滚动条的使用策略
- JScrollPane(Component view)：创建一个含有组件 view 的面板，垂直和水平滚动条始终出现
- JScrollPane(Component view, int vsbPolicy, int hsbPolicy)创建一个含有组件 view 的面板，后两个参数分别指定垂直和水平滚动条的使用策略

JScrollPane 还定义了相关的常量代表垂直和水平滚动条的使用策略：

- HORIZONTAL_SCROLLBAR_ALWAYS：水平滚动条始终出现
- HORIZONTAL_SCROLLBAR_AS_NEEDED：水平滚动条在需要的时候出现
- HORIZONTAL_SCROLLBAR_NEVER：水平滚动条永不出现
- VERTICAL_SCROLLBAR_ALWAYS：垂直滚动条始终出现
- VERTICAL_SCROLLBAR_AS_NEEDED：垂直滚动条在需要的时候出现
- VERTICAL_SCROLLBAR_NEVER：垂直滚动条永不出现

使用 JScrollPane 类的时候会要向其中添加组件，它提供了两种方法：一是在创建时就将组件添加到其中；二是使用 JViewport 类的 add()方法来实现。下面分别介绍。

【例 14.15】在 JScrollPane 创建时添加组件示例

//----------文件名 demoJScrollPane_1.java，程序编号 14.15----------

```
import javax.swing.*;
public class demoJScrollPane_1 {
  JScrollPane myJspane;
  JLabel myJlbl;
  JFrame myJFrame;
  public demoJScrollPane_1() {
   myJFrame=new JFrame("JScrollPane 使用示例");
   //加载一个图片文件
   Icon picture = new ImageIcon("test.jpg");
   myJlbl=new JLabel(picture);
```

```
   //创建面板，同时将标签添加到上面
   myJspane=new JScrollPane(myJlbl);
   //设置面板的布局方式，它只有这一种布局方式
   myJspane.setLayout(new ScrollPaneLayout());
   myJFrame.add(myJspane);
   myJFrame.setSize(300,300);
   myJFrame.setVisible(true);
   myJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJScrollPane_1();
  }
}
```

程序运行截图如 14.36 所示：

图 14.36　程序运行截图

这个程序运行时需要加载一副图片，文件名为 test.jpg，该文件必须与程序放在同一目录下。

将组件添加到 JScrollPane 中的另外一种方法是调用 add()方法，但该方法并非 JScrollPane 所有，而是 JViewport 类所有。JViewport 是指在一个滚动窗口中有一个视点，当用户拖动滚动条时，移动的是视点，就像照相机的镜头一样。通常 JViewport 要与 JScrollPane 配合使用，它们之间的关系如图 14.37 所示：

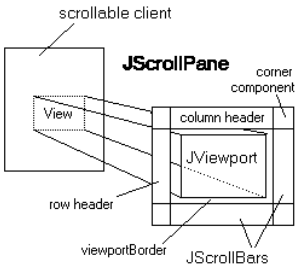

图 14.37　JViewport 与 JScrollPane 的关系

在 JScrollPane 类中，提供了一个方法：getViewport()，可以获得与当前对象相关联的 JViewport 对象，然后再使用该对象的 add()方法就可以向其中添加组件了。

【例 14.16】通过 add()方法添加组件示例

//----------文件名 demoJScrollPane_2.java，程序编号 14.16----------

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

```
public class demoJScrollPane_2 {
  JScrollPane myJspane;
  JLabel myJlbl;
  JFrame myJFrame;
  public demoJScrollPane_2() {
   myJFrame=new JFrame("JScrollPane 使用示例");
   //加载一个图片文件
   Icon picture = new ImageIcon("test.jpg");
   myJlbl=new JLabel(picture);
   //创建空面板
   myJspane=new JScrollPane();
   //设置面板的布局方式
   myJspane.setLayout(new ScrollPaneLayout());
    //向其中添加标签
   myJspane.getViewport().add(myJlbl);
   myJFrame.add(myJspane);
   myJFrame.setSize(300,300);
   myJFrame.setVisible(true);
   myJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJScrollPane_2();
  }
}
```

程序 14.16 和 14.15 的运行情况是完全一样的。

注意：无论采取哪种方法，都只能在 JScrollPane 中放置一个组件，如果多次添加，后面添加的组件将覆盖前面的。如果有多个组件要存放在 JScrollPane 中，需要先将这些组件存放在另外一个中间容器中，再将这个中间容器添加到 JScrollPane 中。

14.7.3 分隔板（JSplitPane）使用示例

JSplitPane 可以将一个窗口分隔成为两个相对独立的区域。分隔方式有水平和垂直两种，用户可以拖动分隔板来改变区域大小。

它有多个构造方法，其中最常用的构造方法为：

```
JSplitPane(int newOrientation, Boolean newContinuousLayout,
                              Component left, Component right);
```

参数 newOrientation 指示分隔方向，可以是 JSplitPane.HORIZONTAL_SPLIT（水平）或 JSplitPane.VERTICAL_SPLIT（垂直）之一。参数 newContinuousLayout 决定当分隔板移动时，组件是否连续变化。left 是放置在左侧或上方的组件。right 是放置在右侧或下方的组件。

除了构造方法，JSplitPane 也提供了对分隔的组件进行重新设置的方法：

- setBottomComponent(Component comp)：在右边或下边添加组件
- setLeftComponent(Component comp)：在左边或上边添加组件
- setContinuousLayout(boolean newContinuousLayout)：设置移动时是否连续显示
- setOrientation(int orientation)：设置分隔方式

【例 14.17】分隔板简单示例 1

//----------文件名 demoJSplitPane_1.java，程序编号 14.17----------

```
import javax.swing.*;
public class demoJSplitPane_1 {
  JSplitPane myJsplitpane;
  JLabel myJlbl1,myJlbl2;
  JFrame myJFrame;
  public demoJSplitPane_1() {
    myJFrame=new JFrame("JSplitPane 使用示例");
    myJlbl1=new JLabel("这是左边窗口");
    myJlbl2=new JLabel("这是右边窗口");
    //创建分隔板，设置分割方式为水平
    myJsplitpane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,
                              myJlbl1,myJlbl2);
    myJFrame.add(myJsplitpane);
    myJFrame.setSize(200,100);
    myJFrame.setVisible(true);
    myJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJSplitPane_1();
  }
}
```

程序运行截图如 14.38 所示：

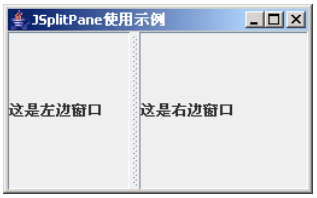

图 14.38　程序运行截图

图中的分隔条可以左右拖动，其中的文字也会随之移动。

与 JScrollPane 相同，在 JSplitPane 的一侧只能放置一个组件。如果需要多个组件，需要将这些组件放在另外一个中间容器中（这个中间容器也可以是 JSplitPane），再添加进来。下面的例子中，将一个窗口分割成为 4 个区域，就是采用的这种方法。

【例 14.18】分隔板示例 2

//----------文件名 demoJSplitPane_2.java，程序编号 14.18----------

```
import javax.swing.*;
public class demoJSplitPane_2 {
  JSplitPane leftJsplitpane,rightJsplitpane,mainJsplitpane;
  JLabel myJlbl1,myJlbl2,myJlbl3,myJlbl4;
  JFrame mainJFrame;
  public demoJSplitPane_2() {
    mainJFrame=new JFrame("JSplitPane 使用示例");
    myJlbl1=new JLabel("这是左侧上边的窗口");
    myJlbl2=new JLabel("这是左侧下边的窗口");
    //设置分隔板为垂直分割，两个标签分别添加到上、下区域
    leftJsplitpane=new JSplitPane(JSplitPane.VERTICAL_SPLIT,true,
                                myJlbl1,myJlbl2);
```

```
    myJlbl3=new JLabel("这是右侧上边的窗口");
    myJlbl4=new JLabel("这是右侧下边的窗口");
    //设置分隔板为垂直分割，两个标签分别添加到上、下区域
    rightJsplitpane=new JSplitPane(JSplitPane.VERTICAL_SPLIT,true,
                                   myJlbl3,myJlbl4);
    //设置分隔板为水平分割，前面两个分隔板分别添加到左、右区域
    mainJsplitpane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,true,
                                  leftJsplitpane,rightJsplitpane);
    mainJFrame.add(mainJsplitpane);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJSplitPane_2();
  }
}
```

程序运行截图如 14.39 所示：

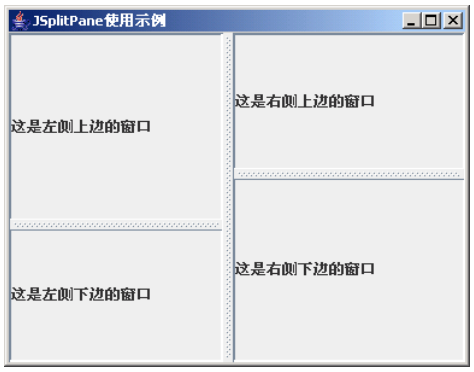

图 14.39　程序运行截图

无论是水平分割还是垂直分割，默认的方式都是将左侧或上方的区域压缩到恰好是放置在其中的组件大小，而将剩余区域全部留给右侧和下方。如果这么做不太美观，就需要使用 setDividerLocation()方法来设置两边区域的大小。

14.7.4　选项板（JTabbedPane）使用示例

JTabbedPane 允许用户单击不同的标签来选择不同的选项卡中的组件，中标签的位置可以在上方也可以在下方。

它有 3 个构造方法：

- JTabbedPane()：创建一个空的选项板，标签位置默认在选项板的顶部，布局也采用默认格式。
- JTabbedPane(int tabPlacement)：创建一个空的选项板，标签位置由 tabPlacement 决定，它可以是 JTabbedPane.TOP、JTabbedPane.BOTTOM、JTabbedPane.LEFT 或 JTabbedPane.RIGHT 之一。
- JTabbedPane(int tabPlacement, int tabLayoutPolicy)：创建一个空的选项板，标签位置由 tabPlacement 决定，布局由 tabLayoutPolicy 决定。

JTabbedPane 有两类方法可以向其中添加组件：一是 add()方法，它可以向已经定义好标签选项卡内

添加组件；一是 addTab()，它添加新的选项卡和标签，以及放在选项板内的组件，实际编程中，常使用第二种方法。

addTab()方法有 3 个重载的方法：

- void addTab(String title, Component component)：增加一个标题为 title 的标签，同时将 component 组件放置在其中。
- void addTab(String title, Icon icon, Component component)：增加一个标题为 title、图标为 icon 的标签，同时将 component 组件放置在其中。
- void addTab(String title, Icon icon, Component component, String tip)：增加一个标题为 title 的标签，同时将 component 组件放置在其中，tip 是提示字符串。

【例 14.19】选项板使用示例

//----------文件名 demoJTabbedPane.java，程序编号 14.19----------

```
import javax.swing.*;
public class demoJTabbedPane {
  JTabbedPane myJtabpane;
  JLabel myJlbl1,myJlbl2;
  JFrame mainJFrame;
  public demoJTabbedPane() {
    mainJFrame=new JFrame("JTabbedPane 使用示例");
    myJlbl1=new JLabel("这是第一个选项卡");
    myJlbl2=new JLabel("这是第二个选项卡");
    //创建一个空白选项板
    myJtabpane=new JTabbedPane();
    //添加选项卡，同时指定标签的标题和放置的组件
    myJtabpane.addTab("标签一",myJlbl1);
    myJtabpane.addTab("标签二",myJlbl2);
    mainJFrame.add(myJtabpane);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJTabbedPane();
  }
}
```

程序运行截图如 14.40 所示：

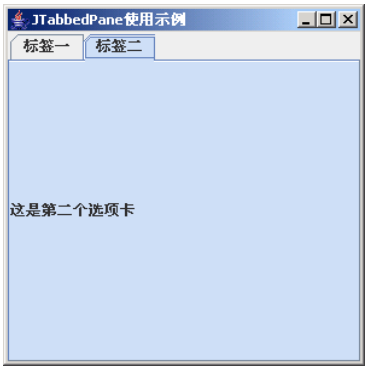

图 14.40　程序运行截图

当用户单击上面的标签时，将自动切换选项卡，并显示放置在其中的组件。但这个时候，程序也需要知道用户到底在哪一个选项卡中进行操作，这可以通过调用 getSelectedIndex()方法来获取。

注意第一个选项卡的索引号为 0。

14.7.5 工具栏（JToolBar）使用示例

大多数的 GUI 程序都会有一个工具栏，用于放置一些快捷按钮，JToolBar 就是 JDK 提供的工具栏，可以将 JButton 加在上面，形成一般的工具栏式样。

在缺省情况下，用户可以把工具栏拖动到容器各个边缘。因此，放置工具栏的容器应该使用 BorderLayout 布局，并且在其他边缘内不能存放其他的组件，而只能在容器的中心摆放一个组件。

JToolBar 提供的构造方法有：

- JToolBar()：创建一个空的工具栏，默认放置方向为水平方向。
- JToolBar(int orientation)：创建一个空的工具栏，放置方向由 orientation 指定。
- JToolBar(String name)：创建一个空的工具栏，name 是它的名字。
- JToolBar(String name, int orientation)：创建一个空的工具栏，name 指定名字，orientation 指定方向。

JToolBar 的使用非常简单，只是要注意往其中添加的按钮通常应该是带图标的。

【例 14.20】工具栏使用示例

//----------文件名 demoJToolBar.java，程序编号 14.20----------

```
import javax.swing.*;
import java.awt.*;
public class demoJToolBar {
  JFrame mainJFrame;
  JToolBar myJToolBar;
  JButton myJbtn1,myJbtn2;
  Container con;
  public demoJToolBar() {
    mainJFrame=new JFrame("JToolBar 使用示例");
    con=mainJFrame.getContentPane();
    myJToolBar=new JToolBar();
    myJbtn1=new JButton(new ImageIcon("OPEN.GIF"));     //为按钮装入图标
    myJbtn1.setToolTipText("打开文档");                  //设置提示信息
    myJbtn2=new JButton(new ImageIcon("CLOSE.GIF"));    //为按钮装入图标
    myJbtn2.setToolTipText("关闭文档");                  //设置提示信息
    myJToolBar.add(myJbtn1);
    myJToolBar.add(myJbtn2);
    con.add(myJToolBar,BorderLayout.NORTH);             //注意布局方式
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJToolBar();
  }
}
```

程序启动时运行截图如 14.41 所示：

图 14.41　程序启动时截图

用户还可以拖动工具栏到窗口的其他位置，比如左侧垂直放置。如图 14.42 所示：

图 14.42　拖动工具栏之后的截图

可以看到，要实现拖动功能，并不需要编一行程序代码，它的表现完全是一个标准的工具栏式样。

14.8　常用组件

本节介绍一些最常用的可视化组件，它们都是 Swing 中的轻量级组件。这些都是基本组件，不能单独存在，需要放在其他容器中。和中间容器不同，大多数的基本组件都要求程序员写事件响应代码，才能充分发挥功能。

Swing 中的组件都采用了一种著名的设计模式：模型-视图-控制器（model-view-controller，简称 MVC）模式。MVC 模式遵循一个基本原则：不要让一个对象承担过多的功能。即便是简单如一个按钮类，它也是由一个对象负责存储内容，一个对象负责组件的外观，另外一个对象负责处理用户的输入。扩展开来，所有的组件都按照下面的模式来设计：

- 模型（model）：存储内容
- 视图（view）：显示内容
- 控制器（controller）：处理用户输入

当然有时候模型中存储的内容并不一定都会显示出来，视图会根据需要选择其中的一部分显示。MVC 的一个优点是一个模型可以有多个视图，其中的每个视图都可以显示全部内容的不同部分。

关于 MVC 模式的优点，本书并不打算深入介绍，有兴趣的读者可以参阅《设计模式——可复用面向对象软件的基础》一书。不过，MVC 模式在某些情况下会让程序员不太习惯，特别是 Delphi 或 VB 程序员会觉得某些地方处理起来过于繁琐。但总体来说，MVC 是一种相当先进的设计模式，当程序员熟悉了这种设计模式之后，在操作一些复杂的组件时会觉得很方便。在本节后面的几个复杂的例子中，

读者将会体会到 MVC 模式的便利。

14.8.1 标签(JLabel)使用示例

标签是用来显示信息最常用的组件，前面的程序都使用过标签。大多数情况下，标签不需要响应事件，所以编程很简单。Swing 中的标签一个突出特点是可以放入图像，让程序外观更漂亮。

要在标签中添加图像，常用的方法是利用它的构造方法：JLabel(Icon image)，或者使用成员方法：setIcon(Icon icon)。这两个方法的参数都是一个加载了图片的 Icon 对象。

【例 14.21】图像标签使用示例

//----------文件名 demoJLabel_1.java，程序编号 14.21----------

```
import javax.swing.*;
import java.awt.*;
public class demoJLabel_1 {
  JLabel imgLabel;
  JFrame mainJFrame;
  Container con;
  public demoJLabel_1() {
    mainJFrame=new JFrame("图像标签使用示例");
    con=mainJFrame.getContentPane();
    //用下面的方法创建一个带图片的标签，图片可以是 GIF 或 JPG 格式
    imgLabel=new JLabel(new ImageIcon("test.jpg"));
    con.add(imgLabel,BorderLayout.CENTER);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJLabel_1();
  }
}
```

程序运行截图如 14.43 所示：

图 14.43　程序运行截图

当外部的窗口改变大小时，标签也会随之对图片进行裁剪。程序中生成了一个 ImageIcon 对象：

```
new ImageIcon("test.jpg")
```

其中的 test.jpg 是一副图片文件，它可以是 jpg 或 gif 格式，但不能是更为简单的 bmp 格式。这大概

与 bmp 是微软的专利格式有关。

如果需要改变标签上的文字或图像，setText()和 setIcon()方法。另外，JLabel 实际上是可以监听鼠标事件的，程序员可以将鼠标变成自己想要的形状。比如在下面的例子中，当鼠标移动到标签位置时，将变成手的形状，以做出链接的效果。

【例 14.22】改变标签上鼠标形状示例

//----------文件名 demoJLabel_2.java，程序编号 14.22----------

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

public class demoJLabel_2 {
  JLabel linkLabel;
  JFrame mainJFrame;
  Container con;
  ListenMouse listenmouse;
  //这个内部类用于监听鼠标事件
  public class ListenMouse extends MouseAdapter{
    //当鼠标移动到标签上时
    public void mouseEntered(MouseEvent e){
      //改变标签上鼠标的形状，变成手的形状
      linkLabel.setCursor(new Cursor(Cursor.HAND_CURSOR));
    }
  }
  public demoJLabel_2() {
    listenmouse=new ListenMouse();
    mainJFrame=new JFrame("图像标签使用示例");
    con=mainJFrame.getContentPane();
    linkLabel=new JLabel("把鼠标移到我这，将变成手的形状");
    //将标签放上方
    con.add(linkLabel,BorderLayout.NORTH);
    //添加鼠标监听器
    linkLabel.addMouseListener(listenmouse);
    mainJFrame.setSize(300,200);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJLabel_2();
  }
}
```

程序运行截图如 14.44 所示：

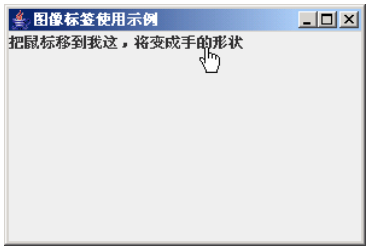

图 14.44　程序运行截图

14.8.2　按钮（JButton）使用示例

JButton 类是 AbstractButton 类的子类，它拥有与 AWT 中 Button 类相同的许多性质，比如设置背景色、设置激活状态、设置可用状态等。但与后者不同的是，它可以同时显示文字和图片，因而显得更美观。JButton 的构造方法如下：

- JButton()：创建一个按钮，没有文字和图标
- JButton(Action a)：创建一个按钮，属性来自于参数 a
- JButton(Icon icon)：创建一个按钮，显示 icon 中的图片
- JButton(String text)：创建一个按钮，显示 text 中的文字
- JButton(String text, Icon icon)：创建一个按钮，显示文字和图片

JButton 中的方法相当多，其中最常用的几个如下：

- void setIcon(Icon icon)：设置按钮上显示的图片
- Icon getIcon()：获取按钮上显示的图片
- void setText(String s)：设置按钮上显示的文字
- String getText()：获取按钮上显示的文字

一般情况下，按钮必须要响应 ActionListener 事件才有意义。在下面的例子中，按钮上的文字和图片会反复变化。

【例 14.23】按钮使用示例

//----------文件名 demoJButton.java，程序编号 14.23----------

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

public class demoJButton implements ActionListener{
  JButton shiftBtn;
  JFrame mainJFrame;
  Container con;
  ImageIcon closeIcon,openIcon;
  public demoJButton() {
    mainJFrame=new JFrame("图像按钮使用示例");
    closeIcon=new ImageIcon("close.gif");
    openIcon=new ImageIcon("open.gif");
    con=mainJFrame.getContentPane();
    //用下面的方法创建一个带图片的按钮，图片可以是 GIF 或 JPG 格式
    shiftBtn=new JButton("打开",openIcon);
    con.add(shiftBtn,BorderLayout.NORTH);
    //安装事件监听器
    shiftBtn.addActionListener(this);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  //监听用户单击按钮事件
  public void actionPerformed(ActionEvent e){
```

```
    if (e.getSource()==shiftBtn){
      //判断当前按钮是显示的文字，并根据文字做出相应的动作
      if(shiftBtn.getText().compareTo("打开")==0){
         shiftBtn.setText("关闭");        //设置文字
         shiftBtn.setIcon(closeIcon);     //设置图片
      }else{
         shiftBtn.setText("打开");
         shiftBtn.setIcon(openIcon);
      }
    }
  }
  public static void main(String[] args) {
     new demoJButton();
  }
}
```

程序启动时截图如 14.45 所示：

图 14.45　程序启动时截图

当单击按钮之后，截图如 14.46 所示：

图 14.46　单击按钮之后的截图

如果反复单击这个按钮，就会循环显示“打开”和“关闭”。这个程序虽然简单，但却演示了一个基本的技巧：如何让一个按钮实现不同的功能。这只需要用分支语句来根据按钮上显示的文字执行不同的代码就可以了。

14.8.3　文本框（JTextField）和密码框（JPasswordField）使用示例

文本框允许用户输入一行字符，也可以由程序添加字符到其中。JTextField 的常用方法如下：

- void setText(String s)：设置文本框中的文字
- String getText()：获取文本框中的文字

- int getColumns()：获取文本框中的可以输入的字符个数
- void setEditable(Boolean b)：设置文本框是否可编辑

JTextField 可以响应的方法比较多，常用的有下面几个：

- TextEvent 事件：当文本内容发生改变时引发
- ActionEvent 事件：当用户按下回车键后引发
- KeyEvent 事件：当用户在文本框中输入按键时引发

密码框是文本框的直接子类，除了父类方法，它新增加了两个方法：

- void setEchoChar(char ch)：设置密码框的显示字符，默认是“*”
- String getPassword()：获取密码框中的密码，该方法取代了父类的 getText()方法

下面的例子模仿了常见的登录界面，这里只使用默认的文本框和密码框，没有为文本框的 ActionEvent 写响应代码。

【例 14.24】文本框和密码框使用示例

//----------文件名 demoJText.java，程序编号 14.24----------

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

public class demoJText {
  JLabel JLabel1,JLabel2;
  JFrame mainJFrame;
  Container con;
  JButton loginBtn,cancelBtn;
  JTextField userText;
  JPasswordField passwordField;
  HandleAction handleAction;
  //按钮事件监听器
  public class HandleAction implements ActionListener{
    public void actionPerformed(ActionEvent e){
      String msg;
      //如果按下确定键，显示用户名和密码
      if (e.getSource()==loginBtn){
          //用 getText()获取文本框中的数据，用 getPassword()获取密码框中的数据
          msg="你的用户名是:"+userText.getText()+"\n 你的密码是："
                          +new String(passwordField.getPassword());
          JOptionPane.showMessageDialog(mainJFrame,msg);
      }
      //如果按下取消键，清空文本框和密码框
      else if(e.getSource()==cancelBtn){
          passwordField.setText("");
          userText.setText("");
      }
    }
  }

  public demoJText() {
    handleAction=new HandleAction();
    JLabel1=new JLabel("用户名");
```

```
    JLabel2=new JLabel("密 码");
    mainJFrame=new JFrame("文本框和密码框使用示例");
    con=mainJFrame.getContentPane();
    loginBtn=new JButton("登录");
    loginBtn.addActionListener(handleAction);
    cancelBtn=new JButton("取消");
    cancelBtn.addActionListener(handleAction);
    userText=new JTextField();
    //设置文本框的宽度，这个很重要
    userText.setColumns(20);
    passwordField=new JPasswordField();
     //设置密码框的宽度
    passwordField.setColumns(20);
    con.setLayout(new FlowLayout());
    con.add(JLabel1);
    con.add(userText);
    con.add(JLabel2);
    con.add(passwordField);
    con.add(loginBtn);
    con.add(cancelBtn);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

   public static void main(String[] args) {
     new demoJText();
    }
}
```

当程序启动后，在文本框和密码框中输入数据时，截图如 14.47 所示：

图 14.47　输入数据时截图

当单击“登录”按钮后截图如 14.48 所示：

图 14.48　按下“登录”按钮后的截图

可以清晰的看到，密码已经被程序读取出来，所以在密码框中输入的数据尽管显示为“*”，但对于程序而言，是完全可以知道的。另外如果需要在输入密码时显示其他诸如“●”之类的符号，可以使

用 setEchoChar('●')来实现。

在程序中有一个地方需要注意：

```
userText.setColumns(20);
```

表示设置文本框的宽度为 20 个字符，如果没有这个设置，文本框的初始宽度会被设置为它拥有的字符数。但本程序中文本框中初始没有字符，所以它的宽度为 0，根本就无法输入数据。

在这个程序中，如果用户在文本框或密码框中按下回车键，程序默认是没有反应的。但在实际使用中，用户常常会有这样的要求：当用户名输入完成后，敲回车键光标进入密码框中；密码输入完成后，敲回车键，就对用户名和密码进行检验（相当于单击了“登录”键）。为了实现这一目标，文本框和密码框都需要响应回车键，也就是 ActionListener 事件，使得程序更人性化。

下面是在程序 14.24 基础上修改后的代码，请特别注意增加的部分：

```
//----------文件名 demoJText.java，程序编号 14.25----------
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class demoJText {
  JLabel JLabel1,JLabel2;
  JFrame mainJFrame;
  Container con;
  JButton loginBtn,cancelBtn;
  JTextField userText;
  JPasswordField passwordField;
  HandleAction handleAction;
  //按钮事件监听器和文本框回车事件监听器
  public class HandleAction implements ActionListener{
    public void actionPerformed(ActionEvent e){
      String msg;
      if (e.getSource()==loginBtn){
          //用 getText()获取文本框中的数据，用 getPassword()获取密码框中的数据
          msg="你的用户名是:"+userText.getText()+"\n 你的密码是："
                          +new String(passwordField.getPassword());
          JOptionPane.showMessageDialog(mainJFrame,msg);
      }
      else if(e.getSource()==cancelBtn){
          passwordField.setText("");
          userText.setText("");
      }
      //这里是新增的事件响应代码
      //如果事件源是文本框
      else if (e.getSource()==userText){
          //指定获取输入焦点的组件为 passwordField
          passwordField.requestFocus();
      }
      //如果事件源是密码框
      else if(e.getSource()==passwordField){
         //注意这个方法，直接调用按钮事件的响应代码
         loginBtn.doClick();
      }
```

```
    }
  }

  public demoJText() {
    handleAction=new HandleAction();
    JLabel1=new JLabel("用户名");
    JLabel2=new JLabel("密 码");
    mainJFrame=new JFrame("文本框和密码框使用示例");
    con=mainJFrame.getContentPane();
    loginBtn=new JButton("登录");
    loginBtn.addActionListener(handleAction);
    cancelBtn=new JButton("取消");
    cancelBtn.addActionListener(handleAction);
    userText=new JTextField();
    userText.setColumns(20);
    //为文本框增加监听器
    userText.addActionListener(handleAction);
    passwordField=new JPasswordField();
    passwordField.setColumns(20);
    //为密码框增加监听器
    passwordField.addActionListener(handleAction);
    con.setLayout(new FlowLayout());
    con.add(JLabel1);
    con.add(userText);
    con.add(JLabel2);
    con.add(passwordField);
    con.add(loginBtn);
    con.add(cancelBtn);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
   mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

   public static void main(String[] args) {
     new demoJText();
    }
}
```

这个程序的运行截图和程序 12.24 是完全一样的，不过操作更方便一些。

注意：在文本框中按下回车键和单击按钮所引发的事件是一样的，都是 ActionListener，所以需要写代码判断事件源。

14.8.4 文本区（JTextArea）使用示例

文本框一次只允许用户输入一行信息，而文本区（JTextArea）则允许用户输入多行信息，可以作为简单的全屏编辑器来使用。

注意：文本区本身是没有滚动条的，所以应该将它放入到一个滚动面板（JScrollPane）中。

文本框属于比较复杂的组件，提供了相当多的方法，表 14.6 列出了文本区常用的构造方法和普通成员方法：

表 14.6　文本区的常用方法

方法	说明
JTextArea()	按默认值创建一个空的文本区
JTextArea(int rows, int columns)	按指定的行和列创建一个文本区
JTextArea(String text)	创建一个文本区，拥有指定的字符串
void append(String str)	在文本区的末尾追加指定字符串
void insert(String str, int pos)	在指定位置插入字符串
String getText()	获取文本区中的文本内容
void setFont(Font f)	指定显示时的字体
void setLineWrap(boolean wrap)	设定是否自动折行
void setText(String t)	设置文本区中的内容为字符串t
void copy()	将选定内容复制到剪贴板
void paste()	将剪贴板的内容粘贴到光标位置
void cut()	将选定内容剪切到剪贴板

文本框可以监听的事件文本区都可以监听，另外，程序员还可以通过添加 UndoableEditListener 监听器，访问 UndoableEdit 记录来实现 redo/undo 功能。

【例 14.25】文本区使用示例

//----------文件名 demoJTextArea.java，程序编号 14.26----------

```
import javax.swing.*;
public class demoJTextArea {
  JFrame mainJFrame;
  JScrollPane JSpane;
  JTextArea JEdit;
  public demoJTextArea() {
    mainJFrame=new JFrame("JTextArea 使用示例");
    //创建一个空白文本区
    JEdit=new JTextArea();
    //将它添加到滚动面板中
    JSpane=new JScrollPane(JEdit);
    mainJFrame.add(JSpane);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    new demoJTextArea();
  }
}
```

程序运行截图如 14.49 所示：

图 14.49　程序运行截图

虽然程序中没有为 JTextArea 编写一行代码，但它已经具备了一个全屏编辑器的基本功能。后面的 14.13 节将利用它来实现一个类似于记事本的程序。关于文本区更详细的介绍也在该程序中。

14.8.5　复选框(JCheckBox)使用示例

程序经常需要用户从多个项目中选出多个条件，JCheckBox 类就可以完成此功能。JCheckBox 继承自 AbstractButton，所以也可以看成是一种特殊的按钮。复选框的状态只有两个：选中和未选中。这只需要调用 isSelected()，其返回值表示当前按钮框是否被选中。复选框拥有的方法很多，但常用的并不多，简要介绍如下：

- JCheckbox()：创建一个空的复选框，标题为空。
- JCheckbox(String s)：创建一个复选框，标题为 s，标题在复选框的右边。
- JCheckbox(String s, boolean b)：创建一个复选框，标题为 s，b 设置初始状态，true 表示选中，默认值为 false。
- boolean isSelected()：如果本按钮被选中，返回 true。
- void setSelected(boolean b)：设置本按钮被选中的状态，如果 b 为 true，就表示选中。该方法不会触发 actionEvent 事件。
- void doClick()：虚拟一个单击按钮事件，它的效果和用户单击该按钮完全相同。

要想知道用户选择了哪些复选框，有两种方法：一是依次调用每个复选框的 isSelected()方法做判断；二是可以监听 ItemEvent 事件，即注册事件接口 ItemListener，可以实时获得用户的单击动作，结合复选框的初始状态，就可以知道复选框目前的状态了。

下面的例子演示了如何获取复选框的状态。由于这里的复选框比较多，所以采用了复选框数组，这样编程比较简洁。

【例 14.26】复选框使用示例

//----------文件名 demoJCheckbox.java，程序编号 14.27----------

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class demoJCheckbox implements ActionListener{
  JFrame mainJFrame;
  Container con;
  JButton OKBtn;
  JCheckBox box[];
```

```
 JLabel  msgJlabel;
 static final String ProvinceName[]={"北京","上海","天津","辽宁",
                                "吉林","四川","湖南","湖北","广东"};
 public demoJCheckbox() {
   mainJFrame=new JFrame("JCheckBox 使用示例");
   con=mainJFrame.getContentPane();
   con.setLayout(new FlowLayout());
   msgJlabel=new JLabel("请至少选择一个你去过的省份");
   con.add(msgJlabel);
    //下面开始创建组件数组
   box=new JCheckBox[ProvinceName.length];
   for(int i=0;i<box.length;i++){
      //依次为每个元素创建复选框对象，初始状态为不选择
      box[i]=new JCheckBox(ProvinceName[i],false);
      con.add(box[i]);
   }
   OKBtn=new JButton("确定");
   OKBtn.addActionListener(this);
   con.add(OKBtn);
   mainJFrame.setSize(200,300);
   mainJFrame.setVisible(true);
   mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }

 public void actionPerformed(ActionEvent e){
   String tmpmsg="";
   int count=0;
   //遍历整个数组，访问每一个复选框组件
   for(int i=0;i<box.length;i++){
      //判断这个复选框是否被选中
      if (box[i].isSelected()){
         count++;
         tmpmsg=tmpmsg+box[i].getText()+" ";
      }
   }
   JOptionPane.showMessageDialog(mainJFrame,"你选择了" +
                                  count+"个省份,它们是:\n"+tmpmsg);
 }

 public static void main(String[] args) {
    new demoJCheckbox();
 }
}
```

程序启动时截图如 14.50 所示：

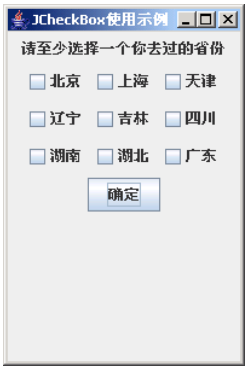

图 14.50　程序启动时截图

选择某些复选框之后，再按“确定”键之后的截图如 14.51 所示：

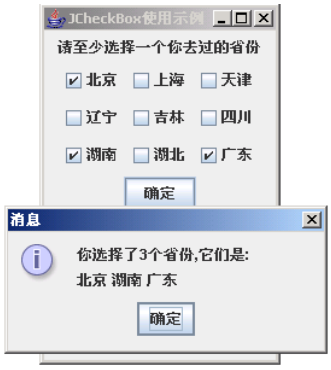

图 14.51　按下确定按钮之后的截图

14.8.6　单选按钮(JRadioButton)使用示例

在多个选项中，如果只允许用户选择一个选项，就需要使用单选按钮，这就好比用收音机收听电台，一次只能听一个台，这也正是 JRadioButton 名字的来源。单选按钮与复选框的本质区别在于：各个复选框是相互独立的，任何一个复选框的选择情况与其他复选框无关。但单选按钮则恰好相反，在一组单选按钮中的各个按钮相互之间的选择关系是互斥的。不过问题是，这些单选按钮本身无法知道自己和其他按钮的关系。所以需要把它们放在一个名为 ButtonGroup 的组件中统一进行管理。该组件本身不可见，但对于 JRadioButton 却是必不可少的。

JRadioButton 也是从 AbstractButton 继承出来的，所以它的大多数方法的使用和 JCheckBox 相同，这里不再重复。使用 JRadioButton 也只需要知道用户到底选择了哪一个单选按钮。这也有三种方法：一是一次调用每一个按钮的 isSelected()方法，判断其是否被选择；二是调用 ButtonGroup 的 getSelection()方法获取被选中的按钮，这样的效率更高一些；三是监听 ActionEvent 事件或是 ItemEvent 事件来实时获取被选中的按钮。

下面的例子中，就采用了第三种方法来获取用户的选择。

【例 14.27】单选按钮使用示例

//----------文件名 demoJRadioButton.java，程序编号 14.28----------

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

```
//本类实现 ActionListener 接口，以便监听按钮事件
public class demoJRadioButton implements ActionListener{
  JFrame mainJFrame;
  Container con;
  JButton OKBtn;
  JRadioButton mRadio,fRadio;
  ButtonGroup sexBtnGroup;
  JLabel msgJlabel;
  String msg;  //用这个类的成员记录用户的选择
  public demoJRadioButton() {
    mainJFrame=new JFrame("JRadioButton 使用示例");
    con=mainJFrame.getContentPane();
    con.setLayout(new FlowLayout());
    msgJlabel=new JLabel("请选择性别");
    con.add(msgJlabel);
    //创建单选按钮
    mRadio=new JRadioButton("男",true);
    //监听 ActionEvent 事件，它和普通按钮共用同一个监听器
    mRadio.addActionListener(this);
    fRadio=new JRadioButton("女",false);
    fRadio.addActionListener(this);
    //创建按钮组
    sexBtnGroup=new ButtonGroup();
    //将单选按钮添加到按钮组中
    sexBtnGroup.add(mRadio);
    sexBtnGroup.add(fRadio);
    con.add(mRadio);
    con.add(fRadio);
    OKBtn=new JButton("确定");
    //普通的下压式按钮也用本监听器
    OKBtn.addActionListener(this);
    con.add(OKBtn);
    mainJFrame.setSize(200,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent e){
    Object obj = e.getSource();
    //注意这个方法，判断是由哪种按钮产生的本事件
    if( obj instanceof JRadioButton)
        msg=e.getActionCommand();  //如果是单选按钮，记录该按钮的文本信息
    else
       JOptionPane.showMessageDialog(mainJFrame,"你选择了"+msg);
  }
  public static void main(String[] args) {
     new demoJRadioButton();
  }
}
```

程序启动时的运行截图如 14.52 所示：

图 14.52　程序启动时截图

当用户按下“确定”按钮之后的截图如 14.53 所示：

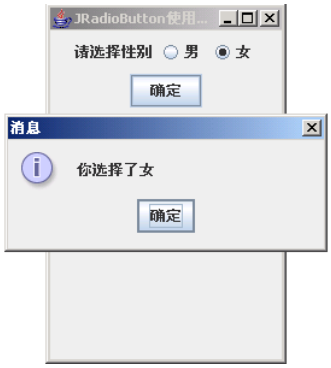

图 14.53　按下确定按钮之后的截图

这个程序能够正常运行。不过有个地方还需要仔细体会一下：在响应按钮单击事件的方法 actionPerformed()中，用到了运行时类型识别，这是因为有两类按钮都用了同一个监听器。当然，可以用多路分支语句分别判断每一个事件源，但采用类型识别的模式来写，程序更加简洁。

14.8.7　列表框（JList）使用示例

列表框的外观类似于 JTextArea，但它不允许用户直接进行编辑，只能以行为单位进行选择。用户可以在列出的可选项目中选择一个或多个条目。列表框本身是没有滚动条的，如果所有条目超过列表框的可视范围，需要将列表框放到 JScrollPane 中。如果程序员有需要，还可以监听 ListSelectionEvent 事件或是 ItemEvent 事件。

JList 的方法非常多，表 14.7 列出一些常用的方法：

表 14.7　JList中的常用方法

方法	说明
JList()	创建一个空的列表，使用默认数目的可见行
JList(ListModel dataModel)	创建一个列表，其中的条目由模型dataModel决定
JList(Object [] listdata)	创建一个列表，其中的条目由数组listdata决定
JList(Vector<?> listData)	创建一个列表，其中的条目由向量listdata决定
void clearSelection()	清除用户所有的选择，没有条目被选中
ListModel getModel()	获得与列表相关联的模型
int getSelectedIndex()	获取第一被选择的条目的索引，-1表示没有条目被选中

int[] getSelectedIndices()	获取所有被选择的条目的索引，按增序存放在数组中
Object getSelectedValue()	获取第一个被选择的条目值，null表示没有条目被选中
Object[] getSelectedValues()	获取所有被选择的条目值
void setListData(Object[] listData)	用listDate构造一个新的模型，然后调用setModel()方法为列表重置所有条目
void setListData(Vector<?> listData)	用listDate构造一个新的模型，然后调用setModel()方法为列表重置所有条目
void setModel(ListModel model)	重置与列表相关联的模型，列表中原先的内容全部被清空
void setSelectedIndex(int index)	将index所指定的条目置为被选择
boolean isSelectedIndex(int index)	判断index所指定的条目是否被选择

从表中可以看到，JList 居然没有提供删除某一个指定条目或者增加一个条目的功能。如果要动态修改 JList 中的条目，需要将其中的所有内容重置，这似乎不太合情理——因为这么做效率显然太低了。答案在于 JList 是一个典型的 MVC 模型组件，JList 只是一个视图，它的数据全部存储在与之相关连的 ListModel 中，而该模型提供了部分修改数据的功能，所以只要获得该模型的引用，然后通过引用对模型中的数据进行修改，就会自动反映到 JList 中。

在下面的例子中，用户可以在左边列表框中选择所需项目，并通过按钮调整到右边列表框，反之也可以。这需要动态的在 JList 中添加和删除项目。正如前所述，JList 本身并不提供这种动态功能，这些功能需要通过一个叫做 DefaultListModel 的组件来完成。在创建 JList 时，先建立二者的联系，以后对 DefaultListModel 的操作就会自动反映到 JList 上来。本程序中使用了比较多的编程技巧，希望读者好好体会。

【例 14.28】列表框使用示例

//----------文件名 demoJList.java，程序编号 14.29----------

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

public class demoJList implements ActionListener{
  JFrame mainJFrame;
  Container con;
  JButton addBtn,delBtn;
  JList orignList,destList;              //这是两个用于显示的列表
  DefaultListModel orignModel,destModel; //这两个不可视的模型包含了 JList 中的条目
  JScrollPane leftJSPane,rightJSPane;    //滚动面板用于存放 JList
  JSplitPane baseSplitPane;              //分割面板存放上面的滚动面板
  JPanel pane;                          //普通面板用来放按钮
  static final String msg[]={"北京","上海","天津","辽宁","吉林",
                         "四川","湖南","湖北","广东"};
   public demoJList() {
     mainJFrame=new JFrame("JList 使用示例");
     con=mainJFrame.getContentPane();
     orignModel=new DefaultListModel();
     //将需要显示的条目逐条加入到 DefaulListModel 中
     for(int i=0;i<msg.length;i++)
         orignModel.addElement(msg[i]);
     //用 DefaulListModel 来创建 JList 对象，建立二者之间的联系
     orignList=new JList(orignModel);
     destModel=new DefaultListModel();
     destList=new JList(destModel);
     //将列表添加到滚动面板中
     leftJSPane=new JScrollPane(orignList);
```

```
    rightJSPane=new JScrollPane(destList);
    //将滚动面板添加到分割面板中
    baseSplitPane=new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                 true,leftJSPane,rightJSPane);
    //将滚动面板添加到窗口的中间
    con.add(baseSplitPane,BorderLayout.CENTER);
    //下面开始创建按钮，并设置各种属性和监听器
    addBtn=new JButton("选中>>");
    delBtn=new JButton("撤消<<");
    pane=new JPanel();
    pane.add(addBtn);
    pane.add(delBtn);
    addBtn.addActionListener(this);
    delBtn.addActionListener(this);
    //将放置了按钮的面板放在窗口的下方
    con.add(pane,BorderLayout.SOUTH);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    //默认情况下，分隔面板左边窗口所占位置太少，用这个方法来调整
    baseSplitPane.setDividerLocation(0.5);
  }
  //响应用户单击按钮事件
  public void actionPerformed(ActionEvent e){
    int i;
    if (e.getSource()==addBtn){
      //将左边选择的条目加入到右边的 JList 中
      for(i=0;i<orignModel.getSize();i++)
        if(orignList.isSelectedIndex(i)) //本条目是否被选择
          destModel.addElement(orignModel.elementAt(i));
      //将左边被选中的条目删除，注意 i 是递减的，可以提高速度
      for(i--;i>=0;i--)
        if(orignList.isSelectedIndex(i))
          orignModel.removeElementAt(i);
    }
    //这个过程和上面的恰好相反，从右边往左边加
    else{
       for(i=0;i<destModel.getSize();i++)
         if(destList.isSelectedIndex(i))
            orignModel.addElement(destModel.elementAt(i));
       for(i--;i>=0;i--)
         if(destList.isSelectedIndex(i))
            destModel.removeElementAt(i);
    }
  }
  public static void main(String[] args) {
      new demoJList();
  }
}
```

程序启动时运行截图如 14.54 所示：

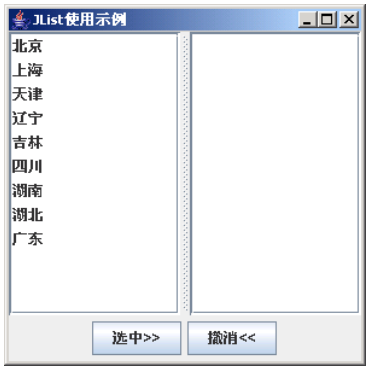

图 14.54　程序启动时截图

当用户选择了某些条目之后，再单击“选中”按钮之后截图如 14.55 所示：

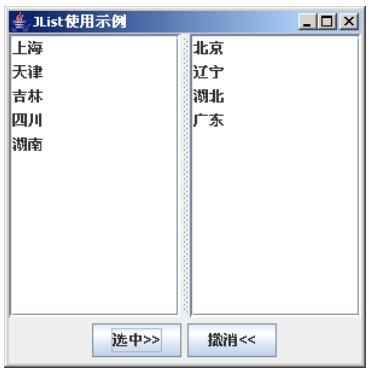

图 14.55　选中了某些条目之后

当用户撤销了某些条目之后如图 14.56 所示：

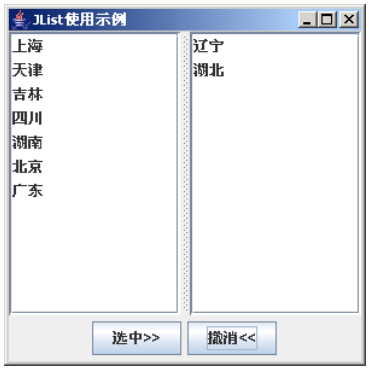

图 14.56　撤销了某些条目的选择之后

这个程序比较长，读者应该把注意力集中在下面几点：

- 在创建列表时，使用了：orignList=new JList(orignModel);这样的语句，这样，orignModel 就是与列表 orignList 相关联的模型。
- 在删除条目时，是对模型进行的操作：orignModel.removeElementAt(i);，这时与之相关联的列表中的内容就被更新。
- 增加条目也是对模型进行操作：destModel.addElement(orignModel.elementAt(i));
- 另外一点，就是在删除条目时，是从后面往前面删除，这比从前面往后面删的速度要稍微快一

点。具体原因读者可以思考线性表的删除操作是如何实现的。

某些书籍上在实现同样功能的时候，没有对模型进行操作，而直接使用了 setListData()方法，这样编程要简单一些。不过当要删除多个条目时，要反复调用这个方法，效率要低得多。

当然，如果是在大量条目中删除大量条目（比如在 10000 条中删除 1000 条），那么上述逐条删除的方法仍然比较低效。更好的方法不是删除，而是逐条复制不要删除的数据，然后一次性全部重置 JList 中所有条目。这已经涉及到算法的优化了，在此不展开论述。

本程序是使用 MVC 模式的第一个示例，后面还会有其他程序来演示如何使用 MVC 模式。

14.8.8 组合框（JComboBox）使用示例

组合框是由一个下拉列表框和一个文本框组合而成，又称为下拉列表组合框。它既允许用户通过下拉列表来选择项目，也允许用户在文本框中输入自己所需要的项目（这个功能可由程序员来控制）。但无论采用哪种方法，用户都只能有一个选择。

JComboBox 的常用方法如表 14.8 所示。

表 14.8 JComboBox常用方法

方法	说明
JComboBox()	创建一个空的组合框，下拉列表中的条目为空
JComboBox(Object [] items)	创建一个组合框，下拉列表中的条目由iteems确定
void addItem(Object anObject)	向列表中添加一个条目anObject
Object getItemAt(int index)	返回列表中index所指定的条目，索引值从0开始
int getItemCount()	返回列表中条目的数目
ComboBoxModel getModel()	获得与组合框相关联的模型
int getSelectedIndex()	在文本框可编辑的情况下，返回列表中与给定条目相匹配的第一条项目的索引值；否则就是用户选中的那个条目的索引。
Object getSelectedItem()	返回被选中的条目
void removeItem(Object anObject)	删除指定条目
void removeItemAt(int anIndex)	删除索引值为anIndex的条目，但该方法仅仅在组合框用mutable data model创建时才可用
void setEditable(boolean aFlag)	设置文本框的可编辑性
void setEnabled(boolean b)	设置组合框是否可用

尽管 JComboBox 也是基于 MVC 模式编写的组件，但它还是可以用 addItem 和 removeItem 直接添加和删除一些条目。程序员也可以通过 getSelectedItem 或 getSelectedIndex 来获取用户最终的选择。如果不允许用户在文本框中输入，可以使用 setEditable(false)方法来限制。程序还可以监听 ItemEvent 事件来实时监控选择条目的变化。

需要注意的是：每当用户改变一次选择，ItemEvent 事件将会发生两次，需要调用 getStateChange()方法来确定被选条目当前的状况，其中一个条目被选中，伴随着另外一个条目被撤销选中。

下面是一个使用组合框的简单例子，用户可以单击按钮来切换组合框中文本框的可编辑性。程序同时还监视用户选择的条目，将它实时显示在标签上。

【例 14.29】组合框使用示例

//----------文件名 demoJComboBox.java，程序编号 14.30----------

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

public class demoJComboBox implements ItemListener,ActionListener{
```

```
JLabel JLabel1,JLabel2;
JFrame mainJFrame;
Container con;
JButton shiftBtn;
JComboBox Jcombobox1;
static final String msg[]={"北京","上海","天津","辽宁",
                      "吉林","四川","湖南","湖北","广东"};
public demoJComboBox() {
  JLabel1=new JLabel("您的选择是: ");
  JLabel2=new JLabel("          ");
  mainJFrame=new JFrame("JComboBox 使用示例");
  con=mainJFrame.getContentPane();
  shiftBtn=new JButton("不可编辑");
  shiftBtn.addActionListener(this);
  //创建一个组合框，列表框中的内容由 msg 指定
  Jcombobox1=new JComboBox(msg);
  //设置文本框为可编辑
  Jcombobox1.setEditable(true);
  //添加监听器
  Jcombobox1.addItemListener(this);
  con.setLayout(new FlowLayout());
  con.add(JLabel1);
  con.add(JLabel2);
  con.add(Jcombobox1);
  con.add(shiftBtn);
  mainJFrame.setSize(300,300);
  mainJFrame.setVisible(true);
  mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void actionPerformed(ActionEvent e){
  if (e.getSource()==shiftBtn){
     if(shiftBtn.getText().compareTo("不可编辑")==0){
        shiftBtn.setText("可以编辑");
        Jcombobox1.setEditable(false);
     }
     else{
        shiftBtn.setText("不可编辑");
        Jcombobox1.setEditable(true);
     }
  }
}
//监听用户选择或输入条目事件
 public void itemStateChanged(ItemEvent e){
    //需要判断条目是被选中还是撤销选中
    if (e.getStateChange() ==ItemEvent.SELECTED)
       //在标签上显示用户选择或输入的条目
      JLabel2.setText((String)Jcombobox1.getSelectedItem());
 }
 public static void main(String[] args) {
```

```
        new demoJComboBox();
    }
}
```

图 14.57-14.59 是程序运行的截图：

图 14.57　选择了一个条目

图 14.58　用户输入了一个条目

图 14.59　将组合框设置为不可编辑

14.8.9　表格(JTable)使用示例

二维表格在程序中也是很常用的组件，Swing 中提供了 JTable 组件来实现一个规则的二位表格。它的方法非常多，它也是基于 MVC 模式的组件，使用上也比较麻烦。这里不打算一一介绍它所有的方法，而是通过几个例子来演示最常用的一些方法。

一般情况下，可以通过 JTable(int row, int col)来指定表格拥有的行和列。如果需要动态的调整列数，可以直接使用它的方法 addColumn()和 removeColumn()。如果需要调整行数，则象 JList 一样，需要调用和它的联系的 DefaultTableModel 的 addRow()和 removeRow()方法。还有，JTable 本身不提供滚动条，需要把它放在 JScrollPane 中才能滚动。

一般情况下，JTable 是显示一些信息给用户看，然后由用户进行修改。JTabel 构造方法如下：

- JTable()：建立一个新的 JTables，并使用系统默认的 Model。
- JTable(int numRows,int numColumns)：建立一个具有 numRows 行，numColumns 列的空表格，使用的是 DefaultTableModel。
- JTable(Object[][] rowData,Object[][] columnNames)：建立一个显示二维数组数据的表格，且可以显示列的名称。

- JTable(TableModel dm)：建立一个 JTable，有默认的字段模式以及选择模式，并设置数据模式。
- JTable(TableModel dm,TableColumnModel cm)：建立一个 JTable,设置数据模式与字段模式，并有默认的选择模式。
- JTable(TableModel dm,TableColumnModel cm,ListSelectionModel sm)：建立一个 JTable，设置数据模式、字段模式与选择模式。
- JTable(Vector rowData,Vector columnNames)：建立一个以 Vector 为输入来源的数据表格，可显示行的名称。

在下面的例子中，简单的显示了一个有 10 行 3 列的表格给用户编辑。

【例 14.30】表格使用示例 1

//----------文件名 demoJTable_1.java，程序编号 14.31----------

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

public class demoJTable_1 {
  JFrame mainJFrame;
  Container con;
  JScrollPane JSPane;
  JTable DataTable;
  public demoJTable_1() {
     mainJFrame=new JFrame("JLable 使用示例");
     con=mainJFrame.getContentPane();
     //创建一个有 10 行 3 列的空表格
     DataTable=new JTable(10,3);
     //把它放在滚动面板中
     JSPane = new JScrollPane(DataTable);
     con.add(JSPane);
     mainJFrame.setSize(300,300);
     mainJFrame.setVisible(true);
     mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public static void main(String[] args) {
      new demoJTable_1();
   }
}
```

程序运行截图如 14.60 所示：

图 14.60　程序运行截图

在下面的例子中，会显示一些信息给用户看，并且表头也可以指定，这要用到另外的构造方法。

【例 14.31】表格使用示例 2

//----------文件名 demoJTable_2.java，程序编号 14.32----------

```
import javax.swing.*;
import java.awt.*;
import java.util.*;
public class demoJTable_2{
   JFrame mainJFrame;
   JScrollPane JSPane;
   JTable DataTable;
   public demoJTable_2(){
     mainJFrame = new JFrame();
     Object[][] playerInfo={
        {"小王",new Integer(66),new Integer(72),new Integer(98),new Boolean(false)},
        {"小张",new Integer(82),new Integer(69),new Integer(78),new Boolean(true)},
     };
     String[] Names={"姓名","语文","数学","总分","及格"};
     //创建带内容和表头信息的表格
     DataTable=new JTable(playerInfo,Names);
     JSPane=new JScrollPane(DataTable);
     mainJFrame.add(JSPane);
     mainJFrame.setTitle("JTable 使用示例");
     mainJFrame.setSize(300,200);
     mainJFrame.setVisible(true);
     mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public static void main(String[] args) {
      new demoJTable_2();
   }
}
```

程序运行截图如 14.61 所示：

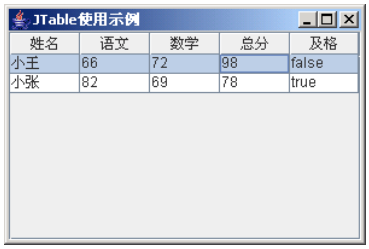

图 14.61　程序运行截图

图中可以看出，表格的头部和内容都是在创建表格时赋予它的。

注意：这里必须要使用滚动面板来显示表格，如果直接将表格加入到 JFrame 中，表头将无法正常显示。

上面的两个例子中，表格每一列的宽度都是相等的，当然用户可以通过拖拽来改变某列的宽度；但如果程序员想用程序设置列宽的值，可以利用 TableColumn 类所提供的 setPreferredWidth()方法来设置，并可利用 JTable 类所提供的 setAutoResizeMode()方法来设置调整某个列宽时其他列宽的变化情况，看下面这个例子：

【例 14.32】表格使用示例 3

//----------文件名 demoJTable_3.java，程序编号 14.33----------

```
import javax.swing.*;
import javax.swing.table.*;  //TableColumn 在这个包中
import java.awt.*;
import java.util.*;
public class demoJTable_3{
   JFrame mainJFrame;
   Container con;
   JScrollPane JSPane;
   JTable DataTable;
   public demoJTable_3(){
     mainJFrame = new JFrame();
     Object[][] playerInfo={
                            {"小王",new Integer(66),new Integer(72),new Integer(98),
                               new Boolean(false),new Boolean(false)},
                             {"小张",new Integer(82),new Integer(69),new Integer(78),
                               new Boolean(true),new Boolean(false)},
     };
     String[] Names={"姓名","语文","数学","总分","及格","作弊"};
     //创建带内容和表头信息的表格
     DataTable=new JTable(playerInfo,Names);
     //利用 JTable 中的 getColumnModel()方法取得 TableColumnModel 对象;
     //再利用 TableColumnModel 接口所定义的 getColumn()方法取得 TableColumn 对象的引用,
     //利用此对象的 setPreferredWidth()方法就可以控制字段的宽度
     for (int i=0;i<6;i++){
        TableColumn  column=DataTable.getColumnModel().getColumn(i);
        if ((i%2)==0)
           column.setPreferredWidth(150);
        else
           column.setPreferredWidth(50);
     }
     JSPane=new JScrollPane(DataTable);
     mainJFrame.add(JSPane);
     mainJFrame.setTitle("JTable 使用示例");
     mainJFrame.setSize(400,200);
     mainJFrame.setVisible(true);
     mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
   public static void main(String[] args) {
      new demoJTable_3();
   }
}
```

程序运行截图如 14.62 所示：

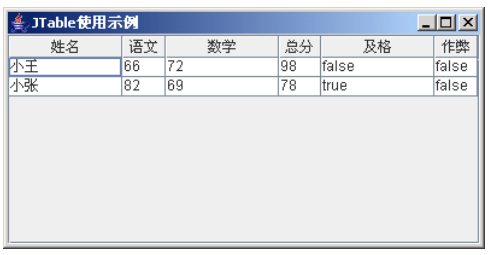

图 14.62　程序运行截图

可以看到各个列的宽度按照设计者的意图进行了调整。如果觉得这样一列一列来调整宽度比较麻烦，还可以使用预定义好的 5 个常数：

- AUTO_RESIZE_SUBSEQUENT_COLUMENS：当调整某一列宽时，此字段之后的所有字段列宽都会跟着一起变动。此为系统默认值。
- AUTO_RESIZE_ALL_COLUMNS：当调整某一列宽时，此表格上所有字段的列宽都会跟着一起变动。
- AUTO_RESIZE_OFFL：当调整某一列宽时，此表格上所有字段列宽都不会跟着改变。
- AUTO_RESIZE_NEXT_COLUMN：当调整某一列宽时，此字段的下一个字段的列宽会跟着改变，其余均不会变。
- AUTO_RESIZE_LAST_COLUMN：当调整某一列宽时，最后一个字段的列宽会跟着改变，其余均不会改变。

前面的三个例子都只是显示数据给用户看，如果需要获取用户修改后的数据，程序只需利用 getColumnCount()和 getRowCount()方法获得 JTable 的列数和行数，然后遍历整个 JTable，利用 getValueAt()方法获取单元格里的内容，就可以了。如果要修改某个单元格里的值，也可以利用 setValueAt()来完成。下面的例子就演示了如何获取以及设置表格中的数据，它会将用户在一个表格中输入的数据做矩阵转置，然后复制到另外一个表格中去。

【例 14.33】表格使用示例 4

//----------文件名 demoJTable_4.java，程序编号 14.34----------

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

public class demoJTable_4 implements ActionListener{
  JFrame mainJFrame;
  Container con;
  JScrollPane JSPane1,JSPane2;
  Panel  panel;
  JTable sourceTable,destTable;
  JButton  copyBtn;
  int rowCnt, colCnt;
  public demoJTable_4() {
     mainJFrame=new JFrame("JLable 使用示例");
     con=mainJFrame.getContentPane();
     //创建一个有 2 行 3 列的空表格,它给用户输入数据
     sourceTable=new JTable(2,3);
     //把它放在滚动面板中
     JSPane1 = new JScrollPane(sourceTable);
     //另外创建一个和上面这个表格行列恰好转置的表格
     rowCnt = sourceTable.getRowCount();    //获取行数
     colCnt = sourceTable.getColumnCount(); //获取列数
     //这个目的表格的行列恰好要换过来
     destTable = new JTable(colCnt,rowCnt);
     JSPane2 = new JScrollPane(destTable);
     //创建按钮，并添加监听器
     copyBtn = new JButton("复制数据");
```

```
    copyBtn.addActionListener(this);
    panel = new Panel();
    panel.add(copyBtn);
    //设置网格布局
    con.setLayout(new GridLayout(3,1));
    con.add(JSPane1);
    con.add(panel);
    con.add(JSPane2);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  //响应按钮单击事件
  public void actionPerformed(ActionEvent e){
    if (e.getSource()==copyBtn){
      int i,j;
      Object tobj;
      //遍历表格，将每个单元中的数据依次赋值到另外一个表格中
      for(i=0;i<rowCnt;++i)
        for(j=0;j<colCnt;++j){
          tobj = sourceTable.getValueAt(i,j);
          destTable.setValueAt(tobj,j,i);
        }
    }
  }
  public static void main(String[] args) {
    new demoJTable_4();
  }
}
```

程序启动时，截图如 14.63 所示：

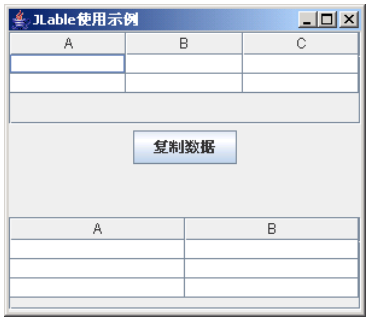

图 14.63　程序启动时截图

当用户输入数据后，并单击“复制数据”按钮之后截图如 14.64 所示：

图 14.64　复制数据之后截图

前面的例子仅仅只是简单地使用 JTable 提供好的功能。但在某些情况下，比如表格中每个单元中的数据类型将会被视为同一种。在前面的例子中，数据类型皆被显示为 String 的类型，因此，原来的数据类型声明为 Boolean 的数据会以 String 的形式出现而不是以 Check Box 的形式出现。

除此之外，如果所要显示的数据是不固定的，或是随情况而变。例如同样是一份成绩单，老师与学生所看到的表格应该不一样，显示的外观或操作模式也许也不相同。为了应付这些复杂的情况，上面简单的使用方式已不能完成任务，这就要借助与 JTable 相联系的模型来实现。

与 JTable 相关的模型有三个：TableModel、TableColumnModel 和 ListSelectionModel。其中最常用的是 TableModel，下面就对 TableModel 来做一个全面的介绍。

TableModel 本身是一个接口，在这个接口里面定义了若干的方法：包括了存取表格单元的内容、计算表格的列数等等基本存取操作，设计者可以简单地利用 TableModel 来实现他所想要的表格。它的全部方法如表 14.9 所示：

表 14.9　TableModel中的全部方法

方法	说明
void addTableModelListener(TableModelListener l)	增加一个TableModelEvent的事件监听器。当表格的TableModel有所变化时，会监听到该事件
Class getColumnClass(int columnIndex)	返回指定列的数据类型的类名称
int getColumnCount()	返回列的数目
String getColumnName(int columnIndex)	返回列名称
int getRowCount()	返回行的数目
Object getValueAt(int rowIndex,int columnIndex)	返回指定单元中的数据
boolean isCellEditable(int rowIndex,int columnIndex)	判断指定单元是否可编辑，true表示可编辑
void removeTableModelListener(TableModelListener l)	删除一个监听器
void setValueAt(Object aValue,int rowIndex,int columnIndex)	设置指定单元的值

由于 TableModel 本身是一个接口，若要直接实现此接口来建立表格并不是件轻松的事。因此系统提供了两个类分别实现了这个接口：一个是 AbstractTableModel 抽象类，一个是 DefaultTableModel 实例类。前者实现了大部份的 TableModel 方法，让用户可以很有弹性地构造自己的表格模型，后者继承前面这个抽象类。它是 JTable 默认的模型。

AbstractTableModel 是一个抽象类，这个类实现大部份的 TableModel 方法——除了 getRowCount()、getColumnCount()和 getValueAt()这三个方法。因此程序员的主要任务就是去实现这三个方法。另外这个抽象方法还提供了一些其他的辅助方法，常用的辅助方法如表 14.10 所示：

表 14.10　AbstractTableModel的常用方法

方法	说明
int findColumn(String columnName)	寻找在列名称中是否含有columnName这个值，若有，则返回其所在行的位置；反之则返回-1表示未找到

void fireTableCellUpdated(int row, int column)	通知所有的监听器在这个表格中的(row,column)字段的内容已经改变了
void fireTableChanged(TableModelEvent e)	将所收到的事件传送给所有在这个table model中注册过的TableModelListeners
void fireTableDataChanged()	通知所有的监听器在这个表格中列的内容已经改变了，列的数目可能已经改变了，因此JTable可能需要重新显示此表格的结构
void fireTableRowsDeleted(int firstRow, int lastRow)	通知所有的监听器在这个表格中第firstrow行至lastrow行已经被删除了
void fireTableRowsUpdated(int firstRow, int lastRow)	通知所有的监听器在这个表格中第firstrow行至lastrow行已经被修改了
void fireTableRowsInserted(int firstRow, int lastRow)	通知所有的监听器在这个表格中第firstrow行至lastrow行已经被加入了
void fireTableStructureChanged()	通知所有的监听器在这个表格的结构已经改变了，列的数目、名称以及数据类型都可能已经改变了
Public EventListener[] getListeners(Class listenerType)	返回所有在这个table model所建立的监听器中符合listenerType的监听器，以数组形式返回

表中没有列出这个抽象类实现 TableModel 接口中的 6 个方法，另外，由于 getRowCount()、getColumnCount()和 getValueAt()这 3 个方法没有实现，如果自己写一个表格能用的模型，必须要实现这 3 个方法。

在下面的例子中，重写一个 MyTableModel 来代替系统提供的 DefaultTableModel，这样可以在显示 Boolean 类型的数据时，不以字符串的形式显示，而是以 checkBox 的形式显示，而且字符串会左对齐，而数值则右对齐。

【例 14.34】表格使用示例 5

//----------文件名 MyTableModel.java，程序编号 14.35----------

```
//本类实现了一个表格用的模型，取代默认的模型
import javax.swing.table.AbstractTableModel;
final class MyTableModel extends AbstractTableModel{
   private Object[][] date;  //存储表格中的数据
   private String [] tableName;      //存储表头
   //这个构造方法，由调用者提供数据和表头
   public MyTableModel(Object [][] date, String []tableName){
      this.date = date;
      this.tableName = tableName;
   }
   //这个构造方法，只需要提供数据，表头依次显示 A、B……
   public MyTableModel(Object [][] date){
      this.date = date;
      tableName = new String[date[0].length];
      char [] tch = {'A'};
      for (int i=0;i<tableName.length;++i){
        tableName[i] = new String(tch);
        tch[0]++;
      }
   }
   //下面三个方法必须要提供
   public int getColumnCount(){
      return date[0].length;
   }
   public int getRowCount(){
     return date.length;
   }
```

```
  public String getColumnName(int col) {
    return tableName[col];
  }
  public Object getValueAt(int row, int col){
    return date[row][col];
  }
  //覆盖父类的方法，改变数据显示的形式
  public Class getColumnClass(int c) {
    return date[0][c].getClass();
  }
}
```

在这个类中，真正和显示有关的就只有 getColumnClass()方法，它返回实际存储的数据的类类型，这样在显示的时候，表格会根据这些类型来做调整。

然后对例 14.32 中的程序稍做修改，用上面的模型取代默认模型创建表格。

//----------文件名 demoJTable_5.java，程序编号 14.36----------

```
import javax.swing.*;
import javax.swing.table.*;  //TableColumn 在这个包中
import java.awt.*;
import java.util.*;
public class demoJTable_5{
  JFrame mainJFrame;
  Container con;
  JScrollPane JSPane;
  JTable DataTable;
  MyTableModel myModel;
  public demoJTable_5(){
    mainJFrame = new JFrame();
    Object[][] playerInfo={
                           {"小王",new Integer(66),new Integer(72),new Integer(98),
                             new Boolean(false),new Boolean(false)},
                           {"小张",new Integer(82),new Integer(69),new Integer(78),
                             new Boolean(true),new Boolean(false)},
                          };
    String[] Names={"姓名","语文","数学","总分","及格","作弊"};
    //创建带内容和表头信息的模型
    myModel = new MyTableModel(playerInfo,Names);
    //用模型来创建表格，取代了默认的模型
    DataTable=new JTable(myModel);
    //设置宽度
    for (int i=0;i<6;i++){
      TableColumn  column=DataTable.getColumnModel().getColumn(i);
      if ((i%2)==0)
        column.setPreferredWidth(150);
      else
        column.setPreferredWidth(50);
    }
    JSPane=new JScrollPane(DataTable);
    mainJFrame.add(JSPane);
    mainJFrame.setTitle("JTable 使用示例");
```

```
    mainJFrame.setSize(400,200);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
     new demoJTable_5();
  }
}
```

程序运行截图如 14.65 所示：

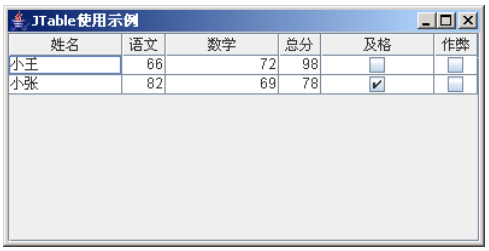

图 14.65　程序运行截图

在这个例子中，只对模型稍作修改就可以改变表格的显示式样，很好地说明了 MVC 设计模式的灵活性。

14.8.10　树（JTree）使用示例

树越来越多地出现在各种软件中，例如 Windows 的资源管理器左侧的目录管理就是一棵典型的树。对于用户而言，树操作方便，界面简洁美观。但对于程序员而言，对树的控制是一件比较麻烦的事情。

JTree 常用的构造方法有：

- JTree()：按照默认情形建一棵空树
- JTree(TreeModel newModel)：用参数 newModel 来创建一棵树
- JTree(TreeNode root)：创建一棵以 root 为根节点的树
- JTree(Vector <?>value)：用向量 value 创建一棵树，参数中的每一个元素都是根节点的子节点

JTree 的方法很多，表 14.11 列出了其中的一些常用方法。

表 14.11　JTree的常用方法

方法	说明
void addSelectionPath(TreePath path)	将由path所确定的节点设置为被选中
void addTreeExpansionListener(TreeExpansionListener tel)	增加一个TreeExpansion事件监听器
void addTreeSelectionListener(TreeSelectionListener tsl)	增加一个TreeSelection事件监听器
void ddTreeWillExpandListener(TreeWillExpandListener tel)	增加一个TreeWillExpand事件监听器
void cancelEditing()	取消当前的编辑会话
void clearSelection()	清除所有的选择
void collapsePath(TreePath path)	将path所确定的节点收缩，并保证可见
void expandPath(TreePath path)	将path所确定的节点展开，并保证可见
TreeCellEditor getCellEditor()	获取当前树的编辑者
TreePath getEditingPath()	获取正在编辑的元素的路径
int getMaxSelectionRow()	获取被选中的最后一行

int getMinSelectionRow()	获取被选中的第一行
TreeModel getModel()	获取提供数据的模型
TreePath getPathForRow(int row)	获取指定行的路径
int getRowCount()	获取被显示的行的数目
int getRowForPath(TreePath path)	获取由path所确定的节点所在的行
int getSelectionCount()	获取被选择的节点的数目
TreeSelectionModel getSelectionModel()	获得被选择数据的模型
TreePath getSelectionPath()	获得第一个被选择的节点的路径
int getVisibleRowCount()	获取显示区中被显示的行数
boolean isCollapsed(int row)	判断指定的行是否被收缩
boolean isCollapsed(TreePath path)	判断由path所确定的节点是否被收缩
boolean isEditable()	如果树可编辑，返回true
boolean isExpanded(int row)	如果row所指定行中的节点被展开，返回true
boolean isExpanded(TreePath path)	如果path所确定的节点被展开，返回true
boolean isRootVisible()	如果根节点被显示，返回true
boolean isRowSelected(int row)	如果row所指定行中的节点被选中，返回true
boolean isSelectionEmpty()	如果当前选择为空，返回true
boolean isVisible(TreePath path)	path中任意节点被显示，返回true
void removeSelectionPath(TreePath path)	从被选择的节点中，取消由path所确定的节点
void removeSelectionRow(int row)	从被选择的节点中，取消由row所确定的节点
void setCellEditor(TreeCellEditor cellEditor)	设置单元编辑者
void setEditable(boolean flag)	设置可编辑性
void setModel(TreeModel newModel)	设置提供数据的模型
void setRootVisible(boolean rootVisible	设置根节点的可见性
void setUI(TreeUI ui)	设置树的观感界面
boolean stopEditing()	试图停止编辑会话

当用户单击非叶子节点前的符号或是双击该节点时，将会展开或是收缩该节点的子树，同时会产生 TreeExpansionEvent 事件，程序员如果需要监听这个事件，可以用 addTreeExpansionListener (TreeExpansionListener tel)来添加监听器。其中 TreeExpansionEvent 事件接口定义了一个 getPath 方法，可以获得节点的完整信息。

如果用户只是选择了一个节点，会产生 TreeSelectionEvent 事件，如果需要监听该事件，可以用 addTreeSelectionListener(TreeSelectionListener tsl)方法来添加监听器。

JTree 还提供了一个很有用的方法：public TreePath getPathForLocation(int x, int y)，将返回指定位置的节点的完整信息。程序员可以获得当前鼠标单击位置，然后由该方法计算出被单击的节点。

注意：JTree 和 JList 一样，本身是没有滚动条的，需要把它放到一个 JScrollPane 中才可以滚动。

下面这个简单的例子，演示了如何创建一棵有层次的树，以及如何获得用户选择的节点。

【例 14.35】创建 JTree 示例

//----------文件名 demoJTree.java，程序编号 14.37----------

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

public class demoJTree implements TreeSelectionListener{
   JFrame mainJFrame;
   Container con;
   JLabel msgLabel;
```

```
  JScrollPane JSPane;
  JTree simpleTree;
  private DefaultMutableTreeNode tmpNode,root;

  public demoJTree() {
    mainJFrame=new JFrame("JTree 使用示例");
    con=mainJFrame.getContentPane();
    msgLabel=new JLabel();
    //创建根节点
    root=new DefaultMutableTreeNode("Option");
    //创建根节点的第一个孩子节点
    tmpNode=new DefaultMutableTreeNode("A");
    //将这个孩子节点添加进来
    root.add(tmpNode);
    //为这个孩子节点添加两个子节点
    tmpNode.add(new DefaultMutableTreeNode("a1"));
    tmpNode.add(new DefaultMutableTreeNode("a2"));
    //创建根节点的第二个孩子节点
    tmpNode=new DefaultMutableTreeNode("B");
    //添加进来
    root.add(tmpNode);
    //为这个孩子节点添加四个子节点
    tmpNode.add(new DefaultMutableTreeNode("b1"));
    tmpNode.add(new DefaultMutableTreeNode("b2"));
    tmpNode.add(new DefaultMutableTreeNode("b3"));
    tmpNode.add(new DefaultMutableTreeNode("b4"));
    //以 root 为根，创建树
    simpleTree=new JTree(root);
    JSPane=new JScrollPane(simpleTree);
    simpleTree.addTreeSelectionListener(this);
    con.add(JSPane,BorderLayout.CENTER);
    con.add(msgLabel,BorderLayout.SOUTH);
    mainJFrame.setSize(300,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  //在标签上显示用户选择的节点
  public void valueChanged(TreeSelectionEvent e){
    msgLabel.setText(e.getPath().toString());
  }

  public static void main(String[] args) {
    new demoJTree();
  }
}
```

图 14.66 是程序启动时的截图：

图 14.66　程序启动时截图

当用户选择某个节点之后的截图如 14.67 所示：

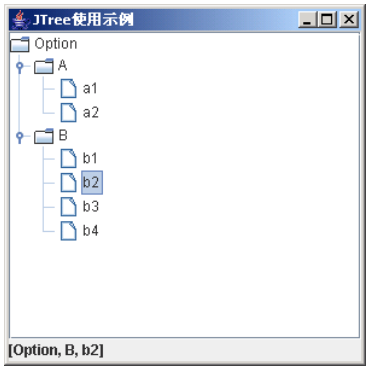

图 14.67　选择节点之后的截图

上面的程序已经可以在大多数情况下使用，但在某些情况下，需要动态的增加或者减少节点。这要把上面的例子改动一下，增加这个功能。

JTree 也是基于 MVC 模式的，和 JList 一样，本身没有提供增加、删除节点功能，这些需要依靠保存其数据、并与其密切联系的 DefaulteTreeModel 来实现。只要是利用 DefaulteTreeModel 来创建 JTree 对象，就可以通过调用 DefaulteTreeModel 的数据修改功能来修改 JTree 的节点。看下面的例子：

【例 14.36】在 JTree 中增加节点示例

//----------文件名 addNodeInJTree.java，程序编号 14.38----------

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.event.*;
import java.awt.*;
import javax.swing.event.*;
import javax.swing.tree.DefaultTreeModel;

public class addNodeInJTree implements TreeSelectionListener,ActionListener{
  JFrame mainJframe;
  Container con;
  JLabel msgLabel;
  JScrollPane JSPane;
  JPanel panel;
  JTextField text;
  JButton addBtn;
```

```
JTree simpleTree;
private DefaultMutableTreeNode tmpNode,root;
private DefaultTreeModel rt;  //准备利用该对象来创建树

public addNodeInJTree() {
  mainJframe=new JFrame("JTree 使用示例");
  con=mainJframe.getContentPane();
  msgLabel=new JLabel();
  root=new DefaultMutableTreeNode("Option");
  tmpNode=new DefaultMutableTreeNode("A");
  root.add(tmpNode);
  tmpNode.add(new DefaultMutableTreeNode("a1"));
  tmpNode.add(new DefaultMutableTreeNode("a2"));
  tmpNode=new DefaultMutableTreeNode("B");
  root.add(tmpNode);
  tmpNode.add(new DefaultMutableTreeNode("b1"));
  tmpNode.add(new DefaultMutableTreeNode("b2"));
  tmpNode.add(new DefaultMutableTreeNode("b3"));
  tmpNode.add(new DefaultMutableTreeNode("b4"));
  //利用前面的数据生成 Model 对象
  rt=new DefaultTreeModel(root);
  //利用 Model 对象创建树
  simpleTree=new JTree(rt);
  JSPane=new JScrollPane(simpleTree);
  simpleTree.addTreeSelectionListener(this);
  con.add(JSPane,BorderLayout.CENTER);
  panel=new JPanel();
  panel.setLayout(new FlowLayout());
  //增加一个按钮，用户通过单击按钮来通知程序增加节点
  addBtn=new JButton("增加子节点");
  addBtn.addActionListener(this);
  text=new JTextField("请在这输入子节点的内容");
  text.setColumns(11);
  panel.add(msgLabel);
  panel.add(text);
  panel.add(addBtn);
  con.add(panel,BorderLayout.SOUTH);
  mainJframe.setSize(400,300);
  mainJframe.setVisible(true);
  mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void valueChanged(TreeSelectionEvent e){
  msgLabel.setText(e.getPath().toString());
}

public void actionPerformed(ActionEvent e){
  DefaultMutableTreeNode tp;
  tp=new DefaultMutableTreeNode(text.getText());
  //利用 Model 来增加节点，这里将其加入到根节点下面
```

```
    rt.insertNodeInto(tp,root,0);
  }

  public static void main(String[] args) {
    new addNodeInJTree();
  }
}
```

程序运行中，插入一个节点之后的截图如 14.68 所示：

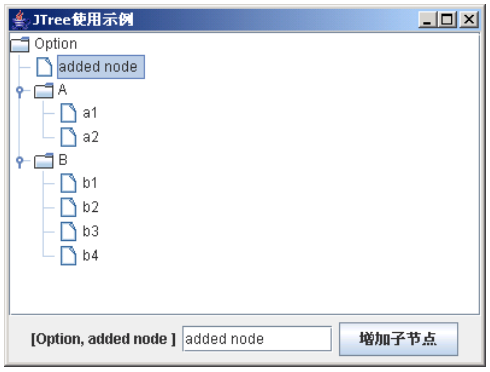

图 14.68　插入节点之后

这里的程序写得比较简单，插入的节点都是增加在根节点的下面。其实可以稍加改动，将节点加入在任何想要的位置，请读者自行完成。

14.8.11　菜单使用示例

绝大多数的窗口应用程序都拥有菜单，它是程序接受用户命令最主要的组件。菜单所占区域少，且使用灵活，因此得到广泛应用。

一个完整的菜单通常由三部分构成：菜单条（JMenuBar）、下拉式菜单（JMenu）和菜单项（JMenuItem）。要向窗口增加菜单，第一步是创建菜单条，并添加到窗口中；然后创建下拉式菜单，添加到菜单条中对应的位置；然后创建菜单项，并添加到每个菜单中。至此，菜单结构已经创建完成。随后需要添加对每个菜单项的的事件监听器，完成对单击事件的处理。

菜单项中有一种特殊的子项：JCheckBoxMenuItem，它象 JCheckBox 一样，具有“可选择性”。和下拉式菜单对应的，还有一种称为弹出式菜单的 JPopupMenu。除了不属于某个菜单条之外，它拥有下拉式菜单的所有功能，编程上也极为相似。

表 14.12 列出了菜单类的一些常用方法。

表 14.12　菜单类的常用方法

所属类	方法	说明
JMenuBar	JMenuBar()	创建一个空菜单条
	JMenu add(JMenu c)	添加一个下拉菜单
	void remove(Component comp)	删除一个下拉菜单
JMenu	JMenu(String s)	创建一个标题为s的下拉菜单
	JMenuItem add(String s)	添加一个以s为显示内容的菜单项
	JMenuItem add(JMenuItem menuItem)	添加一个菜单项
	void addMenuListener(MenuListener l)	添加菜单事件监听器
	void addSeparator()	增加一个分隔条

	void insert(String s, int pos)	在指定位置插入一个以s显示内容的菜单项
	JMenuItem insert(JMenuItem mi, int pos)	在指定位置插入一个菜单项
	boolean isSelected()	本菜单是否被选中
	void remove(int pos)	删除指定位置的菜单项
	void remove(JMenuItem item)	删除指定的菜单项
	void removeAll()	删除所有菜单项
	void setMenuLocation(int x, int y)	设置菜单弹出的位置
	JMenuItem getItem(int pos)	返回指定位置的菜单项
JMenuItem	JMenuItem(String text, Icon icon)	用指定的文字和图标创建一个菜单项
	JMenuItem(String text)	用指定的文字创建一个菜单项
	void addActionListener(ActionListener l)	添加单击事件监听器
	void addMenuKeyListener(MenuKeyListener l)	添加一个键盘监听器
	void setEnabled(boolean b)	设置菜单项是否可用
	void doClick()	虚拟菜单项被单击事件
JCheckBox MenuItem	boolean getState()	返回菜单项被选择的状态
	void setState(boolean b)	设置菜单项被选择的状态
JPopupMenu	void setVisible(boolean b)	设置弹出菜单的可见性
	void show(Component invoker, int x, int y)	在相对于invoker的坐标位置显示菜单
	void pack()	按照最少需要空间排列菜单
	void setPopupSize(int width, int height)	指定弹出窗口的大小

对菜单编程比较简单，方法很死板，不过如果菜单项目比较多，编起来会有点繁琐。下面来看一个仿照记事本的样子做出来的菜单。

【例 14.37】菜单使用示例

//----------文件名 demoJMenu.java，程序编号 14.39----------

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

public class demoJMenu {
  JFrame mainJFrame;
  Container con;
  JScrollPane JSPane;
  JTextArea text;
  JMenuBar mainMenuBar;
  JMenu fileMenu,editMenu,formatMenu,helpMenu;
  //“文件”菜单下的菜单项
  JMenuItem newItem,openItem,saveItem,saveasItem,pageItem,printItem,exitItem;
  //“编辑”菜单下的菜单项
  JMenuItem undoItem,cutItem,copyItem,pasteItem,findItem,replaceItem,selectallItem;
  //“设置”菜单下的菜单项
  JCheckBoxMenuItem wrapItem;
  JMenuItem fontItem;
  //“帮助”菜单下的菜单项
  JMenuItem helpItem,aboutItem;

  public demoJMenu() {
     mainJFrame=new JFrame("菜单使用示例");
     con=mainJFrame.getContentPane();
     text=new JTextArea();
     JSPane=new JScrollPane(text);
     //调用自定义的方法创建菜单结构
```

```
    createMenu();
    //添加菜单到窗口
    mainJFrame.setJMenuBar(mainMenuBar);
    con.add(JSPane,BorderLayout.CENTER);
    mainJFrame.setSize(400,300);
    mainJFrame.setVisible(true);
    mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void createMenu(){
    //创建 JMenuBar
    mainMenuBar=new JMenuBar();
    //创建四个 JMenu
    fileMenu=new JMenu("文件");
    editMenu=new JMenu("编辑");
    formatMenu=new JMenu("格式");
    helpMenu=new JMenu("帮助");
    //创建 JMenuItem 并添加到对应的 JMenu 中
    //创建文件菜单下面的菜单项
    mainMenuBar.add(fileMenu);
    newItem=new JMenuItem("新建");
    openItem=new JMenuItem("打开..");
    saveItem=new JMenuItem("保存..");
    saveasItem=new JMenuItem("另存为..");
    pageItem=new JMenuItem("页面设置..");
    printItem=new JMenuItem("打印..");
    exitItem=new JMenuItem("退出");
    fileMenu.add(newItem);
    fileMenu.add(openItem);
    fileMenu.add(saveItem);
    fileMenu.add(saveasItem);
    fileMenu.addSeparator();
    fileMenu.add(pageItem);
    fileMenu.add(printItem);
    fileMenu.addSeparator();
    fileMenu.add(exitItem);
    //创建编辑菜单下面的菜单项
    mainMenuBar.add(editMenu);
    undoItem=new JMenuItem("撤消");
    cutItem=new JMenuItem("剪切");
    copyItem=new JMenuItem("复制");
    pasteItem=new JMenuItem("粘贴");
    findItem=new JMenuItem("查找..");
    replaceItem=new JMenuItem("替换..");
    selectallItem=new JMenuItem("全选");
    editMenu.add(undoItem);
    editMenu.addSeparator();
    editMenu.add(cutItem);
    editMenu.add(copyItem);
    editMenu.add(pasteItem);
```

```
    editMenu.addSeparator();
    editMenu.add(findItem);
    editMenu.add(replaceItem);
    editMenu.addSeparator();
    editMenu.add(selectallItem);
    //创建格式菜单下面的菜单项
    mainMenuBar.add(formatMenu);
    wrapItem=new JCheckBoxMenuItem("自动换行");
    fontItem=new JMenuItem("设置字体..");
    formatMenu.add(wrapItem);
    formatMenu.add(fontItem);
    //创建帮助菜单下面的菜单项
    mainMenuBar.add(helpMenu);
    helpItem=new JMenuItem("帮助主题");
    aboutItem=new JMenuItem("关于..");
    helpMenu.add(helpItem);
    helpMenu.add(aboutItem);
  }

  public static void main(String[] args) {
     new demoJMenu();
  }
}
```

程序运行截图如 14.69 所示：

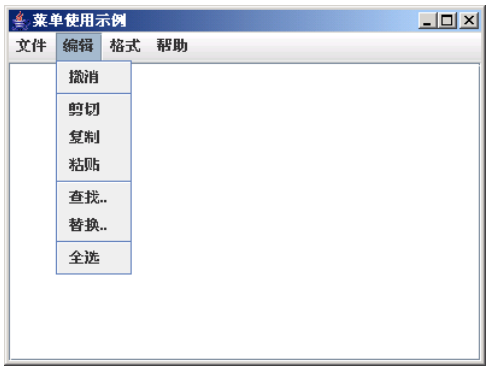

图 14.69　程序运行截图

文字菜单项中间的分隔线是用的这样的方法：fileMenu.addSeparator()。

上面的 creatMenu()方法看上去比较烦琐，也可以将所有的显示信息存放到一个 String 数组中，然后利用 JMenuItem 数组来完成菜单的建立工作。不过这样一来，每个菜单项的名字就不再有意义。

上面的程序已经具备了记事本的外观，但并没有实现任何功能。这是因为还没有对菜单事件进行编程。要监听菜单事件，和监听按钮事件是一样的。只要用 addActionListener()就可以加上监听器，然后再覆盖 actionPerformed()方法，写入所需的程序代码。

这里的一个问题是，由于菜单项非常多，如果全部由一个监听器来完成，则显然需要在 actionPerformed()方法中进行大量的代码判断，像下面这个样子：

```
if(e.getSource()==****)      ....;
else if(e.getSource==*****)   ....;
else if(e.getSource==*****)   ....;
```

```
......
else                          ....;
```

这样显得程序比较繁杂，一种替代方法是为每一个菜单项写单独的监听器。本章 14.13 节的例子中，将看到后一种方法。

最后，还讨论一下 JPopupMenu 的使用。它的创建和 JMenu 的创建并没有什么不同。只是它不需要加入到 JMenuBar 中，而是需要绑定到某个组件上（比如上例中的 JTextArea），这可以使用组件的 setComponentPopupMenu(JPopupMenu popup)方法，以后当用户在这个组件上按鼠标右键时，该菜单将会自动弹出来，这就无需程序员计算菜单显示的位置。

14.9 布局管理

Java 的 GUI 程序设计中，最大亮点是引入了布局管理。每一种布局管理都是依据一定的算法来排列容器上的组件。它使得程序员在设计界面是只需要考虑各个组件之间的相对位置关系，而不必仔细考虑它具体的位置。而且当窗口的大小发生变化时，能够保证组件的位置也发生相应的变化。这就大大化简了设计和编程时的烦琐，使得程序员不必借助 RAD 工具仍能高效地开发 GUI 程序。

Java 中提供了 AWT 和 Swing 两种布局管理，前者的布局管理完全对后者的组件适用。下面先介绍 AWT 的布局管理。

14.9.1 流式布局（FlowLayout）回顾

前面大多数例子使用的就是该布局。它将组件按照先后加入容器的顺序，从左到右，从上到下依次排列。一行排满自动转入到下一行。每一行的组件都是居中排列。JPanel 的默认布局就是 FlowLayout。如果是 Container，则需要用 setLayout(new FlowLayout())来指定。

由于前面已经多次使用流式布局，这里不再举例说明。

14.9.2 边框布局（BorderLayout）使用示例

该布局将整个容器分成 NORTH、SOUTH、EAST、WEST、CENTER 共 5 个区域，这几个区域按照地图位置排列，分别是：上、下、右、左、中间，每个区域只能存放一个组件。它主要的方法有：

- BorderLayout(int hgap, int vgap)：创建布局，并指定组件间水平和垂直方向上的空白
- void setHgap(int hgap)：设置组件间水平方向的间隔
- void setVgap(int vgap)：设置组件间垂直方向的间隔

Container 的默认布局就是该布局，一般情况下，会这么来用：con.add(组件名,BorderLayout.NORTH)。后面那个参数就是指定组件放置的位置。下面看个实际的例子。

【例 14.38】边框布局使用示例

//----------文件名 demoBorderLayout.java，程序编号 14.40----------

```
import javax.swing.*;
import java.awt.*;
public class demoBorderLayout {
  JPanel con;
  JFrame mainJFrame;
  JLabel Label1,Label2,Label3,Label4,Label5;
```

```
 public demoBorderLayout() {
   mainJFrame=new JFrame("布局使用示例");
   con=new JPanel();
   //JPanel 的对象默认布局是 FlowLayout，要这样来改变它
   con.setLayout(new BorderLayout());
   Label1=new JLabel("这是在北方");
   Label2=new JLabel("这是在南方");
   Label3=new JLabel("这是在东方");
   Label4=new JLabel("这是在西方");
   Label5=new JLabel("这是在中间");
   con.add(Label1,BorderLayout.NORTH);  //也可用 con.add("North",Label1);
   con.add(Label2,BorderLayout.SOUTH);
   con.add(Label3,BorderLayout.EAST);
   con.add(Label4,BorderLayout.WEST);
   con.add(Label5,BorderLayout.CENTER);
   mainJFrame.add(con);
   mainJFrame.setSize(300,300);
   mainJFrame.setVisible(true);
   mainJFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
 public static void main(String[] args) {
    new demoBorderLayout();
 }
}
```

在程序中，凡是使用：

```
con.add(Label1,BorderLayout.NORTH);
```

也可用

```
con.add("North",Label1);
```

来代替，只要将常量改成对应的字符串即可。

程序运行截图如图 14.70 所示：

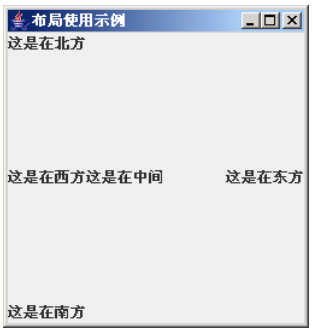

图 14.70　程序运行截图

这个显示并不好看，所以边框布局往往还要配合其他的容器来使用。

14.9.3　网格布局（GridLayout）使用示例

GridLayout 允许用户将容器设置成网格状——相当于一个标准的二维表格，每个单元格可以放置一个组件。放置的时候按照从左到右、从上到下的顺序依次存放。每个单元格都不允许为空。如果希望某

个单元格显示为空，可以在其中加入一个空白标签。

它最常用的构造方法是：GridLayout(row,col)，第一个参数指定行数，第二个参数指定列数。

【例 14.39】网格布局使用示例

//----------文件名 demoGridLayout.java，程序编号 14.41----------

```
import javax.swing.*;
import java.awt.*;
public class demoGridLayout{
  JPanel con;
  JFrame mainJframe;
  JLabel []Label;
  public demoGridLayout() {
    int i,j,k=0;
    mainJframe=new JFrame("布局使用示例");
    con=new JPanel();
    con.setLayout(new GridLayout(2,3));
    Label=new JLabel[6];
    for(i=0; i<2; i++)
       for(j=0; j<3; j++){
         Label[k]=new JLabel("这是第"+i+","+j+"号单元");
         //将标签依次添加到各个单元中
         con.add(Label[k++]);
    }
    mainJframe.getContentPane().add(con);
    mainJframe.setSize(400,100);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
     new demoGridLayout();
  }
}
```

程序运行截图如图 14.71 所示：

图 14.71　程序运行截图

14.9.4　卡片布局（CardLayout）使用示例

当组件比较多的时候，一个窗口可能放置不下。于是人们采用类似于卡片的布局。把相关组件添加到同一张卡片中，一个窗口可以放置多张卡片。然后根据需要选择某一张卡片。这张被选中的卡片将占据整个窗口区域，其余的卡片不可见。

由于一次只能看到一张卡片，而且不能任意地切换卡片，所以 CardLayout 比较适合分类操作或是有多个操作步骤、每个步骤有先后关系的情况。当第一步完成后，切换到第二张卡片，然后切换到第三张卡片……典型的例子是程序的安装向导。

注意：如果需要在多张卡片之间来回切换，一般会选择 JTabbedPane，通过它的标签来选择卡片。

当一个容器拥有了 CardLayout 布局之后，每次调用该容器的 add(String name, Component comp)方法，它都会自动生成一张以 name 命名的卡片，并将组件 comp 放入这张卡片。

与前面介绍的普通布局不同，本布局提供了一些辅助方法帮助程序员操作卡片。比如，需要在一张卡片上放置多个组件，可以有两种方法：一是将这多个组件放到一个 JPanel 中，再将 JPanel 加到卡片上。另一种方法是调用 CardLayout 的 addLayoutComponent(Component comp, Object constraints)方法，方法的第一个参数是组件名，第二个参数是由调用者给卡片起的名字，只要名字相同，多个组件就可以放到同一张卡片中。而且如果使用了该方法为卡片起名字，后面就还可以使用 show(Container parent,String name)方法来任意指定要显示的卡片。

CardLayout 提供的常用方法如表 14.13 所示：

表 14.13　CardLayout提供的常用方法

方法	说明
void addLayoutComponent(Component comp, Object constraints)	将组件comp添加到名为constraints的卡片中
void first(Container parent)	弹出指定容器中的第一张卡片
void last(Container parent)	弹出指定容器中的最后一张卡片
void layoutContainer(Container parent)	让容器parent也使用本布局对象
void next(Container parent)	弹出指定容器中的下一张卡片
void previous(Container parent)	弹出指定容器中的前一张卡片
void removeLayoutComponent(Component comp)	删除布局中的组件
void show(Container parent, String name)	如果组件是用addLayoutComponent加入到卡片中的，则此方法可以显示由name指定的卡片上的组件

下面的程序模拟了一般的软件安装过程。

【例 14.40】卡片布局使用示例

//----------文件名 demoCardLayout.java，程序编号 14.42----------

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class demoCardLayout implements ActionListener{
   Container con;
   JFrame mainJframe;
   JLabel Label1,Label2,Label3;
   JButton nextBtn1,prevBtn1,nextBtn2,prevBtn2,OKBtn;
   CardLayout myCard;
   JPanel panel1,panel2,panel3;
   public demoCardLayout() {
      mainJframe=new JFrame("卡片布局使用示例");
      con=mainJframe.getContentPane();
      myCard=new CardLayout();
      con.setLayout(myCard);
     //为第一张卡片添加组件
      Label1=new JLabel("这是第一步");
      nextBtn1=new JButton("下一步");
      panel1=new JPanel();
      panel1.add(Label1);
      panel1.add(nextBtn1);
      con.add("first",panel1);
```

```
    //为第二张卡片添加组件
     Label2=new JLabel("这是第二步");
     prevBtn1=new JButton("上一步");
     nextBtn2=new JButton("下一步");
     panel2=new JPanel();
     panel2.add(Label2);
     panel2.add(prevBtn1);
     panel2.add(nextBtn2);
     con.add("second",panel2);
    //为第三张卡片添加组件
     Label3=new JLabel("这是第三步");
     prevBtn2=new JButton("上一步");
     OKBtn=new JButton("完成");
     panel3=new JPanel();
     panel3.add(Label3);
     panel3.add(prevBtn2);
     panel3.add(OKBtn);
     con.add("third",panel3);
    //添加事件监听器
     nextBtn1.addActionListener(this);
     prevBtn1.addActionListener(this);
     nextBtn2.addActionListener(this);
     prevBtn2.addActionListener(this);
     OKBtn.addActionListener(this);
     mainJframe.setSize(300,300);
     mainJframe.setVisible(true);
     mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  //根据用户单击的按钮显示所需的卡片
  public void actionPerformed(ActionEvent e){
     Object tp;
     tp=e.getSource();
     if (tp==nextBtn1 || tp==nextBtn2) myCard.next(con);
     if (tp==prevBtn1 || tp==prevBtn2) myCard.previous(con);
     if (tp==OKBtn)                    mainJframe.dispose();
  }
  public static void main(String[] args) {
     new demoCardLayout();
  }
}
```

程序运行情况如图 14.72 所示：

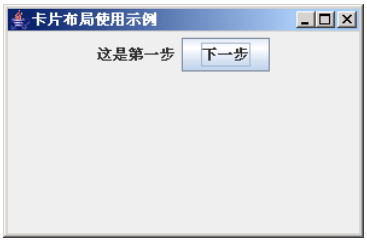

图 14.72　程序启动时截图

单击“下一步”按钮之后的截图如 14.73 所示：

图 14.73　程序运行截图

用户可以继续单击“上一步”或者“下一步”切换到前面或者后面的卡片，这里不再截图。

14.9.5　增强网格布局（GridBagLayout）使用示例

GridBagLayout 直译过来应该是网格包布局，不过笔者认为从它的功能来看，翻译成增强网格布局更准确。GridBagLayout 的功能类似于 GridLayout，但比后者的功能更强大，使用起来也要复杂得多。它能够制造出跨行和跨列的单元格，像下面这个样子：

图 14.74　GridBagLayout 设置的网格布局

这不再是一个规则的二维表格，每个单元格的大小不再相同。不过要达到这个效果，需要程序员编写比较多的代码。

GridBagLayout 设置单元格的规则和 HTML 语言的规则有点相似。不过为了指定单元格的行列值，还需要一个类 GridBagConstraints 配合。该类有以下几个常用属性：

- int gridx：当它为 0 时，指定组件存放的单元格是本行的第一列，默认值是 RELATIVE。
- int gridy：当它为 0 时，指定组件存放的单元格是本列的第一行，默认值是 RELATIVE。
- static final int RELATIVE：上面两个值是绝对定位，该值是相对定位。如果 gridy=n，而 gridx 设定该值后，后面的组件都从第 n 行一直往右放，直到出现下一个指定值。如果 gridx=n，而 gridy 设定该值后，后面的组件都从第 n 列一直往下放，直到出现下一个指定值。
- int gridwidth：指定该单元格所占列数。若该值设定为 RELATIVE，表示占据本行中除最后一格外的所有单元格。
- int gridheight：指定该单元格所占行数。若该值设定为 RELATIVE，表示占据本列中除最后一格外的所有单元格。
- public static final int REMAINDER：指定该单元格占据本行或本列剩余的所有空间。
- double weightx：指定如何分配额外的水平空间，默认值是 0.0。
- double weighty：指定如何分配额外的垂直空间，默认值是 0.0。
- Insets insets：为单元格中的组件设置与边框的间隔像素数。

例如，要创建如图 14.75 所示的这个布局：

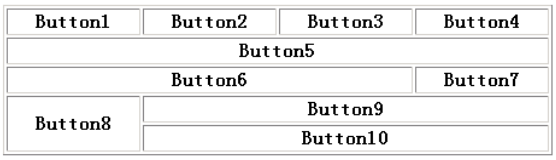

图 14.75　GridBagLayout 设置的布局示意图

需要先创建一个 GridBagConstraints 对象，并将其 fill 属性设置为 GridBagConstraints.BOTH，然后按照下面的规则设置属性：

- Button1，Button2，Button3：weightx = 1.0
- Button4：weightx = 1.0, gridwidth = GridBagConstraints.REMAINDER
- Button5：gridwidth = GridBagConstraints.REMAINDER
- Button6：gridwidth = GridBagConstraints.RELATIVE
- Button7：gridwidth = GridBagConstraints.REMAINDER
- Button8：gridheight = 2, weighty = 1.0
- Button9，Button10：gridwidth = GridBagConstraints.REMAINDER

下面的程序演示了如何用 GridBagLayout 和 GridBagConstraints 配合编制出不规则的表格布局。

【例 14.41】增强网格布局使用示例

//----------文件名 demoGridBagLayout.java，程序编号 14.43----------

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

public class demoGridBagLayout {
   Container con;
   JFrame mainJframe;
   GridBagLayout gridbag;
   //按照指定的属性创建并添加按钮到单元格中
   private void makebutton(String name,
                        GridBagLayout gridbag, GridBagConstraints c) {
      JButton btn = new JButton(name);
      //参数 c 决定了如何放置这个按钮
      gridbag.setConstraints(btn, c);
      con.add(btn);
   }

   public demoGridBagLayout(){
      GridBagConstraints c = new GridBagConstraints();
      gridbag = new GridBagLayout();
      mainJframe=new JFrame("增强网格布局使用示例");
      con=mainJframe.getContentPane();
      con.setLayout(gridbag);
      c.fill = GridBagConstraints.BOTH;
      //下面每个按钮占一行一列
      c.weightx = 1.0;
      makebutton("Button1", gridbag, c);
      makebutton("Button2", gridbag, c);
      makebutton("Button3", gridbag, c);
      c.gridwidth = GridBagConstraints.REMAINDER; //占据到本行结束
```

```
        makebutton("Button4", gridbag, c);
        //开始布置第二行的按钮
        c.weightx = 0.0; //重设为默认值
        //这里的 gridwidth 属性仍然是 REMAINDER，所以占据一整行
        makebutton("Button5", gridbag, c);
        //开始布置第三行
        c.gridwidth = GridBagConstraints.RELATIVE; //重起一行，占三格
        makebutton("Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; //占据到本行结束
        makebutton("Button7", gridbag, c);
        //开始布置第四行和第五行
        c.gridwidth = 1; //本单元格占两行一列
        c.gridheight = 2;
        c.weighty = 1.0;
        makebutton("Button8", gridbag, c);
        //开始布置其他的按钮，它们分在两行中
        c.weighty = 0.0;
        c.gridwidth = GridBagConstraints.REMAINDER; //占据到本行结束
        c.gridheight = 1; //只占一行
        makebutton("Button9", gridbag, c);
        makebutton("Button10", gridbag, c);
        //布局设置完毕
        mainJframe.setSize(400,180);
        mainJframe.setVisible(true);
        mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        new demoGridBagLayout();
    }
}
```

程序运行截图如 14.76 所示：

图 14.76　程序运行截图

应该承认，尽管 GridBagLay　B　　B　　B　　.　　9

管理器。ViewportLayout 是 JViewport 的内置管理器。由 OverLayout 布局管理器放置的组件将放置在其他组件的上面。BoxLayout 和 FlowLayout 很象，但它可以实现将容器中的组件按照水平方向或是垂直方向布置。14.10 中的程序就使用了它。关于这些布局管理的具体用，请查阅 API 手册。

到这里为止，关于编写 GUI 程序的基本知识都介绍完毕了，下面就将利用这些知识编写几个小的 GUI 程序，读者也可以从中学习编写这类程序的一些技巧。

14.10　GUI 程序设计实例 1——色盲检测

这一节介绍一个简单的色盲检测软件。该软件的基本原理是：所有的色彩都是由红、绿、黄三原色组成，每种原色的值在 0-255 之间变化；调节每一种原色的值，可以合成任意的色彩。

程序的基本思路就是：给出一个标准的色彩，要求用户调整三原色的值，如果认为混合得到的色彩与标准色彩相同，就可以按下“确定”键。程序再判断混合的色彩中各个原色的值与标准色彩之间的差距，如果误差在规定范围以内，就认为合格，反之则不合格。如果是色盲，是无法混合得到标准色彩的 。

为了让用户能够调节三原色，程序设置了三个滚动条，每个滚动条代表一种颜色的值。用户拖动滚动条的时候，实时显示混合得到的色彩，供用户做出判断。

图 14.77 就是程序设计的界面：

图 14.77　程序设计界面

图 14.77 中，左边将显示标准色，右边显示用户混合得到的颜色，它们都是用 JTextArea 显示背景色来实现的。下面来介绍这个程序的编制。

14.10.1 界面的实现

这个程序的界面上有较多的组件，但放置很有规律。首先介绍需要用到的组件和一些变量，声明如下：

```
private final static int KIND=3,MIN=0,MAX=256;
private int tolerance=15;   //允许的误差，读者可以修改这个值，越小则越难通过
private int standardRed,standardBlue,standardGreen; //机器产生的标准颜色
private final static String colorMsg[]={"红 最小","绿 最小","蓝 最小"};
protected JTextArea modelArea,userArea;  //前者显示标准色彩，后者显示用户混合的色彩
protected JScrollBar colorBar[]; //三个滚动条
```

```
protected JButton nextBtn,OKBtn;
protected JLabel color_min_Label[],maxLabel[];
protected JPanel areaPanel,barPanel[],btnPanel;//用于放置按钮的中间容器
protected Container con;
protected JFrame mainJframe;                    //主窗口
private HandleButtonClick handleBtn;            //处理按钮单击事件的内部类对象
private HandleScrollbarChange  handleScrollbar;//处理滚动条拖动事件的内部类对象
```

程序界面设计所用到的代码全部放在构造方法中：

```
  public CheckAchromatopsiat() {
     mainJframe=new JFrame("色盲检测");
    //用两个只读的 JTextArea 来显示颜色
     modelArea=new JTextArea();
     modelArea.setColumns(12);
     modelArea.setRows(8);
     modelArea.setEditable(false);
     userArea=new JTextArea();
     userArea.setColumns(12);
     userArea.setRows(8);
     userArea.setEditable(false);
     areaPanel=new JPanel();
     FlowLayout tmpLayout=new FlowLayout();
     //设置两个 JTextArea 之间的垂直距离
     tmpLayout.setHgap(20);
     areaPanel.setLayout(tmpLayout);
     areaPanel.add(modelArea);
     areaPanel.add(userArea);
     //创建滚动条和相应的提示标签，并放在各自的面板中
     color_min_Label=new JLabel[KIND];
     maxLabel=new JLabel[KIND];
     barPanel=new JPanel[KIND];
     colorBar=new JScrollBar[KIND];
     for(int i=0;i<3;i++){
        color_min_Label[i]=new JLabel(colorMsg[i]);
        maxLabel[i]=new JLabel("最大");
        colorBar[i]=new JScrollBar(JScrollBar.HORIZONTAL,MAX-1,1,MIN,MAX);
        barPanel[i]=new JPanel();
        barPanel[i].setLayout(new BorderLayout());
        barPanel[i].add(color_min_Label[i],BorderLayout.WEST);
        barPanel[i].add(colorBar[i],BorderLayout.NORTH);
        barPanel[i].add(maxLabel[i],BorderLayout.EAST);
     }
     //创建两个按钮，并放在面板中
     nextBtn=new JButton("开始");
     OKBtn=new JButton("确定");
     OKBtn.setEnabled(false);
     btnPanel=new JPanel();
     btnPanel.setLayout(new FlowLayout());
     btnPanel.add(nextBtn);
     btnPanel.add(OKBtn);
     //把前面的 Panel 放置到 frame 中
```

```
    con=mainJframe.getContentPane();
    //用 swing 中的 BoxLayout 布局，垂直布局
    con.setLayout(new BoxLayout(con,BoxLayout.Y_AXIS));
    con.add(areaPanel);
    con.add(barPanel[0]);
    con.add(barPanel[1]);
    con.add(barPanel[2]);
    con.add(btnPanel);
    //配置窗口的大小并显示
    mainJframe.setSize(400,350);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

14.10.2 “开始”按钮的事件处理

当用户按下“开始”按钮后，程序应当实现这样一些功能：

- 随机产生一种颜色方案
- 将标准颜色面板置成这种颜色
- 将用户颜色面板置成白色
- 将三个滚动条的值置成最大值
- 如果“确定”按钮不可用，将其变成可用
- 如果自己显示的文本是“开始”，将其变成“下一幅”

笔者为外部类加上一个内部类，处理按钮单击事件的代码如下：

```
public class HandleButtonClick implements ActionListener{
  public void actionPerformed(ActionEvent e){
    if (e.getSource()==nextBtn){
      //随即产生标准色彩
      standardRed=(int)(Math.random()*1000)%MAX;
      standardGreen=(int)(Math.random()*1000)%MAX;
      standardBlue=(int)(Math.random()*1000)%MAX;
      for(int i=0;i<KIND;i++) colorBar[i].setValue(MAX-1);
      //显示标准颜色
      modelArea.setBackground(new Color (standardRed,standardGreen,standardBlue));
      //显示要用户调整的颜色，为白色
      userArea.setBackground(new Color(MAX-1,MAX-1,MAX-1));
      //改变两个按钮的外观
      if (!OKBtn.isEnabled()){
        OKBtn.setEnabled(true);
        nextBtn.setText("下一幅");
      }
   }
}
```

14.10.3 “确定”按钮的事件处理

当用户按下“确定”按钮后，需要程序判断用户调整的颜色是否正确（即与标准颜色的误差是否在规

定范围内），并给出相应的提示。

为此，需要将上面的 actionPerformed 方法做一下修改，增加逻辑判断：

```
boolean correct=true;
if (Math.abs(colorBar[0].getValue()-standardRed)>tolerance)
      correct=false;
if (Math.abs(colorBar[1].getValue()-standardGreen)>tolerance)
      correct=false;
if (Math.abs(colorBar[2].getValue()-standardBlue)>tolerance)
      correct=false;
if (correct){
      JOptionPane.showMessageDialog(mainJframe,"颜色匹配正确");
}else{
      JOptionPane.showMessageDialog(mainJframe,"颜色匹配错误");
}
```

14.10.4 滚动条的事件处理

当用户拖动滚动条上的指示块时，用户颜色板必须随之变化，所以程序要响应滚动条的 AdjustmentEvent 事件，这需要为滚动条添加一个事件监听器，所用方法是：addAdjustmentListener(AdjustmentListener l)。

下面的这个内部类，实现了实时获取滚动条的值，并立即改变颜色板的颜色功能。

```
public class HandleScrollbarChange implements AdjustmentListener{
  public void adjustmentValueChanged(AdjustmentEvent e){
     int red,green,blue;
     red=colorBar[0].getValue();
     green=colorBar[1].getValue();
     blue=colorBar[2].getValue();
     //设置用户调整的颜色
     userArea.setBackground(new Color(red,green,blue));
  }
}
```

只要给组件安装上这两个监听器，就可以实现程序的基本功能了。完整的程序请看下面。

14.10.5 完整的程序

【例 14.42】色盲检测程序

//----------文件名 CheckAchromatopsiat.java，程序编号 14.44----------

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

public class CheckAchromatopsiat {
  private final static int KIND=3,MIN=0,MAX=256;
  private int tolerance=15;  //允许的误差
  private int  standardRed,standardBlue,standardGreen;  //机器产生的标准颜色
  private final static String colorMsg[]={"红 最小","绿 最小","蓝 最小"};
  protected JTextArea modelArea,userArea;
```

```
protected JScrollBar  colorBar[];
protected JButton    nextBtn,OKBtn;
protected JLabel  color_min_Label[],maxLabel[];
protected JPanel  areaPanel,barPanel[],btnPanel;
protected Container con;
protected JFrame mainJframe;

private HandleButtonClick handleBtn;
private HandleScrollbarChange  handleScrollbar;

public CheckAchromatopsiat() {
  mainJframe=new JFrame("色盲检测");
  modelArea=new JTextArea();
  modelArea.setColumns(12);
  modelArea.setRows(8);
  modelArea.setEditable(false);
  userArea=new JTextArea();
  userArea.setColumns(12);
  userArea.setRows(8);
  userArea.setEditable(false);
  areaPanel=new JPanel();
  FlowLayout tmpLayout=new FlowLayout();
  tmpLayout.setHgap(20);
  areaPanel.setLayout(tmpLayout);
  areaPanel.add(modelArea);
  areaPanel.add(userArea);
  handleScrollbar=new HandleScrollbarChange();
  color_min_Label=new JLabel[KIND];
  maxLabel=new JLabel[KIND];
  barPanel=new JPanel[KIND];
  colorBar=new JScrollBar[KIND];
  for(int i=0;i<3;i++){
    color_min_Label[i]=new JLabel(colorMsg[i]);
    maxLabel[i]=new JLabel("最大");
    colorBar[i]=new JScrollBar(JScrollBar.HORIZONTAL,MAX-1,1,MIN,MAX);
    colorBar[i].addAdjustmentListener(handleScrollbar);
    barPanel[i]=new JPanel();
    barPanel[i].setLayout(new BorderLayout());
    barPanel[i].add(color_min_Label[i],BorderLayout.WEST);
    barPanel[i].add(colorBar[i],BorderLayout.NORTH);
    barPanel[i].add(maxLabel[i],BorderLayout.EAST);
  }
  handleBtn=new HandleButtonClick();
  nextBtn=new JButton("开始");
  nextBtn.addActionListener(handleBtn);
  OKBtn=new JButton("确定");
  OKBtn.setEnabled(false);
  OKBtn.addActionListener(handleBtn);
  btnPanel=new JPanel();
  btnPanel.setLayout(new FlowLayout());
```

```
 btnPanel.add(nextBtn);
 btnPanel.add(OKBtn);
 con=mainJframe.getContentPane();
 con.setLayout(new BoxLayout(con,BoxLayout.Y_AXIS));
 con.add(areaPanel);
 con.add(barPanel[0]);
 con.add(barPanel[1]);
 con.add(barPanel[2]);
 con.add(btnPanel);
 mainJframe.setSize(400,350);
 mainJframe.setVisible(true);
 mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public static void main(String[] args) {
   new CheckAchromatopsiat();
}

public class HandleButtonClick implements ActionListener{
  public void actionPerformed(ActionEvent e){
    if (e.getSource()==nextBtn){
      standardRed=(int)(Math.random()*1000)%MAX;
      standardGreen=(int)(Math.random()*1000)%MAX;
      standardBlue=(int)(Math.random()*1000)%MAX;
      for(int i=0;i<KIND;i++) colorBar[i].setValue(MAX-1);
      modelArea.setBackground(new Color(standardRed,standardGreen,standardBlue));
      userArea.setBackground(new Color(MAX-1,MAX-1,MAX-1));
      if (!OKBtn.isEnabled()){
        OKBtn.setEnabled(true);
        nextBtn.setText("下一幅");
      }
   }else{
       boolean correct=true;
       if (Math.abs(colorBar[0].getValue()-standardRed)>tolerance)
           correct=false;
       if (Math.abs(colorBar[1].getValue()-standardGreen)>tolerance)
           correct=false;
       if (Math.abs(colorBar[2].getValue()-standardBlue)>tolerance)
           correct=false;
       if (correct){
           JOptionPane.showMessageDialog(mainJframe,"颜色匹配正确");
       }else{
          JOptionPane.showMessageDialog(mainJframe,"颜色匹配错误");
       }
   }
  }//事件响应方法结束
}//内部类结束

public class HandleScrollbarChange implements AdjustmentListener{
   public void adjustmentValueChanged(AdjustmentEvent e){
```

```
        int red,green,blue;
        red=colorBar[0].getValue();
        green=colorBar[1].getValue();
        blue=colorBar[2].getValue();
        userArea.setBackground(new Color(red,green,blue));
    }
  }
}
```

程序运行时截图如 14.78 所示：

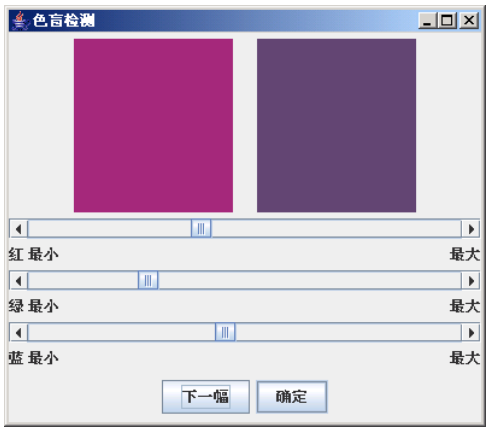

图 14.78　程序运行截图

这个程序功能比较简单，还可以增加一些功能：比如更多的帮助信息、可由用户选择精确程度（即通过的难易程度）、综合评分等等。请读者自行完成。

14.11　GUI 程序设计实例 2——小闹钟

在本节中，介绍一个简单的实用程序——闹钟的实现。它可以让根据用户的设置来报时。设置的时间既可以指定绝对时间，也可以指定间隔时间。提醒的方式也有两种：播放一段声音和跳出窗口到最前面。

图 14.79 是该程序的运行截图：

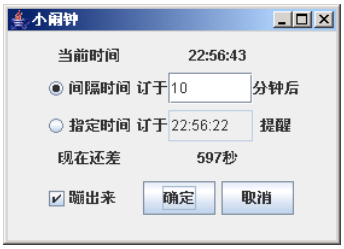

图 14.79　程序运行截图

14.11.1 程序界面的实现

这个程序的界面比上一个要复杂一些，没有那么有规律，这里采用 GridBagLayout 来布局。需要用到的组件和变量如下：

```
JLabel Label[];
ButtonGroup BtnGroup;                          //管理单选按钮
JRadioButton intervalRadio,specifyRadio;    //选择指定时间的方式
JTextField minuteText, timeText; //前者用来输入间隔时间，后者用来输入指定时间
JCheckBox chkBox;                              //用户是否选择让窗口弹出来
JButton OKBtn,CancleBtn;
Container con;
GridBagLayout gridBag;                         //布局要用到不规则的表格
JFrame mainJframe;
Date today;                                    //记录当前时间
```

程序界面设计所用到的代码全部放在构造方法中：

```
  public AlarmClock() {
     final String msg[]={"当前时间","","订于","分钟后","订于",
                                "提醒","现在还差"," 0 秒"};
     GridBagConstraints c = new GridBagConstraints();
     Label=new JLabel[msg.length];
     for(int i=0; i<Label.length; i++)
         Label[i]=new JLabel(msg[i],JLabel.CENTER);
     mainJframe=new JFrame("小闹钟");
     today=new Date();    //获取机器时间
     gridBag = new GridBagLayout();
     con=mainJframe.getContentPane();
     con.setLayout(gridBag);
     c.fill = GridBagConstraints.BOTH;
     c.insets=new Insets(0,0,5,0);
     //设置第一行的两个标签
     c.gridwidth=1;
     addComponents(Label[0],c);
     c.gridwidth = GridBagConstraints.REMAINDER;
     Label[1].setText(TimeToString(today));  //显示当前时间
     addComponents(Label[1],c);
     //设置第二行的组件
     c.gridwidth=1;
     intervalRadio=new JRadioButton("间隔时间",true);
     BtnGroup=new ButtonGroup();
     BtnGroup.add(intervalRadio);
     addComponents(intervalRadio,c);
     addComponents(Label[2],c);
     minuteText=new JTextField("10",6);  //设置默认间隔时间为 10 分钟
     addComponents(minuteText,c);
     c.gridwidth = GridBagConstraints.REMAINDER;
     addComponents(Label[3],c);
     //设置第三行的组件
     c.gridwidth=1;
     specifyRadio=new JRadioButton("指定时间",false);
```

```
    BtnGroup.add(specifyRadio);
    addComponents(specifyRadio,c);
    addComponents(Label[4],c);
    timeText=new JTextField(TimeToString(today),6);
    timeText.setEditable(false);
    addComponents(timeText,c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    addComponents(Label[5],c);
    //设置第四行的组件
    c.gridwidth=1;
    addComponents(Label[6],c);
    c.gridwidth = GridBagConstraints.REMAINDER;
    addComponents(Label[7],c);
    //设置第五行的组件
    c.gridwidth=1;
    chkBox=new JCheckBox("蹦出来",true);
    addComponents(chkBox,c);
    JPanel panel=new JPanel();
    panel.setLayout(new FlowLayout());
    OKBtn=new JButton("确定");
    CancleBtn=new JButton("取消");
    panel.add(OKBtn);
    panel.add(CancleBtn);
    c.gridwidth = GridBagConstraints.REMAINDER;
    addComponents(panel,c);

    mainJframe.setSize(280,200);
    mainJframe.setVisible(true);
    mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

14.11.2 时间的刷新代码

这个程序的实现主要有几个难点，第一个是如何让时间“动”起来。即在 Label[1]上显示的时间应该是机器时间，每一秒钟要变化一次。

为了实现这一功能，需要不断地读取并刷新时间。一种方法是在写一个循环，不断地去做这件事情 。但显然这是很浪费 CPU 资源的，所以这个循环肯定不能写在主线程中，只能写在一个优先级很低的线程中。

还有一种方法是利用 Timer 类。该类对象可以设定间隔时间，当到达指定的时间间隔后，自动去做指定的事情，所以编写比较简单。而这个“指定的事情”，就是一个 TimerTask 类，该类实际上也是一个线程，所以只要实现其中的 run()方法就可以了。下面就是利用 Timer 来实现刷新时间的功能的代码实例部分。

```
//这是实现 TimerTask 类。由于要直接访问 Label[1]，所以该类最好是一个内部类。
class Refresh extends TimerTask{
  public Refresh(){
    super();
  }
```

```
  public void run(){
    today=new Date();
    Label[1].setText(TimeToString(today));
  }
}
```

当然上面只是一个刷新了一个标签上的时间，实际上它还要做一些其他的事情，具体代码将在后面的程序中介绍。

然后在外部类的构造方法 AlarmClock()方法中添加下列语句：

```
Timer   myTimer=new Timer();
Refresh task=new Refresh();
myTimer.schedule(task,1000,1000); //每隔 1000ms 刷新一次时间
```

14.11.3 JRadioButton 的事件响应代码

程序允许用户在“间隔时间”和“指定时间”中选择一个选项，同时根据用户的选择将 minuteText 和 timeText 中的一个置为可写状态。这需要监听 JRadioButton 的 ActionEvent 事件。下面是代码：

```
class HandleBtn implements ActionListener{ //这是一个内部类
   public void actionPerformed(ActionEvent e){
     Object obj;
     obj=e.getSource();
     if(obj==intervalRadio){
        minuteText.setEditable(true);
        timeText.setEditable(false);
     }
     else if (obj==specifyRadio){
        minuteText.setEditable(false);
        timeText.setEditable(true);
     }
  }
}
```

14.11.4 “确定”按钮的事件响应代码

当用户按下“确定”按钮后，程序应该做下面几件事情：

- 判断用户输入的提醒时间或间隔时间是否合法。
- 如果合法，换算成间隔的秒数，并保存。
- 启动一个 Timer，开始倒计时。当计时完成时，给用户一个提醒。
- 将窗口最小化。

为了完成第二条，需要增加一个类变量 remainSeconds。为了完成第三条，需要增加一个 Timer，这样本程序中就有了两个 Timer，这样比较浪费资源。实际上可以修改 Refresh 类的 run()方法来新增加这个功能，这需要一些编程技巧。下面是该按钮的事件响应部分代码：

```
try{
       if (intervalRadio.isSelected()){ //指定了间隔时间
          //remainSeconds 是一个类的成员变量，它记录了剩余的秒数
          remainSeconds=Integer.parseInt(minuteText.getText(),10)*60;
       }else{ //指定了绝对时间
```

```
        Date tmpDate=new Date();
        int curTime,endTime;
        curTime=tmpDate.getHours()*3600
                + tmpDate.getMinutes()*60
                + tmpDate.getSeconds();
        //将用户输入的"时：分：秒"转换成为一个整型数
        endTime=parseTime(timeText.getText());
       //计算间隔时间并保存起来
        remainSeconds=endTime-curTime;
      }
      if (remainSeconds>0){
         startTime=true; //让 Timer 开始倒计时
         mainJframe.setState(JFrame.ICONIFIED); //窗口最小化
      }else{ //剩余时间小于 0，取消本次设置
        CancleBtn.doClick();
      }
}catch(NumberFormatException el){
      JOptionPane.showMessageDialog(mainJframe, "对不起，输入的时间间隔有错误");
}catch(ParseException el){
      JOptionPane.showMessageDialog(mainJframe, "对不起，指定的时间有错误");
}//try-catch 语句块结束
```

在上面的代码中，有很重要的一个方法：parseTime()，它可以将用户输入的“时:分:秒”数据转换成一个整型数，然后才得以计算与当前时间的间隔。这里必须要考虑到用户输入是可能出错的，作为一个实用程序，健壮性是一个重要的衡量标准。这里可以采用第 13.5.2 小节中介绍的 SimpleDateFormat 类的 parse()方法将字符串转换成为 Java 中的标准时间，同时捕获异常，判断用户的输入是否正确。

不过笔者采用了更为“低级”的办法，完全由自己编程来做判断。这么做程序要复杂很多，但对于提高编程能力也是一个很好的锻炼。下面来分析具体的代码：

```
//将用户输入的"时：分：秒"转换成为一个整型数，如果有错误，抛出一个异常
   private int parseTime(String str) throws ParseException {
      int i = 0, hour = 0, minute = 0, second = 0;
      int ch;
      //取出其中的小时
      ch = str.charAt(i);
      while (i < str.length() && ch != ':') {
         if (ch < '0' || ch > '9')  //不允许非数字的出现
            throw new ParseException(str, i);
         hour = hour * 10 + ch - '0';
         i++;
         if (i < str.length())
            ch = str.charAt(i);
         else
            throw new ParseException(str, i);
      }
      //越过中间的“：”
      i++;
      //取出其中的分钟
      ch = str.charAt(i);
      while (i < str.length() && ch != ':') {
         if (ch < '0' || ch > '9')
```

```
            throw new ParseException(str, i);
         minute = minute * 10 + ch - '0';
         i++;
         if (i < str.length())
            ch = str.charAt(i);
         else
            throw new ParseException(str, i);
      }
      //越过中间的“：”
      i++;
      //取出其中的秒数
      ch = str.charAt(i);
      while (i < str.length()) {
         if (ch < '0' || ch > '9')
            throw new ParseException(str, i);
         second = second * 10 + ch - '0';
         i++;
         if (i < str.length())
            ch = str.charAt(i);
      }
      //判断时分秒是否在合法的范围内
      if (hour > 23 || minute > 59 || second > 59)
         throw new ParseException(str, i);
      //求与 0:0:0 相间隔的秒数
      return hour * 3600 + minute * 60 + second;
   }
```

现在来改造原来的 TimerTask 的 run()方法：

```
public void run() {
   today = new Date();
   Label[1].setText(TimeToString(today));
   //下面是新加入的报时功能
   if (startTime) {
      Label[7].setText(remainSeconds + "秒");    //刷新剩余时间
      remainSeconds--;           //倒计时
      if (remainSeconds == 0) { //预定时间已到
         startTime = false;      //本次报时完成，把标志改掉
         if (chkBox.isSelected())
            mainJframe.setState(JFrame.NORMAL);  //让窗口弹出来
         try {                    //播放一段声音
            FileInputStream fileau = new FileInputStream("alert.wav");
            AudioStream as = new AudioStream(fileau);
            AudioPlayer.player.start(as);
         } catch (Exception e) {   }
      }
   }
}
```

14.11.5 “取消”按钮的事件响应代码

当用户按下“取消”按钮时，程序应当清除本次报时任务。这里只需要将标志 startTime 改成 false，将剩余时间置成小于等于 0 的任意数值就可以了。下面是代码段：

```
else if(obj==CancleBtn){
   startTime=false;
   remainSeconds=0;
   Label[7].setText(remainSeconds + "秒");
}
```

14.11.6 完整的程序

【例 14.43】小闹钟程序

//----------文件名 AlarmClock.java，程序编号 14.45----------

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.util.Date;
import java.util.Formatter;
import java.util.Timer;
import java.util.TimerTask;
import java.lang.Integer;
import java.text.*;
import sun.audio.*;
import java.io.*;
public class AlarmClock {
   JLabel Label[];
   ButtonGroup BtnGroup;
   JRadioButton intervalRadio, specifyRadio;
   JTextField minuteText, timeText;
   JCheckBox chkBox;
   JButton OKBtn, CancleBtn;
   Container con;
   GridBagLayout gridBag;
   JFrame mainJframe;
   Date today;
   Timer myTimer;
   int remainSeconds = 0;
   boolean startTime = false;
   private void addComponents(Component obj, GridBagConstraints c) {
      gridBag.setConstraints(obj, c);
      con.add(obj);
   }
   //将时间转换成为“时：分：秒”格式的字符串
   private String TimeToString(Date day) {
       Formatter fmt = new Formatter();
      fmt.format("%tT",day);
      return fmt.toString();
```

```
}
//将用户输入的"时：分：秒"转换成为一个整型数，如果有错误，抛出一个异常
private int parseTime(String str) throws ParseException {
    int i = 0, hour = 0, minute = 0, second = 0;
    int ch;
    ch = str.charAt(i);
    while (i < str.length() && ch != ':') {
        if (ch < '0' || ch > '9')
            throw new ParseException(str, i);
        hour = hour * 10 + ch - '0';
        i++;
        if (i < str.length())
            ch = str.charAt(i);
        else
            throw new ParseException(str, i);
    }
    i++;
    ch = str.charAt(i);
    while (i < str.length() && ch != ':') {
        if (ch < '0' || ch > '9')
            throw new ParseException(str, i);
        minute = minute * 10 + ch - '0';
        i++;
        if (i < str.length())
            ch = str.charAt(i);
        else
            throw new ParseException(str, i);
    }
    i++;
    ch = str.charAt(i);
    while (i < str.length()) {
        if (ch < '0' || ch > '9')
            throw new ParseException(str, i);
        second = second * 10 + ch - '0';
        i++;
        if (i < str.length())
            ch = str.charAt(i);
    }
    if (hour > 23 || minute > 59 || second > 59)
        throw new ParseException(str, i);
    return hour * 3600 + minute * 60 + second;
}

public AlarmClock() {
    final String msg[] = { "当前时间", "", "订于", "分钟后", "订于",
                           "提醒", "现在还差"," 0 秒"
                         };
    Refresh task;
    HandleBtn handle = new HandleBtn();
    GridBagConstraints c = new GridBagConstraints();
```

```
Label = new JLabel[msg.length];
for (int i = 0; i < Label.length; i++)
   Label[i] = new JLabel(msg[i], JLabel.CENTER);
mainJframe = new JFrame("小闹钟");
today = new Date();
gridBag = new GridBagLayout();
con = mainJframe.getContentPane();
con.setLayout(gridBag);
c.fill = GridBagConstraints.BOTH;

c.gridwidth = 1;
c.insets = new Insets(0, 0, 5, 0);
addComponents(Label[0], c);
c.gridwidth = GridBagConstraints.REMAINDER;
Label[1].setText(TimeToString(today));
addComponents(Label[1], c);

c.gridwidth = 1;
intervalRadio = new JRadioButton("间隔时间", true);
intervalRadio.addActionListener(handle);
BtnGroup = new ButtonGroup();
BtnGroup.add(intervalRadio);
addComponents(intervalRadio, c);
addComponents(Label[2], c);
minuteText = new JTextField("10", 6);
addComponents(minuteText, c);
c.gridwidth = GridBagConstraints.REMAINDER;
addComponents(Label[3], c);

c.gridwidth = 1;
specifyRadio = new JRadioButton("指定时间", false);
specifyRadio.addActionListener(handle);
BtnGroup.add(specifyRadio);
addComponents(specifyRadio, c);
addComponents(Label[4], c);
timeText = new JTextField(TimeToString(today), 6);
timeText.setEditable(false);
addComponents(timeText, c);
c.gridwidth = GridBagConstraints.REMAINDER;
addComponents(Label[5], c);

c.gridwidth = 1;
addComponents(Label[6], c);
c.gridwidth = GridBagConstraints.REMAINDER;
addComponents(Label[7], c);

c.gridwidth = 1;
chkBox = new JCheckBox("蹦出来", true);
addComponents(chkBox, c);
JPanel panel = new JPanel();
```

```
        panel.setLayout(new FlowLayout());
        OKBtn = new JButton("确定");
        OKBtn.addActionListener(handle);
        CancleBtn = new JButton("取消");
        CancleBtn.addActionListener(handle);
        panel.add(OKBtn);
        panel.add(CancleBtn);
        c.gridwidth = GridBagConstraints.REMAINDER;
        addComponents(panel, c);

        mainJframe.setSize(280, 200);
        mainJframe.setVisible(true);
        mainJframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        myTimer = new Timer();
        task = new Refresh();
        myTimer.schedule(task, 1000, 1000);
    }

    class Refresh extends TimerTask {
        public Refresh() {
            super();
        }

        public void run() {
            today = new Date();
            Label[1].setText(TimeToString(today));
            if (startTime) {
                Label[7].setText(remainSeconds + "秒");
                remainSeconds--;
                if (remainSeconds == 0) {
                    startTime = false;
                    if (chkBox.isSelected())
                        mainJframe.setState(JFrame.NORMAL);
                    try {
                        FileInputStream fileau = new FileInputStream("alert.wav");
                        AudioStream as = new AudioStream(fileau);
                        AudioPlayer.player.start(as);
                    } catch (Exception e) { }
                }//内层 if 结束
            }//外层 if 结束
        }//run 方法结束
    }//内部类 Refresh 结束

    class HandleBtn implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            Object obj;
            obj = e.getSource();
            if (obj == intervalRadio) {
                minuteText.setEditable(true);
```

```
            timeText.setEditable(false);
         } else if (obj == specifyRadio) {
            minuteText.setEditable(false);
            timeText.setEditable(true);
         } else if (obj == OKBtn) {
            try {
               if (intervalRadio.isSelected()) {
                  remainSeconds = Integer.parseInt(minuteText.getText(),10)*60;
               } else {
                  Date tmpDate = new Date();
                  int curTime, endTime;
                  curTime = tmpDate.getHours() * 3600
                        + tmpDate.getMinutes() * 60
                        + tmpDate.getSeconds();
                  endTime = parseTime(timeText.getText());
                  remainSeconds = endTime - curTime;
               }
               if (remainSeconds > 0) {
                  startTime = true; // 让 Timer 开始倒计时
                  mainJframe.setState(JFrame.ICONIFIED); // 窗口最小化
               }
            }catch(NumberFormatException el){
              JOptionPane.showMessageDialog(mainJframe,"对不起，输入的时间间隔有错误");
            }catch(ParseException el){
              JOptionPane.showMessageDialog(mainJframe, "对不起，指定的时间有错误");
            }
         }else if(obj==CancleBtn){
            startTime=false;
            remainSeconds=0;
            Label[7].setText(remainSeconds + "秒");
         }
      }
   }

   public static void main(String[] args) {
      new AlarmClock();
   }
}
```

这个程序已经基本上完成了，但还有一些可以改进的地方。比如：窗口最大化时变得不大美观，显示的时间格式也不够标准。更为重要的是：由用户输入时间所用的组件是 JTextField，对用户的输入没有任何控制，所以很容易出错，编写的差错代码很长。作为学习，这么做是可以的，但如果是实际的编程，则应该选用对用户输入有限制的 JFormattedTextField 组件，这样可以提高开发的效率。还有，本程序一次只能定一个时间，还可以改进成多个定时闹钟。

14.12　GUI 程序设计实例 3——字体选择对话框

JDK1.5 中没有提供字体选择对话框，而这又是很常用的对话框之一。本节中，自己编程来实现一

个字体选择对话框。

本例仿照 Windows 提供的标准字体选择对话框来做界面。为了简单，笔者省略了对话框中的字体示例。程序运行界面如图 14.80 所示：

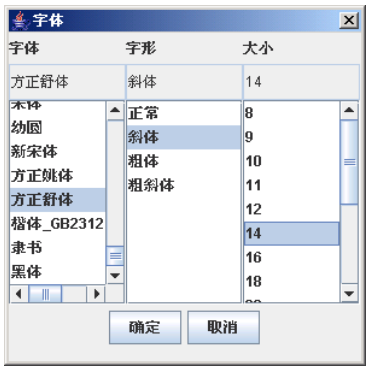

图 14.80　字体对话框运行截图

我们希望该对话框的使用能象系统提供的 JFileChooser 一样，调用者通过下面三步来获取用户选择的字体：

- 创建对话框对象；
- 调用对话框类提供的方法，显示该对话框。并能在对话框关闭后，知道用户是按下“确定”还是“取消”按钮；
- 如果用户按下确定，通过调用对话框类提供的方法，获取用户选择的字体。

这其中有几个难点：

- 如何获取系统中的字体名称，显示在一个列表框中？
- 当用户按下“确定”或“取消”按钮后，对话框必须关闭，而用户的动作是如何记录下来的？
- 如何将上图中的字体、字形、大小组合成一个 Font 对象返回给调用者？

14.12.1　界面的实现

这个对话框的界面很有规律，它是由 4 个 panel 构成，其中前 3 个是用表格布局，一行三列。分别放置标签、文本框和列表框。最后一个 panel 采用流失布局，存放两个按钮。4 个 panel 用 BoxLayout 布局按照从上到下的方式存放窗口的中间容器上面就可以了。

另外，由于这是一个对话框，所以它必须由 JDialog 派生出来，不过这也决定了它不能独立存在，必须要一个附属的窗体。

界面的程序代码如下：

//----------文件名 fontDialog.java，程序编号 14.46----------

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

public class fontDialog extends JDialog
        implements ActionListener,ListSelectionListener{
   public static final int Cancle=0;
   public static final int OK=1;
```

```
 public static final String [] style={"正常","斜体","粗体","粗斜体"};
 public static final String [] size={"8","9","10","11","12","14","16",
                      "18","20","22","24","26","28","36","48","72"};
//这个对象记录了用户选择的字体信息，准备用来返回给调用者
 private Font userFont=null;
 private int userSelect=Cancle;  //标记用户按下了哪个按钮
 private JFrame parent=null;     //对话框所属的窗体
 private Container con;
 private JScrollPane nameSPane,styleSPane,sizeSPane;
 private JPanel panel[];
 private JLabel nameLbl,styleLbl,sizeLbl;
 private JTextField nameText,styleText,sizeText;
 private JList nameList,styleList,sizeList;
 private JButton OKBtn,cancleBtn;
 //无参数的构造方法
 public fontDialog() {
    this(null);
 }
 //本构造方法只是构造出对话框，并没有显示它
 public fontDialog(JFrame owner){ //此参数是对话框所属的窗口
   super(owner,true);              //本对话框必须以 model 形态显示
   parent=owner;
   setTitle("字体");
   con=getContentPane();
   //设置中间容器的布局，从上到下排列
   BoxLayout box=new BoxLayout(con,BoxLayout.Y_AXIS);
   con.setLayout(box);
   panel=new JPanel[4];
   for(int i=0;i<3;i++){
      panel[i]=new JPanel();
      panel[i].setLayout(new GridLayout(1,3));
   }
   panel[3]=new JPanel();
   panel[3].setLayout(new FlowLayout());
   //创建第一行的三个标签，存放在第一个 panel 中
   nameLbl=new JLabel("字体");
   styleLbl=new JLabel("字形");
   sizeLbl=new JLabel("大小");
   panel[0].add(nameLbl);
   panel[0].add(styleLbl);
   panel[0].add(sizeLbl);
   //创建第二行的三个文本框存放在第二个 panel 中
   nameText=new JTextField("宋体");  //默认为宋体
   nameText.setColumns(5);
   //将文本框设置为只读，降低用户输入出错的可能
   nameText.setEditable(false);
   styleText=new JTextField("正常");
   styleText.setColumns(5);
   styleText.setEditable(false);
   sizeText=new JTextField("12");
```

```
    sizeText.setColumns(5);
    sizeText.setEditable(false);
    panel[1].add(nameText);
    panel[1].add(styleText);
    panel[1].add(sizeText);
    //下面创建第三行的三个列表框
    //首先获取系统所安装的字体名称，以便显示在第一个列表框中
    GraphicsEnvironment eq = GraphicsEnvironment.
                                    getLocalGraphicsEnvironment();
    String[] availableFonts= eq.getAvailableFontFamilyNames();
    nameList=new JList(availableFonts);
    nameList.addListSelectionListener(this); //安装事件监听器
    nameSPane=new JScrollPane(nameList);
    styleList=new JList(style);
    styleList.addListSelectionListener(this);//安装事件监听器
    styleSPane=new JScrollPane(styleList);
    sizeList=new JList(size);
    sizeList.addListSelectionListener(this); //安装事件监听器
    sizeSPane=new JScrollPane(sizeList);
    panel[2].add(nameSPane);
    panel[2].add(styleSPane);
    panel[2].add(sizeSPane);
    //创建两个按钮，放置在第四个 panel 中
    OKBtn=new JButton("确定");
    OKBtn.addActionListener(this);
    cancleBtn=new JButton("取消");
    cancleBtn.addActionListener(this);
    panel[3].add(OKBtn);
    panel[3].add(cancleBtn);
    //把四个 panel 都加入到中间容器中
    for(int i=0;i<4;i++)
       con.add(panel[i]);
  }
}
```

14.12.2 监听 ListSelectionEvent 事件

当用户选择字体、字形或是大小时，对应的文本框中会显示用户的选择，这需要监听 ListSelectionEvent 事件，下面实现了 ListSelectionListener 接口中的唯一方法：

```
//本事件有三个事件源，所以需要区分来自哪个列表
 public void valueChanged(ListSelectionEvent e){
   if (e.getSource()==nameList)
     nameText.setText((String)nameList.getSelectedValue());
   if (e.getSource()==styleList)
     styleText.setText((String)styleList.getSelectedValue());
   if (e.getSource()==sizeList)
     sizeText.setText((String)sizeList.getSelectedValue());
}
```

14.12.3 按钮响应事件

对话框中有两个按钮，共用下面同一个监听方法：

```
public void actionPerformed(ActionEvent e){
   int styleIndex=Font.PLAIN,fontSize;
    //按下了“确定”按钮
   if(e.getSource()==OKBtn){
     //将用户选择的字形转换成 Font 类所定义的整型量
      if(styleText.getText().equals("正常"))
         styleIndex=Font.PLAIN;
      if(styleText.getText().equals("斜体"))
         styleIndex=Font.ITALIC;
      if(styleText.getText().equals("粗体"))
         styleIndex=Font.BOLD;
      if(styleText.getText().equals("粗斜体"))
         styleIndex=Font.BOLD | Font.ITALIC;
     //获取用户选择的字体大小
      fontSize=Integer.parseInt(sizeText.getText());
     //根据上述信息生成一个 Font 对象
      userFont=new Font(nameText.getText(),styleIndex,fontSize);
      userSelect=OK;      //标记用户做出的选择
      setVisible(false); //关闭对话框
   }
   //按下了取消按钮
   else{
      userSelect=Cancle;
      setVisible(false);
   }
}
```

14.12.4 对话框的显示

由于创建对话框时并没有显示它，所以还需要写一个方法，显示该对话框。而且当对话框关闭后应该要返回一个值给调用者，通知它用户最后单击的是“确定”还是“取消”按钮。

```
public int showFontDialog(){
   setSize(300,300);
   int x,y;
   //计算要显示的位置
   if (parent!=null){
      x=parent.getX()+30;
      y=parent.getY()+30;
   }else{
      x=150;
      y=100;
   }
   setLocation(new Point(x,y));  //确定显示位置
   setVisible(true);             //显示对话框
   //由于该对话框是以 modal 形式显示，所以直到该对话框关闭，下面这条语句都不会执行
```

```
    return userSelect;
}
```

14.12.5 返回用户选择的字体

如果用户按下的是“确定”键，调用者就会想知道用户到底选择了哪种字体，所以需要提供一个方法给调用者。这个方法很简单，如下：

```
public Font getFont(){
    return userFont;
}
```

14.12.6 如何使用字体选择对话框

到这里，对话框所需要的全部方法都已经编写完毕了，读者应该很容易地把前面各个方法拼装在一起（注意，所有的方法都是属于 fontDialog 类的）。由于篇幅的原因，这里不再提供拼装在一起的完整的源程序。

如果需要使用该对话框，应该像下面这个样子：

```
void doChangeFont(){
    if (myFontDialog==null)              //myFontDialog 是一个类的实例成员变量
        myFontDialog=new fontDialog(this);//显示对话框
    if(myFontDialog.showFontDialog()==fontDialog.OK)
        text.setFont(myFontDialog.getFont());//获得对话框返回的字体
}
```

一共只有三步：

（1）创建对象，需要传递一个参数给它，这个参数是本对话框所属的窗口。

（2）用 showFontDialog()显示对话框，同时获得它的返回值确定用户的按键。

（3）调用 getFont()获取用户选择的字体。

本对话框的基本功能已经实现，但还有一些需要完善的地方

- 应该可以由调用者来指定默认的字体、字形和大小。
- 三个文本框应该是可以编辑的。
- 字体大小应该要列出“四号字”之类的中文。
- 当用户选择了某种字体后，应该要显示一个示例出来。

以上这些功能，请读者自己实现。关于本对话框的使用，在下一节的记事本啊中，还会看到。

14.13 GUI 程序设计实例 4——记事本

这一节介绍一个仿照 Windows 的记事本实现的文本编辑器。该程序的界面实现在第 14.11.8 节中介绍过了，只需要做小小的修改。

14.13.1 增加弹出式菜单

这里只要为文本区加上一个弹出菜单，代码如下：

```
public void createPopupMenu(){
```

```
  popMenu=new JPopupMenu();
  popMenu.add("撤消");
  popMenu.addSeparator();
  popMenu.add("剪切");
  popMenu.add("复制");
  popMenu.add("粘贴");
  popMenu.addSeparator();
  popMenu.add("全选");
}
```

然后用 setComponentPopupMenu(popMenu)为文本区加上这个菜单就可以使用了。

现在剩下的主要工作就是为主菜单项逐一编写事件响应代码，然后先用 MenuItem.addActionListener(this)方法添加监听器，再在 actionPerformed(ActionEvent e)方法中增加这样的语句：

```
if (e.getSource==某个菜单项名)  调用相应的方法;
```

后面不再重复说明这一步骤。当然，当实现或继承某些监听器或是适配器类时，需要引入相应的包，也不再说明。

添加右键菜单之后的程序界面如图 14.81 所示：

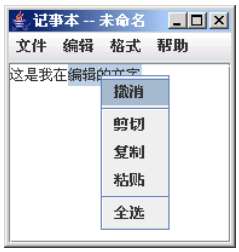

图 14.81　弹出式菜单截图

14.13.2　“退出”菜单的响应代码

当用户单击“退出”时，程序应该做这么几件事情：

- 若文件在最后一次改动后已经存盘，直接退出。
- 若文件发生了改动且未存盘，则弹出一个对话框，询问是否要存盘。
- 若用户选择存盘，则要看文件是新文件还是曾经存过盘的老文件，并执行相应的操作。
- 若用户选择不存盘，则直接退出。
- 若用户选择取消，则返回程序主界面。

为了完成上述工作，需要增加几个成员变量：

- boolean changed=false;　　//标记文件内容是否已被修改
- boolean haveName=false; //标记这是否为新文件
- File　　file;　　//记录文件的有关信息

下面是完成上述功能所需的代码：

```
void doExit(){
  int select;
  if (!changed)   //如果文件没有改变，直接推出
     System.exit(0);
  else{
     select=JOptionPane.showConfirmDialog(this, "文件修改后尚未存盘，要保存吗？");
```

```
    switch (select){  //根据用户的选择来执行相应的操作
      case JOptionPane.YES_OPTION:
              select=doSave();  //执行存盘操作，并获得操作是否成功的标记
              if (select==1)    System.exit(0);  //成功存盘，退出
              break;
      case JOptionPane.NO_OPTION:
              System.exit(0);  //不存盘，直接退出
              break;
      case JOptionPane.CANCEL_OPTION:
              break;           //选择了取消，返回主界面
    }
  }
}
```

14.13.3 覆盖 JFrame 的 processWindowEvent 方法

当用户单击“退出”菜单时，会调用上面这段程序。但是，由于 JFrame 是自动响应窗口关闭事件的，用户有可能通过系统的关闭按钮来直接退出，那么此段代码就不会起作用。因此，程序需要覆盖 JFrame 的窗口关闭方法 processWindowEvent（注意: 不是监听 windowClosing 事件），在此方法中调用 doExit() 方法。

在回顾 14.11.8 节的代码，mainJFrame 是一个成员变量，这是无法覆盖它的成员方法的。所以需要去掉这个成员变量，然后将外部类声明成 JFrame 的子类，再覆盖 processWindowEvent 方法，具体代码如下：

```
//当用户按下窗口的关闭按钮时，会自动调用此方法
protected void processWindowEvent(WindowEvent e){
   if (e.getID() == WindowEvent.WINDOW_CLOSING)
       doExit();
}
```

有了上面两个方法中，无论是选择菜单中的“退出”还是单击系统的关闭按钮，都会出现图 14.82 的提示：

图 14.82　退出时的提示对话框

14.13.4 监听 JTextArea 的 DocumentEvent 事件

在退出菜单的事件响应代码中，需要用到一个变量：changed，如果文件内容发生了变化，该值应该为 true；如果已经存盘或是新建文件，该值应该为 false。

为了实时监测文本内容的变化，需要监听 JTextArea 的 DocumentEvent 事件，这需要添加 DocumentListener 监听器，由于该监听器是一个接口，所以需要实现该接口中的所有方法，下面是代码:

```
//监听文本内容的改变事件
public void changedUpdate(DocumentEvent e){
```

```
    //不需要动作
}
public void insertUpdate(DocumentEvent e){
    changed=true;
}
public void removeUpdate(DocumentEvent e){
    changed=true;
}
```

为 JTextArea 添加该事件监听器时需要加在它的模型上，所以需要这样用：text.getDocument().addDocumentListener(this)。

14.13.5 “另存为...”菜单的响应代码

当用户单击“另存为...”菜单时，程序应完成下述工作：

- 打开一个文件保存对话框，让用户选择存储路径和文件名。
- 如果用户填写的文件已经存在，弹出一个警告框，让用户确认是否覆盖。
- 以用户填写的文件名来保存，更改相关的变量。

为实现这些功能，需要用到 JFileChooser 类和 ExampleFileFilter 类。后者不是 Java 标准类库中的类，它在 jdk 的安装路径下的：demo\jfc\FileChooserDemo 目录下，包含在 FileChooserDemo.jar 文件中。为了让 eclipse 能引入它，需要做相应的配置。当然也可以直接用编译命令的-classpath 来指定查找的路径。或者，也可以将 src\FileChooserDemo.java 文件拷贝到你编辑的源程序所在的目录下面。

由于这里需要处理用户的输入，需要比较多的代码来提高程序的健壮性。下面是具体代码：

```
//用"另存为"对话框保存文件。保存成功返回 1，否则返回 0
int doSaveAs(){
    FileOutputStream fout;
    byte content[];
    int flag=0;
    File tmpfile=null;
    ExampleFileFilter filter = new ExampleFileFilter();
    JFileChooser chooser;
    //设置保存文件对话框中的文件属性过滤器
    filter.addExtension("txt");
    filter.setDescription("文本文件");
    //设置要保存的文件的目录与当前编辑文件所在目录一样
    if (file!=null)
        chooser = new JFileChooser(file.getPath());
    else
        chooser = new JFileChooser();
    //设置保存对话框中的“文件类型”
    chooser.setFileFilter(filter);
    //打开文件保存对话框
    flag = chooser.showSaveDialog(this);
    if(flag == JFileChooser.APPROVE_OPTION) { //如果用户按下了对话框中的“保存”
        //获取用户指定的文件名、目录等信息
        tmpfile=chooser.getSelectedFile();
        //开始做容错处理
         if (tmpfile.exists()){
```

```
            if (JOptionPane.showConfirmDialog(this,"文件已经存在，是否覆盖？",
               "警告",JOptionPane.YES_NO_OPTION)==JOptionPane.YES_OPTION){
                  flag=1;
            }else{
                flag=0;
            }
        }else{
           flag=1;
        }
    }else{  //用户没有选择"保存"
       flag=0;
    }//对话框处理完毕
    //真正开始保存文件
    if (flag==1){ //用户已经确定要以指定名称保存文件
       try{
           //用文件输出流来保存
           fout=new FileOutputStream(tmpfile);
           //将文本内容从 String 转为 byte 类型的数组
           content=text.getText().getBytes();
           fout.write(content);
           fout.close();
           flag = 1; //文件保存成功
       }catch(FileNotFoundException e){
           JOptionPane.showMessageDialog(this,"指定的文件名称或属性有问题！");
           flag = 0;
       }catch(IOException e){
          JOptionPane.showMessageDialog(this,"无法写文件，请检查文件是否被锁定");
          flag = 0;
       }
    }
    //文件保存完毕，开始做其他的处理
    if (flag==1){     //文件保存成功，修改相关变量
       changed=false;
       haveName=true;
       file=tmpfile; //记录用户选择的文件名
       this.setTitle("记事本 -- "+file.getName());
    }
    return flag;    //返回保存标记
  }
}
```

这段代码看起来比较烦琐，这是由于多数代码要用于处理意外情况。这正是一个应用软件和以前编写的示例程序的区别。

图 14.83 是单击“另存为”菜单后的截图：

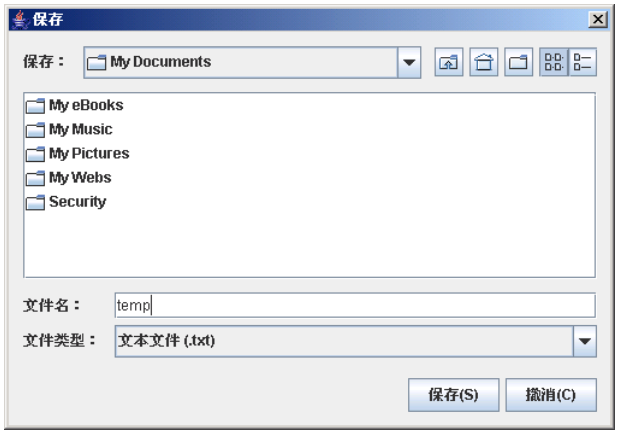

图 14.83　文件另存为截图

14.13.6　“保存”菜单的响应代码

当用户单击“保存”菜单时，程序应完成下述工作：

- 判断文件是否为新文件，如果是新文件，调用 doSaveAs()来保存。
- 否则，看上次保存之后是否修改过，如果修改过，直接以原文件名保存，否则调用 doSaveAs()。
- 如果没有修改过，不做任何动作。

这里大部分工作已经由 doSaveAs()完成，所以程序相对简单，下面是代码：

```
//保存用户编辑的文件，保存成功返回 1，否则返回 0
int doSave(){
   FileOutputStream fout;
   byte content[];
   int flag;
   if (!haveName){  //如果从来没有保存过
      flag = doSaveAs(); //调用另存为…
   }else if(changed){
      try{
         fout=new FileOutputStream(file);
         content=text.getText().getBytes();
         fout.write(content);
         fout.close();
         changed=false;
         flag = 1;
      }catch(FileNotFoundException e){
          JOptionPane.showMessageDialog(this,"指定的文件名称或属性有问题！");
          flag = 0;
      }catch(IOException e){
          JOptionPane.showMessageDialog(this,"无法写文件，请检查文件是否被锁定");
          flag = 0;
      }
   }else{
      flag =1;
   }
   return flag;
```

```
}
```

14.13.7 “新建”菜单的响应代码

当用户单击“新建”菜单时，程序应完成下述工作：

- 判断文件内容是否已经修改过，是则询问用户是否保存。根据用户选择做出相应的动作。
- 除非用户选择“撤消”或保存不成功，否则清除掉当前文本区的内容。
- 设置相应的变量值。

下面是具体代码：

```
void doNewFile(){
   int select,flag=0;
   if (changed){
      select=JOptionPane.showConfirmDialog(this, "文件修改后尚未存盘，要保存吗？");
      switch (select){
        case JOptionPane.YES_OPTION:
             flag=doSave();
             break;
        case JOptionPane.NO_OPTION:
             flag=1;
             break;
        default :
             flag=0;
             break;
      }
   }else{
        flag=1;
   }
   if(flag==1){
      changed=false;
      haveName=false;
      setTitle("记事本 -- 未命名");
      text.setText(""); //清空文本区
   }
}
```

14.13.8 “打开...”菜单的响应代码

当用户单击“打开...”菜单时，程序应完成下述工作：

- 判断文件内容是否已经修改过，是则询问用户是否保存。根据用户选择做出相应的动作。
- 除非用户选择“撤消”或保存不成功，否则弹出一个文件选择对话框由用户选择要打开的文件。
- 读入用户选择的文件内容，复制给 text。
- 设置相应的变量值。

下面是具体的代码：

```
//打开一个已经存在的文件
void doOpen(){
  int select,flag;
```

```
File tmpfile=null;
ExampleFileFilter filter;
JFileChooser chooser;
FileInputStream fin;
byte buf[];
//先做存盘处理
if (changed){
    select=JOptionPane.showConfirmDialog(this, "文件修改后尚未存盘，要保存吗？");
    switch (select){
      case JOptionPane.YES_OPTION:
          flag=doSave();
          break;
      case JOptionPane.NO_OPTION:
          flag=1;
          break;
      default:
          flag=0;
          break;
    }
}else{
   flag = 1;
}
//存盘成功，开始准备打开文件
if(flag==1){
  changed = false;
  //设置文件打开对话框的文件过滤器
  filter = new ExampleFileFilter();
  filter.addExtension("txt");
  filter.setDescription("文本文件");
  //设置打开时默认路径
  if (file!=null)
     chooser = new JFileChooser(file.getPath());
  else
     chooser = new JFileChooser();
  chooser.setFileFilter(filter);
  select = chooser.showOpenDialog(this);
  if(select == JFileChooser.APPROVE_OPTION) {
     tmpfile=chooser.getSelectedFile();
     try{
        //用文件输入流读入文件内容
        fin=new FileInputStream(tmpfile);
        buf=new byte[(int)tmpfile.length()];
        fin.read(buf);
        fin.close();
        //将文件内容显示在文本区中
        text.setText(new String(buf));
        changed=false;
        haveName=true;
        file=tmpfile;
        setTitle("记事本 -- "+file.getName());
```

```
        }catch(FileNotFoundException e){
           JOptionPane.showMessageDialog(this, "指定的文件名称或属性有问题！");
        }catch(IOException e){
           JOptionPane.showMessageDialog(this, "无法读文件，请检查文件是否被锁定");
        }
     }
  }
}
```

图 14.84 是单击“打开..”菜单之后的程序截图：

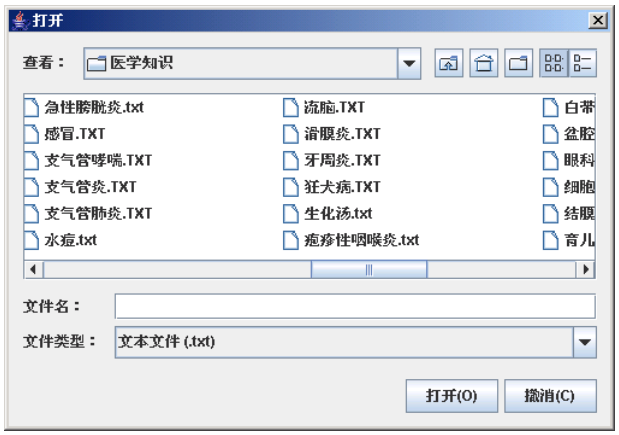

图 14.84　打开文件截图

14.13.9　“打印..”菜单的响应代码

此菜单涉及到打印功能的实现。Java 提供的打印功能一直都比较弱。实际上最初的 JDK 根本不支持打印，直到 JDK1.1 才引入了很少量的打印支持，到 JDK1.4 才逐步完善（当然，和 MFC、Delphi 的功能还不能比）。

Java 的打印 API 主要存在于 java.awt.print 包中。而 JDK1.4 新增的类则主要存在于 javax.print 包及其相应的子包 javax.print.event 和 javax.print.attribute 中。其中 javax.print 包中主要包含打印服务的相关类，而 javax.print.event 则包含打印事件的相关定义，javax.print.attribute 则包括打印服务的可用属性列表等。

要完成打印任务，需要做以下步骤：

- ❑ 定位一台打印机
- ❑ 指定输出格式（即打印内容的格式）
- ❑ 设置打印属性
- ❑ 设置内容
- ❑ 打印

这些步骤中的第三步，一般应由用户通过一个打印对话框来指定。要显示一个打印设置的对话框，在 JDK1.3 中，可以使用 Printable 的打印对话框；在 JDK1.4 以后的版本中，可以使用 ServiceUI 的打印对话框。下面是打印的代码：

```
void doPrint(){
   try{
     //构建打印请求属性集
```

```
     PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
    //设置打印格式，也许由于 JDK1.4 的不足，这里只能选择自动检测
    //如果选择文本格式，可能会由于打印机不支持而失败
    DocFlavor flavor = DocFlavor.BYTE_ARRAY.AUTOSENSE;
    //查找所有的可用打印服务
    PrintService printService[] = PrintServiceLookup.lookupPrintServices(
                                        flavor, pras);
    //定位默认的打印机
    PrintService defaultService = PrintServiceLookup.lookupDefaultPrintService();
   //显示打印对话框
    PrintService service = null;
    service = ServiceUI.printDialog(null, 100, 100,
                    printService, defaultService, flavor, pras);
   //准备开始打印
    if (service!=null){
       //创建打印作业
       DocPrintJob job = service.createPrintJob();
       DocAttributeSet das = new HashDocAttributeSet();
      //建立打印文件格式
       Doc doc = new SimpleDoc(text.getText().getBytes(), flavor, das);
       job.print(doc, pras); //进行文件的打印
    }
  }catch(Exception e){
     JOptionPane.showMessageDialog(this,"打印任务无法完成");
  }
}
```

图 14.85 是单击“打印..”对话框之后的运行截图：

图 14.85　打印对话框运行截图

到这里，基本上这个文本编辑器就可以使用了，只是不太方便，下面继续为它添加辅助功能。

14.13.10　“剪切”菜单的响应代码

一般情况下，如果用户没有选择文本，则剪切和复制菜单应该是不可用。而一旦用户做出了选择，

这两项菜单就应该变成可用状态。

为了完成这一功能，应该为 text 增加两个监听器：键盘事件监听器 KeyListener 和鼠标事件监听器 MouseListener。而且分别实现其中的 keyPressed(KeyEvent e)和 mouseReleased(MouseEvent e)方法就可以了。但这两个监听器需要实现的方法比较多，为了简便，实际上只要分别继承适配器类 KeyAdapter 和 MouseAdapter，并写成内部类的形式。下面是这两个适配器类的子类：

```
//监听鼠标事件
class handleMouse extends MouseAdapter{
    public void mouseReleased(MouseEvent e) {
        chkText();
    }
}
//监听键盘事件
class handleKey extends KeyAdapter{
    public void keyPressed(KeyEvent e) {
        chkText();
    }
}
//根据用户选择文本的情况，修改菜单的状态
void chkText(){
   if(text.getSelectedText()==null){
        cutItem.setEnabled(false);
        copyItem.setEnabled(false);
   }else{
        cutItem.setEnabled(true);
        copyItem.setEnabled(true);
   }
}
```

然后分别用：text.addKeyListener(new handleKey())和 text.addMouseListener(new handleMouse())为 text 安装监听器。

“剪切”菜单本身的响应代码是很简单的，直接调用 JTextArea 自身的方法就行：

```
//将用户选择的文本剪切到剪贴板
void doCut(){
    text.cut();
}
```

14.13.11 “复制”菜单的响应代码

有了前面的准备工作，这个响应代码很简单：

```
//将用户选择的文本复制到剪贴板
void doCut(){
    text.copy();
}
```

14.13.12 “粘贴”菜单的响应代码

有了前面的准备工作，这个响应代码也很简单：

```
//将用户选择的文本复制到剪贴板
```

```
void doCut(){
    text.paste();
}
```

这段代码运行起来并没有什么问题，但在显示上却不是很符合人们的习惯：当剪贴板为空或是内容不是文本时，“粘贴”菜单也是可用状态，只是无法将数据粘贴到文本区中。要解决这个问题，需要注册剪贴板监听器，监视剪贴板中内容的变化。这个问题，请读者自行解决。

14.13.13 “全选”菜单的响应代码

这个直接调用文本区自己的方法就行：

```
void doSelectAll(){
   text.selectAll();
}
```

14.13.14 “时间/日期”菜单的响应代码

这个需要获得系统的时间和日期，将其转换成字符串，插入到光标所在的位置：

```
//插入当前日期和时间
void doDateTime(){
   SimpleDateFormat sdf = new SimpleDateFormat("HH:mm yyyy-MM-dd");
   text.append(sdf.format(new Date()));
}
```

14.13.15 “自动换行”菜单的响应代码

本菜单是一个带选择标记的菜单项，需要根据当前状态做出相应的动作。

```
void doWrap(){
   if(wrapItem.getState()) {
      text.setLineWrap(true);
   }else {
      text.setLineWrap(false);
   }
}
```

14.13.16 “查找..”菜单的响应代码

JDK1.5 没有提供查找对话框，所以需要自己写一个对话框类，这个对话框是非模态对话框，所以控制起来比字体选择对话框更复杂一些。下面的代码实现该对话框的界面：

//----------文件名 findDialog.java，程序编号 14.47----------

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

public class findDialog extends JDialog implements ActionListener{
   Container con;
   JPanel panel1,panel2;
```

```
 JTextArea text;
 JLabel label1;
 JTextField findEdit;
 JCheckBox checkBox;
 JRadioButton upBtn,downBtn;
 ButtonGroup dirBtnGroup;
 JButton OKBtn,CancleBtn;
 int start;  //标志开始查找的位置
 //第一个参数是对话框的父窗口，第二个参数是要查找内容的文本区
 public findDialog(JFrame owner, JTextArea Jtext) {
   super(owner,false);    //以非模态方式创建对话框
   start=0;
   text=Jtext;
   panel1=new JPanel();
   panel1.setLayout(new FlowLayout());
   panel2=new JPanel();
   panel2.setLayout(new FlowLayout());
   label1=new JLabel("查找内容");
   findEdit=new JTextField(12);
   OKBtn=new JButton("查找下一个");
   OKBtn.addActionListener(this);
   panel1.add(label1);
   panel1.add(findEdit);
   panel1.add(OKBtn);
   checkBox=new JCheckBox("区分大小写");
   checkBox.setSelected(true);
   upBtn=new JRadioButton("向上");
   downBtn=new JRadioButton("向下",true);
   dirBtnGroup=new ButtonGroup();
   dirBtnGroup.add(upBtn);
   dirBtnGroup.add(downBtn);
   CancleBtn=new JButton("取消");
   CancleBtn.addActionListener(this);
   panel2.add(checkBox);
   panel2.add(upBtn);
   panel2.add(downBtn);
   panel2.add(CancleBtn);
   con=getContentPane();
   con.setLayout(new FlowLayout());
   con.add(panel1);
   con.add(panel2);
   setTitle("查找");
   setSize(300,120);
   setVisible(true);
}
//响应按钮单击事件
public void actionPerformed(ActionEvent e){
   if (e.getSource()==OKBtn){
      find();
   }else{
```

```
        dispose();
      }
    }
}
```

对话框的截图如 14.86 所示：

图 14.86　查找对话框运行截图

findDialog 并不是记事本这个类的内部类，它是一个独立的类，但为了查找用户输入的字符串，它需要访问记事本中的文本，也就是说，它必须访问另外一个类的私有数据。因为 Java 并没有提供类似于 C++那样的友元函数。所以需要用到一个变通的方法。注意它的构造方法：

```
public findDialog(JFrame owner, JTextArea Jtext)
```

第二个参数是一个文本，调用者必须把文本传递进来。然后本类会保存这个文本的一个引用，以后就可以对它进行操作了。

这个类中最关键部分是 find()方法的实现。看下面这个简单的：

```
//本方法没有从前往后找，区分大小写
public void find(){
  int index;
  //指定开始查找的位置，从上次找到的位置或是光标位置开始查找
  if (start>text.getCaretPosition())
        start=text.getCaretPosition();
     //直接利用 String 的匹配功能，但该功能是区分大大小写的
     index=text.getText().indexOf(findEdit.getText(),start);
     if (index<0){
        JOptionPane.showMessageDialog(this,"查找完毕");
        start=0;
     }else{ //如果找到，显示找到的位置
        text.setSelectionStart(index);
        text.setSelectionEnd(index+findEdit.getText().length()-1);
        //设置下一次开始查找的位置
        start=index+findEdit.getText().length();
     }
}
```

读者需要把这个 find()方法添加到前面的 findDialog 类中间。上面这个查找功能并不太完善，请读者自己完成不区分大小写和从后往前找的功能。

14.13.17　“设置字体..”菜单的响应代码

这需要调用上一节介绍的 fontDialog 对话框，程序很简单：

```
void doChangeFont(){
  if (myFontDialog==null)              //myFontDialog 是一个类的实例成员变量
    myFontDialog=new fontDialog(this);//显示对话框
  if(myFontDialog.showFontDialog()==fontDialog.OK)
    text.setFont(myFontDialog.getFont());//获得对话框返回的字体
```

```
}
```

图 14.87 是设置字体前的截图：

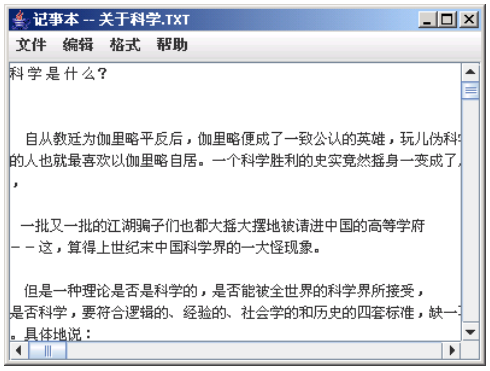

图 14.87　显示默认字体

图 14.88 是设置字体之后的截图：

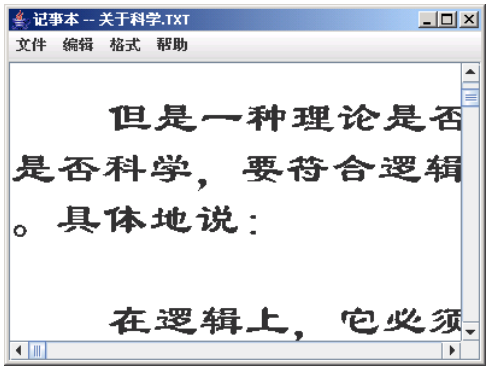

图 14.88　用户设置字体之后截图

到这里程序还有这么几个菜单功能没有完成：撤消、页面设置、替换、帮助主题、关于..，然后还有右键菜单没有提供功能，没有为主菜单指定快捷键。另外本程序还可以进一步改进，比如提供拖放功能、提供工具栏等。这些功能请读者查阅相关书籍和资料自行完成。

14.13.18　完整的程序

最后要将前面介绍的代码拼装成一个完整的程序。这个记事本由 3 个独立的类构成：字体选择对话框（fontDialog）、查找对话框（findDialog）和主类（NoteBook）。另外还有一个系统提供的 FileChooserDemo 类。这 4 个类最好都放在同一个目录下面进行编译，其余 3 个类都需要主类的调用才能运行。为节省篇幅，这两个对话框类不再重写，下面只提供主类的完整代码。

【例 14.43】记事本

//----------文件名 NoteBook.java，程序编号 14.48----------

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

```
import javax.print.attribute.*;
import java.util.*;
import java.text.*;

public class Notebook extends JFrame implements ActionListener,DocumentListener{
  Container   con;
  JScrollPane JSPane;
  JTextArea   text;
  JMenuBar    mainMenuBar;
  JMenu       fileMenu,editMenu,formatMenu,helpMenu;
  JMenuItem   newItem,openItem,saveItem, saveasItem,pageItem,printItem,exitItem;
  JMenuItem   undoItem,cutItem,copyItem,pasteItem,findItem,replaceItem,
              selectallItem,dateItem;
  JCheckBoxMenuItem wrapItem;
  JMenuItem   fontItem;
  JMenuItem   helpItem,aboutItem;
  JPopupMenu  popMenu;
  fontDialog  myFontDialog=null;
  boolean changed=false;
  boolean haveName=false;
  File    file=null;

//主程序入口
  public static void main(String[] args) {
     new Notebook();
  }

//创建界面、安装各种监听器、
  public Notebook() {
    setTitle("记事本 -- 未命名");
    con=getContentPane();
    text=new JTextArea();
    JSPane=new JScrollPane(text);
    createMenu();
    createPopupMenu();
    setJMenuBar(mainMenuBar);
    con.add(JSPane,BorderLayout.CENTER);
    text.setComponentPopupMenu(popMenu);
    text.getDocument().addDocumentListener(this);
    text.addKeyListener(new handleKey());
    text.addMouseListener(new handleMouse());
    setSize(400,300);
    setVisible(true);
  }

//创建主菜单
  public void createMenu(){
    //创建 JMenuBar
    mainMenuBar=new JMenuBar();
    //创建四个 JMenu
```

```
fileMenu=new JMenu("文件");
editMenu=new JMenu("编辑");
formatMenu=new JMenu("格式");
helpMenu=new JMenu("帮助");
//创建 JMenuItem 并添加到对应的 JMenu 中
mainMenuBar.add(fileMenu);
newItem=new JMenuItem("新建");
openItem=new JMenuItem("打开..");
saveItem=new JMenuItem("保存..");
saveasItem=new JMenuItem("另存为..");
pageItem=new JMenuItem("页面设置..");
printItem=new JMenuItem("打印..");
exitItem=new JMenuItem("退出");
fileMenu.add(newItem);
fileMenu.add(openItem);
fileMenu.add(saveItem);
fileMenu.add(saveasItem);
fileMenu.addSeparator();
fileMenu.add(pageItem);
fileMenu.add(printItem);
fileMenu.addSeparator();
fileMenu.add(exitItem);

mainMenuBar.add(editMenu);
undoItem=new JMenuItem("撤消");
cutItem=new JMenuItem("剪切");
copyItem=new JMenuItem("复制");
pasteItem=new JMenuItem("粘贴");
findItem=new JMenuItem("查找..");
replaceItem=new JMenuItem("替换..");
selectallItem=new JMenuItem("全选");
dateItem=new JMenuItem("时间/日期");
editMenu.add(undoItem);
editMenu.addSeparator();
editMenu.add(cutItem);
editMenu.add(copyItem);
editMenu.add(pasteItem);
editMenu.addSeparator();
editMenu.add(findItem);
editMenu.add(replaceItem);
editMenu.addSeparator();
editMenu.add(selectallItem);
editMenu.add(dateItem);

mainMenuBar.add(formatMenu);
wrapItem=new JCheckBoxMenuItem("自动换行");
fontItem=new JMenuItem("设置字体..");
formatMenu.add(wrapItem);
formatMenu.add(fontItem);
mainMenuBar.add(helpMenu);
```

```
    helpItem=new JMenuItem("帮助主题");
    aboutItem=new JMenuItem("关于..");
    helpMenu.add(helpItem);
    helpMenu.add(aboutItem);

    exitItem.addActionListener(this);
    saveItem.addActionListener(this);
    saveasItem.addActionListener(this);
    newItem.addActionListener(this);
    printItem.addActionListener(this);
    openItem.addActionListener(this);
    cutItem.addActionListener(this);
    copyItem.addActionListener(this);
    pasteItem.addActionListener(this);
    selectallItem.addActionListener(this);
    dateItem.addActionListener(this);
    wrapItem.addActionListener(this);
    findItem.addActionListener(this);
    fontItem.addActionListener(this);
  }
//创建弹出式菜单
  public void createPopupMenu(){
    popMenu=new JPopupMenu();
    popMenu.add("撤消");
    popMenu.addSeparator();
    popMenu.add("剪切");
    popMenu.add("复制");
    popMenu.add("粘贴");
    popMenu.addSeparator();
    popMenu.add("全选");
  }

  public void actionPerformed(ActionEvent e){
    Object obj;
    obj=e.getSource();
    if (obj==exitItem)
           doExit();
    else if(obj==saveItem)
           doSave();
    else if(obj==saveasItem)
           doSaveAs();
    else if(obj==newItem)
           doNewFile();
    else if(obj==printItem)
           doPrint();
    else if(obj==openItem)
           doOpen();
    else if(obj==cutItem)
           doCut();
    else if(obj==copyItem)
```

```
            doCopy();
    else if(obj==pasteItem)
            doPaste();
    else if(obj==selectallItem)
            doSelectAll();
    else if(obj==dateItem)
            doDateTime();
    else if(obj==wrapItem)
            doWrap();
    else if (obj==findItem)
            doFind();
    else if (obj==fontItem)
            doChangeFont();
  }

//当用户按下窗口的"关闭"时，会自动调用此方法
  protected void processWindowEvent(WindowEvent e){
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
       doExit();
  }

//监听文本内容的改变事件
  public void changedUpdate(DocumentEvent e){
     //不需要动作
  }

  public void insertUpdate(DocumentEvent e){
     changed=true;
  }

  public void removeUpdate(DocumentEvent e){
     changed=true;
  }

//监听鼠标事件
  class handleMouse extends MouseAdapter{
     public void mouseReleased(MouseEvent e) {chkText();}
  }

//监听键盘事件
  class handleKey extends KeyAdapter{
     public void keyPressed(KeyEvent e) {chkText();}
  }

//根据用户选择文本的情况，修改菜单的状态
  void chkText(){
    if(text.getSelectedText()==null){
       cutItem.setEnabled(false);
       copyItem.setEnabled(false);
    }else{
```

```
      cutItem.setEnabled(true);
      copyItem.setEnabled(true);
    }
  }

//程序退出时的代码
  void doExit(){
    int select;
    if (!changed)
       System.exit(0);
    else{
      select=JOptionPane.showConfirmDialog(this,"文件修改后尚未存盘，要保存吗？");
      switch (select){
        case JOptionPane.YES_OPTION:
              select=doSave();
              if (select==1)System.exit(0);
              break;
        case JOptionPane.NO_OPTION:
              System.exit(0);
              break;
        case JOptionPane.CANCEL_OPTION:
              break;
      }
    }
   }

//保存用户编辑的文件，保存成功返回 1，否则返回 0
  int doSave(){
    FileOutputStream fout;
    byte content[];
    int flag;
    if (!haveName){
           flag = doSaveAs();
    }else if(changed){
      try{
        fout=new FileOutputStream(file);
        content=text.getText().getBytes();
        fout.write(content);
        fout.close();
        changed=false;
        flag = 1;
      }catch(FileNotFoundException e){
         JOptionPane.showMessageDialog(this,"指定的文件名称或属性有问题！");
         flag = 0;
      }catch(IOException e){
         JOptionPane.showMessageDialog(this,"无法写文件，请检查文件是否被锁定");
         flag = 0;
      }
    }else{
        flag =1;
```

```
    }
    return flag;
  }

//用"另存为"对话框保存文件。保存成功返回 1，否则返回 0
  int doSaveAs(){
     FileOutputStream fout;
     byte content[];
     int flag=0;
     File tmpfile=null;
     ExampleFileFilter filter = new ExampleFileFilter();
     JFileChooser  chooser;

     filter.addExtension("txt");
     filter.setDescription("文本文件");
     if (file!=null)
        chooser =  new JFileChooser(file.getPath());
     else
        chooser =  new JFileChooser();
     chooser.setFileFilter(filter);
     flag = chooser.showSaveDialog(this);
     if(flag == JFileChooser.APPROVE_OPTION) {
        tmpfile=chooser.getSelectedFile();
        if (tmpfile.exists()){
          if (JOptionPane.showConfirmDialog(this,"文件已经存在，是否覆盖？",
                       "警告",JOptionPane.YES_NO_OPTION)==JOptionPane.YES_OPTION){
                flag=1;
          }else{
                flag=0;
          }
        }else{
           flag=1;
        }
     }else{
         flag=0;
     }

     if (flag==1){//用户已经确定要以指定名称保存文件
       try{
          fout=new FileOutputStream(tmpfile);
          content=text.getText().getBytes();
          fout.write(content);
          fout.close();
          flag = 1;
       }catch(FileNotFoundException e){
          JOptionPane.showMessageDialog(this,"指定的文件名称或属性有问题！");
          flag = 0;
       }catch(IOException e){
          JOptionPane.showMessageDialog(this,"无法写文件，请检查文件是否被锁定");
          flag = 0;
```

```
      }
     }

     if (flag==1){//文件保存成功，修改相关变量
       changed=false;
       haveName=true;
       file=tmpfile;
       this.setTitle("记事本 -- "+file.getName());
     }
     return flag;
  }

//新建一个文件
  void doNewFile(){
     int select,flag;
     if (changed){
        select=JOptionPane.showConfirmDialog(this,"文件修改后尚未存盘，要保存吗？");
        switch (select){
          case JOptionPane.YES_OPTION:
               flag=doSave();
               break;
          case JOptionPane.NO_OPTION:
               flag=1;
               break;
          default:
               flag=0;
               break;
        }
     }else{
        flag = 1;
     }
     if(flag==1){
        changed=false;
        haveName=false;
        setTitle("记事本 -- 未命名");
        text.setText(null);
     }
  }

//调用打印对话框，给用户打印文档
  void doPrint(){
    try{
       PrintRequestAttributeSet pras = new HashPrintRequestAttributeSet();
       DocFlavor flavor = DocFlavor.BYTE_ARRAY.AUTOSENSE;
       PrintService printService[] = PrintServiceLookup.lookupPrintServices(
                                     flavor, pras);
       PrintService defaultService = PrintServiceLookup.lookupDefaultPrintService();
       PrintService service = null;
       service = ServiceUI.printDialog(null, 100, 100, printService,
                                       defaultService, flavor, pras);
```

```
     if (service!=null){
       DocPrintJob job = service.createPrintJob();
       DocAttributeSet das = new HashDocAttributeSet();
       Doc doc = new SimpleDoc(text.getText().getBytes(), flavor, das);
       job.print(doc, pras); //进行文件的打印
     }
   }catch(Exception e){
         JOptionPane.showMessageDialog(this,"打印任务无法完成");
   }
  }

//打开一个已经存在的文件
  void doOpen(){
     int select,flag;
     File tmpfile=null;
     ExampleFileFilter filter;
     JFileChooser  chooser;
     FileInputStream fin;
     byte   buf[];

    if (changed){
       select=JOptionPane.showConfirmDialog(this,"文件修改后尚未存盘，要保存吗？");
       switch (select){
         case JOptionPane.YES_OPTION:
           flag=doSave();
           break;
         case JOptionPane.NO_OPTION:
           flag=1;
           break;
         default:
           flag=0;
           break;
       }
    }else{
       flag = 1;
    }
    if(flag==1){
       changed = false;
       filter = new ExampleFileFilter();
       filter.addExtension("txt");
       filter.setDescription("文本文件");
       if (file!=null)
         chooser =  new JFileChooser(file.getPath());
       else
         chooser =  new JFileChooser();
       chooser.setFileFilter(filter);
       select = chooser.showOpenDialog(this);
       if(select == JFileChooser.APPROVE_OPTION) {
         tmpfile=chooser.getSelectedFile();
         try{
```

```
            fin=new FileInputStream(tmpfile);
            buf=new byte[(int)tmpfile.length()];
            fin.read(buf);
            fin.close();
            text.setText(new String(buf));
            changed=false;
            haveName=true;
            file=tmpfile;
            setTitle("记事本 -- "+file.getName());
          }catch(FileNotFoundException e){
            JOptionPane.showMessageDialog(this,"指定的文件名称或属性有问题！");
          }catch(IOException e){
            JOptionPane.showMessageDialog(this,"无法读文件，请检查文件是否被锁定");
          }
        }
      }
    }

//将用户选择的文本剪切到剪贴板
  void doCut(){
        text.cut();
  }

//将用户选择的文本复制到剪贴板
  void doCopy(){
        text.copy();
  }

//将剪贴板中的内容复制到文本区
   void doPaste(){
         text.paste();
   }

//全选
   void doSelectAll(){
         text.selectAll();
   }

//插入当前日期和时间
   void doDateTime(){
         SimpleDateFormat sdf = new SimpleDateFormat("HH:mm yyyy-MM-dd");
         text.append(sdf.format(new Date()));
   }

//自动换行
  void doWrap(){
        if(wrapItem.getState()) {
              text.setLineWrap(true);
        }else {
            text.setLineWrap(false);
```

```
        }
  }

//显示查找对话框
  void doFind(){
        new findDialog(this,text);
  }

//设置字体
  void doChangeFont(){
        if     (myFontDialog==null)
              myFontDialog=new fontDialog(this);
        if(myFontDialog.showFontDialog()==fontDialog.OK)
              text.setFont(myFontDialog.getFont());
  }
}//类结束
```

14.14 本章小结

本章全面介绍了如何用 Java 编写 GUI 程序，重点介绍了其中的 Swing 组件，以及与其相关的事件、容器、布局、基本组件等基础知识，并以示例展示了 MVC 编程模式的优势。编写 GUI 程序入门的门槛比较高，但是一旦掌握了其基本模式，编写起来还是很有规律可循。一般的设计过程是：首先设计好布局，选择要使用的组件，安排各种组件的外观，最后为各个交互组件编写事件响应代码。其中的核心部分是编写事件响应代码。

本章最后的几个程序实例都是可以运行的小程序，虽然功能比较简单，但都具备了一个实用程序的基本框架，并且对各种错误情况都做了较为充分的考虑。读者不妨从改进它们入手，提高自己的实际编程能力。